aws

Guida per l'utente

FreeRTOS

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

FreeRTOS: Guida per l'utente

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

I marchi e l'immagine commerciale di Amazon non possono essere utilizzati in relazione a prodotti o servizi che non siano di Amazon, in una qualsiasi modalità che possa causare confusione tra i clienti o in una qualsiasi modalità che denigri o discrediti Amazon. Tutti gli altri marchi non di proprietà di Amazon sono di proprietà delle rispettive aziende, che possono o meno essere associate, collegate o sponsorizzate da Amazon.

Table of Contents

Cos'è FreerTOS? 1
Scaricare il codice sorgente di FreerTOS 1
Controllo delle versioni FreerTOS 1
Architettura FreerTOS 2
Piattaforme hardware certificate FreeRTOS 2
Flusso di lavoro di sviluppo 3
Risorse aggiuntive 4
Aspetti fondamentali del kernel FreeRTOS 5
Pianificatore del kernel FreeRTOS 5
Gestione della memoria 6
Allocazione della memoria del kernel 6
Gestione della memoria dell'applicazione 7
Coordinamento tra attività 7
Queues 7
Semafori e mutex 8
Direct-to-task notifiche 8
Buffer dei flussi 9
Buffer dei messaggi 10
Timer del software 12
Supporto della modalità a basso consumo 12
`FreeRTOSConfig.h` 13
AWS IoT SDK del dispositivo per C integrato 14
Guida introduttiva a FreerTOS 16
Applicazione demo FreerTOS 16
Fase iniziale 17
Configurazione dell'account e delle autorizzazioni AWS 18
Registrazione della scheda MCU con AWS IoT 20
Scaricare FreerTOS 23
Configurazione delle demo di FreerTOS 23
Risoluzione dei problemi 26
Suggerimenti generali sulla risoluzione dei problemi iniziali 26
Installazione di un emulatore di terminale 27
Utilizzo CMake con FreerTOS 28
Prerequisiti 29

Sviluppo di applicazioni FreerTOS con editor di codice e strumenti di debug di terze parti 30
Costruire FreerTOS con CMake 31
Distribuzione delle chiavi in modalità sviluppatore 38
Introduzione 38
Opzione #1: importazione di chiavi private da AWS IoT 38
Opzione 2: Generazione di chiavi private integrate 38
Guide alle operazioni di base specifiche per la scheda 41
Kit di sviluppo Cypress CYW9439 07 AEVAL1 F 42
Kit di sviluppo Cypress CYW9549 07 AEVAL1 F 47
Cypress CKIT-064S0S2-4343W CY8 52
Microchip ATECC6 08A Secure Element con simulatore Windows 59
Espressif - C ESP32 e DevKit ESP-WROVER-KIT 66
Espressif -WROOM-32SE ESP32 110
Kit di connettività IoT Infineon XMC48 00 126
Kit di connettività XMC48 IoT Infineon OPTIGA Trust X e 00 133
Marvell Starter Kit MW32x AWS IoT 141
MediaTek MT7697Kit di sviluppo Hx 163
Microchip Curiosity MZ EF PIC32 168
Nordic n 0-DK RF5284 175
Nuvoton 487 NuMaker-IoT-M 181
Modulo LPC54 IoT NXP 018 189
RX65Renesas Starter Kit+ per N-2MB 194
STMicroelectronics STM32Nodo IoT L4 Discovery Kit 200
Texas Instruments 0SF-LAUNCHXL CC322 203
Windows Device Simulator 209
Kit IoT industriale Xilinx Avnet MicroZed 214
Over-the-Air Aggiornamenti 224
Tagging delle risorse OTA 225
Prerequisiti per l'aggiornamento OTA 225
Crea un bucket Amazon S3 per archiviare l'aggiornamento 226
Creazione di un ruolo del servizio per gli aggiornamenti OTA 227
Creazione di una policy utente OTA 229
Creazione di un certificato di firma del codice 231
Concedi l'accesso alla firma del codice per AWS IoT 242
Scarica FreerTOS con la libreria OTA 243
Prerequisiti per gli aggiornamenti OTA mediante MQTT 243

Prerequisiti per gli aggiornamenti OTA mediante HTTP 247
Tutorial OTA 252
Installazione del firmware iniziale 253
Aggiornare la versione del firmware 267
Creazione di un aggiornamento OTA (console)AWS IoT 268
Creazione di un aggiornamento OTA con AWS CLI 273
Servizio OTA Update Manager 300
Integrazione dell'agente OTA nell'applicazione 301
Gestione delle connessioni 302
Demo OTA semplice 302
Utilizzo di un callback personalizzato per gli eventi di completamento OTA 305
Sicurezza OTA 307
Firma del codice per AWS IoT 308
Risoluzione dei problemi di OTA 309
Configura i CloudWatch registri per gli aggiornamenti OTA 309
Registra le AWS IoT chiamate API OTA con AWS CloudTrail 315
Ottieni i dettagli sull'OTAUpdate errore di Create utilizzando il AWS CLI 318
Ottieni i codici di errore OTA con AWS CLI 320
Risoluzione dei problemi relativi agli aggiornamenti OTA di più dispositivi 322
Risolvi i problemi relativi agli aggiornamenti OTA con il Launchpad 0SF di Texas Instruments CC322 322
Librerie FreerTOS 323
Librerie di porting FreerTOS 323
Librerie di applicazioni FreerTOS 325
Librerie comuni FreerTOS 327
Configurazione delle librerie FreerTOS 327
Librerie comuni 328
Operazioni atomiche 329
Container lineari 330
Registrazione 330
Memoria statica 331
Pool di attività 331
Algoritmo BackOff 336
Introduzione 336
Bluetooth Low Energy 337
Panoramica 337

Architettura 338
Dipendenze e requisiti 341
File di configurazione della libreria 341
Ottimizzazione 342
Limitazioni d'uso 342
Inizializzazione 343
Riferimento API 345
Esempio di utilizzo 345
Portabilità 349
Mobile SDKs per dispositivi Bluetooth FreerTOS 352
I/O comuni 353
AWS IoT Device Defender 354
Introduzione 355
AWS IoT Greengrass 356
Panoramica 356
Dipendenze e requisiti 356
Riferimento API 357
Esempio di utilizzo 357
CoreHTTP 359
Introduzione 359
CoreJSON 360
Introduzione 360
Uso della memoria 360
CoreMQTT 361
Introduzione 361
Agente OTA 363
Panoramica 363
Funzionalità 363
Riferimento API 364
Esempio di utilizzo 364
Portabilità 365
Uso della memoria 365
nucleo PKCS11 366
Panoramica 366
Funzionalità 367
Supporto per sistema di crittografia asimmetrica 368

Portabilità 371
Uso della memoria 371
Secure Sockets 371
Panoramica 371
Dipendenze e requisiti 372
Funzionalità 372
Risoluzione dei problemi 373
Supporto per gli sviluppatori 373
Limitazioni d'uso 373
Inizializzazione 374
Riferimento API 374
Esempio di utilizzo 374
Portabilità 376
AWS IoT Device Shadow 377
Introduzione 377
AWS IoT Lavori 378
Introduzione 378
Transport Layer Security 379
Wi-Fi 379
Panoramica 379
Dipendenze e requisiti 380
Funzionalità 380
Configurazione 382
Inizializzazione 382
Riferimento API 383
Esempio di utilizzo 383
Portabilità 386
Demo FreerTOS 387
Esecuzione delle demo di FreerTOS 387
Configurazione delle demo 388
Bluetooth Low Energy 388
Panoramica 388
Prerequisiti 389
Componenti comuni 392
MQTT su Bluetooth Low Energy 400
Provisioning Wi-Fi 401

Server Generic Attributes 407
Bootloader per Microchip Curiosity MZEF PIC32 410
Stati del bootloader 410
Dispositivo Flash 412
Struttura dell'immagine di un'applicazione 412
Intestazione immagine 413
Descrittore immagine 414
Trailer dell'immagine 415
Configurazione del bootloader 416
Creazione del bootloader 417
AWS IoT Device Defender 418
Introduzione 418
Funzionalità 418
AWS IoT Greengrass 423
Amazon EC2 425
CoreHTTP 427
Autenticazione reciproca CoreHTTP 428
Caricamento CoreHTTP su Amazon S3 430
Scarica CoreHTTP S3 433
CoreHTTP multithread 436
AWS IoT Offerte di lavoro 441
Introduzione 441
Organizzazione del codice sorgente 441
Configura la connessione del broker AWS IoT MQTT 442
Funzionalità 442
CoreMQTT 444
Autenticazione reciproca CoreMQTT 444
Condivisione della connessione CoreMQTT 449
Over-the-air aggiornamenti 454
Over-the-air configurazioni demo 459
Texas Instruments 0SF-LAUNCHXL CC322 460
Microchip Curiosity MZEF PIC32 462
Espressif ESP32 468
Renesas N RX65 469
AWS IoT Device Shadow 494
Introduzione 494

Funzionalità 495
Connect al broker AWS IoT MQTT 497
Eliminare il documento ombra 497
Iscriviti agli argomenti ombra 498
Invia aggiornamenti Shadow 498
Gestisci i messaggi shadow delta e i messaggi shadow update 499
Secure Sockets 500
Utilizzo AWS IoT Device Tester per FreerTOS 502
Versioni supportate di IDT per FreerTOS 503
Ultima versione di IDT per FreerTOS 504
Versioni di IDT precedenti 505
Versioni di IDT non supportate 507
Prerequisiti 511
Scarica FreerTOS 511
Qualifica LTS (qualificazione per FreerTOS che utilizza librerie LTS) 511
Scarica IDT per FreerTOS 512
..... 512
AWS IoT Device Tester politica gestita 514
(Facoltativo) Installa AWS Command Line Interface 515
Preparazione dei test per la prima verifica della scheda del microcontrollore 515
Aggiunta di livelli di porting delle librerie 515
AWS Configura le tue credenziali 515
Crea un pool di dispositivi in IDT per FreerTOS 516
Configurazione delle impostazioni di compilazione, flashing e test 521
Esecuzione dei test Bluetooth Low Energy 534
Prerequisiti 535
Configurazione di Raspberry Pi 535
Configurazione del dispositivo FreerTOS 539
Esecuzione dei test BLE 540
Risoluzione dei problemi relativi ai test BLE 540
Esecuzione della suite di qualificazione FreerTOS 540
Comandi IDT per FreerTOS 544
Test di riqualifica 546
Versioni della suite di test 546
Informazioni su risultati e log 548
Visualizzazione dei risultati 548

Usa IDT per sviluppare ed eseguire le tue suite di test 552
Scarica l'ultima versione di IDT per FreerTOS 553
Flusso di lavoro per la creazione della suite 553
Tutorial: crea ed esegui la suite di test IDT di esempio 554
Tutorial: Sviluppa una semplice suite di test IDT 560
Crea file di configurazione della suite di test IDT 569
Configura la macchina a stati IDT 577
Crea eseguibili per i test case IDT 601
Usa il contesto IDT 609
Configura le impostazioni per i test runner 614
Esegui il debug ed esegui suite di test personalizzate 625
Esamina i risultati e i registri dei test IDT 629
Metriche di utilizzo IDT 635
Risoluzione dei problemi 642
Risoluzione degli errori di configurazione del dispositivo 643
Risoluzione rei problemi relativi agli errori di timeout 654
Funzionalità e costi della rete cellulare AWS 654
Politica di generazione dei report di qualificazione 654
Policy di supporto 655
Sicurezza in AWS 656
Identity and Access Management 657
Destinatari 657
Autenticazione con identità 658
Gestione dell'accesso con policy 661
Come funziona FreerTOS con IAM 664
Esempi di policy basate su identità 671
Risoluzione dei problemi 674
Convalida della conformità 676
Resilienza 677
Sicurezza dell'infrastruttura 678
..... dclxxix

Cos'è FreerTOS?

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Sviluppato in collaborazione con le principali società di chip del mondo per un periodo di 15 anni e ora scaricato ogni 175 secondi, FreerTOS è un sistema operativo in tempo reale (RTOS) leader di mercato per microcontrollori e piccoli microprocessori. Distribuito gratuitamente sotto la licenza open source MIT, FreerTOS include un kernel e un set crescente di librerie adatte all'uso in tutti i settori industriali. FreerTOS è costruito con particolare attenzione all'affidabilità e alla facilità d'uso.

FreerTOS include librerie per connettività, sicurezza over-the-air e aggiornamenti (OTA). FreeRTOS include anche applicazioni demo che mostrano le funzionalità di FreeRTOS su schede qualificate.

FreerTOS è un progetto open source. Puoi scaricare il codice sorgente, contribuire a modifiche o miglioramenti o segnalare problemi sul sito all'indirizzo amazon-freertos. GitHub https://github.com/aws/ Rilasciamo il codice FreerTOS con la licenza open source MIT, in modo da poterlo utilizzare in progetti commerciali e personali.

Accogliamo con favore anche i contributi alla documentazione di FreeRTOS (FreeRTOS User Guide, FreeRTOS Porting Guide e FreeRTOS Qualification Guide). La fonte di markdown per la documentazione è disponibile all'indirizzo. https://github.com/awsdocs/aws-freertos-docs È rilasciato con la licenza Creative Commons (CC BY-ND).

Scaricare il codice sorgente di FreerTOS

Puoi clonare o scaricare FreerTOS da. GitHub Consultare il file README.md per le istruzioni.

Controllo delle versioni FreerTOS

Il kernel e i componenti FreerTOS vengono rilasciati singolarmente e utilizzano il versionamento semantico. Le versioni integrate di FreerTOS vengono rilasciate periodicamente. Tutte le versioni utilizzano la funzione Versioni multiple basata sulla data con il formato AAAAMM.NN, dove:

- Y rappresenta l'anno.
- M rappresenta il mese.
- N rappresenta l'ordine di rilascio entro il mese designato (00 è la prima versione).

Ad esempio, una seconda versione a luglio 2021 sarebbe 202107.01.

In precedenza, le versioni di FreerTOS utilizzavano il controllo delle versioni semantiche per le versioni principali. Sebbene sia passato al controllo delle versioni basato sulla data (FreeRTOS 1.4.8 aggiornato a FreeRTOS Reference Integrations 201906.00), il kernel FreerTOS e AWS ogni singola libreria FreeRTOS mantengono ancora il controllo delle versioni semantiche. Nell'ambito della funzione Versioni multiple semantica, il numero stesso della versione (X.Y.Z) indica se si tratta di una versione principale, secondaria o di una point release. Puoi utilizzare la versione semantica di una libreria per valutare l'ambito e l'impatto di una nuova versione sull'applicazione.

Le versioni LTS vengono mantenute in modo diverso rispetto ad altri tipi di rilascio. Le versioni principali e secondarie vengono aggiornate di frequente con nuove caratteristiche oltre alle risoluzioni dei difetti. Le versioni LTS vengono aggiornate solo con modifiche per risolvere i difetti critici e le vulnerabilità della sicurezza. Nessuna nuova funzionalità viene introdotta in una determinata release LTS dopo l'avvio. Sono mantenute per almeno tre anni di calendario dopo il rilascio e forniscono ai produttori di dispositivi la possibilità di utilizzare una baseline stabile rispetto a una baseline più dinamica rappresentata da rilasci principali e secondari.

Architettura FreerTOS

FreerTOS viene in genere visualizzato sui dispositivi come un'unica immagine compilata con tutti i componenti necessari per le applicazioni del dispositivo. Questa immagine combina le funzionalità per le applicazioni scritte dallo sviluppatore integrato, le librerie software fornite da Amazon, il kernel FreerTOS e i pacchetti di supporto per driver e schede BSPs () per la piattaforma hardware. Indipendentemente dal singolo microcontrollore utilizzato, gli sviluppatori di applicazioni integrate possono aspettarsi le stesse interfacce standardizzate per il kernel FreerTOS e tutte le librerie software FreerTOS.

Piattaforme hardware certificate FreeRTOS

Le seguenti piattaforme hardware sono qualificate per FreerTOS:

- ATECC6Kit di provisioning Zero Touch 08A per AWS IoT

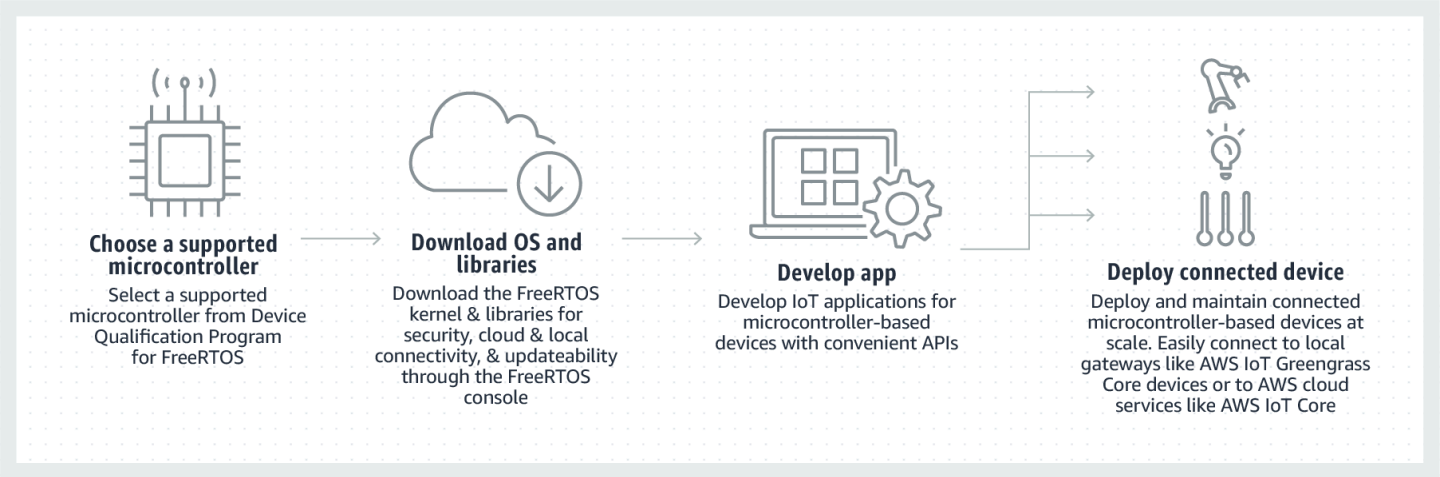
- Kit di sviluppo Cypress CYW9439 07 F AEVAL1
- Kit di sviluppo Cypress CYW9549 07 AEVAL1 F
- Espressif - C ESP32 DevKit
- Espresso ESP-WROVER-KIT
- Kit di connettività IoT Infineon XMC48 00
- Kit iniziale Marvell 0 MW32 AWS IoT
- Kit per principianti Marvell MW322 AWS IoT
- MediaTek MT7697Kit di sviluppo Hx
- Pacchetto Microchip Curiosity PIC32 MZEF
- Nordic in RF5284 0-DK
- NuMaker-IoT-M487
- Modulo LPC54 IoT NXP 018
- Soluzione di sicurezza OPTIGA Trust X
- Modulo IoT Renesas RX65 N RSK
- STMicroelectronicsSTM32Nodo IoT L4 Discovery Kit
- Texas Instruments CC322 0SF-LAUNCHXL
- Microsoft Windows 7 o versioni successive, con almeno un dual core e una connessione Ethernet cablata
- Kit IoT industriale Xilinx Avnet MicroZed

Un elenco dei dispositivi qualificati è disponibile anche nell'AWS Partner Device Catalog.

Per informazioni sulla qualificazione di un nuovo dispositivo, consulta la Guida alla qualificazione di FreerTOS.

Flusso di lavoro di sviluppo

Puoi iniziare lo sviluppo scaricando FreerTOS. Decomprimi il pacchetto e importalo nell'IDE. Puoi quindi sviluppare un'applicazione sulla piattaforma hardware selezionata e produrre e distribuire questi dispositivi utilizzando il processo di sviluppo appropriato per il tuo dispositivo. I dispositivi implementati possono connettersi al AWS IoT servizio o AWS IoT Greengrass come parte di una soluzione IoT completa.



Risorse aggiuntive

Queste risorse possono essere utili per l'utente.

- Documentazione aggiuntiva su FreerTOS è disponibile su freertos.org, incluso il manuale di riferimento FreerTOS v10.0.0.
- Per domande su FreeRTOS per il team di ingegneri di FreeRTOS, puoi aprire un problema nella pagina FreeRTOS. GitHub
- Per domande tecniche su FreerTOS visita i FreerTOS Community Forum.
- Per ulteriori informazioni sulla connessione dei dispositivi a AWS IoT, consulta la Guida per gli AWS IoT Core sviluppatori e il capitolo relativo al Device Provisioning.
- Per il supporto tecnico per AWS, visita il Centro AWS assistenza.
- Per domande su AWS fatturazione, servizi relativi all'account, eventi, abusi o altri problemi relativi AWS, visita la pagina Contattaci.

Aspetti fondamentali del kernel FreeRTOS

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Il kernel FreeRTOS è un sistema operativo in tempo reale che supporta numerose architetture ed è ideale per la creazione di applicazioni microcontroller integrate. Offre:

- Un pianificatore multitasking.
- Numerose opzioni di allocazione di memoria (tra cui la possibilità di creare sistemi completamente allocati staticamente).
- Primitive di coordinamento tra task, che includono notifiche di task, code di messaggi, più tipi di semaforo e buffer dei messaggi e di flusso.

Il kernel FreeRTOS non esegue mai operazioni non deterministiche, ad esempio esplorare un elenco collegato, in una sezione critica o in un interrupt. Il kernel FreeRTOS include un'efficiente implementazione del timer di software che non utilizza tempo CPU, a meno che un timer non necessiti di manutenzione. Le attività bloccate non richiedono una manutenzione periodica dispendiosa in termini di tempo. Direct-to-taskle notifiche consentono una segnalazione rapida delle attività, praticamente senza sovraccarico di RAM. Possono essere utilizzate nella maggior parte degli scenari di intertask e interrupt-to-task segnalazione.

Il kernel FreeRTOS è stato progettato per essere semplice, piccolo e di facile utilizzo. L'immagine binaria del kernel di un RTOS tipica rientra nell'intervallo compreso tra 4.000 e 9.000 byte.

Per la maggior parte della up-to-date documentazione sul kernel FreerTOS, vedere FreerTOS.org. FreeRTOS.org offre una serie di tutorial dettagliati e guide sull'utilizzo del kernel FreeRTOS, tra cui una Guida rapida e la padronanza più approfondita del kernel in tempo reale di FreeRTOS.

Pianificatore del kernel FreeRTOS

Un'applicazione integrata che utilizza un sistema RTOS può essere strutturata come set di task indipendenti. Ogni task viene eseguito all'interno del proprio contesto, senza alcuna dipendenza

da altri task. Nell'applicazione viene eseguito sempre un solo task alla volta. Il pianificatore RTOS in tempo reale determina quando deve essere eseguito ciascun task. Ogni task ha il proprio stack. Quando viene eseguito lo swap out di un task per consentire l'esecuzione di un altro task, il relativo contesto di esecuzione viene salvato nel rispettivo stack, in modo che il task possa essere ripristinato quando ne viene eseguito nuovamente lo swap in per riprenderne l'esecuzione.

Per offrire un comportamento deterministico in tempo reale, il pianificatore di task FreeRTOS permette di assegnare ai task priorità rigorose. RTOS garantisce che venga assegnato tempo di elaborazione ai task con la massima priorità che riesce a eseguire. Ciò comporta che, se sono pronti per essere eseguiti simultaneamente, i task di pari priorità devono condividere il tempo di elaborazione. FreeRTOS inoltre crea un task inattivo che viene eseguito solo quando non ci sono altri task pronti per l'esecuzione.

Gestione della memoria

Questa sezione fornisce informazioni sull'allocazione della memoria kernel e sulla gestione della memoria dell'applicazione.

Allocazione della memoria del kernel

Il kernel RTOS richiede RAM ogni volta che viene creato un task, una coda o un altro oggetto RTOS. La RAM può essere allocata:

- Staticamente in fase di compilazione.
- Dinamicamente dall'heap RTOS dalle funzioni di creazione dell'oggetto API RTOS.

Quando gli oggetti RTOS vengono creati in modo dinamico, l'utilizzo delle funzioni `malloc()` e `free()` della libreria C standard non è sempre appropriato per una serie di motivi.

- Potrebbero non essere disponibili nei sistemi integrati.
- Occupano spazio di codice prezioso.
- Generalmente non sono thread-safe.
- Non sono deterministiche.

Per questi motivi, FreeRTOS mantiene l'API di allocazione della memoria nel livello portatile. Il livello portatile è esterno ai file di origine che implementano la funzionalità di base di RTOS, consentendo di fornire un'implementazione specifica dell'applicazione appropriata per il sistema in tempo reale

che stai sviluppando. Quando il kernel RTOS richiede RAM, chiama `pvPortMalloc()` anziché `malloc()`(). Quando viene liberata RAM, il kernel RTOS chiama `vPortFree()` anziché `free()`.

Gestione della memoria dell'applicazione

Quando le applicazioni hanno bisogno di memoria, possono allocarla dall'heap FreeRTOS. L'heap FreeRTOS offre vari schemi di gestione degli heap di complessità e caratteristiche varie. È anche possibile specificare un'implementazione heap personalizzata.

Il kernel FreeRTOS include cinque implementazioni heap:

`heap_1`

Si tratta del tipo di implementazione più semplice. Non consente che venga liberata memoria.

`heap_2`

Consente che venga liberata memoria, ma non unisce i blocchi liberi adiacenti.

`heap_3`

Racchiude i valori `malloc()` e `free()` standard per la sicurezza per i thread.

`heap_4`

Unisce i blocchi adiacenti per evitare la frammentazione. Include un'opzione di posizionamento dell'indirizzo assoluto.

`heap_5`

È simile a heap_4. Può distribuire l'heap su più aree di memoria non adiacenti.

Coordinamento tra attività

Questa sezione contiene informazioni sulle primitive FreeRTOS.

Queues

Le code sono la principale forma di comunicazione tra task. Esse possono essere utilizzate per lo scambio di messaggi tra i task, e tra interrupt e task. Nella maggior parte dei casi, vengono utilizzate come buffer FIFO (First In First Out) thread-safe con l'inserimento di nuovi dati in coda, ma è possibile anche predisporre un inserimento in testa. I messaggi vengono inviati alle code attraverso

una copia, ciò significa che la coda non si limita ad archiviare un riferimento ai dati, ma vengono copiati nella coda i dati stessi (che possono essere un puntatore ai buffer di dimensioni maggiori).

La coda APIs consente di specificare un periodo di blocco. Quando prova a leggere da una coda vuota, un task viene inserito nello stato bloccato, in modo da non consumare alcun tempo di CPU e da dare la possibilità di mandare in esecuzione altri task, fino a quando nella coda non diventano disponibili dati o fino allo scadere del periodo di blocco. Analogamente, quando prova a scrivere in una coda piena, il task viene inserito nello stato bloccato finché nella coda non si libera spazio o fino allo scadere del periodo di blocco. Se nella stessa coda si bloccano più task, il primo task a essere sbloccato sarà quello con priorità più alta.

Altre primitive FreerTOS, direct-to-task come le notifiche e i buffer di stream e messaggi, offrono alternative leggere alle code in molti scenari di progettazione comuni.

Semafori e mutex

Il kernel FreeRTOS prevede semafori binari, semafori a conteggio e mutex, utilizzati sia per risolvere la mutua esclusione sia per scopi di sincronizzazione.

I semafori binari possono avere solo due valori. Sono la soluzione migliore per implementare la sincronizzazione (tra i task o tra i task e un interrupt). I semafori a conteggio accettano più di due valori e consentono a un numero elevato di task di condividere le risorse o di eseguire operazioni di sincronizzazione più complesse.

I mutex sono semafori binari che includono un meccanismo di ereditarietà della priorità. Questo significa che, se un task a priorità elevata si blocca durante il tentativo di ottenere un mutex attualmente incluso in un task di priorità inferiore, la priorità del task che include il token viene temporaneamente elevata al livello di quella del task che blocca. Questo meccanismo è stato progettato per garantire che il task con priorità maggiore resti nello stato bloccato per il minor tempo possibile, in modo da ridurre al minimo l'inversione di priorità che si verifica.

Direct-to-task notifiche

Le notifiche delle attività consentono alle attività di interagire con altre attività e di sincronizzarsi con le routine del servizio di interruzione (ISRs), senza la necessità di un oggetto di comunicazione separato come un semaforo. Ogni task RTOS presenta un valore di notifica a 32 bit che viene utilizzato per memorizzare eventuale contenuto della notifica. Una notifica di task RTOS è un evento inviato direttamente a un task che può sbloccare il task ricevente e, facoltativamente, aggiornare il rispettivo valore di notifica.

Le notifiche di task RTOS possono essere utilizzate come alternativa semplice e veloce ai semafori a conteggio e binari e, in alcuni casi, alle code. Le notifiche di task presentano vantaggi sia in termini di velocità che di ingombro della RAM rispetto alle altre caratteristiche di FreeRTOS che possono essere utilizzate per eseguire funzionalità equivalenti. Tuttavia, le notifiche dei task possono essere utilizzate solo nei casi un cui un solo task può essere destinatario dell'evento.

Buffer dei flussi

I buffer di flussi consentono a un flusso di byte di passare da una routine di servizi di interrupt a un task o da un task a un altro. Un flusso di byte può essere di lunghezza arbitraria e non ha necessariamente un inizio o una fine. È possibile leggere e scrivere qualsiasi numero di byte contemporaneamente. La funzionalità del buffer di flussi viene abilitata includendo nel progetto il file di origine `stream_buffer.c`.

I buffer di flussi presuppongono che un solo task o un solo interrupt scriva nel buffer (il writer) e che un solo task o un solo interrupt legga dal buffer (il reader). Il writer e il reader possono essere task diversi o diverse routine di servizi di interrupt, ma non è sicuro avere più writer o reader.

L'implementazione dello stream buffer utilizza le notifiche. direct-to-task Pertanto, chiamare un'API del buffer di flussi che inserisce il task chiamante nello stato bloccato può modificare il valore e lo stato della notifica del task.

Invio dei dati

`xStreamBufferSend()` viene utilizzato per inviare dati a un buffer di flussi in un task. `xStreamBufferSendFromISR()` viene utilizzato per inviare dati a un buffer di flussi in una routine di servizi di interrupt (ISR).

`xStreamBufferSend()` consente di specificare un periodo di blocco. Se si chiama `xStreamBufferSend()` con un periodo di blocco diverso da zero per scrivere in un buffer di flussi e il buffer è pieno, il task viene inserito nello stato bloccato finché non si libera spazio o fino allo scadere del periodo di blocco.

`sbSEND_COMPLETED()` e `sbSEND_COMPLETED_FROM_ISR()` sono macro che vengono chiamate (internamente dall'API FreeRTOS) quando vengono scritti dati in un buffer di flussi. Utilizzano l'handle del buffer di flussi che è stato aggiornato. Entrambe le macro verificano la presenza di eventuali task bloccati nel buffer di flussi in attesa di dati e, se presenti, li rimuovono dallo stato bloccato.

È possibile modificare questo comportamento predefinito specificando la propria implementazione di `sbSEND_COMPLETED()` in `FreeRTOSConfig.h`. Questa funzione risulta utile quando

viene utilizzato un buffer di flussi per trasferire dati tra i core in un processore multicore. In tale scenario è possibile implementare `sbSEND_COMPLETED()` per generare un interrupt nell'altro core della CPU e la routine dei servizi di interrupt può quindi utilizzare l'API `xStreamBufferSendCompletedFromISR()` per verificare la presenza di un task in attesa dei dati e, se necessario, sbloccarlo.

Ricezione dei dati

`xStreamBufferReceive()` viene utilizzato per leggere i dati a un buffer di flussi in un task. `xStreamBufferReceiveFromISR()` viene utilizzato per leggere i dati da un buffer di flussi in una routine di servizi di interrupt (ISR).

`xStreamBufferReceive()` consente di specificare un periodo di blocco. Se si chiama `xStreamBufferReceive()` con un periodo di blocco diverso da zero per leggere da un buffer di flussi e il buffer è vuoto, il task viene inserito nello stato bloccato finché nel buffer non diventa disponibile una quantità di dati specificata o fino allo scadere del periodo di blocco.

La quantità di dati che deve trovarsi nel buffer di flussi prima che un task venga sbloccato viene denominata livello di trigger del buffer di flussi. Un task bloccato con un livello di trigger pari a 10 viene sbloccato quando nel buffer vengono scritti almeno 10 byte o allo scadere del periodo di blocco. Se il periodo i blocco del task di lettura scade prima che venga raggiunto il livello di trigger, il task riceve tutti i dati scritti nel buffer. Il livello di trigger di un'attività deve essere impostato su un valore compreso tra 1 e le dimensioni del buffer di flussi. Il livello di trigger di un buffer di flussi viene impostato quando si chiama `xStreamBufferCreate()`. Può essere modificato chiamando `xStreamBufferSetTriggerLevel()`.

`sbRECEIVE_COMPLETED()` e `sbRECEIVE_COMPLETED_FROM_ISR()` sono macro che vengono chiamate (internamente dall'API FreeRTOS) quando vengono letti dati da un buffer di flussi. Le macro verificano la presenza di eventuali task bloccati nel buffer di flussi in attesa che si liberi spazio nel buffer e, se presenti, li rimuovono dallo stato bloccato. È possibile modificare questo comportamento predefinito di `sbRECEIVE_COMPLETED()` specificando un'implementazione alternativa in `FreeRTOSConfig.h`.

Buffer dei messaggi

I buffer di messaggi consentono a singoli messaggi di lunghezza variabile di passare da una routine di servizi di interrupt a un task o da un task a un altro. Ad esempio, i messaggi lunghi 10, 20 e 123 byte possono essere tutti letti da e scritti nello stesso buffer di messaggi. Un messaggio da 10 byte può essere letto solo come messaggio da 10 byte, non come byte singoli. I buffer dei messaggi

vengono creati nella parte superiore dell'implementazione del buffer di flusso. È possibile abilitare la funzionalità di buffer dei messaggi includendo il file di origine `stream_buffer.c` nel progetto.

I buffer di messaggi presuppongono che un solo task o un solo interrupt scriva nel buffer (il writer) e che un solo task o un solo interrupt legga dal buffer (il reader). Il writer e il reader possono essere task diversi o diverse routine di servizi di interrupt, ma non è sicuro avere più writer o reader.

L'implementazione del buffer dei messaggi utilizza direct-to-task le notifiche. Pertanto, chiamare un'API del buffer di flussi che inserisce il task chiamante nello stato bloccato può modificare il valore e lo stato della notifica del task.

Per consentire ai buffer di messaggi di gestire messaggi di dimensioni variabili, prima di ciascun messaggio ne viene scritta la lunghezza. La durata viene memorizzata in una variabile di tipo `size_t`, che in genere in un'architettura a 32 byte è di 4 byte. Pertanto, per scrivere un messaggio di 10 byte in un buffer di messaggi si consumano 14 byte di spazio di buffer effettivi. Analogamente, per scrivere un messaggio di 100 byte in un buffer di messaggi si utilizzano 104 byte di spazio di buffer effettivi.

Invio dei dati

`xMessageBufferSend()` viene utilizzato per inviare dati a un buffer di messaggi da un task. `xMessageBufferSendFromISR()` viene utilizzato per inviare dati a un buffer di messaggi da una routine di servizi di interrupt (ISR).

`xMessageBufferSend()` consente di specificare un periodo di blocco. Se si chiama `xMessageBufferSend()` con un periodo di blocco diverso da zero per scrivere in un buffer di messaggi e il buffer è pieno, il task viene inserito nello stato bloccato finché non si libera spazio nel buffer di messaggi o fino allo scadere del periodo di blocco.

`sbSEND_COMPLETED()` e `sbSEND_COMPLETED_FROM_ISR()` sono macro che vengono chiamate (internamente dall'API FreeRTOS) quando vengono scritti dati in un buffer di flussi. Accettano un parametro singolo, ovvero l'handle del buffer di flussi che è stato aggiornato. Entrambe le macro verificano la presenza di eventuali task bloccati nel buffer di flussi in attesa di dati e, se presenti, li rimuovono dallo stato bloccato.

È possibile modificare questo comportamento predefinito specificando la propria implementazione di `sbSEND_COMPLETED()` in `FreeRTOSConfig.h`. Questa funzione risulta utile quando viene utilizzato un buffer di flussi per trasferire dati tra i core in un processore multicore. In tale scenario è possibile implementare `sbSEND_COMPLETED()` per generare un interrupt

nell'altro core della CPU e la routine dei servizi di interrupt può quindi utilizzare l'API `xStreamBufferSendCompletedFromISR()` per verificare e, se necessario, sbloccare, un task in attesa dei dati.

Ricezione dei dati

`xMessageBufferReceive()` viene utilizzato per leggere i dati da un buffer di messaggi in un task. `xMessageBufferReceiveFromISR()` viene utilizzato per leggere i dati da un buffer di messaggi in una routine di servizio di interrupt (ISR). `xMessageBufferReceive()` consente di specificare un periodo di blocco. Se si chiama `xMessageBufferReceive()` con un periodo di blocco diverso da zero per leggere da un buffer di messaggi e il buffer è vuoto, il task viene inserito nello stato bloccato finché non diventano disponibili dati o fino allo scadere del periodo di blocco.

`sbRECEIVE_COMPLETED()` e `sbRECEIVE_COMPLETED_FROM_ISR()` sono macro che vengono chiamate (internamente dall'API FreeRTOS) quando vengono letti dati da un buffer di flussi. Le macro verificano la presenza di eventuali task bloccati nel buffer di flussi in attesa che si liberi spazio nel buffer e, se presenti, li rimuovono dallo stato bloccato. È possibile modificare questo comportamento predefinito di `sbRECEIVE_COMPLETED()` specificando un'implementazione alternativa in `FreeRTOSConfig.h`.

Timer del software

Un timer del software consente di eseguire una funzione a un'ora futura stabilita. La funzione eseguita dal timer viene denominata la funzione di callback del timer. Il periodo di tempo compreso tra l'avvio di un timer e l'esecuzione della rispettiva funzione di callback viene denominato periodo del timer. Il kernel FreeRTOS offre un'implementazione efficace del timer del software perché:

- Non esegue funzioni di callback del timer da un contesto di interrupt.
- Non consuma tempo di elaborazione, a meno che un timer non sia scaduto.
- Non aggiunge alcun sovraccarico di elaborazione all'interrupt del tick.
- Non esplora nessuna struttura di elenchi di link quando gli interrupt sono disabilitati.

Supporto della modalità a basso consumo

Come la maggior parte dei sistemi operativi integrati, il kernel FreeRTOS utilizza un timer hardware per generare periodici interrupt del tick, il cui scopo è misurare il tempo. Il risparmio di energia delle normali implementazioni dei timer hardware è limitata dalla necessità di uscire e rientrare

periodicamente nella modalità a basso consumo per elaborare gli interrupt dei tick. Se la frequenza dell'interrupt del tick è troppo elevata, l'energia e il tempo consumati per entrare e uscire nella modalità a basso consumo per ogni tick annulla qualsiasi potenziale vantaggio derivante dall'uso di una modalità di risparmio energetico, ad eccezione di quella a più basso consumo.

Per risolvere questa limitazione, FreeRTOS include una modalità del timer senza tick per le applicazioni a basso consumo. La modalità inattiva senza tick FreeRTOS arresta l'interrupt del tick periodico durante i periodi di inattività (periodi in cui non sono presenti task di applicazione in grado di essere eseguiti), quindi apporta un adeguamento correttivo al valore del conteggio dei tick RTOS quando viene riavviato l'interrupt del tick. L'arresto dell'interrupt del tick consente al microcontroller di rimanere in uno stato di risparmio energetico profondo finché non si verifica un interrupt oppure fino a quando il kernel RTOS non deve eseguire la transizione di un task nello stato pronto.

Configurazione dei kernel

È possibile configurare il kernel FreeRTOS per una scheda e un'applicazione specifiche con il file di intestazione `FreeRTOSConfig.h`. Ogni applicazione basata su kernel deve avere un file di intestazione `FreeRTOSConfig.h` nel suo percorso di inclusione del preprocessore. `FreeRTOSConfig.h` è specifico dell'applicazione e deve essere posizionato all'interno della directory di un'applicazione e non in una delle directory del codice sorgente del kernel FreeRTOS.

I `FreeRTOSConfig.h` file per le applicazioni demo e di test di FreerTOS si trovano in e. *`freertos`*`/vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/FreeRTOSConfig.h` *`freertos`*`/vendors/`*`vendor`*`/boards/`*`board`*`/aws_tests/config_files/FreeRTOSConfig.h`

Per un elenco di tutti i parametri di configurazione disponibili da specificare in `FreeRTOSConfig.h`, consulta FreeRTOS.org.

AWS IoT SDK del dispositivo per C integrato

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

> **Note**
>
> Questo SDK è destinato all'uso da parte di sviluppatori di software integrati esperti.

Il SDK per dispositivi AWS IoT per Embedded C (C-SDK) è una raccolta di file sorgente C con licenza open source MIT che possono essere utilizzati in applicazioni integrate per connettere in modo sicuro i dispositivi IoT. AWS IoT Core Include un client MQTT, un client HTTP, JSON Parser e AWS IoT Device Shadow, AWS IoT Jobs e librerie. AWS IoT Device Defender Questo SKD viene distribuito come codice sorgente e può essere integrato nel firmware del cliente con il codice dell'applicazione, altre librerie ed eventualmente un sistema operativo a scelta.

Il SDK per dispositivi AWS IoT per Embedded C (C-SDK) è una raccolta di file sorgente C con licenza open source MIT che possono essere utilizzati in applicazioni integrate per connettere in modo sicuro i dispositivi IoT. AWS IoT Core Include un client MQTT, un client HTTP, JSON Parser e AWS IoT Device Shadow, AWS IoT Jobs, AWS IoT Fleet Provisioning e librerie. AWS IoT Device Defender Questo SKD viene distribuito come codice sorgente e può essere integrato nel firmware del cliente con il codice dell'applicazione, altre librerie ed eventualmente un sistema operativo a scelta.

SDK per dispositivi AWS IoT per Embedded C È generalmente destinato a dispositivi con risorse limitate che richiedono un runtime ottimizzato in linguaggio C. È possibile utilizzare l'SDK su qualsiasi sistema operativo e ospitarlo su qualsiasi tipo di processore (ad esempio, MCUs e). MPUs Tuttavia, se i tuoi dispositivi dispongono di memoria e risorse di elaborazione sufficienti, ti consigliamo di utilizzare un AWS IoT dispositivo SDKs di ordine superiore.

Per ulteriori informazioni, consulta gli argomenti seguenti:

- AWS IoT Device SDK per Embedded C attivo GitHub

- AWS IoT SDK del dispositivo per il file Readme C incorporato
- AWS IoT SDK del dispositivo per esempi C incorporati

Guida introduttiva a FreerTOS

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Questo tutorial Getting Started with FreerTOS mostra come scaricare e configurare FreerTOS su una macchina host, quindi compilare ed eseguire una semplice applicazione demo su una scheda microcontrollore qualificata.

In questo tutorial, supponiamo che tu conosca e conosca la console. AWS IoT AWS IoT In caso contrario, consigliamo di completare il tutorial sulle nozioni di base su AWS IoT prima di continuare.

Argomenti:

- Applicazione demo FreerTOS
- Fase iniziale
- Nozioni di base sulla risoluzione dei problemi
- Utilizzo CMake con FreerTOS
- Distribuzione delle chiavi in modalità sviluppatore
- Guide alle operazioni di base specifiche per la scheda

Applicazione demo FreerTOS

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

L'applicazione demo di questo tutorial è la demo di CoreMQTT Mutual Authentication definita nel file. *freertos*`/demos/coreMQTT/mqtt_demo_mutual_auth.c` La utilizza libreria CoreMQTT

per connettersi al AWS Cloud e quindi pubblicare periodicamente messaggi su un argomento MQTT ospitato dal broker MQTT.AWS IoT

È possibile eseguire solo una singola applicazione demo FreerTOS alla volta. Quando si crea un progetto demo FreerTOS, la prima demo abilitata nel file header è *freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h l'applicazione che viene eseguita. Per questo tutorial, non è necessario abilitare o disabilitare le demo. La demo di autenticazione reciproca di CoreMQTT è abilitata per impostazione predefinita.

Per ulteriori informazioni sulle applicazioni demo incluse in FreerTOS, vedere. Demo FreerTOS

Fase iniziale

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Per iniziare a utilizzare FreerTOS AWS IoT con, devi disporre di AWS un account, un utente IAM con autorizzazione all'accesso AWS IoT e servizi cloud FreeRTOS. Devi anche scaricare FreerTOS e configurare il progetto demo FreeRTOS della tua scheda su cui lavorare. AWS IoT Le seguenti sezioni ti forniscono istruzioni dettagliate per tutti e tre i requisiti.

> **Note**
>
> - Se stai usando Espressif ESP32 - DevKit C o ESP32 -WROOM-32SE ESP-WROVER-KIT, salta questi passaggi e vai a. Guida introduttiva all'Espressif ESP32 - DevKit C e al ESP-WROVER-KIT
> - Se stai usando Nordic n 0-DK, salta questi passaggi e vai a. RF5284 Guida introduttiva al Nordic n 0-DK RF5284

1. Configurazione dell'account e delle autorizzazioni AWS
2. Registrazione della scheda MCU con AWS IoT
3. Scaricare FreerTOS

4. Configurazione delle demo di FreerTOS

Configurazione dell'account e delle autorizzazioni AWS

Iscriviti per un Account AWS

Se non ne hai uno Account AWS, completa i seguenti passaggi per crearne uno.

Per iscriverti a un Account AWS

1. Apri la https://portal.aws.amazon.com/billing/registrazione.
2. Segui le istruzioni online.

   Parte della procedura di registrazione prevede la ricezione di una telefonata o di un messaggio di testo e l'immissione di un codice di verifica sulla tastiera del telefono.

   Quando ti iscrivi a un Account AWS, Utente root dell'account AWSviene creato un. L'utente root dispone dell'accesso a tutte le risorse e tutti i Servizi AWS nell'account. Come best practice di sicurezza, assegna l'accesso amministrativo a un utente e utilizza solo l'utente root per eseguire attività che richiedono l'accesso di un utente root.

AWS ti invia un'email di conferma dopo il completamento della procedura di registrazione. In qualsiasi momento, puoi visualizzare l'attività corrente del tuo account e gestirlo accedendo a https://aws.amazon.com/e scegliendo Il mio account.

Crea un utente con accesso amministrativo

Dopo esserti registrato Account AWS, proteggi Utente root dell'account AWS AWS IAM Identity Center, abilita e crea un utente amministrativo in modo da non utilizzare l'utente root per le attività quotidiane.

Proteggi i tuoi Utente root dell'account AWS

1. Accedi AWS Management Consolecome proprietario dell'account scegliendo Utente root e inserendo il tuo indirizzo Account AWS email. Nella pagina successiva, inserisci la password.

   Per informazioni sull'accesso utilizzando un utente root, consulta la pagina Signing in as the root user della Guida per l'utente di Accedi ad AWS .
2. Abilita l'autenticazione a più fattori (MFA) per l'utente root.

Per istruzioni, consulta Abilitare un dispositivo MFA virtuale per l'utente Account AWS root (console) nella Guida per l'utente IAM.

Crea un utente con accesso amministrativo

1. Abilita Centro identità IAM.

   Per istruzioni, consulta Abilitazione di AWS IAM Identity Center nella Guida per l'utente di AWS IAM Identity Center .

2. In IAM Identity Center, assegna l'accesso amministrativo a un utente.

   Per un tutorial sull'utilizzo di IAM Identity Center directory come fonte di identità, consulta Configurare l'accesso utente con l'impostazione predefinita IAM Identity Center directory nella Guida per l'AWS IAM Identity Center utente.

Accesso come utente amministratore

- Per accedere con l'utente IAM Identity Center, utilizza l'URL di accesso che è stato inviato al tuo indirizzo e-mail quando hai creato l'utente IAM Identity Center.

  Per informazioni sull'accesso utilizzando un utente IAM Identity Center, consulta AWS Accedere al portale di accesso nella Guida per l'Accedi ad AWS utente.

Assegna l'accesso a ulteriori utenti

1. In IAM Identity Center, crea un set di autorizzazioni conforme alla best practice dell'applicazione di autorizzazioni con il privilegio minimo.

   Segui le istruzioni riportate nella pagina Creazione di un set di autorizzazioni nella Guida per l'utente di AWS IAM Identity Center .

2. Assegna al gruppo prima gli utenti e poi l'accesso con autenticazione unica (Single Sign-On).

   Per istruzioni, consulta Aggiungere gruppi nella Guida per l'utente di AWS IAM Identity Center .

Per fornire l'accesso, aggiungi autorizzazioni agli utenti, gruppi o ruoli:

- Utenti e gruppi in AWS IAM Identity Center:

Crea un set di autorizzazioni. Segui le istruzioni riportate nella pagina Create a permission set (Creazione di un set di autorizzazioni) nella Guida per l'utente di AWS IAM Identity Center .

- Utenti gestiti in IAM tramite un provider di identità:

  Crea un ruolo per la federazione delle identità. Segui le istruzioni riportate nella pagina Create a role for a third-party identity provider (federation) della Guida per l'utente IAM.

- Utenti IAM:
  - Crea un ruolo che l'utente possa assumere. Segui le istruzioni riportate nella pagina Create a role for an IAM user della Guida per l'utente IAM.
  - (Non consigliato) Collega una policy direttamente a un utente o aggiungi un utente a un gruppo di utenti. Segui le istruzioni riportate nella pagina Aggiunta di autorizzazioni a un utente (console) nella Guida per l'utente IAM.

Dopo aver configurato l' AWS account e le autorizzazioni, continua conRegistrazione della scheda MCU con AWS IoT.

Registrazione della scheda MCU con AWS IoT

La tua scheda deve essere registrata AWS IoT per comunicare con il AWS Cloud. Per registrare la tua bacheca AWS IoT, devi avere:

Una AWS IoT politica

La AWS IoT politica concede al dispositivo le autorizzazioni per accedere AWS IoT alle risorse. È archiviata nel cloud. AWS

Qualsiasi AWS IoT cosa

Qualsiasi AWS IoT cosa ti permette di gestire i tuoi dispositivi in AWS IoT. È archiviato sul AWS cloud.

Una chiave privata e un certificato X.509

La chiave privata e il certificato consentono al dispositivo di autenticarsi con AWS IoT.

Per registrare la tua scheda, segui le procedure riportate di seguito.

Per creare una AWS IoT politica

1. Per creare una policy IAM, devi conoscere la tua AWS regione e il numero di AWS account.

   Per trovare il numero del tuo AWS account, apri la console di AWS gestione, individua ed espandi il menu sotto il nome del tuo account nell'angolo in alto a destra e seleziona Il mio account. Il tuo ID account viene visualizzato in Account Settings (Impostazioni account).

   Per trovare la AWS regione del tuo AWS account, usa il. AWS Command Line Interface Per installare il AWS CLI, segui le istruzioni contenute nella Guida AWS Command Line Interface per l'utente. Dopo aver installato AWS CLI, aprite una finestra del prompt dei comandi e immettete il seguente comando:

   ```
   aws iot describe-endpoint
   ```

   L'output dovrebbe essere simile al seguente:

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

   Nell'esempio, la regione è `us-west-2`.
2. Passare alla console AWS IoT.
3. Nel riquadro di navigazione, selezionare Secure (Sicurezza), scegliere Policies (Policy) e poi Create (Crea).
4. Inserire un nome per identificare la policy.
5. Nella sezione Add statements (Aggiungi istruzioni), scegliere Advanced mode (Modalità avanzata). Copiare e incollare il seguente JSON nella finestra dell'editor policy. Sostituisci *`aws-region`* e *`aws-account`* con la tua AWS regione e l'ID dell'account.

   ```
   {
       "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:*"
    }
    ]
}
```

La policy concede le seguenti autorizzazioni:

`iot:Connect`

Concede al dispositivo l'autorizzazione a connettersi al broker di AWS IoT messaggi con qualsiasi ID cliente.

`iot:Publish`

Concede al dispositivo l'autorizzazione per pubblicare un messaggio MQTT su qualsiasi argomento MQTT.

`iot:Subscribe`

Concede al dispositivo l'autorizzazione per eseguire la sottoscrizione a qualsiasi filtro di argomenti MQTT.

`iot:Receive`

Concede al dispositivo l'autorizzazione per ricevere messaggi dal broker di messaggi AWS IoT su qualsiasi argomento MQTT.

6. Scegli Create (Crea).

Per creare un oggetto IoT, una chiave privata e un certificato per il tuo dispositivo

1. Passare alla console AWS IoT.

2. Nel riquadro di navigazione scegliere Manage (Gestisci), quindi Things (Oggetti).
3. Se non si dispone di oggetti IoT registrati nel proprio account, compare la pagina You don't have any things yet (Non hai ancora oggetti). Se si consulta questa pagina, scegliere Register a thing (Registra un oggetto). In caso contrario, scegliere Create (Crea).
4. Nella pagina Creazione di AWS IoT oggetti, scegli Crea una singola cosa.
5. Sulla pagina Add your device to the thing registry (Aggiungi il tuo dispositivo al registro degli oggetti), immettere un nome per il proprio oggetto, quindi scegliere Next (Successivo).
6. Sulla pagina Add a certificate for your thing (Aggiungi un certificato per il tuo oggetto), sotto One-click certificate creation (Creazione di un certificato con un clic), scegliere Create certificate (Crea certificato).
7. Scaricare la chiave privata e il certificato scegliendo i link Download (Scarica) per ognuno.
8. Scegliere Activate (Attiva) per attivare il certificato. I certificati devono essere attivati prima dell'uso.
9. Scegli Allega una policy per allegare al tuo certificato una policy che garantisca al tuo dispositivo l'accesso alle AWS IoT operazioni.
10. Scegliere la policy appena creata e poi Register thing (Registra l'oggetto).

Dopo aver registrato la tua scheda AWS IoT, puoi continuare a Scaricare FreerTOS farlo.

Scaricare FreerTOS

Puoi scaricare FreerTOS dal repository FreerTOS. GitHub

> **Note**
>
> Se stai iniziando a usare i kit di sviluppo Cypress CYW9549 07 AEVAL1 F o CYW9439 07 AEVAL1 F, devi scaricare FreerTOS da. GitHub Consultare il file README.md per le istruzioni. Le configurazioni di FreerTOS per queste schede non sono attualmente disponibili nella console FreerTOS.

Configurazione delle demo di FreerTOS

È necessario modificare alcuni file di configurazione nella directory FreerTOS prima di poter compilare ed eseguire qualsiasi demo sulla scheda.

Per configurare il tuo endpoint AWS IoT

Devi fornire a FreerTOS AWS IoT il tuo endpoint in modo che l'applicazione in esecuzione sulla tua scheda possa inviare richieste all'endpoint corretto.

1. Passare alla console AWS IoT.
2. Nel pannello di navigazione scegli Impostazioni.

   L'endpoint viene visualizzato AWS IoT in Endpoint. L'URL dovrebbe essere del tipo *1234567890123*`-ats.iot.`*us-east-1*`.amazonaws.com`. Prendere nota di questo endpoint.
3. Nel riquadro di navigazione scegliere Manage (Gestisci), quindi Things (Oggetti).

   Il dispositivo deve avere un nome per l' AWS IoT oggetto. Prendere nota di questo nome.
4. Aprire `demos/include/aws_clientcredential.h`.
5. Specificare i valori per le seguenti costanti :

   - `#define clientcredentialMQTT_BROKER_ENDPOINT "`*`Your AWS IoT endpoint`*`";`
   - `#define clientcredentialIOT_THING_NAME "`*`The AWS IoT thing name of your board`*`"`

Per configurare il Wi-Fi

Se la scheda si connette a Internet tramite una connessione Wi-Fi, è necessario fornire a FreerTOS le credenziali Wi-Fi per connettersi alla rete. Se la scheda non supporta il Wi-Fi, è possibile ignorare questi passaggi.

1. `demos/include/aws_clientcredential.h`.
2. Specificare i valori per le seguenti costanti `#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`*

     I tipi di sicurezza validi sono:

     - `eWiFiSecurityOpen` (Aperto, nessuna protezione)

- `eWiFiSecurityWEP` (Sicurezza WEP)
- `eWiFiSecurityWPA` (Sicurezza WPA)
- `eWiFiSecurityWPA2`(sicurezza) WPA2

Per formattare le tue AWS IoT credenziali

FreerTOS deve avere AWS IoT il certificato e le chiavi private associati all'oggetto registrato e alle relative politiche di autorizzazione per comunicare AWS IoT con successo per conto del dispositivo.

> **Note**
>
> Per configurare AWS IoT le credenziali, è necessario disporre della chiave privata e del certificato scaricati dalla AWS IoT console al momento della registrazione del dispositivo. Dopo aver registrato il dispositivo come dispositivo, AWS IoT puoi recuperare i certificati del dispositivo dalla AWS IoT console, ma non puoi recuperare le chiavi private.

FreerTOS è un progetto in linguaggio C e il certificato e la chiave privata devono essere formattati in modo speciale per essere aggiunti al progetto.

1. In una finestra del browser, aprire `tools/certificate_configuration/CertificateConfigurator.html`.
2. In File PEM del certificato, scegliere *`ID`*`-certificate.pem.crt` scaricato dalla console AWS IoT .
3. In File PEM della chiave privata, scegliere la *`ID`*`-private.pem.key` scaricata dalla console AWS IoT .
4. Scegliere Generate and save aws_clientcredential_keys.h (Genera e salva aws_clientcredential_keys.h) e salvare il file in `demos/include`. Questa operazione sovrascrive il file esistente nella directory.

   > **Note**
   >
   > Il certificato e la chiave privata sono hardcoded solo a scopo dimostrativo. Le applicazioni a livello di produzione devono archiviare questi file in un percorso sicuro.

Dopo aver configurato FreerTOS, puoi continuare con la guida introduttiva per la tua scheda madre per configurare l'hardware della piattaforma e il relativo ambiente di sviluppo software, quindi compilare ed eseguire la demo sulla tua scheda. Per istruzioni specifiche sulla scheda, consulta la documentazione Guide alle operazioni di base specifiche per la scheda. L'applicazione demo utilizzata nel tutorial Getting Started è la demo di CoreMQTT Mutual Authentication, che si trova all'indirizzo. `demos/coreMQTT/mqtt_demo_mutual_auth.c`

Nozioni di base sulla risoluzione dei problemi

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

I seguenti argomenti possono aiutarti a risolvere i problemi che riscontri quando inizi a usare FreerTOS:

Argomenti

- Suggerimenti generali sulla risoluzione dei problemi iniziali
- Installazione di un emulatore di terminale

Per informazioni sulla risoluzione di problemi specifici per la scheda, consulta la Guida introduttiva a FreerTOS guida relativa alla tua scheda.

Suggerimenti generali sulla risoluzione dei problemi iniziali

Non viene visualizzato alcun messaggio nella AWS IoT console dopo aver eseguito il progetto demo Hello World. Cosa devo fare?

Eseguire quanto segue:

1. Aprire la finestra di un terminale per visualizzare l'output di registrazione dell'esempio. Questo può aiutare a determinare la causa di eventuali errori.
2. Verificare che le credenziali di rete siano valide.

I log visualizzati nel mio terminale durante l'esecuzione di una demo vengono troncati. Come posso aumentarne la lunghezza?

Aumentate il valore `configLOGGING_MAX_MESSAGE_LENGTH` a 255 nel `FreeRTOSconfig.h` file della demo in esecuzione:

```
#define configLOGGING_MAX_MESSAGE_LENGTH    255
```

Installazione di un emulatore di terminale

Un emulatore di terminale può aiutarti a diagnosticare i problemi o a verificare che il codice del dispositivo venga eseguito correttamente. Ci sono vari emulatori di terminale disponibili per macOS, Windows e Linux.

È necessario collegare la scheda al computer prima di tentare di stabilire una connessione seriale per la scheda con un emulatore di terminale.

Usa le seguenti impostazioni per configurare l'emulatore di terminale:

Impostazione del terminale	Valore
BAUD	115200
Dati	8 bit
Parità	nessuno
Stop	1 bit
Controllo di flusso	nessuno

Trovare la porta seriale della scheda

Se non conosci la porta seriale della scheda, puoi inviare uno dei seguenti comandi dalla riga di comando o terminale per ottenere le porte seriali per tutti i dispositivi connessi al tuo computer host:

Windows

```
chgport
```

Linux

```
ls /dev/tty*
```

macOS

```
ls /dev/cu.*
```

Utilizzo CMake con FreerTOS

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Puoi usarlo CMake per generare file di build del progetto dal codice sorgente dell'applicazione FreerTOS e per creare ed eseguire il codice sorgente.

Puoi anche utilizzare un IDE per modificare, eseguire il debug, compilare, eseguire il flashing ed eseguire il codice su dispositivi certificati FreeRTOS. Ogni guida alle operazioni di base specifiche della scheda include istruzioni per configurare l'IDE per una determinata piattaforma. Se preferisci lavorare senza un IDE, puoi utilizzare altri strumenti di modifica e debug del codice di terze parti per lo sviluppo e il debug del codice, e quindi utilizzarli per creare ed eseguire le applicazioni. CMake

Le seguenti schede supportano: CMake

- Espressif - C ESP32 DevKit
- Espresso ESP-WROVER-KIT
- Kit di connettività IoT Infineon XMC48 00
- Kit iniziale Marvell 0 MW32 AWS IoT
- Kit per principianti Marvell MW322 AWS IoT
- Pacchetto Microchip Curiosity MZEF PIC32
- Kit di sviluppo SDK Nordic n 10 RF5284

- STMicroelectronicsSTM32Nodo IoT L4 Discovery Kit
- Texas Instruments CC322 0SF-LAUNCHXL
- Simulatore Microsoft Windows

Consulta gli argomenti seguenti per ulteriori informazioni sull'utilizzo CMake con FreerTOS.

Argomenti

- Prerequisiti
- Sviluppo di applicazioni FreerTOS con editor di codice e strumenti di debug di terze parti
- Costruire FreerTOS con CMake

Prerequisiti

Assicurati che il computer host soddisfi i seguenti prerequisiti prima di continuare:

- La toolchain di compilazione del dispositivo deve supportare il sistema operativo della macchina. CMake supporta tutte le versioni di Windows, macOS e Linux

  Il sottosistema Windows per Linux (WSL) non è supportato. Utilizza la versione nativa CMake su macchine Windows.
- È necessario che sia installata CMake la versione 3.13 o successiva.

  È possibile scaricare la distribuzione binaria di CMake CMakeda.org.

  > **Note**
  >
  > Se scaricate la distribuzione binaria di CMake, assicuratevi di aggiungere l' CMake eseguibile alla variabile di ambiente PATH prima di utilizzarla CMake dalla riga di comando.

  Puoi anche scaricare e installare CMake usando un gestore di pacchetti, come homebrew su macOS e scoop o chocolatey su Windows.

> **Note**
>
> Le versioni dei CMake pacchetti fornite nei gestori di pacchetti per molte distribuzioni Linux sono. out-of-date Se il gestore di pacchetti della tua distribuzione non fornisce la versione più recente di CMake, puoi provare gestori di pacchetti alternativi, come `linuxbrew` `onix`.

- Devi disporre di un sistema di compilazione nativo compatibile.

  CMake può indirizzarsi a molti sistemi di build nativi, inclusi GNU Make o Ninja. Make e Ninja possono essere entrambi installati con il programma di gestione dei pacchetti su Linux, macOS e Windows. Se utilizzi Make su Windows, puoi installare una versione autonoma da Equation o installare MinGW, che include Make.

  > **Note**
  >
  > L'eseguibile Make in MinGW è chiamato `mingw32-make.exe`, anziché `make.exe`.

  Ti consigliamo di utilizzare Ninja, perché è più veloce di Make e fornisce anche supporto nativo per tutti i sistemi operativi desktop.

Sviluppo di applicazioni FreerTOS con editor di codice e strumenti di debug di terze parti

Puoi utilizzare un editor di codice e un'estensione di debug o uno strumento di debug di terze parti per sviluppare applicazioni per FreerTOS.

Se, ad esempio, utilizzi Visual Studio Code come editor di codice, puoi installare l'estensione Cortex-Debug di VS Code come strumento di debug. Al termine dello sviluppo dell'applicazione, puoi richiamare lo strumento da riga di CMake comando per creare il tuo progetto dall'interno di VS Code. Per ulteriori informazioni sull'utilizzo CMake per creare applicazioni FreerTOS, vedere. Costruire FreerTOS con CMake

Per le operazioni di debug, puoi specificare una configurazione di debug simile alla seguente:

```
"configurations": [
    {
        "name": "Cortex Debug",
```

```
        "cwd": "${workspaceRoot}",
        "executable": "./build/st/stm32l475_discovery/aws_demos.elf",
        "request": "launch",
        "type": "cortex-debug",
        "servertype": "stutil"
    }
]
```

Costruire FreerTOS con CMake

CMake utilizza come sistema di destinazione il sistema operativo host come sistema di destinazione per impostazione predefinita. Per utilizzarlo per la compilazione incrociata, è CMake necessario un file toolchain, che specifica il compilatore che si desidera utilizzare. In FreerTOS, forniamo file di toolchain predefiniti in. *freertos*`/tools/cmake/toolchains` Il modo in cui fornire questo file CMake dipende dal fatto che si stia utilizzando l'interfaccia a riga di CMake comando o la GUI. Per ulteriori dettagli, segui le istruzioni Generazione di file di build (strumento da riga di CMake comando) di seguito. Per ulteriori informazioni sulla compilazione incrociata in CMake, consulta il CrossCompilingwiki ufficiale. CMake

Per creare un progetto basato CMake

1. Esegui CMake per generare i file di build per un sistema di build nativo, come Make o Ninja.

   Puoi utilizzare lo strumento da CMake riga di comando o la CMake GUI per generare i file di build per il tuo sistema di compilazione nativo.

   Per informazioni sulla generazione di file build FreerTOS, consulta e. Generazione di file di build (strumento da riga di CMake comando) Generazione di file di build (GUI) CMake
2. Richiamare il sistema di compilazione nativo per rendere eseguibile il progetto.

   Per informazioni sulla creazione di file build per FreerTOS, consulta. Creazione di FreerTOS da file di build generati

Generazione di file di build (strumento da riga di CMake comando)

Puoi usare lo strumento da CMake riga di comando (cmake) per generare file di build per FreerTOS. Per generare i file di compilazione, devi specificare una scheda di destinazione, un compilatore e la posizione del codice sorgente e della directory di compilazione.

Puoi usare le seguenti opzioni per cmake:

- `-DVENDOR`— Specifica la scheda di destinazione.
- `-DCOMPILER`— specifica il compilatore.
- `-S`— specifica la posizione del codice sorgente.
- `-B`— specifica la posizione dei file di build generati.

> **Note**
>
> Il compilatore deve trovarsi nella variabile `PATH`. In alternativa, è necessario specificare la posizione del compilatore.

Ad esempio, se il fornitore è Texas Instruments e la scheda è CC322 0 Launchpad e il compilatore è GCC for ARM, puoi eseguire il seguente comando per creare i file sorgente dalla directory corrente in una directory denominata: *`build-directory`*

```
cmake -DVENDOR=ti -DBOARD=cc3220_launchpad -DCOMPILER=arm-ti -S . -B build-directory
```

> **Note**
>
> Se utilizzi Windows, devi specificare il sistema di compilazione nativo perché CMake utilizza Visual Studio per impostazione predefinita. Per esempio:
>
> ```
> cmake -DVENDOR=ti -DBOARD=cc3220_launchpad -DCOMPILER=arm-ti -S . -B build-directory -G Ninja
> ```
>
> O:
>
> ```
> cmake -DVENDOR=ti -DBOARD=cc3220_launchpad -DCOMPILER=arm-ti -S . -B build-directory -G "MinGW Makefiles"
> ```

Le espressioni regolari `${VENDOR}.*` e `${BOARD}.*` vengono utilizzate per cercare una scheda corrispondente, pertanto non è necessario utilizzare i nomi completi del fornitore e della scheda per le opzioni `VENDOR` e `BOARD`. È possibile utilizzare nomi parziali, purché esista una sola corrispondenza. Ad esempio, i seguenti comandi generano gli stessi file di compilazione dalla stessa origine:

```
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
```

Puoi utilizzare l'opzione `CMAKE_TOOLCHAIN_FILE` se desideri utilizzare un file di toolchain che non si trova nella directory predefinita `cmake/toolchains`. Per esempio:

```
cmake -DBOARD=cc3220 -DCMAKE_TOOLCHAIN_FILE='/path/to/toolchain_file.cmake' -S . -
B build-directory
```

Se il file della toolchain non utilizza percorsi assoluti per il compilatore e non hai aggiunto il compilatore alla variabile di `PATH` ambiente, CMake potresti non riuscire a trovarlo. Per assicurarti che CMake trovi il tuo file della toolchain, puoi usare l'opzione. `AFR_TOOLCHAIN_PATH` Questa opzione esegue una ricerca nel percorso di directory della toolchain specificato e nella sottocartella della toolchain in `bin`. Per esempio:

```
cmake -DBOARD=cc3220 -DCMAKE_TOOLCHAIN_FILE='/path/to/toolchain_file.cmake' -
DAFR_TOOLCHAIN_PATH='/path/to/toolchain/' -S . -B build-directory
```

Per abilitare il debug, imposta `CMAKE_BUILD_TYPE` su `debug`. Con questa opzione abilitata, CMake aggiunge flag di debug alle opzioni di compilazione e crea FreerTOS con simboli di debug.

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

Puoi anche impostare `CMAKE_BUILD_TYPE` su `release` per aggiungere flag di ottimizzazione alle opzioni di compilazione.

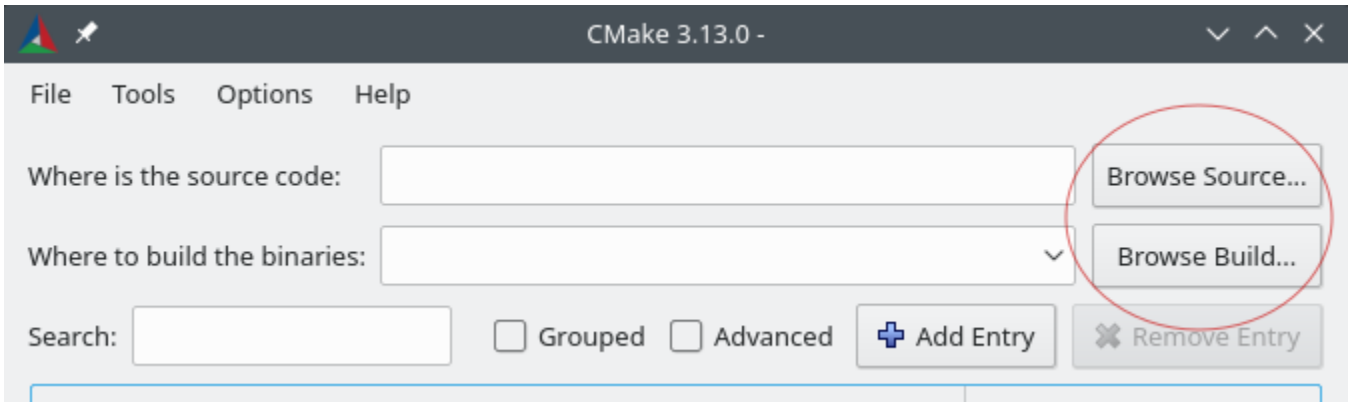
Generazione di file di build (GUI) CMake

Puoi usare la CMake GUI per generare file di build FreerTOS.

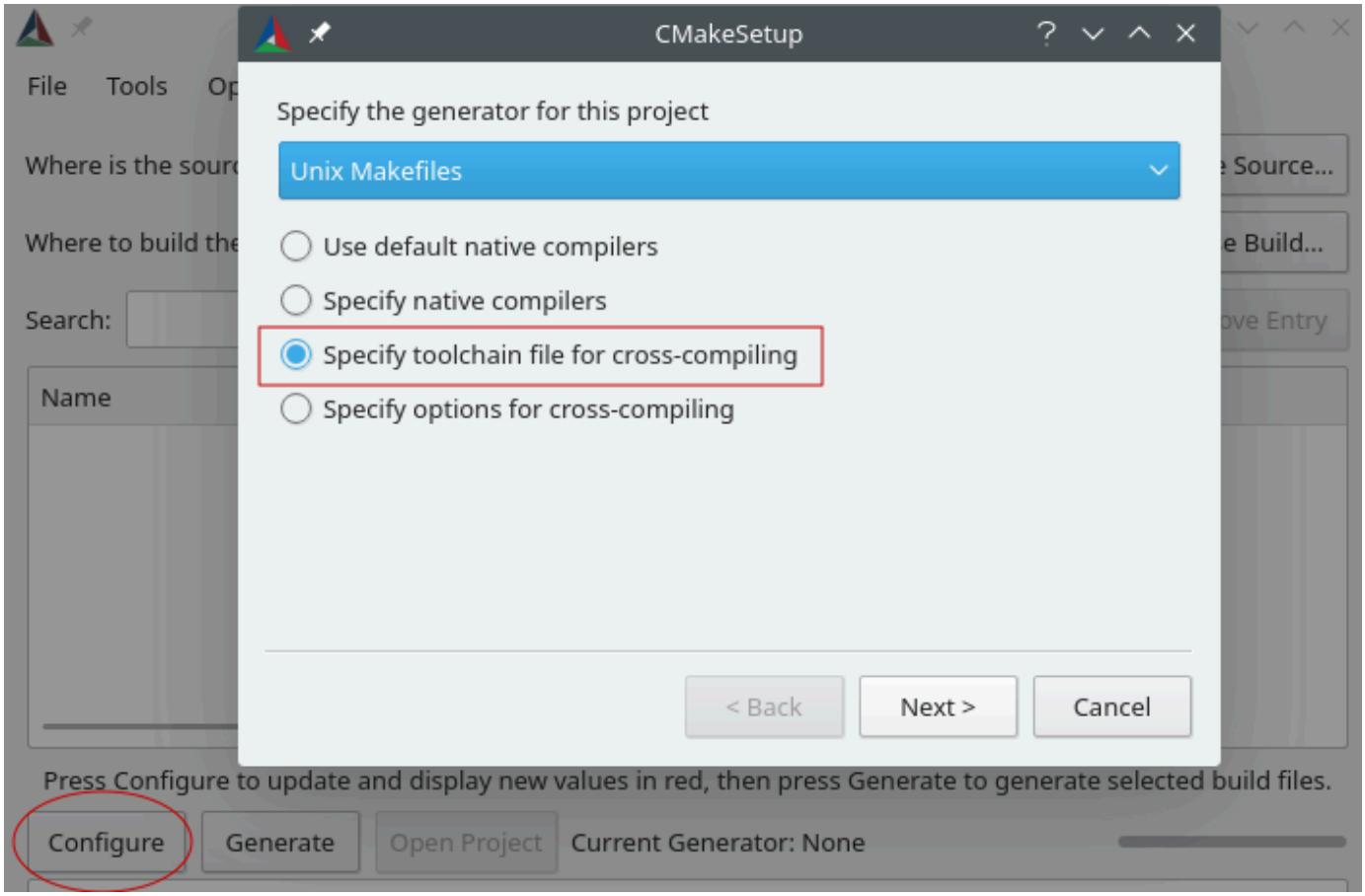
Per generare file di build con la GUI CMake

1. Dalla riga di comando, eseguire `cmake-gui` per avviare la GUI.

2. Scegliere Browse Source (Sfoglia origine) e specificare l'input di origine, quindi scegliere Browse Build (Sfoglia compilazione) e specificare l'output di compilazione.



3. Scegliere Configure (Configura) e in Specify the build generator for this project (Specifica il generatore di compilazione per questo progetto), individuare e scegliere il sistema di compilazione che si desidera utilizzare per i file di compilazione generati. Se la finestra popup non viene visualizzata, è possibile che si stia riutilizzando una directory di compilazione esistente. In questo caso, eliminate la CMake cache scegliendo Elimina cache dal menu File.

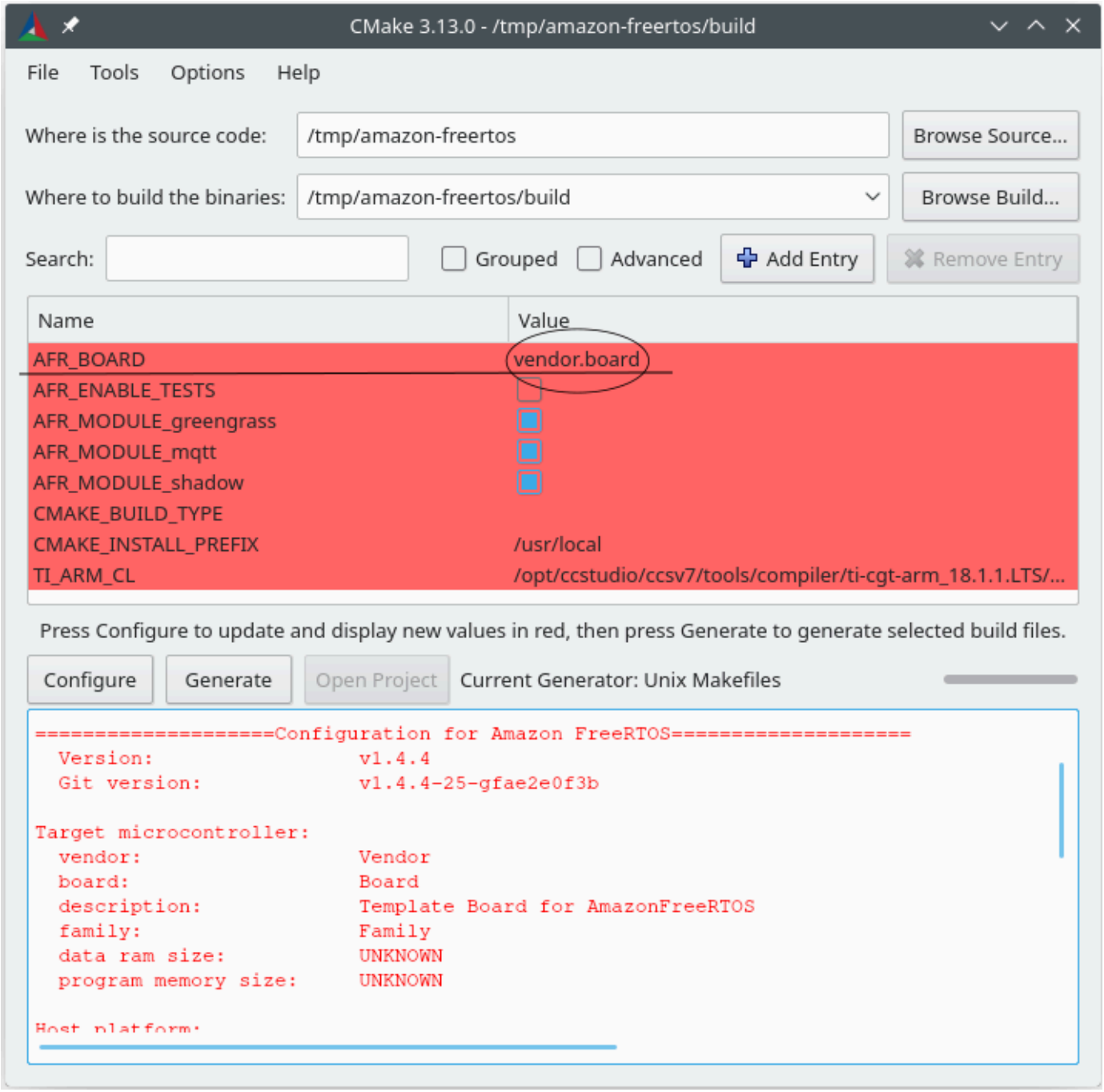


4. Scegliere Specify toolchain file for cross-compiling (Specifica file di toolchain per la compilazione incrociata), quindi selezionare Next (Successivo).
5. Scegliere il file di toolchain (ad esempio, *freertos*`/tools/cmake/toolchains/arm-ti.cmake`), quindi selezionare Fine.

   La configurazione predefinita per FreerTOS è la scheda modello, che non fornisce obiettivi di livello portatili. Di conseguenza, viene visualizzata una finestra con il messaggio Error in configuration process.

   > **Note**
   >
   > Se viene visualizzato il seguente errore:
   >
   > ```
   > CMake Error at tools/cmake/toolchains/find_compiler.cmake:23 (message):
   > Compiler not found, you can specify search path with AFR_TOOLCHAIN_PATH.
   > ```

   Ciò significa che il compilatore non è nella variabile di ambiente `PATH`. È possibile impostare la `AFR_TOOLCHAIN_PATH` variabile nella GUI per indicare CMake dove è stato installato il compilatore. Se la variabile `AFR_TOOLCHAIN_PATH` non è visualizzata, scegliere Add Entry (Aggiungi voce). Nella finestra popup, in Nome, digitare **`AFR_TOOLCHAIN_PATH`**. In Compiler Path (Percorso compilatore) digitare il percorso del compilatore, ad esempio `C:/toolchains/arm-none-eabi-gcc`.
6. L'aspetto della GUI dovrebbe essere simile al seguente:



Scegliere AFR_BOARD, selezionare la scheda, quindi scegliere nuovamente Configure (Configura) .

7. Scegli Genera. CMake genera i file del sistema di compilazione (ad esempio, makefile o ninja) e questi file vengono visualizzati nella directory di compilazione specificata nel primo passaggio. Segui le istruzioni nella sezione successiva per generare l'immagine binaria.

Creazione di FreerTOS da file di build generati

Compilazione con un sistema di compilazione nativo

Puoi creare FreerTOS con un sistema di build nativo chiamando il comando build system dalla directory dei binari di output.

Ad esempio, se la directory di output del file di compilazione è `<build_dir>` e utilizzi Make come sistema di compilazione nativo, esegui i comandi seguenti:

```
cd <build_dir>
make -j4
```

Costruire con CMake

Puoi anche usare lo strumento da CMake riga di comando per creare FreerTOS. CMake fornisce un livello di astrazione per chiamare sistemi di build nativi. Per esempio:

```
cmake --build build_dir
```

Ecco alcuni altri usi comuni della modalità di compilazione dello strumento CMake da riga di 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
```

Per ulteriori informazioni sulla modalità di CMake compilazione, consulta la CMake documentazione.

Distribuzione delle chiavi in modalità sviluppatore

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Introduzione

In questa sezione vengono illustrate due opzioni per ottenere un certificato client X.509 attendibile su un dispositivo IoT per i test di laboratorio. A seconda delle funzionalità del dispositivo, è possibile o meno supportare varie operazioni correlate al provisioning, tra cui la generazione di chiavi ECDSA integrate, l'importazione di chiavi private e la registrazione di certificati X.509. Inoltre, diversi casi d'uso richiedono diversi livelli di protezione delle chiavi, che vanno dall'archiviazione flash integrata all'uso di hardware crittografico dedicato. Questa sezione fornisce la logica per lavorare nelle funzionalità crittografiche del dispositivo.

Opzione #1: importazione di chiavi private da AWS IoT

Per scopi di test di laboratorio, se il dispositivo consente l'importazione di chiavi private, seguire le istruzioni riportate in Configurazione delle demo di FreerTOS.

Opzione 2: Generazione di chiavi private integrate

Se il dispositivo dispone di un elemento di sicurezza o se preferisci generare una coppia di chiavi e un certificato personalizzati, segui le istruzioni riportate di seguito.

Configurazione iniziale

Per prima cosa, esegui i passaggi indicatiConfigurazione delle demo di FreerTOS, ma salta l'ultimo passaggio (ovvero, non farlo per formattare AWS IoT le tue credenziali). Il risultato finale dovrebbe essere che il file `demos/include/aws_clientcredential.h` è stato aggiornato con le impostazioni, ma non il file `demos/include/aws_clientcredential_keys.h`.

Configurazione del progetto demo

Apri la demo di CoreMQTT Mutual Authentication come descritto nella guida relativa alla tua scheda. Guide alle operazioni di base specifiche per la scheda Nel progetto,

aprire il file `aws_dev_mode_key_provisioning.c` e modificare la definizione di `keyprovisioningFORCE_GENERATE_NEW_KEY_PAIR` (che è impostata su zero per impostazione predefinita) su uno:

```
#define keyprovisioningFORCE_GENERATE_NEW_KEY_PAIR 1
```

Quindi creare ed eseguire il progetto demo e passare alla fase successiva.

Estrazione di chiavi pubbliche

Poiché il dispositivo non è stato dotato di una chiave privata e di un certificato client, la demo non riuscirà ad autenticarsi. AWS IoT Tuttavia, la demo di CoreMQTT Mutual Authentication inizia eseguendo il provisioning delle chiavi in modalità sviluppatore, con conseguente creazione di una chiave privata se non era già presente. Dovresti vedere qualcosa di simile al seguente all'inizio dell'output della console seriale.

```
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
```

Copiare le sei righe di byte della chiave in un file denominato `DevicePublicKeyAsciiHex.txt`. Quindi utilizzare lo strumento a riga di comando "xxd" per analizzare i byte esadecimali in formato binario:

```
xxd -r -ps DevicePublicKeyAsciiHex.txt DevicePublicKeyDer.bin
```

Utilizzare "openssl" per formattare la chiave pubblica codificata in formato binario (DER) del dispositivo come PEM:

```
openssl ec -inform der -in DevicePublicKeyDer.bin -pubin -pubout -outform pem -out
 DevicePublicKey.pem
```

Non dimenticare di disabilitare l'impostazione temporanea di generazione delle chiavi abilitata in precedenza. Altrimenti, il dispositivo creerà un'altra coppia di chiavi e sarà necessario ripetere le fasi precedenti:

```
#define keyprovisioningFORCE_GENERATE_NEW_KEY_PAIR 0
```

Configurazione dell'infrastruttura a chiave pubblica

Seguire le istruzioni riportate in Registrazione del certificato CA per creare una gerarchia di certificati per il certificato del laboratorio dei dispositivi. Arrestare prima dell'esecuzione la sequenza descritta nella sezione Creazione di un certificato del dispositivo tramite il certificato CA.

In questo caso, il dispositivo non firmerà la richiesta di certificato (ovvero la Certificate Service Request o CSR) perché la logica di codifica X.509 richiesta per creare e firmare una CSR è stata esclusa dai progetti demo FreerTOS per ridurre le dimensioni della ROM. Invece, per scopi di test di laboratorio, creare una chiave privata sulla workstation e utilizzarla per firmare il CSR.

```
openssl genrsa -out tempCsrSigner.key 2048
openssl req -new -key tempCsrSigner.key -out deviceCert.csr
```

Una volta creata e registrata l'Autorità di certificazione con AWS IoT, utilizzate il seguente comando per emettere un certificato client basato sulla CSR del dispositivo firmata nel passaggio precedente:

```
openssl x509 -req -in deviceCert.csr -CA rootCA.pem -CAkey rootCA.key
 -CAcreateserial -out deviceCert.pem -days 500 -sha256 -force_pubkey
 DevicePublicKey.pem
```

Anche se il CSR è stato firmato con una chiave privata temporanea, il certificato emesso può essere utilizzato solo con la chiave privata effettiva del dispositivo. Lo stesso meccanismo può essere utilizzato in produzione se si memorizza la chiave del firmatario CSR in un hardware separato e si configura l'autorità di certificazione in modo che emetta solo certificati per le richieste firmate da tale chiave specifica. Tale chiave dovrebbe inoltre rimanere sotto il controllo di un amministratore designato.

Importazione di certificati

Con il certificato emesso, la fase successiva è importarlo nel dispositivo. Dovrete inoltre importare il certificato di Certificate Authority (CA), poiché è necessario per il successo della prima autenticazione quando si utilizza AWS IoT JITP. Nel file `aws_clientcredential_keys.h` nel progetto, impostare la macro `keyCLIENT_CERTIFICATE_PEM` come contenuto di deviceCert.pem e impostare la macro `keyJITR_DEVICE_CERTIFICATE_AUTHORITY_PEM` come contenuto di `rootCA.pem`.

Autorizzazione dispositivo

Importalo `deviceCert.pem` nel AWS IoT registro come descritto in Use Your Own Certificate. È necessario creare un nuovo AWS IoT elemento, allegare il certificato PENDING e una politica al documento, quindi contrassegnare il certificato come ATTIVO. Tutti questi passaggi possono essere eseguiti manualmente nella AWS IoT console.

Una volta che il nuovo certificato client è ATTIVO e associato a un oggetto e a una policy, esegui nuovamente la demo di CoreMQTT Mutual Authentication. Questa volta, la connessione al broker AWS IoT MQTT avrà esito positivo.

Guide alle operazioni di base specifiche per la scheda

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Dopo aver completatoFase iniziale, consulta la guida della tua scheda per istruzioni specifiche su come iniziare a usare FreerTOS:

- Guida introduttiva al Cypress CYW9439 07 AEVAL1 F Development Kit
- Guida introduttiva al Cypress CYW9549 07 AEVAL1 F Development Kit
- Guida introduttiva al kit Cypress CY8 CKIT-064S0S2-4343W
- Guida introduttiva al kit di connettività IoT Infineon XMC48 00
- Guida introduttiva allo MW32x AWS IoT Starter Kit
- Guida introduttiva al kit di sviluppo MediaTek MT7697 Hx
- Guida introduttiva a Microchip Curiosity PIC32 MZ EF
- Guida introduttiva al Nuvoton 487 NuMaker-IoT-M
- Guida introduttiva al modulo LPC54 IoT NXP 018
- Guida introduttiva al Renesas Starter Kit+ per N-2MB RX65
- Guida introduttiva al nodo IoT STMicroelectronics STM32 L4 Discovery Kit
- Guida introduttiva a Texas Instruments CC322 0SF-LAUNCHXL

- Nozioni di base su Windows Device Simulator
- Guida introduttiva allo Xilinx Avnet MicroZed Industrial IoT Kit

> **Note**
>
> Non è necessario completare le seguenti guide introduttive Fase iniziale per iniziare a usare FreerTOS:
>
> - Guida introduttiva al simulatore Microchip ATECC6 08A Secure Element con Windows
> - Guida introduttiva all'Espressif ESP32 - DevKit C e al ESP-WROVER-KIT
> - Guida introduttiva all' ESP32Espressif -WROOM-32SE
> - Guida introduttiva al kit di connettività XMC48 IoT Infineon OPTIGA Trust X e 00
> - Guida introduttiva al Nordic n 0-DK RF5284

Guida introduttiva al Cypress CYW9439 07 AEVAL1 F Development Kit

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Questo tutorial fornisce istruzioni per iniziare con il Cypress CYW9439 07 AEVAL1 F Development Kit. Se non disponi del Cypress CYW9439 07 AEVAL1 F Development Kit, visita il AWS Partner Device Catalog per acquistarne uno dal nostro partner.

> **Note**
>
> Questo tutorial illustra la configurazione e l'esecuzione della demo di CoreMQTT Mutual Authentication. La porta FreerTOS per questa scheda attualmente non supporta le demo del server e del client TCP.

Prima di iniziare, devi configurare AWS IoT e scaricare i FreerTOS per connettere il tuo dispositivo al Cloud. AWS Per istruzioni, consulta Fase iniziale.

Important

- In questo argomento, il percorso della directory di download di FreerTOS viene definito come. *freertos*
- Gli spazi contenuti nel percorso *freertos* possono causare errori di compilazione. Quando si clona o si copia il repository, assicurarsi che il percorso creato non contenga spazi.
- La lunghezza massima di un percorso di file su Microsoft Windows è di 260 caratteri. I lunghi percorsi delle directory di download di FreerTOS possono causare errori di compilazione.
- Come indicato inScaricare FreerTOS, le porte FreerTOS per Cypress sono attualmente disponibili solo su. GitHub

Panoramica

Questo tutorial contiene le istruzioni per i seguenti passaggi iniziali:

1. Installazione di software sul computer host per lo sviluppo e il debug di applicazioni integrate per la scheda a microcontroller.
2. Compilazione incrociata di un'applicazione demo FreerTOS con un'immagine binaria.
3. Caricamento dell'immagine binaria dell'applicazione sulla scheda in uso e successiva esecuzione dell'applicazione.
4. Interazione con l'applicazione in esecuzione sulla scheda attraverso una connessione seriale, per scopi di monitoraggio e debug.

Configurazione dell'ambiente di sviluppo

Scaricare e installare l'SDK WICED Studio

In questa guida introduttiva, utilizzi l'SDK Cypress WICED Studio per programmare la tua scheda con la demo di FreerTOS. Visita il sito Web del software WICED per scaricare l'SDK WICED Studio da

Cypress. Per scaricare il software è necessario effettuare la registrazione per un account Cypress gratuito. L'SDK WICED Studio è compatibile con i sistemi operativi Windows, macOS e Linux.

> **Note**
>
> Alcuni sistemi operativi richiedono fasi di installazione aggiuntive. Assicurarsi di leggere e seguire tutte le istruzioni di installazione del sistema operativo e della versione di WICED Studio che si stanno installando.

Impostazione delle variabili di ambiente

Prima di utilizzare WICED Studio per programmare la scheda, è necessario creare una variabile di ambiente per la directory di installazione dell'SDK WICED Studio. Se WICED Studio è in esecuzione durante la creazione delle variabili, sarà necessario riavviare l'applicazione dopo l'impostazione delle variabili.

> **Note**
>
> Il programma di installazione di WICED Studio crea due cartelle separate denominate `WICED-Studio-m.n` sul computer in cui `m` e `n` sono rispettivamente i numeri di versione principale e secondaria. Questo documento presuppone il nome della cartella `WICED-Studio-6.2` ma assicurati di utilizzare il nome corretto per la versione installata. Una volta definita la variabile di ambiente `WICED_STUDIO_SDK_PATH` assicurati di specificare il percorso completo di installazione dell'SDK WICED Studio e non il percorso di installazione dell'interfaccia utente WICED Studio. In Windows e macOS, la cartella `WICED-Studio-m.n` per l'SDK viene creata per impostazione predefinita nella cartella `Documents`.

Per creare la variabile di ambiente in Windows

1. Aprire il Pannello di controllo di Windows fare clic su Sistema, Impostazioni di sistema avanzate.
2. Nella scheda Avanzate, scegliere Variabili di ambiente.
3. In Variabili utente scegliere Nuova.
4. In Nome variabile, inserire **`WICED_STUDIO_SDK_PATH`**. Per Valore della variabile, immettere la directory di installazione dell'SDK WICED Studio.

Per creare la variabile di ambiente in Linux o macOS

1. Aprire il file `/etc/profile` nel compute, quindi aggiungere quanto segue all'ultima riga del file:

```
export WICED_STUDIO_SDK_PATH=installation-path/WICED-Studio-6.2
```

2. Riavviare il computer.
3. Aprire una finestra del terminale ed eseguire i comandi seguenti:

```
cd freertos/vendors/cypress/WICED_SDK
```

```
perl platform_adjust_make.pl
```

```
chmod +x make
```

Stabilire una connessione seriale

Per stabilire una connessione seriale tra il computer host e la scheda

1. Collegare la scheda al computer host tramite un cavo USB da Standard-A a Micro-B.
2. Identificare il numero della porta seriale USB per la connessione alla scheda sul computer host.
3. Avviare un terminale seriale e aprire una connessione con le impostazioni seguenti:

   - Velocità in baud: 115200
   - Dati: 8 bit
   - Parità: nessuna
   - Bit di stop: 1
   - Controllo di flusso: nessuno

Per ulteriori informazioni sull'installazione di un terminale e la configurazione di una connessione seriale, consultare Installazione di un emulatore di terminale.

Crea ed esegui il progetto demo FreerTOS

Dopo aver impostato una connessione seriale alla tua scheda, puoi creare il progetto demo FreerTOS, eseguire il flashing della demo sulla tua scheda e quindi eseguire la demo.

Per creare ed eseguire il progetto demo FreerTOS in WICED Studio

1. Avviare WICED Studio.
2. Dal menu File scegliere Import (Importa). Espandere la cartella `General`, scegliere Existing Projects into Workspace (Progetti esistenti in Workspace) e Next (Successivo).
3. In Select root directory (Seleziona directory root), selezionare Browse... (Sfoglia...), passare al percorso *`freertos`*`/projects/cypress/CYW943907AEVAL1F/wicedstudio`, quindi selezionare OK.
4. In Projects (Progetti), selezionare la casella solo per il progetto aws_demo . Selezionare Finish (Fine) per importare il progetto. Il progetto di destinazione aws_demo (aws_demo) dovrebbe comparire nella finestra Make Target (Crea destinazione).
5. Espandere il menu WICED Platform (Piattaforma WICED) e selezionare WICED Filters off (Disattiva filtri WICED).
6. Nella finestra Make Target (Crea destinazione), espandere aws_demo (aws_demo), fare clic con il tasto destro del mouse sul file `demo.aws_demo`, quindi selezionare Build Target (Compila destinazione) per compilare e scaricare la demo sulla scheda. L'esecuzione della demo dovrebbe avvenire automaticamente dopo che viene compilata e scaricata nella scheda.

Monitoraggio dei messaggi MQTT in cloud

Puoi utilizzare il client MQTT nella AWS IoT console per monitorare i messaggi che il tuo dispositivo invia al Cloud. AWS

Per sottoscrivere l'argomento MQTT con il AWS IoT client MQTT

1. Accedi alla console AWS IoT.
2. Nel riquadro di navigazione scegliere Test per aprire il client MQTT.
3. In Argomento sottoscrizione, digitare ***`your-thing-name`*`/example/topic`**, quindi scegliere Effettua sottoscrizione all'argomento.

Risoluzione dei problemi

- Se si utilizza Windows, quando si compila e si esegue il progetto demo si potrebbe ricevere il seguente errore:

```
: recipe for target 'download_dct' failed
```

```
make.exe[1]: *** [download_dct] Error 1
```

Per risolvere questo errore, eseguire questi passaggi:

1. Passa a *WICED-Studio-SDK-PATH*\WICED-Studio-6.2\43xxx_Wi-Fi\tools\OpenOCD\Win32 e fai doppio clic su openocd-all-brcm-libftdi.exe.
2. Passa a *WICED-Studio-SDK-PATH*\WICED-Studio-6.2\43xxx_Wi-Fi\tools\drivers\CYW9WCD1EVAL1 e fai doppio clic su InstallDriver.exe.

- Se si utilizza Linux o macOS, quando si compila e si esegue il progetto demo si potrebbe ricevere il seguente errore:

```
make[1]: *** [download_dct] Error 127
```

Per risolvere questo errore, utilizza il comando seguente per aggiornare il pacchetto libusb-dev:

```
sudo apt-get install libusb-dev
```

Per informazioni generali sulla risoluzione dei problemi su Getting Started with FreerTOS, consulta. Nozioni di base sulla risoluzione dei problemi

Guida introduttiva al Cypress CYW9549 07 AEVAL1 F Development Kit

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Questo tutorial fornisce istruzioni per iniziare con il Cypress CYW9549 07 AEVAL1 F Development Kit. Se non disponi del Cypress CYW9549 07 AEVAL1 F Development Kit, visita il AWS Partner Device Catalog per acquistarne uno dal nostro partner.

> **Note**
>
> Questo tutorial illustra la configurazione e l'esecuzione della demo di CoreMQTT Mutual Authentication. La porta FreerTOS per questa scheda attualmente non supporta le demo del server e del client TCP.

Prima di iniziare, devi configurare AWS IoT e scaricare FreerTOS per connettere il tuo dispositivo al Cloud. AWS Per istruzioni, consulta Fase iniziale. In questo tutorial, il percorso della directory di download di FreerTOS è indicato come. *`freertos`*

> **Important**
>
> - In questo argomento, il percorso della directory di download di FreerTOS viene definito come. *`freertos`*
> - Gli spazi contenuti nel percorso *`freertos`* possono causare errori di compilazione. Quando si clona o si copia il repository, assicurarsi che il percorso creato non contenga spazi.
> - La lunghezza massima di un percorso di file su Microsoft Windows è di 260 caratteri. I lunghi percorsi delle directory di download di FreerTOS possono causare errori di compilazione.
> - Come indicato inScaricare FreerTOS, le porte FreerTOS per Cypress sono attualmente disponibili solo su. GitHub

Panoramica

Questo tutorial contiene le istruzioni per i seguenti passaggi iniziali:

1. Installazione di software sul computer host per lo sviluppo e il debug di applicazioni integrate per la scheda a microcontroller.
2. Compilazione incrociata di un'applicazione demo FreerTOS con un'immagine binaria.
3. Caricamento dell'immagine binaria dell'applicazione sulla scheda in uso e successiva esecuzione dell'applicazione.
4. Interazione con l'applicazione in esecuzione sulla scheda attraverso una connessione seriale, per scopi di monitoraggio e debug.

Configurazione dell'ambiente di sviluppo

Scaricare e installare l'SDK WICED Studio

In questa guida introduttiva, utilizzi l'SDK Cypress WICED Studio per programmare la tua scheda con la demo di FreerTOS. Visita il sito Web del software WICED per scaricare l'SDK WICED Studio da Cypress. Per scaricare il software è necessario effettuare la registrazione per un account Cypress gratuito. L'SDK WICED Studio è compatibile con i sistemi operativi Windows, macOS e Linux.

> **Note**
>
> Alcuni sistemi operativi richiedono fasi di installazione aggiuntive. Assicurarsi di leggere e seguire tutte le istruzioni di installazione del sistema operativo e della versione di WICED Studio che si stanno installando.

Impostazione delle variabili di ambiente

Prima di utilizzare WICED Studio per programmare la scheda, è necessario creare una variabile di ambiente per la directory di installazione dell'SDK WICED Studio. Se WICED Studio è in esecuzione durante la creazione delle variabili, sarà necessario riavviare l'applicazione dopo l'impostazione delle variabili.

> **Note**
>
> Il programma di installazione di WICED Studio crea due cartelle separate denominate `WICED-Studio-`*`m`*`.`*`n`* sul computer in cui `m` e `n` sono rispettivamente i numeri di versione principale e secondaria. Questo documento presuppone il nome della cartella `WICED-Studio-6.2` ma assicurati di utilizzare il nome corretto per la versione installata. Una volta definita la variabile di ambiente `WICED_STUDIO_SDK_PATH` assicurati di specificare il percorso completo di installazione dell'SDK WICED Studio e non il percorso di installazione dell'interfaccia utente WICED Studio. In Windows e macOS, la cartella `WICED-Studio-`*`m`*`.`*`n`* per l'SDK viene creata per impostazione predefinita nella cartella `Documents`.

Per creare la variabile di ambiente in Windows

1. Aprire il Pannello di controllo di Windows fare clic su Sistema, Impostazioni di sistema avanzate.
2. Nella scheda Avanzate, scegliere Variabili di ambiente.

3. In Variabili utente scegliere Nuova.
4. In Nome variabile, inserire **`WICED_STUDIO_SDK_PATH`**. Per Valore della variabile, immettere la directory di installazione dell'SDK WICED Studio.

Per creare la variabile di ambiente in Linux o macOS

1. Aprire il file `/etc/profile` nel compute, quindi aggiungere quanto segue all'ultima riga del file:

```
export WICED_STUDIO_SDK_PATH=installation-path/WICED-Studio-6.2
```

2. Riavviare il computer.
3. Aprire una finestra del terminale ed eseguire i comandi seguenti:

```
cd freertos/vendors/cypress/WICED_SDK
```

```
perl platform_adjust_make.pl
```

```
chmod +x make
```

Stabilire una connessione seriale

Per stabilire una connessione seriale tra il computer host e la scheda

1. Collegare la scheda al computer host tramite un cavo USB da Standard-A a Micro-B.
2. Identificare il numero della porta seriale USB per la connessione alla scheda sul computer host.
3. Avviare un terminale seriale e aprire una connessione con le impostazioni seguenti:
   - Velocità in baud: 115200
   - Dati: 8 bit
   - Parità: nessuna
   - Bit di stop: 1
   - Controllo di flusso: nessuno

Per ulteriori informazioni sull'installazione di un terminale e la configurazione di una connessione seriale, consultare Installazione di un emulatore di terminale.

Crea ed esegui il progetto demo FreerTOS

Dopo aver impostato una connessione seriale alla tua scheda, puoi creare il progetto demo FreerTOS, eseguire il flashing della demo sulla tua scheda e quindi eseguire la demo.

Per creare ed eseguire il progetto demo FreerTOS in WICED Studio

1. Avviare WICED Studio.
2. Dal menu File scegliere Import (Importa). Espandere la cartella `General`, scegliere Existing Projects into Workspace (Progetti esistenti in Workspace) e Next (Successivo).
3. In Select root directory (Seleziona directory root), selezionare Browse... (Sfoglia...), passare al percorso *`freertos`*`/projects/cypress/CYW954907AEVAL1F/wicedstudio`, quindi selezionare OK.
4. In Projects (Progetti), selezionare la casella solo per il progetto aws_demo . Selezionare Finish (Fine) per importare il progetto. Il progetto di destinazione aws_demo (aws_demo) dovrebbe comparire nella finestra Make Target (Crea destinazione).
5. Espandere il menu WICED Platform (Piattaforma WICED) e selezionare WICED Filters off (Disattiva filtri WICED).
6. Nella finestra Make Target (Crea destinazione), espandere aws_demo (aws_demo), fare clic con il tasto destro del mouse sul file `demo.aws_demo`, quindi selezionare Build Target (Compila destinazione) per compilare e scaricare la demo sulla scheda. L'esecuzione della demo dovrebbe avvenire automaticamente dopo che viene compilata e scaricata nella scheda.

Monitoraggio dei messaggi MQTT in cloud

Puoi utilizzare il client MQTT nella AWS IoT console per monitorare i messaggi che il tuo dispositivo invia al Cloud. AWS

Per abbonarsi all'argomento MQTT con il client MQTT AWS IoT

1. Accedere alla console AWS IoT.
2. Nel riquadro di navigazione scegliere Test per aprire il client MQTT.
3. In Argomento sottoscrizione, digitare ***`your-thing-name`*`/example/topic`**, quindi scegliere Effettua sottoscrizione all'argomento.

Risoluzione dei problemi

- Se si utilizza Windows, quando si compila e si esegue il progetto demo si potrebbe ricevere il seguente errore:

  ```
  : recipe for target 'download_dct' failed
  make.exe[1]: *** [download_dct] Error 1
  ```

  Per risolvere questo errore, eseguire questi passaggi:

  1. Passa a *WICED-Studio-SDK-PATH*\WICED-Studio-6.2\43xxx_Wi-Fi\tools\OpenOCD\Win32 e fai doppio clic su openocd-all-brcm-libftdi.exe.
  2. Passa a *WICED-Studio-SDK-PATH*\WICED-Studio-6.2\43xxx_Wi-Fi\tools\drivers\CYW9WCD1EVAL1 e fai doppio clic su InstallDriver.exe.

- Se si utilizza Linux o macOS, quando si compila e si esegue il progetto demo si potrebbe ricevere il seguente errore:

  ```
  make[1]: *** [download_dct] Error 127
  ```

  Per risolvere questo errore, utilizza il comando seguente per aggiornare il pacchetto libusb-dev:

  ```
  sudo apt-get install libusb-dev
  ```

Per informazioni generali sulla risoluzione dei problemi su Getting Started with FreerTOS, consulta. Nozioni di base sulla risoluzione dei problemi

Guida introduttiva al kit Cypress CY8 CKIT-064S0S2-4343W

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Questo tutorial fornisce istruzioni per iniziare con il kit CY8CKIT-064S0S2-4343W. Se non ne hai già uno, puoi utilizzare quel link per acquistare un kit. Puoi anche utilizzare quel link per accedere alla guida per l'utente del kit.

Nozioni di base

Prima di iniziare, devi configurare AWS IoT FreerTOS per connettere il tuo dispositivo al Cloud. AWS Per istruzioni, consulta Fase iniziale. Dopo aver completato i prerequisiti, avrai un pacchetto FreerTOS con le credenziali. AWS IoT Core

> **Note**
>
> In questo tutorial, il percorso della directory di download di FreerTOS creata nella sezione «Primi passi» è indicato come. *`freertos`*

Configurazione dell'ambiente di sviluppo

FreerTOS funziona sia con CMake un flusso di compilazione che con Make. Puoi usarlo ModusToolbox per il tuo flusso di compilazione Make. È possibile utilizzare l'IDE Eclipse fornito con ModusToolbox o un IDE partner come IAR EW-Arm, Arm MDK o Microsoft Visual Studio Code. L'IDE Eclipse è compatibile con i sistemi operativi Windows, macOS e Linux.

Prima di iniziare, scarica e installa il software più recenteModusToolbox . Per ulteriori informazioni, consulta la Guida all'ModusToolbox installazione.

Strumenti di aggiornamento per la ModusToolbox versione 2.1 o precedente

Se utilizzi l'IDE ModusToolbox 2.1 Eclipse per programmare questo kit, dovrai aggiornare gli strumenti OpenOCD e Firmware-Loader.

Nei passaggi seguenti, per impostazione predefinita, il percorso per: *`ModusToolbox`*

- Windows è`C:\Users\`*`user_name`*`\ModusToolbox`.
- Linux è *`user_home`*`/ModusToolbox` o dove scegli di estrarre il file di archivio.
- macOS si trova nella cartella Applicazioni del volume selezionato nella procedura guidata.

Aggiornamento di OpenOCD

Questo kit richiede Cypress OpenOCD 4.0.0 o successivo per cancellare e programmare correttamente il chip.

Per aggiornare Cypress OpenOCD

1. Vai alla pagina di rilascio di Cypress OpenOCD.
2. Scarica il file di archivio per il tuo sistema operativo (). Windows/Mac/Linux
3. Eliminare i file esistenti in*ModusToolbox*`/tools_2.x/openocd`.
4. Sostituisci i file *ModusToolbox*`/tools_2.x/openocd` con il contenuto estratto dell'archivio scaricato in un passaggio precedente.

Aggiornamento del firmware-Loader

Questo kit richiede Cypress Firmware-Loader 3.0.0 o versione successiva.

Per aggiornare Cypress Firmware-Loader

1. Vai alla pagina di rilascio di Cypress Firmware-Loader.
2. Scarica il file di archivio per il tuo sistema operativo (). Windows/Mac/Linux
3. Eliminare i file esistenti in*ModusToolbox*`/tools_2.x/fw-loader`.
4. Sostituisci i file *ModusToolbox*`/tools_2.x/fw-loader` con il contenuto estratto dell'archivio scaricato in un passaggio precedente.

In alternativa, puoi usarlo CMake per generare file di build del progetto dal codice sorgente dell'applicazione FreerTOS, creare il progetto usando il tuo strumento di compilazione preferito e quindi programmare il kit usando OpenOCD. Se preferisci utilizzare uno strumento GUI per la programmazione con il CMake flusso, scarica e installa Cypress Programmer dalla pagina web di Cypress Programming Solutions. Per ulteriori informazioni, consulta Utilizzo CMake con FreerTOS.

Configurazione dell'hardware

Segui questi passaggi per configurare l'hardware del kit.

1. Esegui il provisioning del kit

Segui le istruzioni della Guida alla fornitura del kit CY8 CKIT-064S0S2-4343W per fornire il kit in modo sicuro. AWS IoT

2. Configurare una connessione seriale

   a. Connect il kit al computer host.

   b. La porta seriale USB del kit viene automaticamente enumerata sul computer host. Identifica il numero di porta. In Windows, è possibile identificarlo utilizzando il Device Manager in Porte (COM e LPT).

   c. Avviare un terminale seriale e aprire una connessione con le impostazioni seguenti:

      - Velocità in baud: 115200
      - Dati: 8 bit
      - Parità: nessuna
      - Bit di stop: 1
      - Controllo di flusso: nessuno

Crea ed esegui il progetto FreerTOS Demo

In questa sezione crei ed esegui la demo.

1. Assicurati di seguire i passaggi indicati nella Guida al provisioning per il kit CY8 CKIT-064S0S2-4343W.
2. Crea la demo di FreerTOS.

   a. Apri l'IDE di Eclipse ModusToolbox e scegli, o crea, uno spazio di lavoro.

   b. Dal menu File scegliere Import (Importa).

      Espandi Generale, scegli Progetto esistente in area di lavoro, quindi scegli Avanti.

   c. In Root Directory, immettete *freertos*`/projects/cypress/CY8CKIT-064S0S2-4343W/mtb/aws_demos` e selezionate il nome `aws_demos` del progetto. Dovrebbe essere selezionato per impostazione predefinita.

   d. Scegliete Fine per importare il progetto nel vostro spazio di lavoro.

   e. Create l'applicazione effettuando una delle seguenti operazioni:

      - Dal pannello rapido, seleziona Crea applicazione aws_demos.
      - Scegli Project e scegli Build All.

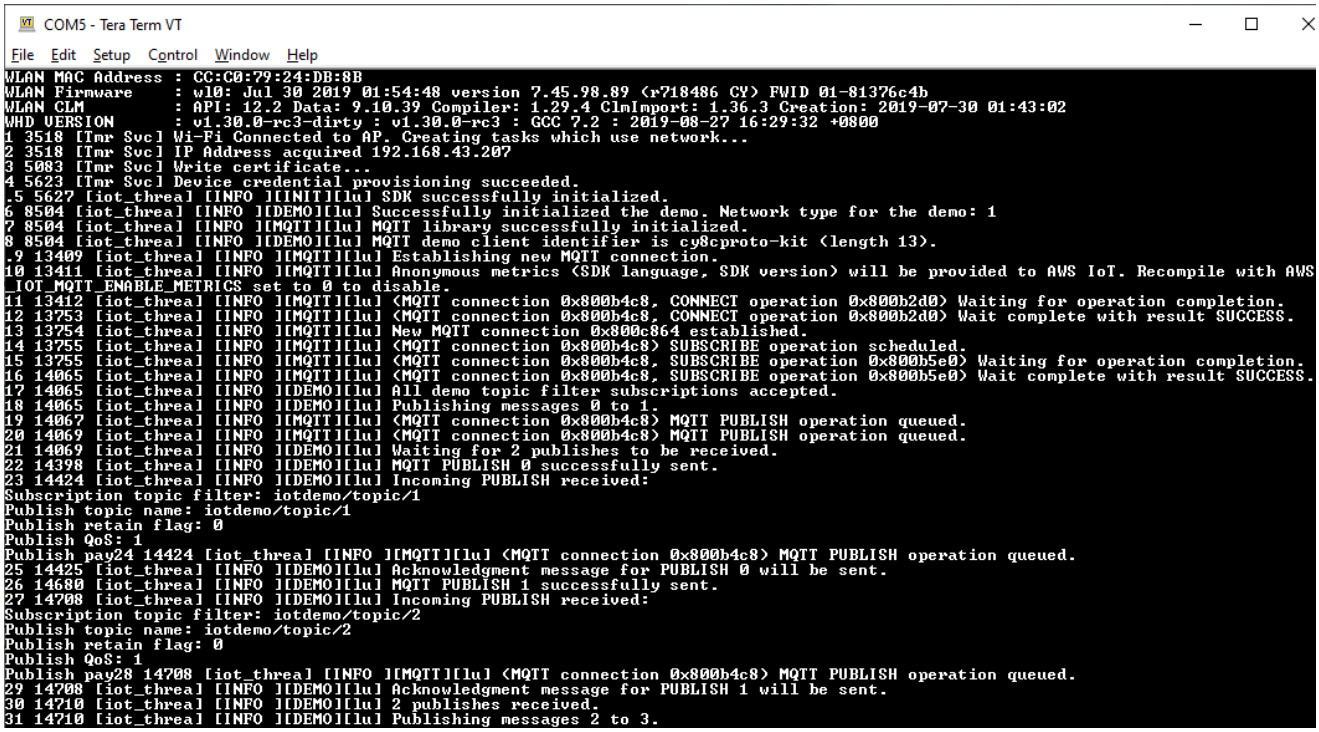
Assicurati che il progetto venga compilato senza errori.

3. Esegui il progetto demo FreerTOS

   a. Seleziona il progetto `aws_demos` nell'area di lavoro.

   b. Dal pannello rapido, seleziona aws_demos Program (3). KitProg Questo programma la scheda e l'applicazione demo inizia a funzionare al termine della programmazione.

   c. È possibile visualizzare lo stato dell'applicazione in esecuzione nel terminale seriale. La figura seguente mostra una parte dell'uscita del terminale.



La demo MQTT pubblica messaggi su quattro argomenti diversi (`iotdemo/topic/`*n*dove da n=1 a 4) e sottoscrive tutti questi argomenti per ricevere gli stessi messaggi. Quando viene ricevuto un messaggio, la demo pubblica un messaggio di conferma sull'argomento. `iotdemo/acknowledgements` L'elenco seguente descrive i messaggi di debug che appaiono nell'output del terminale, con riferimenti ai numeri di serie dei messaggi. Nell'output, i dettagli del driver WICED Host Driver (WHD) vengono stampati per primi senza numerazione seriale.

1. Da 1 a 4: il dispositivo si connette all'Access Point (AP) configurato e viene fornito connettendosi al AWS server utilizzando l'endpoint e i certificati configurati.

2. Da 5 a 13: la libreria CoreMQTT viene inizializzata e il dispositivo stabilisce una connessione MQTT.
3. Da 14 a 17 — Il dispositivo si iscrive a tutti gli argomenti per ricevere di nuovo i messaggi pubblicati.
4. Da 18 a 30: il dispositivo pubblica due messaggi e attende di riceverli. Quando ogni messaggio viene ricevuto, il dispositivo invia un messaggio di conferma.

Lo stesso ciclo di pubblicazione, ricezione e conferma continua fino alla pubblicazione di tutti i messaggi. Vengono pubblicati due messaggi per ciclo fino al completamento del numero di cicli configurati.

4. Utilizzo CMake con FreerTOS

Puoi anche usarlo CMake per creare ed eseguire l'applicazione demo. Per configurare CMake e creare un sistema di compilazione nativo, vediPrerequisiti.

a. Utilizzate il seguente comando per generare file di build. Specificate la scheda di destinazione con l'`-DBOARD`opzione.

```
cmake -DVENDOR=cypress -DBOARD=CY8CKIT_064S0S2_4343W -DCOMPILER=arm-gcc -
S freertos -B build_dir
```

Se utilizzi Windows, devi specificare il sistema di compilazione nativo utilizzando l'`-G`opzione perché CMake utilizza Visual Studio per impostazione predefinita.

Example

```
cmake -DVENDOR=cypress -DBOARD=CY8CKIT_064S0S2_4343W -DCOMPILER=arm-gcc -
S freertos -B build_dir -G Ninja
```

Se non `arm-none-eabi-gcc` è nel percorso della shell, devi anche impostare la `AFR_TOOLCHAIN_PATH` CMake variabile.

Example

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

b. Usa il seguente comando per creare il progetto usando CMake.

```
cmake --build build_dir
```

c. Infine, programma i `cm4.hex` file `cm0.hex` and generati *`build_dir`* da Cypress Programmer.

5. Monitoraggio dei messaggi MQTT nel cloud

   Puoi utilizzare il client MQTT nella AWS IoT console per monitorare i messaggi che il tuo dispositivo invia al Cloud. AWS Per iscriverti all'argomento MQTT con il client AWS IoT MQTT, segui questi passaggi.

   a. Accedere alla console AWS IoT.

   b. Nel riquadro di navigazione scegliere Test per aprire il client MQTT.

   c. Per l'argomento Abbonamento, immettete***your-thing-name*/example/topic**, quindi scegliete Sottoscrivi all'argomento.

   d. Reimpostate il kit per forzarlo a pubblicare messaggi MQTT, che possono poi essere visualizzati sul client di test della console.

Esecuzione di altre demo

Le seguenti applicazioni demo sono state testate e verificate per funzionare con la versione corrente. Puoi trovare queste demo nella *`freertos`*`/demos` directory. Per informazioni su come eseguire queste demo, consulta. Demo FreerTOS

- Dimostrazione Bluetooth Low Energy
- Over-the-Air Demo di aggiornamenti
- Dimostrazione del client Secure Sockets Echo
- AWS IoT Dimostrazione di Device Shadow

Debug

I KitProg 3 del kit supportano il debug tramite il protocollo SWD.

- Per eseguire il debug dell'applicazione FreerTOS, seleziona il progetto aws_demos nell'area di lavoro, quindi seleziona aws_demos Debug (3) dal Pannello rapido. KitProg

Aggiornamenti OTA

PSoI C 64 MCUs hanno superato tutti i test di qualificazione FreerTOS richiesti. Tuttavia, la funzionalità opzionale over-the-air (OTA) implementata nella libreria AWS firmware PSo C 64 Standard Secure è ancora in attesa di valutazione. La funzionalità OTA così come implementata attualmente supera tutti i test di qualificazione OTA ad eccezione di aws_ota_test_case_rollback_if_unable_to_connect_after_update.py.

Quando un'immagine OTA convalidata con successo viene applicata a un dispositivo che utilizza l' AWS MCU PSo C64 Standard Secure con cui il dispositivo non è in grado di comunicare AWS IoT Core, il dispositivo non può tornare automaticamente all'immagine originale valida nota. Ciò potrebbe rendere il dispositivo irraggiungibile per ulteriori aggiornamenti. AWS IoT Core Questa funzionalità è ancora in fase di sviluppo da parte del team di Cypress.

Per ulteriori informazioni, vedere OTA Updates with AWS and the CY8 CKIT-064S0S2-4343W Kit. Se hai ulteriori domande o hai bisogno di supporto tecnico, contatta la Cypress Developer Community.

Guida introduttiva al simulatore Microchip ATECC6 08A Secure Element con Windows

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Questo tutorial fornisce istruzioni per iniziare a utilizzare Microchip ATECC6 08A Secure Element with Windows Simulator.

È necessario il seguente hardware:

- Clickboard Microchip ATECC6 08A Secure Element
- SAMD21 XPlained Pro
- Adattatore mikroBUS Xplained Pro

Prima di iniziare, devi configurare AWS IoT e scaricare FreerTOS per connettere il tuo dispositivo al Cloud. AWS In questo tutorial, il percorso della directory di download di FreerTOS è indicato come. *freertos*

Panoramica

Questa esercitazione contiene i seguenti passaggi:

1. Connessione della scheda a un computer host.
2. Installazione di software sul computer host per lo sviluppo e il debug di applicazioni integrate per la scheda del microcontroller.
3. Compila in modo incrociato un'applicazione demo FreerTOS su un'immagine binaria.
4. Caricamento dell'immagine binaria dell'applicazione sulla scheda in uso e successiva esecuzione dell'applicazione.

Configurare l'hardware Microchip 08A ATECC6

Prima di poter interagire con il dispositivo Microchip ATECC6 08A, è necessario programmare il. SAMD21

Per configurare la scheda Pro SAMD21 XPlained

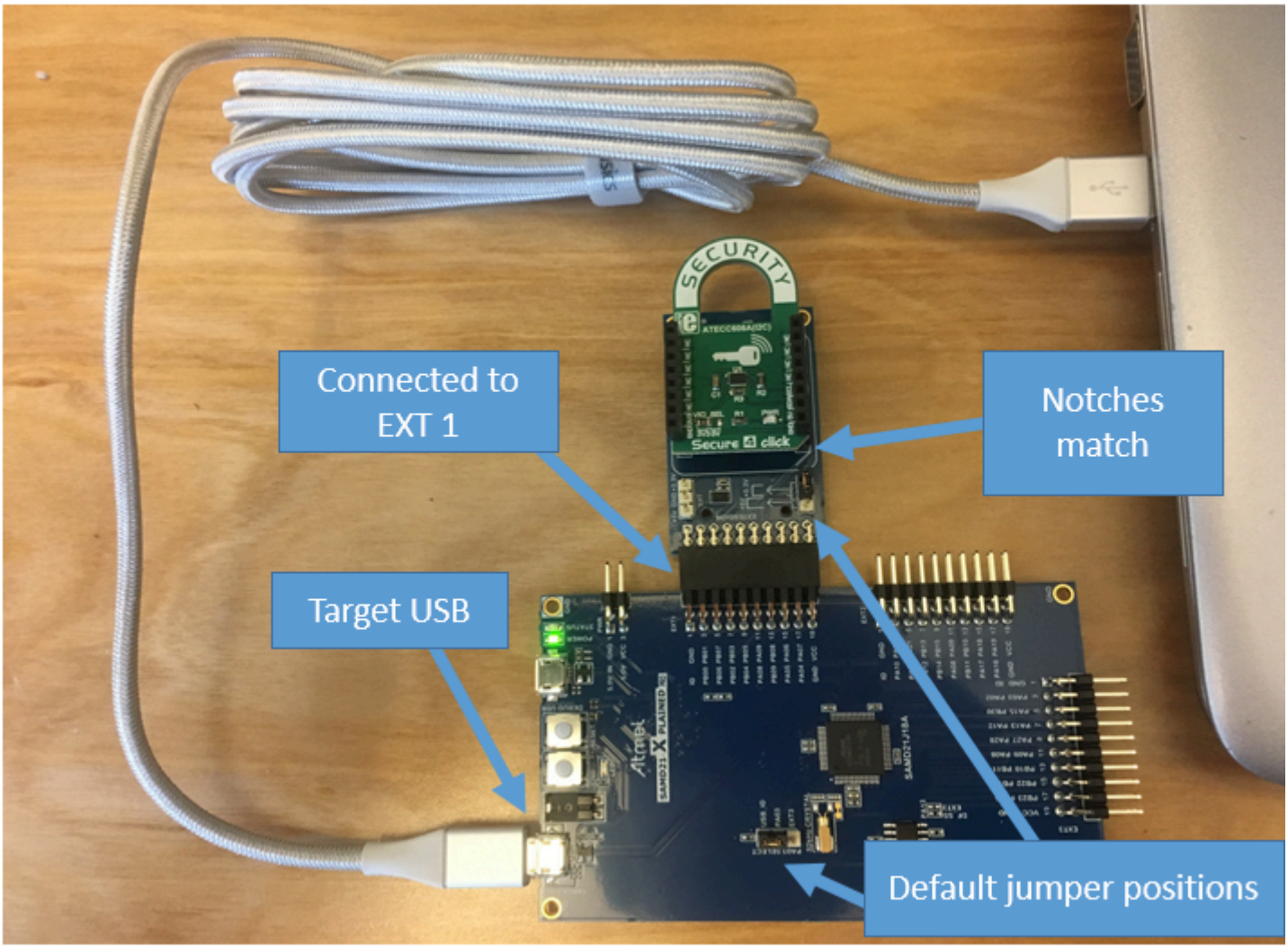
1. Segui il link CryptoAuthSSH-XSTK (DM320109) - Latest Firmware per scaricare un file.zip contenente istruzioni (PDF) e un file binario che può essere programmato sulla D21.
2. Scarica e installa Atmel Studio 7 IDP. Assicurarsi di selezionare l'architettura del driver SMART ARM MCU durante l'installazione.
3. Utilizzare un cavo USB 2.0 Micro B per collegare il connettore "Debug USB" al computer e seguire le istruzioni contenute nel PDF. (Il connettore "Debug USB" è la porta USB più vicina al LED di alimentazione e ai pin.)

Per collegare l'hardware

1. Scollegare il cavo micro USB da Debug USB.
2. Collega l'adattatore XPlained mikroBus Pro alla scheda nella SAMD21 posizione desiderata. EXT1

3. Collega la scheda ATECC6 08A Secure 4 Click all'adattatore mikrobusX Pro XPlained . Assicurarsi che l'angolo dentellato della click board corrisponda all'icona dentellata sulla scheda adattatore.
4. Collegare il cavo micro USB all'USB di destinazione.

La configurazione avrà un aspetto simile alla seguente immagine.



Configurazione dell'ambiente di sviluppo

Iscriviti per un Account AWS

Se non ne hai uno Account AWS, completa i seguenti passaggi per crearne uno.

Per iscriverti a un Account AWS

1. Apri la https://portal.aws.amazon.com/billing/registrazione.
2. Segui le istruzioni online.

   Parte della procedura di registrazione prevede la ricezione di una telefonata o di un messaggio di testo e l'immissione di un codice di verifica sulla tastiera del telefono.

   Quando ti iscrivi a Account AWS, Utente root dell'account AWSviene creato un. L'utente root dispone dell'accesso a tutte le risorse e tutti i Servizi AWS nell'account. Come best practice di sicurezza, assegna l'accesso amministrativo a un utente e utilizza solo l'utente root per eseguire attività che richiedono l'accesso di un utente root.

AWS ti invia un'email di conferma dopo il completamento della procedura di registrazione. In qualsiasi momento, puoi visualizzare l'attività corrente del tuo account e gestirlo accedendo a https://aws.amazon.com/e scegliendo Il mio account.

Crea un utente con accesso amministrativo

Dopo esserti registrato Account AWS, proteggi Utente root dell'account AWS AWS IAM Identity Center, abilita e crea un utente amministrativo in modo da non utilizzare l'utente root per le attività quotidiane.

Proteggi i tuoi Utente root dell'account AWS

1. Accedi AWS Management Consolecome proprietario dell'account scegliendo Utente root e inserendo il tuo indirizzo Account AWS email. Nella pagina successiva, inserisci la password.

   Per informazioni sull'accesso utilizzando un utente root, consulta la pagina Signing in as the root user della Guida per l'utente di Accedi ad AWS .
2. Abilita l'autenticazione a più fattori (MFA) per l'utente root.

   Per istruzioni, consulta Abilitare un dispositivo MFA virtuale per l'utente Account AWS root (console) nella Guida per l'utente IAM.

Crea un utente con accesso amministrativo

1. Abilita Centro identità IAM.

Per istruzioni, consulta Abilitazione di AWS IAM Identity Center nella Guida per l'utente di AWS IAM Identity Center .

2. In IAM Identity Center, assegna l'accesso amministrativo a un utente.

   Per un tutorial sull'utilizzo di IAM Identity Center directory come fonte di identità, consulta Configurare l'accesso utente con l'impostazione predefinita IAM Identity Center directory nella Guida per l'AWS IAM Identity Center utente.

Accesso come utente amministratore

- Per accedere con l'utente IAM Identity Center, utilizza l'URL di accesso che è stato inviato al tuo indirizzo e-mail quando hai creato l'utente IAM Identity Center.

  Per informazioni sull'accesso utilizzando un utente IAM Identity Center, consulta AWS Accedere al portale di accesso nella Guida per l'Accedi ad AWS utente.

Assegna l'accesso a ulteriori utenti

1. In IAM Identity Center, crea un set di autorizzazioni conforme alla best practice dell'applicazione di autorizzazioni con il privilegio minimo.

   Segui le istruzioni riportate nella pagina Creazione di un set di autorizzazioni nella Guida per l'utente di AWS IAM Identity Center .

2. Assegna al gruppo prima gli utenti e poi l'accesso con autenticazione unica (Single Sign-On).

   Per istruzioni, consulta Aggiungere gruppi nella Guida per l'utente di AWS IAM Identity Center .

Per fornire l'accesso, aggiungi autorizzazioni agli utenti, gruppi o ruoli:

- Utenti e gruppi in AWS IAM Identity Center:

  Crea un set di autorizzazioni. Segui le istruzioni riportate nella pagina Create a permission set (Creazione di un set di autorizzazioni) nella Guida per l'utente di AWS IAM Identity Center .

- Utenti gestiti in IAM tramite un provider di identità:

  Crea un ruolo per la federazione delle identità. Segui le istruzioni riportate nella pagina Create a role for a third-party identity provider (federation) della Guida per l'utente IAM.

- Utenti IAM:
  - Crea un ruolo che l'utente possa assumere. Segui le istruzioni riportate nella pagina Create a role for an IAM user della Guida per l'utente IAM.
  - (Non consigliato) Collega una policy direttamente a un utente o aggiungi un utente a un gruppo di utenti. Segui le istruzioni riportate nella pagina Aggiunta di autorizzazioni a un utente (console) nella Guida per l'utente IAM.

Configurazione

1. Per scaricare FreerTOS da: GitHub
   1. Vai al repository FreerTOS GitHub .
   2. Scegli Codice e poi Clona o Scarica ZIP.
   3. Dalla riga di comando del computer, clonare il repository in una directory sul computer host.

      ```
      git clone https://github.com/aws/amazon-freertos.git --recurse-submodules
      ```

      Important
      - In questo argomento, il percorso della directory di download di FreerTOS viene definito come. *freertos*
      - Gli spazi contenuti nel percorso *freertos* possono causare errori di compilazione. Quando si clona o si copia il repository, assicurarsi che il percorso creato non contenga spazi.
      - La lunghezza massima di un percorso di file su Microsoft Windows è di 260 caratteri. I lunghi percorsi delle directory di download di FreerTOS possono causare errori di compilazione.

   4. Dalla directory *freertos*, selezionare la ramificazione da utilizzare.
2. Configurazione dell'ambiente di sviluppo.
   a. Installa l'ultima versione di Win. PCap
   b. Installare Microsoft Visual Studio.

      Visual Studio versioni 2017 e 2019 funzionano. Sono supportate tutte le versioni di Visual Studio (Community, Professional o Enterprise).

Oltre all'ambiente di sviluppo integrato (IDE), installare il componente Sviluppo di applicazioni desktop con C++. Quindi, in Optional (Facoltativo), installare l'SDK di Windows 10 più recente.

c. Verificare di disporre di una connessione Ethernet cablata.

Crea ed esegui il progetto demo FreerTOS

> **Important**
>
> Il dispositivo Microchip ATECC6 08A ha un'inizializzazione unica che viene bloccata sul dispositivo la prima volta che viene eseguito un progetto (durante la chiamata a). `C_InitToken` Tuttavia, il progetto demo FreerTOS e il progetto di test hanno configurazioni diverse. Se il dispositivo è bloccato durante le configurazioni del progetto demo, non sarà possibile che tutti i test nel progetto di test abbiano esito positivo.

Per creare ed eseguire il progetto demo FreerTOS con l'IDE di Visual Studio

1. Caricare il progetto in Visual Studio.

   Nel menu File, scegliere Open (Apri). Scegliere File/Solution (File/Soluzione), accedere al file *`freertos`*`\projects\microchip\ecc608a_plus_winsim\visual_studio\aws_demos\aws_demos.sln` e quindi scegliere Open (Apri).

2. Definire una nuova destinazione per il progetto demo.

   Il progetto demo fornito dipende da Windows SDK, ma non ha una versione di Windows SDK specifica. Per impostazione predefinita, l'IDE potrebbe tentare di creare il progetto demo con una versione dell'SDK non presente sul computer in uso. Per impostare la versione di Windows SDK, fare clic con il pulsante destro del mouse su aws_demos e quindi scegliere Retarget Projects (Ridestina progetti). Viene visualizzata la finestra Review Solution Actions (Esamina azioni della soluzione). Scegliere una versione di Windows SDK presente sul computer in uso (è possibile scegliere il valore iniziale nell'elenco a discesa), quindi scegliere OK.

3. Creare ed eseguire il progetto.

   Dal menu Build, scegli Build Solution e assicurati che la soluzione venga compilata senza errori. Scegliere Debug, Start Debugging (Avvia debug) per eseguire il progetto. Alla prima esecuzione,

è necessario configurare l'interfaccia del dispositivo e ricompilare. Per ulteriori informazioni, consulta Configurazione dell'interfaccia di rete.

4. Fornisci il Microchip ATECC6 08A.

   Microchip ha fornito diversi strumenti di scripting per facilitare la configurazione dei componenti 08A. ATECC6 Passare al file *freertos*`\vendors\microchip\secure_elements\app\example_trust_chain_tool` e aprire il file README.md.

   Seguire le istruzioni contenute nel file `README.md` per effettuare il provisioning del dispositivo. Queste fasi includono quanto segue:

   1. Creare e registrare un'autorità di certificazione con AWS.
   2. Genera le tue chiavi sul Microchip ATECC6 08A ed esporta la chiave pubblica e il numero di serie del dispositivo.
   3. Generazione di un certificato per il dispositivo e registrare tale certificato con AWS.
   4. Caricamento del certificato CA e il certificato del dispositivo sul dispositivo.

5. Crea ed esegui esempi FreerTOS.

   Rieseguire nuovamente il progetto demo. Questa volta la connessione dovrebbe funzionare.

Risoluzione dei problemi

Per informazioni generiche sulla risoluzione dei problemi, consultare Nozioni di base sulla risoluzione dei problemi.

Guida introduttiva all'Espressif ESP32 - DevKit C e al ESP-WROVER-KIT

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Segui questo tutorial per iniziare a usare Espressif ESP32 - DevKit C dotato dei moduli ESP32 - WROOM-32, -SOLO-1 o ESP-WROVER e il. ESP32 ESP-WROVER-KIT-VB Per acquistarne uno dal nostro partner sul catalogo Partner Device, utilizza i seguenti link: AWS

- ESP32-WROOM-32 C DevKit
- ESP32-SOLO-1
- ESP32-KIT WROVER

Queste versioni di schede di sviluppo sono supportate su FreerTOS.

Per ulteriori informazioni sulle ultime versioni di queste schede, vedere ESP32- DevKit C V4 o ESP-WROVER-KITv4.1 sul sito Web di Espressif.

> **Note**
>
> Attualmente, la porta FreerTOS ESP32 per -WROVER-KIT ed DevKit ESP C non supporta la funzionalità Symmetric multiprocessing (SMP).

Panoramica

Questo tutorial descrive le seguenti procedure:

1. Collegamento della scheda a una macchina host
2. Installazione del software sulla macchina host per lo sviluppo e il debug di applicazioni integrate per la scheda microcontrollore
3. Compilazione incrociata di un'applicazione demo FreerTOS su un'immagine binaria
4. Caricamento dell'immagine binaria dell'applicazione sulla scheda e quindi esecuzione dell'applicazione
5. Interazione con l'applicazione in esecuzione sulla scheda tramite una connessione seriale, per scopi di monitoraggio e debug

Prerequisiti

Prima di iniziare a usare FreerTOS sulla tua scheda Espressif, devi configurare il tuo account e le autorizzazioni. AWS

Iscriviti per un Account AWS

Se non ne hai uno Account AWS, completa i seguenti passaggi per crearne uno.

Per iscriverti a un Account AWS

1. Apri la https://portal.aws.amazon.com/billing/registrazione.
2. Segui le istruzioni online.

   Parte della procedura di registrazione prevede la ricezione di una telefonata o di un messaggio di testo e l'immissione di un codice di verifica sulla tastiera del telefono.

   Quando ti iscrivi a un Account AWS, Utente root dell'account AWSviene creato un. L'utente root dispone dell'accesso a tutte le risorse e tutti i Servizi AWS nell'account. Come best practice di sicurezza, assegna l'accesso amministrativo a un utente e utilizza solo l'utente root per eseguire attività che richiedono l'accesso di un utente root.

AWS ti invia un'email di conferma dopo il completamento della procedura di registrazione. In qualsiasi momento, puoi visualizzare l'attività corrente del tuo account e gestirlo accedendo a https://aws.amazon.com/e scegliendo Il mio account.

Crea un utente con accesso amministrativo

Dopo esserti registrato Account AWS, proteggi Utente root dell'account AWS AWS IAM Identity Center, abilita e crea un utente amministrativo in modo da non utilizzare l'utente root per le attività quotidiane.

Proteggi i tuoi Utente root dell'account AWS

1. Accedi AWS Management Consolecome proprietario dell'account scegliendo Utente root e inserendo il tuo indirizzo Account AWS email. Nella pagina successiva, inserisci la password.

   Per informazioni sull'accesso utilizzando un utente root, consulta la pagina Signing in as the root user della Guida per l'utente di Accedi ad AWS .
2. Abilita l'autenticazione a più fattori (MFA) per l'utente root.

   Per istruzioni, consulta Abilitare un dispositivo MFA virtuale per l'utente Account AWS root (console) nella Guida per l'utente IAM.

Crea un utente con accesso amministrativo

1. Abilita Centro identità IAM.

Per istruzioni, consulta Abilitazione di AWS IAM Identity Center nella Guida per l'utente di AWS IAM Identity Center .

2. In IAM Identity Center, assegna l'accesso amministrativo a un utente.

   Per un tutorial sull'utilizzo di IAM Identity Center directory come fonte di identità, consulta Configurare l'accesso utente con l'impostazione predefinita IAM Identity Center directory nella Guida per l'AWS IAM Identity Center utente.

Accesso come utente amministratore

- Per accedere con l'utente IAM Identity Center, utilizza l'URL di accesso che è stato inviato al tuo indirizzo e-mail quando hai creato l'utente IAM Identity Center.

  Per informazioni sull'accesso utilizzando un utente IAM Identity Center, consulta AWS Accedere al portale di accesso nella Guida per l'Accedi ad AWS utente.

Assegna l'accesso a ulteriori utenti

1. In IAM Identity Center, crea un set di autorizzazioni conforme alla best practice dell'applicazione di autorizzazioni con il privilegio minimo.

   Segui le istruzioni riportate nella pagina Creazione di un set di autorizzazioni nella Guida per l'utente di AWS IAM Identity Center .

2. Assegna al gruppo prima gli utenti e poi l'accesso con autenticazione unica (Single Sign-On).

   Per istruzioni, consulta Aggiungere gruppi nella Guida per l'utente di AWS IAM Identity Center .

Per fornire l'accesso, aggiungi autorizzazioni agli utenti, gruppi o ruoli:

- Utenti e gruppi in AWS IAM Identity Center:

  Crea un set di autorizzazioni. Segui le istruzioni riportate nella pagina Create a permission set (Creazione di un set di autorizzazioni) nella Guida per l'utente di AWS IAM Identity Center .

- Utenti gestiti in IAM tramite un provider di identità:

  Crea un ruolo per la federazione delle identità. Segui le istruzioni riportate nella pagina Create a role for a third-party identity provider (federation) della Guida per l'utente IAM.

- Utenti IAM:
  - Crea un ruolo che l'utente possa assumere. Segui le istruzioni riportate nella pagina Create a role for an IAM user della Guida per l'utente IAM.
  - (Non consigliato) Collega una policy direttamente a un utente o aggiungi un utente a un gruppo di utenti. Segui le istruzioni riportate nella pagina Aggiunta di autorizzazioni a un utente (console) nella Guida per l'utente IAM.

> **Note**
>
> La versione FreerTOS 202012.00 supporta ESP-IDF v3.3. Se stai usando una versione successiva di FreerTOS, segui le istruzioni specifiche IDFv4 al punto .2 all'indirizzo. Inizia con ESP-IDF v4.2

Inizia con ESP-IDF v3.3

Questa sezione mostra come utilizzare ESP-IDF v3.3 sull'hardware Espressif. Per utilizzare ESP-IDF v4.2, vedere. Inizia con ESP-IDF v4.2

Configurazione dell'hardware di Espressif

Per ulteriori informazioni sulla configurazione dell'hardware della scheda di sviluppo ESP32 - DevKit C, vedere la ESP32- DevKit C V4 Getting Started Guide.

Per informazioni sulla configurazione dell'hardware della scheda di ESP-WROVER-KIT sviluppo, vedere la Guida introduttiva alla ESP-WROVER-KITversione 4.1.

> **Important**
>
> Quando arrivi alla sezione Guida introduttiva delle guide di Espressif, fermati e segui i passaggi seguenti.

Configurazione dell'ambiente di sviluppo

Per comunicare con la tua scheda, devi scaricare e installare una toolchain.

Configurazione della toolchain

> **Note**
>
> ESP-IDF v3.3 non supporta l'ultima versione del compilatore. ESP32 Se è già stata installata la versione più recente del ESP32 compilatore, è necessario disinstallarla e utilizzare la versione compatibile del compilatore inclusa nell'installazione della toolchain. Consultate i collegamenti in questa sezione. Per controllare la versione del compilatore, esegui il seguente comando.
>
> ```
> xtensa-esp32-elf-gcc --version
> ```

Per configurare la toolchain, segui queste istruzioni per il sistema operativo della tua macchina host:

- Configurazione standard di Toolchain (ESP-IDF v3.3) per Windows
- Configurazione standard di Toolchain (ESP-IDF v3.3) per macOS
- Configurazione standard di Toolchain (ESP-IDF v3.3) per Linux

> **Important**
>
> Quando raggiungi le istruzioni «Scarica ESP-IDF» riportate nella sezione Passaggi successivi, interrompi e poi torna alle istruzioni riportate in questa pagina.
> Se avete seguito le istruzioni «Scarica ESP-IDF» riportate nei passaggi successivi, proprio ora o in un'occasione precedente, è necessario cancellare la variabile di `IDF_PATH` ambiente dal sistema prima di continuare. Questa variabile di ambiente è stata impostata automaticamente se hai seguito le istruzioni «Get ESP-IDF».

Installa CMake

Il sistema di CMake build è necessario per creare la demo e le applicazioni di test di FreerTOS per questo dispositivo. FreerTOS supporta le versioni 3.13 e successive.

È possibile scaricare l'ultima versione di from.org. CMake CMake Sono disponibili sia il codice sorgente che la distribuzione binaria.

Per maggiori dettagli sull'utilizzo CMake con FreerTOS, consulta. Utilizzo CMake con FreerTOS

Stabilire una connessione seriale

Per stabilire una connessione seriale tra la macchina host e ESP32 - DevKit C, è necessario installare i driver CP21 VCP 0x USB to UART Bridge. È possibile scaricare i driver da Silicon Labs.

Per stabilire una connessione seriale tra la macchina host e ESP32 -WROVER-KIT, è necessario installare alcuni driver di porta COM virtuale FTDI. Puoi scaricare tali driver da FTDI.

Per ulteriori informazioni, vedere Stabilire una connessione seriale con. ESP32 Dopo aver stabilito una connessione seriale, annotare la porta seriale per la connessione della scheda. Tale informazione sarà necessaria durante il processo di creazione della demo.

Scarica e configura FreerTOS

Dopo aver configurato l'ambiente, puoi scaricare FreerTOS GitHubda .Per istruzioni, consulta il file README.md sul sito web. GitHub

Configura le applicazioni demo FreerTOS

1. Se utilizzi macOS o Linux, apri un prompt del terminale. Se utilizzi Windows, apri il. `mingw32.exe` MinGW è un ambiente di sviluppo minimalista per applicazioni native di Microsoft Windows.
2. Per verificare che Python 2.7.10 o versione successiva sia installato, esegui

   ```
   python --version
   ```

   Viene visualizzata la versione installata. Se non avete installato Python 2.7.10 o versioni successive, potete installarlo dal sito Web di Python.
3. È necessario il AWS Command Line Interface (AWS CLI) per eseguire i comandi. AWS IoT Se utilizzi Windows, usa il easy_install awscli per installarlo AWS CLI nell'ambiente mingw32.

   Se utilizzi macOS o Linux, consulta Installazione di. AWS CLI
4. aws configureEseguili e configurali AWS CLI con l'ID della chiave di AWS accesso, la chiave di accesso segreta e la AWS regione. Per ulteriori informazioni, consultare la pagina relativa alla configurazione di AWS CLI.
5. Usa il seguente comando per installare l' AWS SDK per Python (boto3):

   - Su Windows, nell'ambiente mingw32, esegui

```
easy_install boto3
```

- Su macOS o Linux, esegui

```
pip install tornado nose --user
```

e poi esegui

```
pip install boto3 --user
```

FreerTOS include lo script per semplificare `SetupAWS.py` la configurazione della scheda Espressif a cui connettersi. AWS IoT Per configurare lo script, apri *`freertos`*`/tools/aws_config_quick_start/configure.json` e imposta i seguenti attributi:

`afr_source_dir`

Il percorso completo della directory *`freertos`* sul computer. Assicurarsi di utilizzare le barre per specificare questo percorso.

`thing_name`

Il nome che vuoi assegnare all'oggetto che rappresenta la AWS IoT tua scheda.

`wifi_ssid`

Il SSID della rete Wi-Fi.

`wifi_password`

La password della rete Wi-Fi.

`wifi_security`

Il tipo di sicurezza della rete Wi-Fi.

I seguenti sono tipi di sicurezza validi:

- `eWiFiSecurityOpen` (Aperto, nessuna protezione)
- `eWiFiSecurityWEP` (Sicurezza WEP)
- `eWiFiSecurityWPA` (Sicurezza WPA)
- `eWiFiSecurityWPA2`(WPA2 sicurezza)

Per eseguire lo script di configurazione

1. Se utilizzi macOS o Linux, apri un prompt del terminale. Se usi Windows, apri. `mingw32.exe`
2. Vai alla *`freertos`*`/tools/aws_config_quick_start` directory ed esegui

```
python SetupAWS.py setup
```

Lo script svolge le seguenti funzioni:

- Crea un oggetto, un certificato e una policy IoT
- Collega la policy IoT al certificato e il certificato all'oggetto AWS IoT
- Compila il `aws_clientcredential.h` file con l' AWS IoT endpoint, l'SSID Wi-Fi e le credenziali
- Formatta il certificato e la chiave privata e li scrive nel file di intestazione `aws_clientcredential_keys.h`

> **Note**
>
> Il certificato è codificato solo a scopo dimostrativo. Le applicazioni a livello di produzione devono archiviare questi file in un percorso sicuro.

Per ulteriori informazioni in merito`SetupAWS.py`, vedere `README.md` nella directory. *`freertos`*`/tools/aws_config_quick_start`

Crea, esegui il flashing ed esegui il progetto demo FreerTOS

Puoi usare CMake per generare i file di build, Make per creare il file binario dell'applicazione e l'utilità IDF di Espressif per eseguire il flashing della scheda.

Crea FreerTOS su Linux e macOS (ESP-IDF v3.3)

Se usi Windows, vedi. Crea FreerTOS su Windows (ESP-IDF v3.3)

CMake Utilizzatelo per generare i file di compilazione, quindi utilizzate Make per creare l'applicazione.

Per generare i file di build dell'applicazione demo con CMake

1. Cambia le directory nella cartella principale della cartella di download di FreerTOS.

2. Usa il seguente comando per generare i file di build.

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

> **Note**
>
> Se si desidera creare l'applicazione per il debug, aggiungere il flag `-DCMAKE_BUILD_TYPE=Debug` al comando.
> Se si desidera generare i file compilati dell'applicazione di test, aggiungere il flag `-DAFR_ENABLE_TESTS=1`.
> Il codice fornito da Espressif utilizza lo stack IP leggero (lwIP) come stack di rete predefinito. Per utilizzare invece lo stack di rete FreeRTOS+TCP, aggiungi il flag al comando. `–DAFR_ESP_FREERTOS_TCP` CMake
> Per aggiungere la dipendenza LWip per il codice fornito non dal fornitore, aggiungi le seguenti righe al file `CMakeLists.txt` delle dipendenze, per CMake il tuo componente Wi-Fi personalizzato.
>
> ```
> # Add a dependency on the bluetooth espressif component to the common
>  component
> set(COMPONENT_REQUIRES lwip)
> ```

Per compilare l'applicazione con Make

1. Passare alla directory `build`.
2. Usa il comando seguente per creare l'applicazione con Make.

```
make all -j4
```

> **Note**
>
> È necessario generare i file compilati con il comando cmake ogni volta che si passa dal progetto `aws_demos` al progetto `aws_tests`.

Crea FreerTOS su Windows (ESP-IDF v3.3)

Su Windows, è necessario specificare un generatore di build per. CMake Altrimenti, il CMake valore predefinito è Visual Studio. Espressif consiglia ufficialmente il sistema di compilazione Ninja perché funziona in Windows, Linux e MacOS. È necessario eseguire CMake i comandi in un ambiente Windows nativo come cmd o. PowerShell L'esecuzione di CMake comandi in un ambiente Linux virtuale, ad esempio MSYS2 o WSL, non è supportata.

CMake Utilizzatelo per generare i file di compilazione, quindi utilizzate Make per creare l'applicazione.

Per generare i file di build dell'applicazione demo con CMake

1. Cambia le directory nella cartella principale della cartella di download di FreerTOS.
2. Usa il seguente comando per generare i file di build.

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

> **Note**
>
> Se si desidera creare l'applicazione per il debug, aggiungere il flag `-DCMAKE_BUILD_TYPE=Debug` al comando.
> Se si desidera generare i file compilati dell'applicazione di test, aggiungere il flag `-DAFR_ENABLE_TESTS=1`.
> Il codice fornito da Espressif utilizza lo stack IP leggero (lwIP) come stack di rete predefinito. Per utilizzare invece lo stack di rete FreeRTOS+TCP, aggiungi il flag al comando. `–DAFR_ESP_FREERTOS_TCP` CMake
> Per aggiungere la dipendenza LWip per il codice fornito non dal fornitore, aggiungi le seguenti righe al file `CMakeLists.txt` delle dipendenze, per CMake il tuo componente Wi-Fi personalizzato.
>
> ```
> # Add a dependency on the bluetooth espressif component to the common
>  component
> set(COMPONENT_REQUIRES lwip)
> ```

Per compilare l'applicazione

1. Passare alla directory `build`.
2. Invoca Ninja per creare l'applicazione.

```
ninja
```

Oppure, usa l' CMake interfaccia generica per creare l'applicazione.

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

> **Note**
>
> È necessario generare i file compilati con il comando cmake ogni volta che si passa dal progetto `aws_demos` al progetto `aws_tests`.

Esegui il flashing ed esegui FreerTOS (ESP-IDF v3.3)

Questa sezione mostra come utilizzare ESP-IDF v3.3 sull'hardware Espressif. Per utilizzare ESP-IDF v4.2, vedere. Inizia con ESP-IDF v4.2

Utilizza la utilità IDF di Espressif (*`freertos`*`/vendors/espressif/esp-idf/tools/idf.py`) per eseguire il flashing della scheda, eseguire l'applicazione e visualizzare i log.

Per cancellare il flash della scheda, vai alla *`freertos`* directory e usa il seguente comando.

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

Per eseguire il flashing del file binario dell'applicazione sulla tua scheda, usa `make`

```
make flash
```

È inoltre possibile utilizzare lo script IDF per eseguire il flashing della scheda:

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

Per monitorare:

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

> **Note**
>
> È possibile combinare questi comandi. Per esempio:
>
> ```
> ./vendors/espressif/esp-idf/tools/idf.py erase_flash flash monitor -p /dev/
> ttyUSB1 -B build-directory
> ```

Inizia con ESP-IDF v4.2

Questa sezione mostra come utilizzare ESP-IDF v4.2 sull'hardware Espressif. Per utilizzare ESP-IDF v3.3, vedere. Inizia con ESP-IDF v3.3

> **Note**
>
> I comandi Linux in questo tutorial richiedono l'uso della shell Bash.

Configurazione dell'hardware di Espressif

Per informazioni sulla configurazione dell'hardware della scheda di sviluppo ESP32 - DevKit C, consultate la ESP32- Guida introduttiva di DevKit C V4.

Per informazioni sulla configurazione dell'hardware della scheda di ESP-WROVER-KIT sviluppo, vedere la Guida introduttiva alla ESP-WROVER-KIT V4.1.

> **Important**
>
> Quando arrivi alla sezione Guida introduttiva delle guide di Espressif, fermati e poi torna alle istruzioni in questa pagina.

Configurazione dell'ambiente di sviluppo

Per comunicare con la tua scheda, devi scaricare e installare una toolchain.

Configurazione della toolchain

Per configurare la toolchain, segui queste istruzioni per il sistema operativo della tua macchina host:

- Configurazione standard della toolchain e prerequisiti (ESP-IDF v4.2) per Windows
- Configurazione standard dei prerequisiti della Toolchain (ESP-IDF v4.2) per macOS
- Configurazione standard dei prerequisiti della Toolchain (ESP-IDF v4.2) per Linux

> **Important**
>
> Quando raggiungi le istruzioni «Scarica ESP-IDF» riportate nella sezione Passaggi successivi, interrompi e torna alle istruzioni riportate in questa pagina.

Completa l'installazione (Linux/macOS)

Il programma di installazione di ESP-IDF per Windows installa tutti gli strumenti necessari. Le piattaforme Linux e macOS richiedono un passaggio aggiuntivo per completare l'installazione.

1. Aprire una finestra della riga di comando
2. Vai alla directory di download di FreerTOS, quindi esegui lo script seguente per scaricare e installare la toolchain espressif per la tua piattaforma.

   ```
   vendors/espressif/esp-idf/install.sh
   ```

3. Aggiungi gli strumenti della toolchain ESP-IDF al percorso del tuo terminale con il seguente comando.

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

Stabilire una connessione seriale

Per stabilire una connessione seriale tra la macchina host e ESP32 - DevKit C, è necessario installare i driver CP21 VCP 0x USB to UART Bridge. È possibile scaricare i driver da Silicon Labs.

Per stabilire una connessione seriale tra la macchina host e ESP32 -WROVER-KIT, è necessario installare il driver della porta COM virtuale FTDI. È possibile scaricare questo driver da FTDI.

Per ulteriori informazioni, vedere Stabilire una connessione seriale con ESP32. Dopo aver stabilito una connessione seriale, annotare la porta seriale per la connessione della scheda. Tale informazione sarà necessaria durante il processo di creazione della demo.

Scarica e configura FreerTOS

Dopo aver configurato il tuo ambiente, puoi scaricare FreerTOS da. GitHub Per istruzioni, consultate il file README.md sul sito Web. GitHub

Configura le applicazioni demo FreerTOS

1. Se utilizzi macOS o Linux, apri un prompt del terminale. Se usi Windows, apri. mingw32.exe (MinGW è un ambiente di sviluppo minimalista per applicazioni Microsoft Windows native.)
2. Per verificare che Python3 sia installato, esegui

   ```
   python --version
   ```

   Viene visualizzata la versione installata. Se non avete installato Python 3.0.1 o versioni successive, potete installarlo dal sito Web di Python.
3. È necessario il AWS Command Line Interface (AWS CLI) per eseguire i comandi. AWS IoT Se utilizzi Windows, usa il easy_install awscli per installarlo AWS CLI nell'ambiente mingw32.

   Se utilizzi macOS o Linux, consulta Installazione di. AWS CLI
4. Esecuzione

   ```
   aws configure
   ```

   e configurali AWS CLI con l'ID della chiave di AWS accesso, la chiave di accesso segreta e la AWS regione predefinita. Per ulteriori informazioni, consultare la pagina relativa alla configurazione di AWS CLI.
5. Usa il seguente comando per installare l' AWS SDK per Python (boto3):

   - Su Windows, nell'ambiente mingw32, esegui

     ```
     easy_install boto3
     ```

   - Su macOS o Linux, esegui

```
pip install tornado nose --user
```

e poi esegui

```
pip install boto3 --user
```

FreerTOS include lo script per semplificare `SetupAWS.py` la configurazione della scheda Espressif a cui connettersi. AWS IoT Per configurare lo script, apri *`freertos`*`/tools/aws_config_quick_start/configure.json` e imposta i seguenti attributi:

`afr_source_dir`

Il percorso completo della directory *`freertos`* sul computer. Assicurarsi di utilizzare le barre per specificare questo percorso.

`thing_name`

Il nome che vuoi assegnare all'oggetto che rappresenta la AWS IoT tua scheda.

`wifi_ssid`

Il SSID della rete Wi-Fi.

`wifi_password`

La password della rete Wi-Fi.

`wifi_security`

Il tipo di sicurezza della rete Wi-Fi.

I seguenti sono tipi di sicurezza validi:

- `eWiFiSecurityOpen` (Aperto, nessuna protezione)
- `eWiFiSecurityWEP` (Sicurezza WEP)
- `eWiFiSecurityWPA` (Sicurezza WPA)
- `eWiFiSecurityWPA2`(WPA2 sicurezza)

Per eseguire lo script di configurazione

1. Se utilizzi macOS o Linux, apri un prompt del terminale. Se usi Windows, apri il `mingw32.exe` file.
2. Vai alla *`freertos`*`/tools/aws_config_quick_start` directory ed esegui

```
python SetupAWS.py setup
```

Lo script svolge le seguenti funzioni:

- Crea un oggetto, un certificato e una policy IoT
- Collega la policy IoT al certificato e il certificato all'oggetto AWS IoT
- Compila il `aws_clientcredential.h` file con l' AWS IoT endpoint, l'SSID Wi-Fi e le credenziali
- Formatta il certificato e la chiave privata e li scrive nel file di intestazione `aws_clientcredential_keys.h`

> **Note**
>
> Il certificato è codificato solo a scopo dimostrativo. Le applicazioni a livello di produzione devono archiviare questi file in un percorso sicuro.

Per ulteriori informazioni in merito`SetupAWS.py`, vedere `README.md` nella directory. *`freertos`*`/tools/aws_config_quick_start`

Compila, esegui il flashing ed esegui il progetto demo FreerTOS utilizzando lo script idf.py

Puoi usare l'utilità IDF di Espressif per generare i file di build, creare il binario dell'applicazione e aggiornare la tua scheda.

Crea ed esegui il flashing di FreerTOS su Windows, Linux e macOS (ESP-IDF v4.2)

Usa `idf.py` lo script per creare il progetto ed esegui il flashing dei file binari sul tuo dispositivo.

> **Note**
>
> Alcune configurazioni potrebbero richiedere l'utilizzo dell'opzione port "`-p` *`port-name`*" with `idf.py` per specificare la porta corretta, come nell'esempio seguente.

```
idf.py -p /dev/cu.usbserial-00101301B flash
```

Per creare e aggiornare il progetto

1. Vai alla directory principale della tua cartella di download di FreerTOS.
2. In una finestra della riga di comando, inserisci il seguente comando per aggiungere gli strumenti ESP-IDF al PATH del tuo terminale.

   Windows

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

   Linux/ macOS

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

3. Configura cmake nella `build` directory e crea l'immagine del firmware con il seguente comando.

   ```
   idf.py -DVENDOR=espressif -DBOARD=esp32_wrover_kit -DCOMPILER=xtensa-esp32 build
   ```

   L'output restituito dovrebbe essere simile al seguente.

   ```
   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 non ci sono errori, la build genererà i file.bin binari del firmware.

4. Cancellate la memoria flash della scheda di sviluppo con il seguente comando.

```
idf.py erase_flash
```

5. Usa `idf.py` lo script per eseguire il flashing del file binario dell'applicazione sulla tua scheda.

```
idf.py flash
```

6. Monitora l'uscita dalla porta seriale della scheda con il seguente comando.

```
idf.py monitor
```

> **Note**
>
> È possibile combinare questi comandi come nell'esempio seguente.
>
> ```
> idf.py erase_flash flash monitor
> ```

> **Note**
>
> Per alcune configurazioni della macchina host, è necessario specificare la porta quando si esegue il flashing della scheda, come nell'esempio seguente.
>
> ```
> idf.py erase_flash flash monitor -p /dev/ttyUSB1
> ```

Crea ed esegui il flashing di FreerTOS con CMake

Oltre allo `idf.py` script fornito dall'IDF SDK per creare ed eseguire il codice, puoi anche creare il progetto con. CMake Attualmente supporta Unix Makefiles o il sistema di build Ninja.

Per creare e aggiornare il progetto

1. In una finestra a riga di comando, vai alla radice della tua directory di download di FreerTOS.
2. Esegui lo script seguente per aggiungere gli strumenti ESP-IDF al PATH della tua shell.

   Windows

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

   Linux/ macOS

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

3. Immettete il seguente comando per generare i file di build.

   Con Unix Makefiles

   ```
   cmake -DVENDOR=espressif -DBOARD=esp32_wrover_kit -DCOMPILER=xtensa-esp32 -S . -
   B ./YOUR_BUILD_DIRECTORY -DAFR_ENABLE_ALL_MODULES=1 -DAFR_ENABLE_TESTS=0
   ```

   Con Ninja

   ```
   cmake -DVENDOR=espressif -DBOARD=esp32_wrover_kit -DCOMPILER=xtensa-esp32 -S . -
   B ./YOUR_BUILD_DIRECTORY -DAFR_ENABLE_ALL_MODULES=1 -DAFR_ENABLE_TESTS=0 -GNinja
   ```

4. Cancella il flash e poi fai lampeggiare la lavagna.

   Con Unix Makefiles

   ```
   make -C ./YOUR_BUILD_DIRECTORY erase_flash
   ```

   ```
   make -C ./YOUR_BUILD_DIRECTORY flash
   ```

   Con Ninja

   ```
   ninja -C ./YOUR_BUILD_DIRECTORY erase_flash
   ```

   ```
   ninja -C ./YOUR_BUILD_DIRECTORY flash
   ```

Monitoraggio dei messaggi MQTT in cloud

Puoi utilizzare il client MQTT nella AWS IoT console per monitorare i messaggi che il tuo dispositivo invia al AWS Cloud.

Per abbonarsi all'argomento MQTT con il client MQTT AWS IoT

1. Passare alla console AWS IoT.
2. Nel riquadro di navigazione scegliere Test per aprire il client MQTT.
3. In Argomento sottoscrizione, digitare ***your-thing-name*****/example/topic**, quindi scegliere Effettua sottoscrizione all'argomento.

Esecuzione delle demo Bluetooth Low-Energy

FreerTOS supporta la connettività Bluetooth Low Energy.

Per eseguire il progetto demo FreerTOS su Bluetooth Low Energy, è necessario eseguire l'applicazione demo FreerTOS Bluetooth Low Energy Mobile SDK su un dispositivo mobile iOS o Android.

Per configurare l'applicazione demo SDK mobile FreerTOS Bluetooth Low Energy

1. Segui le istruzioni in Mobile SDKs for FreerTOS Bluetooth Devices per scaricare e installare l'SDK per la tua piattaforma mobile sul tuo computer host.
2. Segui le istruzioni nell'applicazione demo FreerTOS Bluetooth Low Energy Mobile SDK per configurare l'applicazione mobile demo sul tuo dispositivo mobile.

Per istruzioni su come eseguire la demo MQTT su Bluetooth Low Energy sulla scheda, consulta l'applicazione demo MQTT su Bluetooth Low Energy.

Per istruzioni su come eseguire la demo Provisioning Wi-Fi sulla scheda, consulta l'applicazione demo Provisioning Wi-Fi.

Usare FreerTOS nel proprio progetto per CMake ESP32

Se vuoi utilizzare FreerTOS nel CMake tuo progetto, puoi configurarlo come sottodirectory e crearlo insieme alla tua applicazione. Per prima cosa, procurati una copia di FreerTOS da. GitHub Puoi anche configurarlo come sottomodulo Git con il seguente comando in modo che sia più facile aggiornarlo in futuro.

```
git submodule add -b release https://github.com/aws/amazon-freertos.git freertos
```

Se viene rilasciata una versione successiva, puoi aggiornare la tua copia locale con questi comandi.

```
# 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 il progetto ha la seguente struttura di cartelle:

```
- freertos (the copy that you obtained from GitHub or the AWS IoT console)
- src
  - main.c (your application code)
- CMakeLists.txt
```

Quindi quello che segue è un esempio del `CMakeLists.txt` file di primo livello che può essere usato per creare la tua applicazione insieme a 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
    )
```

Per creare il progetto, esegui i seguenti comandi. CMake Assicurati che il ESP32 compilatore sia nella variabile di ambiente PATH.

```
cmake -S . -B build-directory -DCMAKE_TOOLCHAIN_FILE=freertos/tools/cmake/toolchains/
xtensa-esp32.cmake -GNinja
```

```
cmake --build build
```

Per eseguire il flashing dell'applicazione sulla scheda, esegui il comando seguente.

```
cmake --build build-directory --target flash
```

Utilizzo di componenti di FreerTOS

Dopo l'esecuzione CMake, puoi trovare tutti i componenti disponibili nell'output di riepilogo. Dovrebbe assomigliare all'esempio seguente.

```
====================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:
==============================================================================
```

È possibile fare riferimento a qualsiasi componente dall'`Modules to build`elenco. Per collegarli all'applicazione, inserite lo spazio dei `AFR::` nomi davanti al nome, ad esempio `AFR::mqttAFR::ota`, e così via.

Aggiungi componenti personalizzati a ESP-IDF

È possibile aggiungere altri componenti all'ambiente di compilazione ESP-IDF. Ad esempio, supponiamo che vogliate aggiungere un componente chiamato `example_component` e che il vostro progetto abbia questo aspetto

```
- freertos
- components
  - example_component
    - include
      - example_component.h
    - src
      - example_component.c
    - CMakeLists.txt
- src
  - main.c
- CMakeLists.txt
```

quindi il seguente è un esempio del `CMakeLists.txt` file per il 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)
```

È inoltre possibile specificare le dipendenze utilizzando la CMake funzione `target_link_libraries` standard. Si noti che il nome di destinazione per il componente è memorizzato nella variabile `COMPONENT_TARGET`, definita da 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)
```

Per i componenti ESP, ciò avviene impostando due variabili `COMPONENT_REQUIRES` e. `COMPONENT_PRIV_REQUIRES` Vedi Build System (CMake) nella Guida alla programmazione 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)
```

Quindi, nel file `CMakeLists.txt` di livello superiore, indicare a ESP-IDF dove trovare questi componenti. Aggiungete prima le righe seguenti in qualsiasi punto. `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})
```

Questo componente è ora automaticamente collegato al codice dell'applicazione per impostazione predefinita. Dovresti essere in grado di includere i relativi file di intestazione e chiamare le funzioni che definisce.

Sovrascrivi le configurazioni per FreerTOS

Al momento non esiste un approccio ben definito per ridefinire le configurazioni al di fuori dell'albero dei sorgenti di FreerTOS. Per impostazione predefinita, CMake cercherà le directory and. *freertos*/vendors/espressif/boards/esp32/aws_demos/config_files/ *freertos*/demos/include/ Tuttavia, è possibile utilizzare una soluzione alternativa per indicare al compilatore di cercare prima altre directory. Ad esempio, puoi aggiungere un'altra cartella per le configurazioni di FreerTOS.

```
- freertos
- freertos-configs
  - aws_clientcredential.h
  - aws_clientcredential_keys.h
  - iot_mqtt_agent_config.h
  - iot_config.h
- components
- src
- CMakeLists.txt
```

I file in freertos-configs vengono copiati dalle directory *freertos*/vendors/espressif/boards/esp32/aws_demos/config_files/ e *freertos*/demos/include/. Quindi, nel CMakeLists.txt file di primo livello, aggiungi prima questa riga add_subdirectory(freertos) in modo che il compilatore cerchi prima questa directory.

```
include_directories(BEFORE freertos-configs)
```

Fornire il file sdkconfig per ESP-IDF

Nel caso in cui desideri fornire la tuasdkconfig.default, puoi impostare la CMake variabile IDF_SDKCONFIG_DEFAULTS dalla riga di 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 non specifichi una posizione per il tuo sdkconfig.default file, FreerTOS utilizzerà il file predefinito che si trova in. *freertos*/vendors/espressif/boards/esp32/aws_demos/sdkconfig.defaults

Riepilogo

Se hai un progetto con un componente denominato `example_component`, e vuoi sovrascrivere alcune configurazioni, ecco un esempio completo del file `CMakeLists.txt` di livello superiore.

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)
```

Per ulteriori informazioni sul sistema di compilazione Espressif con CMake, consulta Build System nelle guide alle API di Espressif

Risoluzione dei problemi

- Se utilizzi macOS e il sistema operativo non riconosce il tuo ESP-WROVER-KIT, assicurati di non avere i driver D2XX installati. Per disinstallarli, segui le istruzioni contenute in FTDI Drivers Installation Guide for macOS X (Guida per l'installazione dei driver FTDI per macOS X).
- L'utility di monitoraggio fornita da ESP-IDF (e richiamata utilizzando make monitor) consente di decodificare gli indirizzi. Per questo motivo, può aiutarti a ottenere alcuni backtrace significativi nel caso in cui l'applicazione smetta di funzionare. Per ulteriori informazioni, consulta Automatic Address Decoding sul sito web di Espressif.

- È anche possibile abilitare GDBstub la comunicazione con gdb senza richiedere alcun hardware JTAG speciale. Per maggiori informazioni, consulta Launching GDB with GDBStub sul sito web di Espressif.
- Per informazioni sulla configurazione di un ambiente basato su OpenOCD se è necessario il debug basato su hardware JTAG, vedere JTAG Debugging sul sito Web di Espressif.
- Se non `pyserial` può essere `pip` installato utilizzando macOS, scaricalo dal sito Web pyserial.
- Se la scheda si resetta continuamente, prova a cancellare il flash inserendo il seguente comando sul terminale.

```
make erase_flash
```

- Se riscontri errori quando esegui `idf_monitor.py`, utilizza Python 2.7.
- Le librerie richieste da ESP-IDF sono incluse in FreerTOS, quindi non è necessario scaricarle esternamente. Se la variabile di `IDF_PATH` ambiente è impostata, ti consigliamo di cancellarla prima di creare FreerTOS.
- In Windows, la creazione del progetto può richiedere 3-4 minuti. Puoi utilizzare lo switch `-j4` sul comando make per ridurre il tempo di creazione:

```
make flash monitor -j4
```

- Se il dispositivo ha problemi di connessione AWS IoT, apri il `aws_clientcredential.h` file e verifica che le variabili di configurazione siano definite correttamente nel file. `clientcredentialMQTT_BROKER_ENDPOINT[]`dovrebbe assomigliare a*`1234567890123`*`-ats.iot.`*`us-east-1`*`.amazonaws.com`.
- Se stai seguendo la procedura descritta in Usare FreerTOS nel proprio progetto per CMake ESP32 e visualizzi errori di riferimento non definiti dal linker, in genere è dovuto alla mancanza di librerie dipendenti o demo. Per aggiungerli, aggiorna il `CMakeLists.txt` file (nella directory principale) usando la CMake funzione standard`target_link_libraries`.

- ESP-IDF v4.2 supporta l'uso di xtensa-esp32-elf-gcc 8.2.0. toolchain. Se utilizzi una versione precedente della toolchain Xtensa, scarica la versione richiesta.

- Se vedi un registro degli errori come il seguente sulle dipendenze di Python che non vengono soddisfatte per 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
```

Installa le dipendenze python sulla tua piattaforma usando il seguente comando Python:

```
root/vendors/espressif/esp-idf/requirements.txt
```

Per ulteriori informazioni sulla risoluzione dei problemi, vedere. Nozioni di base sulla risoluzione dei problemi

Debug

Codice di debug su Espressif ESP32 - DevKit C e (ESP-IDF v3.3) ESP-WROVER-KIT

Questa sezione mostra come eseguire il debug dell'hardware Espressif utilizzando ESP-IDF v3.3. Per eseguire il debug utilizzando ESP-IDF v4.2, vedere. Codice di debug su Espressif ESP32 - DevKit C e (ESP-IDF v4.2) ESP-WROVER-KIT

È necessario un cavo da JTAG a USB. Usiamo un cavo da USB a MPSSE (per esempio, il FTDI C232HM-DDHSL-0).

Configurazione ESP-C JTAG DevKit

Per il cavo FTDI C232HM-DDHSL-0, queste sono le connessioni al devKitC. ESP32

C232HM-DDHSL-0 Wire Color	ESP32 Pin GPIO	JTAG Signal Name
Brown (pin 5)	IO14	TMS
Yellow (pin 3)	IO12	TDI

C232HM-DDHSL-0 Wire Color	ESP32 Pin GPIO	JTAG Signal Name
Black (pin 10)	GND	GND
Orange (pin 2)	IO13	TCK
Verde (pin 4)	IO15	TDO

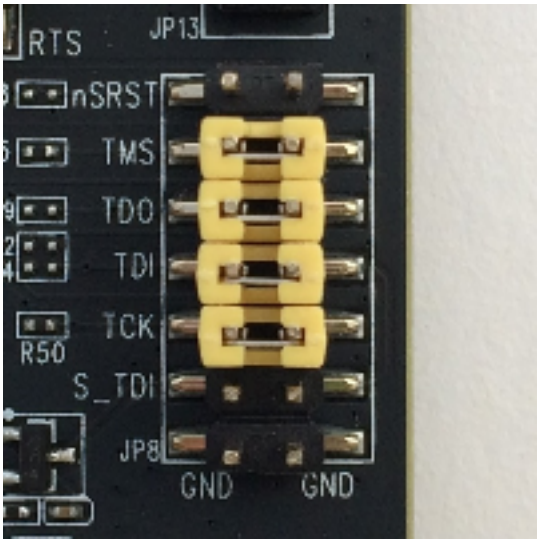
ESP-WROVER-KIT Configurazione JTAG

Per il cavo FTDI C232HM-DDHSL-0, queste sono le connessioni al -WROVER-KIT. ESP32

C232HM-DDHSL-0 Wire Color	ESP32 Pin GPIO	JTAG Signal Name
Brown (pin 5)	IO14	TMS
Yellow (pin 3)	IO12	TDI
Orange (pin 2)	IO13	TCK
Verde (pin 4)	IO15	TDO

Queste tabelle sono state sviluppate dalla scheda tecnica FTDI C232HM-DDHSL-0. Per ulteriori informazioni, vedere la sezione «Connessione via cavo MPSSE C232HM e dettagli meccanici» nella scheda tecnica.

Per abilitare JTAG su ESP-WROVER-KIT, posizionate i ponticelli sui pin TMS, TDO, TDI, TCK e S_TDI come mostrato qui.



Debug su Windows (ESP-IDF v3.3)

Per configurare il debug su Windows

1. Collegare il lato USB di FTDI C232HM-DDHSL-0 sul tuo computer e sull'altro lato come descritto in Codice di debug su Espressif ESP32 - DevKit C e (ESP-IDF v3.3) ESP-WROVER-KIT. Il dispositivo FTDI C232HM-DDHSL-0 deve essere visualizzato in Device Manager (Gestione dispositivi) in Universal Serial Bus Controllers (Controller bus seriale universale).
2. Nell'elenco dei dispositivi USB, fare clic con il tasto destro sul dispositivo C232HM-DDHSL-0 e selezionare Properties (Proprietà).

   > **Note**
   >
   > Il dispositivo potrebbe essere elencato come USB Serial Port (Porta seriale USB).

   Nella finestra proprietà, scegliere la scheda Details (Dettagli) per visualizzare le proprietà del dispositivo. Se il dispositivo non è presente nell'elenco, installate il driver Windows per FTDI C232HM-DDHSL-0.
3. Nella scheda Dettagli, scegliete Proprietà, quindi scegliete Hardware. IDs Dovresti vedere qualcosa di simile nel campo Valore.

   ```
   FTDIBUS\COMPORT&VID_0403&PID_6014
   ```

   In questo esempio, l'ID fornitore è 0403 e l'ID prodotto è 6014.

Verifica che IDs corrispondano al IDs pin`projects/espressif/esp32/make/aws_demos/esp32_devkitj_v1.cfg`. IDs Sono specificati in una riga che inizia con `ftdi_vid_pid` seguita da un ID fornitore e da un ID prodotto.

```
ftdi_vid_pid 0x0403 0x6014
```

4. Scaricare OpenOCD per Windows.
5. Decomprimere il file su `C:\` e aggiungere `C:\openocd-esp32\bin` al percorso di sistema.
6. OpenOCD richiede libusb, che non è installato per impostazione predefinita in Windows.

   Per installare libusb

   a. Scaricare zadig.exe.
   b. Esegui `zadig.exe`. Nel menu, Options (Opzioni) scegliere List All Devices (Elenca tutti i dispositivi).
   c. Dal menu a discesa del metodo, scegliere C232HM-DDHSL-0.
   d. Nel campo driver di destinazione, a destra della freccia verde, selezionare WinUSB (WinUSB).
   e. Dall'elenco a discesa sotto il campo del driver di destinazione, scegli la freccia, quindi scegli Installa driver. Scegliere Replace Driver (Sostituisci driver).
7. Apri un prompt dei comandi, accedi `projects/espressif/esp32/make/aws_demos` ed esegui il comando seguente.

   Per ESP32 -WROOM-32 e -WROVER ESP32

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

   Per ESP32 -SOLO-1

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

   Lasciare questo prompt dei comandi aperto.
8. Aprire un nuovo prompt dei comandi, accedere alla directory `msys32` ed eseguire `mingw32.exe`.
9. Nel terminale mingw32, accedi ed esegui `projects/espressif/esp32/make/aws_demos`

```
make flash monitor
```

10. Aprire un altro terminale mingw32, passare a `projects/espressif/esp32/make/aws_demos` e attendere l'avvio dell'esecuzione della demo nella scheda. Quando inizia, eseguire `xtensa-esp32-elf-gdb -x gdbinit build/aws_demos.elf`. Il programma deve arrestarsi nella funzione `main`.

> **Note**
>
> ESP32 Supporta un massimo di due punti di interruzione.

Debug su macOS (ESP-IDF v3.3)

1. Scaricare il driver FTDI per macOS.
2. Scaricare OpenOCD.
3. Estrarre il file .tar scaricato e impostare il percorso in `.bash_profile` su *`OCD_INSTALL_DIR`*`/openocd-esp32/bin`.
4. Usa il seguente comando per l'installazione `libusb` su macOS.

   ```
   brew install libusb
   ```

5. Usa il seguente comando per scaricare il driver della porta seriale.

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

6. Se utilizzi una versione macOS successiva alla 10.9, usa il seguente comando per scaricare il driver FTDI Apple.

   ```
   sudo kextunload -b com.apple.driver.AppleUSBFTDI
   ```

7. Utilizzare il comando seguente per ottenere l'ID prodotto e l'ID fornitore del cavo FTDI. Elenca i dispositivi USB collegati.

   ```
   system_profiler SPUSBDataType
   ```

   L'output di `system_profiler` dovrebbe essere simile al seguente.

```
DEVICE:

Product ID: product-ID
Vendor ID: vendor-ID (Future Technology Devices International Limited)
```

8. Apri il file `projects/espressif/esp32/make/aws_demos/esp32_devkitj_v1.cfg`. L'ID fornitore e l'ID prodotto sono specificati in una riga che inizia con `ftdi_vid_pid`. Modifica il valore IDs in modo che IDs corrisponda a quello dell'`system_profiler`output del passaggio precedente.
9. Aprire la finestra del terminale, passare a `projects/espressif/esp32/make/aws_demos` e utilizzare il comando seguente per eseguire OpenOCD:

   Per ESP32 -WROOM-32 e -WROVER: ESP32

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

   Per ESP32 -SOLO-1:

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

10. Aprire un nuovo terminale e utilizzare il seguente comando per caricare il driver della porta seriale FTDI.

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

11. Passare a `projects/espressif/esp32/make/aws_demos` ed eseguire il comando seguente:

    ```
    make flash monitor
    ```

12. Apri un altro nuovo terminale, naviga `projects/espressif/esp32/make/aws_demos` ed esegui il comando seguente.

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

    Il programma deve interrompersi su `main()`.

Debug su Linux (ESP-IDF v3.3)

1. Scaricare OpenOCD. Estrarre il tarball e seguire le istruzioni per l'installazione del file readme.
2. Usa il seguente comando per installare libusb su Linux.

```
sudo apt-get install libusb-1.0
```

3. Aprire un terminale e immettere **`ls -l /dev/ttyUSB*`** per elencare tutti i dispositivi USB collegati al computer. Questo ti aiuta a verificare se le porte USB della scheda sono riconosciute dal sistema operativo. L'output restituito dovrebbe essere simile al seguente.

```
$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. Uscire ed effettuare di nuovo l'accesso e spegnere e accendere la scheda per rendere le modifiche effettive. In un prompt del terminale, elencare i dispositivi USB. Assicurati che il proprietario del gruppo sia passato da `dialout` a `plugdev`.

```
$ls -l /dev/ttyUSB*
crw-rw----    1    root    plugdev    188,    0    Jul    10    19:04    /dev/
ttyUSB0
crw-rw----    1    root    plugdev    188,    1    Jul    10    19:04    /dev/
ttyUSB1
```

   L'interfaccia `/dev/ttyUSBn` con il numero inferiore viene utilizzato per le comunicazioni JTAG. L'altra interfaccia viene indirizzata alla ESP32 porta seriale (UART) e viene utilizzata per caricare il codice nella memoria flash ESP32 del computer.

5. In una finestra del terminale, passare a `projects/espressif/esp32/make/aws_demos` e utilizzare il comando seguente per eseguire OpenOCD:

   Per ESP32 -WROOM-32 e -WROVER: ESP32

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

   Per ESP32 -SOLO-1:

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

6. Apri un altro terminale, naviga ed `projects/espressif/esp32/make/aws_demos` esegui il seguente comando.

```
make flash monitor
```

7. Aprire un altro terminale, passare a `projects/espressif/esp32/make/aws_demos` ed eseguire il comando seguente:

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

   Il programma deve interrompersi su `main()`.

Codice di debug su Espressif ESP32 - DevKit C e (ESP-IDF v4.2) ESP-WROVER-KIT

Questa sezione mostra come eseguire il debug dell'hardware Espressif utilizzando ESP-IDF v4.2. Per eseguire il debug utilizzando ESP-IDF v3.3, vedere. Codice di debug su Espressif ESP32 - DevKit C e (ESP-IDF v3.3) ESP-WROVER-KIT

È necessario un cavo da JTAG a USB. Usiamo un cavo da USB a MPSSE (per esempio, il FTDI C232HM-DDHSL-0).

Configurazione ESP-C JTAG DevKit

Per il cavo FTDI C232HM-DDHSL-0, queste sono le connessioni al devKitC. ESP32

C232HM-DDHSL-0 Wire Color	ESP32 Pin GPIO	JTAG Signal Name
Brown (pin 5)	IO14	TMS
Yellow (pin 3)	IO12	TDI
Black (pin 10)	GND	GND
Orange (pin 2)	IO13	TCK
Verde (pin 4)	IO15	TDO

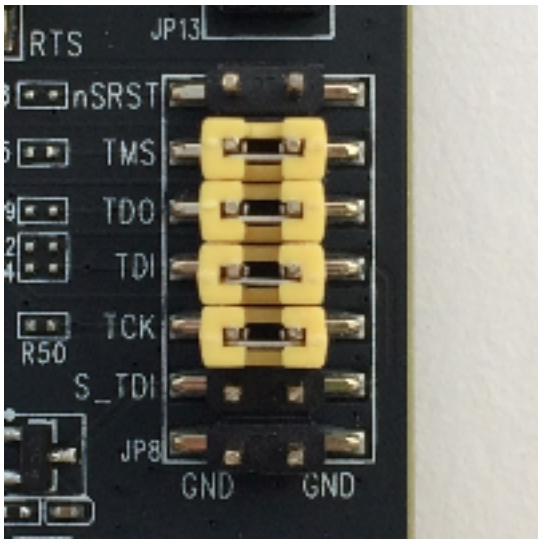
ESP-WROVER-KIT Configurazione JTAG

Per il cavo FTDI C232HM-DDHSL-0, queste sono le connessioni al -WROVER-KIT. ESP32

C232HM-DDHSL-0 Wire Color	ESP32 Pin GPIO	JTAG Signal Name
Brown (pin 5)	IO14	TMS
Yellow (pin 3)	IO12	TDI
Orange (pin 2)	IO13	TCK
Verde (pin 4)	IO15	TDO

Queste tabelle sono state sviluppate dalla scheda tecnica FTDI C232HM-DDHSL-0. Per ulteriori informazioni, vedere la sezione «Connessione via cavo MPSSE C232HM e dettagli meccanici» nella scheda tecnica.

Per abilitare JTAG su ESP-WROVER-KIT, posizionate i ponticelli sui pin TMS, TDO, TDI, TCK e S_TDI come mostrato qui.



Debug su Windows (ESP-IDF v4.2)

Per configurare il debug su Windows

1. Collegare il lato USB di FTDI C232HM-DDHSL-0 sul tuo computer e sull'altro lato come descritto in Codice di debug su Espressif ESP32 - DevKit C e (ESP-IDF v4.2) ESP-WROVER-KIT. Il

dispositivo FTDI C232HM-DDHSL-0 deve essere visualizzato in Device Manager (Gestione dispositivi) in Universal Serial Bus Controllers (Controller bus seriale universale).

2. Nell'elenco dei dispositivi USB, fare clic con il tasto destro sul dispositivo C232HM-DDHSL-0 e selezionare Properties (Proprietà).

   > **Note**
   >
   > Il dispositivo potrebbe essere elencato come USB Serial Port (Porta seriale USB).

   Nella finestra proprietà, scegliere la scheda Details (Dettagli) per visualizzare le proprietà del dispositivo. Se il dispositivo non è presente nell'elenco, installate il driver Windows per FTDI C232HM-DDHSL-0.

3. Nella scheda Dettagli, scegliete Proprietà, quindi scegliete Hardware. IDs Dovresti vedere qualcosa di simile nel campo Valore.

   ```
   FTDIBUS\COMPORT&VID_0403&PID_6014
   ```

   In questo esempio, l'ID fornitore è 0403 e l'ID prodotto è 6014.

   Verifica che IDs corrispondano al IDs pin`projects/espressif/esp32/make/aws_demos/esp32_devkitj_v1.cfg`. IDs Sono specificati in una riga che inizia con `ftdi_vid_pid` seguita da un ID fornitore e da un ID prodotto.

   ```
   ftdi_vid_pid 0x0403 0x6014
   ```

4. Scaricare OpenOCD per Windows.
5. Decomprimere il file su `C:\` e aggiungere `C:\openocd-esp32\bin` al percorso di sistema.
6. OpenOCD richiede libusb, che non è installato per impostazione predefinita in Windows.

   Per installare libusb

   a. Scaricare zadig.exe.
   b. Esegui `zadig.exe`. Nel menu, Options (Opzioni) scegliere List All Devices (Elenca tutti i dispositivi).
   c. Dal menu a discesa del metodo, scegliere C232HM-DDHSL-0.

d. Nel campo driver di destinazione, a destra della freccia verde, selezionare WinUSB (WinUSB).

e. Dall'elenco a discesa sotto il campo del driver di destinazione, scegli la freccia, quindi scegli Installa driver. Scegliere Replace Driver (Sostituisci driver).

7. Apri un prompt dei comandi, vai alla radice della directory di download di FreerTOS ed esegui il seguente comando.

```
idf.py openocd
```

Lasciare questo prompt dei comandi aperto.

8. Apri un nuovo prompt dei comandi, vai alla radice della directory di download di FreerTOS ed esegui

```
idf.py flash monitor
```

9. Apri un altro prompt dei comandi, vai alla directory principale della cartella di download di FreerTOS e attendi che la demo inizi a funzionare sulla tua scheda. Quando lo fa, esegui

```
idf.py gdb
```

Il programma deve arrestarsi nella funzione `main`.

> **Note**
>
> ESP32 Supporta un massimo di due punti di interruzione.

Debug su macOS (ESP-IDF v4.2)

1. Scaricare il driver FTDI per macOS.
2. Scaricare OpenOCD.
3. Estrarre il file .tar scaricato e impostare il percorso in `.bash_profile` su *`OCD_INSTALL_DIR`*`/openocd-esp32/bin`.
4. Usa il seguente comando per l'installazione `libusb` su macOS.

```
brew install libusb
```

5. Usa il seguente comando per scaricare il driver della porta seriale.

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

6. Se utilizzi una versione macOS successiva alla 10.9, usa il seguente comando per scaricare il driver FTDI Apple.

```
sudo kextunload -b com.apple.driver.AppleUSBFTDI
```

7. Utilizzare il comando seguente per ottenere l'ID prodotto e l'ID fornitore del cavo FTDI. Elenca i dispositivi USB collegati.

```
system_profiler SPUSBDataType
```

   L'output di `system_profiler` dovrebbe essere simile al seguente.

```
DEVICE:

Product ID: product-ID
Vendor ID: vendor-ID (Future Technology Devices International Limited)
```

8. Apri il file `projects/espressif/esp32/make/aws_demos/esp32_devkitj_v1.cfg`. L'ID fornitore e l'ID prodotto sono specificati in una riga che inizia con `ftdi_vid_pid`. Modifica il valore IDs in modo che IDs corrisponda a quello dell'`system_profiler`output del passaggio precedente.
9. Apri una finestra di terminale, vai alla radice della tua directory di download di FreerTOS e usa il seguente comando per eseguire OpenOCD.

```
idf.py openocd
```

   Lascia aperta questa finestra del terminale.

10. Apri un nuovo terminale e usa il seguente comando per caricare il driver della porta seriale FTDI.

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

11. Vai alla radice della tua directory di download di FreerTOS ed esegui

```
idf.py flash monitor
```

12. Apri un altro nuovo terminale, vai alla radice della cartella di download di FreerTOS ed esegui

```
idf.py gdb
```

Il programma deve interrompersi su `main`.

Debug su Linux (ESP-IDF v4.2)

1. Scaricare OpenOCD. Estrarre il tarball e seguire le istruzioni per l'installazione del file readme.
2. Usa il seguente comando per installare libusb su Linux.

```
sudo apt-get install libusb-1.0
```

3. Aprire un terminale e immettere **`ls -l /dev/ttyUSB*`** per elencare tutti i dispositivi USB collegati al computer. Questo ti aiuta a verificare se le porte USB della scheda sono riconosciute dal sistema operativo. L'output restituito dovrebbe essere simile al seguente.

```
$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. Uscire ed effettuare di nuovo l'accesso e spegnere e accendere la scheda per rendere le modifiche effettive. In un prompt del terminale, elencare i dispositivi USB. Assicurati che il proprietario del gruppo sia passato da `dialout` a `plugdev`.

```
$ls -l /dev/ttyUSB*
crw-rw----    1    root    plugdev    188,    0    Jul    10    19:04    /dev/
ttyUSB0
crw-rw----    1    root    plugdev    188,    1    Jul    10    19:04    /dev/
ttyUSB1
```

L'interfaccia `/dev/ttyUSBn` con il numero inferiore viene utilizzato per le comunicazioni JTAG. L'altra interfaccia viene indirizzata alla ESP32 porta seriale (UART) e viene utilizzata per caricare il codice nella memoria flash ESP32 del computer.

5. In una finestra di terminale, vai alla radice della tua directory di download di FreerTOS e usa il seguente comando per eseguire OpenOCD.

```
idf.py openocd
```

6. Apri un altro terminale, vai alla radice della directory di download di FreerTOS ed esegui il seguente comando.

```
idf.py flash monitor
```

7. Apri un altro terminale, naviga nella directory principale della cartella di download di FreerTOS ed esegui il seguente comando:

```
idf.py gdb
```

   Il programma deve interrompersi su `main()`.

Guida introduttiva all' ESP32Espressif -WROOM-32SE

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Segui questo tutorial per iniziare con ESP32 Espressif -WROOM-32SE. Per acquistarne uno dal nostro partner nel catalogo Partner Device, vedi -WROOM-32SE. AWS ESP32

> **Note**
>
> Attualmente, la porta FreerTOS ESP32 per -WROOM-32SE non supporta la funzionalità di multiprocessing simmetrico (SMP).

Panoramica

Questo tutorial descrive le seguenti procedure:

1. Connect la scheda a una macchina host
2. Installa il software sulla tua macchina host per sviluppare ed eseguire il debug di applicazioni integrate per la tua scheda microcontrollore
3. Compilazione incrociata di un'applicazione demo FreerTOS su un'immagine binaria
4. Carica l'immagine binaria dell'applicazione sulla tua scheda, quindi esegui l'applicazione
5. Monitora ed esegui il debug dell'applicazione in esecuzione utilizzando una connessione seriale

Prerequisiti

Prima di iniziare a usare FreerTOS sulla tua scheda Espressif, devi configurare il tuo account e le autorizzazioni. AWS

Iscriviti per un Account AWS

Se non ne hai uno Account AWS, completa i seguenti passaggi per crearne uno.

Per iscriverti a un Account AWS

1. Apri la https://portal.aws.amazon.com/billing/registrazione.
2. Segui le istruzioni online.

   Parte della procedura di registrazione prevede la ricezione di una telefonata o di un messaggio di testo e l'immissione di un codice di verifica sulla tastiera del telefono.

   Quando ti iscrivi a un Account AWS, Utente root dell'account AWSviene creato un. L'utente root dispone dell'accesso a tutte le risorse e tutti i Servizi AWS nell'account. Come best practice di sicurezza, assegna l'accesso amministrativo a un utente e utilizza solo l'utente root per eseguire attività che richiedono l'accesso di un utente root.

AWS ti invia un'email di conferma dopo il completamento della procedura di registrazione. In qualsiasi momento, puoi visualizzare l'attività corrente del tuo account e gestirlo accedendo a https://aws.amazon.com/e scegliendo Il mio account.

Crea un utente con accesso amministrativo

Dopo esserti registrato Account AWS, proteggi Utente root dell'account AWS AWS IAM Identity Center, abilita e crea un utente amministrativo in modo da non utilizzare l'utente root per le attività quotidiane.

Proteggi i tuoi Utente root dell'account AWS

1. Accedi AWS Management Consolecome proprietario dell'account scegliendo Utente root e inserendo il tuo indirizzo Account AWS email. Nella pagina successiva, inserisci la password.

   Per informazioni sull'accesso utilizzando un utente root, consulta la pagina Signing in as the root user della Guida per l'utente di Accedi ad AWS .

2. Abilita l'autenticazione a più fattori (MFA) per l'utente root.

   Per istruzioni, consulta Abilitare un dispositivo MFA virtuale per l'utente Account AWS root (console) nella Guida per l'utente IAM.

Crea un utente con accesso amministrativo

1. Abilita Centro identità IAM.

   Per istruzioni, consulta Abilitazione di AWS IAM Identity Center nella Guida per l'utente di AWS IAM Identity Center .

2. In IAM Identity Center, assegna l'accesso amministrativo a un utente.

   Per un tutorial sull'utilizzo di IAM Identity Center directory come fonte di identità, consulta Configurare l'accesso utente con l'impostazione predefinita IAM Identity Center directory nella Guida per l'AWS IAM Identity Center utente.

Accesso come utente amministratore

- Per accedere con l'utente IAM Identity Center, utilizza l'URL di accesso che è stato inviato al tuo indirizzo e-mail quando hai creato l'utente IAM Identity Center.

  Per informazioni sull'accesso utilizzando un utente IAM Identity Center, consulta AWS Accedere al portale di accesso nella Guida per l'Accedi ad AWS utente.

Assegna l'accesso a ulteriori utenti

1. In IAM Identity Center, crea un set di autorizzazioni conforme alla best practice dell'applicazione di autorizzazioni con il privilegio minimo.

Segui le istruzioni riportate nella pagina Creazione di un set di autorizzazioni nella Guida per l'utente di AWS IAM Identity Center .

2. Assegna al gruppo prima gli utenti e poi l'accesso con autenticazione unica (Single Sign-On).

   Per istruzioni, consulta Aggiungere gruppi nella Guida per l'utente di AWS IAM Identity Center .

Per fornire l'accesso, aggiungi autorizzazioni agli utenti, gruppi o ruoli:

- Utenti e gruppi in AWS IAM Identity Center:

  Crea un set di autorizzazioni. Segui le istruzioni riportate nella pagina Create a permission set (Creazione di un set di autorizzazioni) nella Guida per l'utente di AWS IAM Identity Center .

- Utenti gestiti in IAM tramite un provider di identità:

  Crea un ruolo per la federazione delle identità. Segui le istruzioni riportate nella pagina Create a role for a third-party identity provider (federation) della Guida per l'utente IAM.

- Utenti IAM:
  - Crea un ruolo che l'utente possa assumere. Segui le istruzioni riportate nella pagina Create a role for an IAM user della Guida per l'utente IAM.
  - (Non consigliato) Collega una policy direttamente a un utente o aggiungi un utente a un gruppo di utenti. Segui le istruzioni riportate nella pagina Aggiunta di autorizzazioni a un utente (console) nella Guida per l'utente IAM.

> **Note**
>
> La versione FreerTOS 202012.00 supporta ESP-IDF v3.3. Se stai usando una versione successiva di FreerTOS, segui le istruzioni specifiche al punto .2 at. IDFv4 Inizia con ESP-IDF v4.2

Inizia con ESP-IDF v3.3

Questa sezione mostra come utilizzare ESP-IDF v3.3 sull'hardware Espressif. Per utilizzare ESP-IDF v4.2, vedere. Inizia a usare ESP-IDF v4.2

Configurazione dell'hardware di Espressif

Consulta la Guida introduttiva ESP32 - DevKit C per informazioni sulla configurazione dell'hardware della scheda di sviluppo ESP32 -WROOM-32SE.

> **Important**
>
> Quando raggiungi la sezione Get Started della guida Espressif, fermati e segui i passaggi seguenti.

Configurazione dell'ambiente di sviluppo

Per comunicare con la tua scheda, devi scaricare e installare una toolchain.

Configurazione della toolchain

> **Note**
>
> ESP-IDF v3.3 (la versione utilizzata da FreerTOS) non supporta l'ultima versione del compilatore. ESP32 È necessario utilizzare il compilatore compatibile con la versione 3.3 di ESP-IDF. Consulta i collegamenti precedenti. Per controllare la versione del compilatore, esegui il seguente comando.
>
> ```
> xtensa-esp32-elf-gcc --version
> ```

Per configurare la toolchain, segui le istruzioni per il sistema operativo del computer host:

- Configurazione standard di Toolchain (ESP-IDF v3.3) per Windows
- Configurazione standard di Toolchain (ESP-IDF v3.3) per macOS
- Configurazione standard di Toolchain (ESP-IDF v3.3) per Linux

> **Important**
>
> Quando arrivi alle istruzioni "Ottieni ESP-IDF" in Next Steps (Fasi successive), fermati e torna alle istruzioni riportate in questa pagina.

Se avete seguito le istruzioni «Scarica ESP-IDF» riportate nei passaggi successivi, proprio ora o in un'occasione precedente, è necessario cancellare la variabile di `IDF_PATH` ambiente dal sistema prima di continuare. Questa variabile di ambiente è stata impostata automaticamente se hai seguito le istruzioni «Get ESP-IDF».

Installa CMake

Il sistema di CMake build è necessario per creare la demo e le applicazioni di test di FreerTOS per questo dispositivo. FreerTOS supporta le versioni 3.13 e successive.

È possibile scaricare l'ultima versione di CMake da.org. CMake Sono disponibili il codice sorgente e la distribuzione binaria.

Per maggiori dettagli sull'utilizzo CMake con FreerTOS, consulta. Utilizzo CMake con FreerTOS

Stabilire una connessione seriale

1. Per stabilire una connessione seriale tra la macchina host e ESP32 -WROOM-32SE, installa i driver VCP da USB a UART Bridge CP21 0x. È possibile scaricare i driver da Silicon Labs.
2. Segui i passaggi per stabilire una connessione seriale con. ESP32
3. Dopo aver stabilito una connessione seriale, annotare la porta seriale per la connessione della scheda. Tale informazione sarà necessaria durante il processo di creazione della demo.

Scarica e configura FreerTOS

Dopo aver configurato il tuo ambiente, puoi scaricare FreerTOS da. GitHub Per istruzioni, consultate il file README.md sul sito Web. GitHub

Important

Il dispositivo ATECC6 08A dispone di un'inizializzazione unica che viene bloccata sul dispositivo la prima volta che viene eseguito un progetto (durante la chiamata a). `C_InitToken` Tuttavia, il progetto demo FreerTOS e il progetto di test hanno configurazioni diverse. Se il dispositivo è bloccato durante le configurazioni del progetto demo, non tutti i test nel progetto di test avranno esito positivo.

1. Configura il FreerTOS Demo Project seguendo i passaggi seguenti. Configurazione delle demo di FreerTOS Ignorare l'ultimo passaggio per formattare le credenziali AWS IoT e seguire invece i passaggi riportati di seguito.
2. Microchip ha fornito diversi strumenti di scripting per facilitare la configurazione delle parti 08A. ATECC6 Accedere alla *freertos*`/vendors/microchip/secure_elements/app/example_trust_chain_tool` directory, quindi aprire il file. `README.md`
3. Seguire le istruzioni contenute nel file `README.md` per effettuare il provisioning del dispositivo. Queste fasi includono quanto segue:

   a. Crea e registra un'autorità di certificazione con AWS

   b. Genera le tue chiavi sullo ATECC6 08A ed esporta la chiave pubblica e il numero di serie del dispositivo

   c. Genera un certificato per il dispositivo e registralo con AWS
4. Caricare il certificato CA e il certificato del dispositivo sul dispositivo seguendo le istruzioni in Distribuzione delle chiavi in modalità sviluppatore.

Crea, esegui il flashing ed esegui il progetto demo FreerTOS

Puoi usare CMake per generare i file di build, Make per creare il file binario dell'applicazione e l'utilità IDF di Espressif per eseguire il flashing della scheda.

Crea FreerTOS su Linux o macOS (ESP-IDF v3.3)

Se usi Windows, puoi passare a Crea FreerTOS su Windows (ESP-IDF v3.3).

CMake Usalo per generare i file di build, quindi usa Make per creare l'applicazione.

Per generare i file di build dell'applicazione demo con CMake

1. Vai alla directory principale della tua cartella di download di FreerTOS.
2. In una finestra della riga di comando, immettere il seguente comando per generare i file di compilazione.

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

> **Note**
>
> Per creare l'applicazione per il debug, aggiungere il flag `-DCMAKE_BUILD_TYPE=Debug`.
> Per generare i file di compilazione dell'applicazione di test, aggiungere il flag `-DAFR_ENABLE_TESTS=1`.
> Il codice fornito da Espressif utilizza lo stack IP leggero (lwIP) come stack di rete predefinito. Per utilizzare invece lo stack di rete FreeRTOS+TCP, aggiungi il flag al comando. `–DAFR_ESP_FREERTOS_TCP` CMake
> Per aggiungere la dipendenza LWip per il codice fornito non dal fornitore, aggiungi le seguenti righe al file `CMakeLists.txt` delle dipendenze, per CMake il tuo componente Wi-Fi personalizzato.
>
> ```
> # Add a dependency on the bluetooth espressif component to the common
>  component
> set(COMPONENT_REQUIRES lwip)
> ```

Per compilare l'applicazione con Make

1. Passa alla directory `build`.
2. In una finestra della riga di comando, immettere il seguente comando per compilare l'applicazione con Make.

```
make all -j4
```

> **Note**
>
> È necessario generare i file compilati con il comando cmake ogni volta che si passa dal progetto `aws_demos` al progetto `aws_tests`.

Crea FreerTOS su Windows (ESP-IDF v3.3)

Su Windows, è necessario specificare un generatore di build per. CMake Altrimenti, il CMake valore predefinito è Visual Studio. Espressif consiglia ufficialmente il sistema di compilazione Ninja perché

funziona in Windows, Linux e MacOS. È necessario eseguire CMake i comandi in un ambiente Windows nativo come cmd o. PowerShell L'esecuzione di CMake comandi in un ambiente Linux virtuale, ad esempio MSYS2 o WSL, non è supportata.

CMake Utilizzatelo per generare i file di compilazione, quindi utilizzate Make per creare l'applicazione.

Per generare i file di build dell'applicazione demo con CMake

1. Vai alla directory principale della tua cartella di download di FreerTOS.
2. In una finestra della riga di comando, immettere il seguente comando per generare i file di compilazione.

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

> **Note**
>
> Per creare l'applicazione per il debug, aggiungere il flag `-DCMAKE_BUILD_TYPE=Debug`.
> Per generare i file di compilazione dell'applicazione di test, aggiungere il flag `-DAFR_ENABLE_TESTS=1`.
> Il codice fornito da Espressif utilizza lo stack IP leggero (lwIP) come stack di rete predefinito. Per utilizzare invece lo stack di rete FreeRTOS+TCP, aggiungi il flag al comando. `–DAFR_ESP_FREERTOS_TCP` CMake
> Per aggiungere la dipendenza LWip per il codice fornito non dal fornitore, aggiungi le seguenti righe al file `CMakeLists.txt` delle dipendenze, per CMake il tuo componente Wi-Fi personalizzato.
>
> ```
> # Add a dependency on the bluetooth espressif component to the common
>  component
> set(COMPONENT_REQUIRES lwip)
> ```

Per compilare l'applicazione

1. Passa alla directory `build`.
2. In una finestra della riga di comando, immettere il seguente comando per richiamare Ninja per compilare l'applicazione.

```
ninja
```

Oppure, usa l'interfaccia generica CMake per creare l'applicazione.

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

> **Note**
>
> È necessario generare i file compilati con il comando cmake ogni volta che si passa dal progetto `aws_demos` al progetto `aws_tests`.

Esegui il flashing ed esegui FreerTOS (ESP-IDF v3.3)

Utilizza la utilità IDF di Espressif (*`freertos`*`/vendors/espressif/esp-idf/tools/idf.py`) per eseguire il flashing della scheda, eseguire l'applicazione e visualizzare i log.

Per cancellare il flashing della scheda, vai alla directory *`freertos`* e inserisci il seguente comando.

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

Per eseguire il flashing del file binario dell'applicazione sulla scheda, utilizza `make`.

```
make flash
```

È inoltre possibile utilizzare lo script IDF per eseguire il flashing della scheda.

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

Utilizzate il seguente comando per visualizzare i log:

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

> **Tip**
>
> È inoltre possibile combinare questi comandi.

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

Inizia a usare ESP-IDF v4.2

Questa sezione mostra come utilizzare ESP-IDF v4.2 sull'hardware Espressif. Per utilizzare ESP-IDF v3.3, vedere. Inizia con ESP-IDF v3.3

> **Note**
>
> I comandi Linux in questo tutorial richiedono l'uso della shell Bash.

Configurazione dell'hardware di Espressif

Per informazioni sulla configurazione dell'hardware della scheda di sviluppo ESP32 -WROOM-32SE, consultate la ESP32- DevKit C V4 Getting Started Guide.

> **Important**
>
> Quando raggiungi la sezione Get Started delle guide Espressif, fermati e segui i passaggi seguenti.

Configurazione dell'ambiente di sviluppo

Per comunicare con la scheda, è necessario scaricare e installare una toolchain.

Configurazione della toolchain

Per configurare la toolchain, segui le istruzioni per il sistema operativo del computer host:

- Configurazione standard di Toolchain e prerequisiti (ESP-IDF v4.2) per Windows
- Configurazione standard dei prerequisiti della Toolchain (ESP-IDF v4.2) per macOS
- Configurazione standard dei prerequisiti della Toolchain (ESP-IDF v4.2) per Linux

> **Important**
>
> Quando raggiungi le istruzioni «Scarica ESP-IDF» riportate nella sezione Passaggi successivi, interrompi e torna alle istruzioni riportate in questa pagina.

Completa l'installazione (Linux/ Mac)

Il programma di installazione di ESP-IDF per Windows installa tutti gli strumenti necessari. Le piattaforme Linux e MAC OSX richiedono un passaggio aggiuntivo per completare l'installazione.

Apri una finestra della riga di comando, vai alla directory di download di FreerTOS, quindi esegui lo script seguente per scaricare e installare la toolchain espressif per la tua piattaforma.

```
vendors/espressif/esp-idf/install.sh
```

Successivamente, aggiungi gli strumenti della toolchain ESP-IDF al percorso del tuo terminale con il seguente comando.

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

Stabilire una connessione seriale

1. Per stabilire una connessione seriale tra la macchina host e ESP32 -WROOM-32SE, installa i driver VCP da USB a UART Bridge 0x. CP21 È possibile scaricare i driver da Silicon Labs.
2. Segui i passaggi per stabilire una connessione seriale con. ESP32
3. Dopo aver stabilito una connessione seriale, annotare la porta seriale per la connessione della scheda. Tale informazione sarà necessaria durante il processo di creazione della demo.

Scarica e configura FreerTOS

Dopo aver configurato il tuo ambiente, puoi scaricare FreerTOS da. GitHub Per istruzioni, consultate il file README.md sul sito Web. GitHub

> **Important**
>
> Il dispositivo ATECC6 08A dispone di un'inizializzazione unica che viene bloccata sul dispositivo la prima volta che viene eseguito un progetto (durante la chiamata a).

`C_InitToken` Tuttavia, il progetto demo FreerTOS e il progetto di test hanno configurazioni diverse. Se il dispositivo è bloccato durante le configurazioni del progetto demo, non tutti i test nel progetto di test avranno esito positivo.

1. Configura il FreerTOS Demo Project seguendo i passaggi seguenti. Configurazione delle demo di FreerTOS Quando arrivi all'ultimo passaggio per formattare AWS IoT le tue credenziali, FERMATI e segui invece i passaggi seguenti.
2. Microchip ha fornito diversi strumenti di scripting per facilitare la configurazione dei componenti 08A. ATECC6 Passare alla directory *`freertos`*`/vendors/microchip/secure_elements/app/example_trust_chain_tool` e aprire il file `README.md`.
3. Seguire le istruzioni contenute nel file `README.md` per effettuare il provisioning del dispositivo. Queste fasi includono quanto segue:
   a. Crea e registra un'autorità di certificazione con AWS
   b. Genera le tue chiavi sullo ATECC6 08A ed esporta la chiave pubblica e il numero di serie del dispositivo
   c. Genera un certificato per il dispositivo e registralo con AWS
4. Caricare il certificato CA e il certificato del dispositivo sul dispositivo seguendo le istruzioni in Distribuzione delle chiavi in modalità sviluppatore.

Compila, esegui il flashing ed esegui il progetto demo FreerTOS utilizzando lo script idf.py

Puoi usare l'utilità IDF di Espressif per generare i file di build, creare il binario dell'applicazione e aggiornare la tua scheda.

Crea ed esegui il flashing di FreerTOS su Windows, Linux e macOS (ESP-IDF v4.2)

Usa `idf.py` lo script per creare il progetto ed esegui il flashing dei file binari sul tuo dispositivo.

> **Note**
>
> Alcune configurazioni potrebbero richiedere l'utilizzo dell'opzione port "`-p` *`port-name`*" with `idf.py` per specificare la porta corretta, come nell'esempio seguente.
>
> ```
> idf.py -p /dev/cu.usbserial-00101301B flash
> ```

Per creare e aggiornare il progetto

1. Vai alla directory principale della tua cartella di download di FreerTOS.
2. In una finestra a riga di comando, inserisci il seguente comando per aggiungere gli strumenti ESP-IDF al PATH del tuo terminale:

   Windows

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

   Linux /macOS

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

3. Configura cmake nella `build` directory e crea l'immagine del firmware con il seguente comando.

   ```
   idf.py -DVENDOR=espressif -DBOARD=esp32_ecc608a_devkitc -DCOMPILER=xtensa-esp32
    build
   ```

   Dovresti vedere un output come questo nell'esempio seguente.

   ```
   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 non ci sono errori, la build genererà i file.bin binari del firmware.

4. Cancellate la memoria flash della scheda di sviluppo con il seguente comando.

```
idf.py erase_flash
```

5. Usa `idf.py` lo script per eseguire il flashing del file binario dell'applicazione sulla tua scheda.

```
idf.py flash
```

6. Monitora l'uscita dalla porta seriale della scheda con il seguente comando.

```
idf.py monitor
```

> **Note**
>
> È possibile combinare questi comandi come nell'esempio seguente.
>
> ```
> idf.py erase_flash flash monitor
> ```

> **Note**
>
> Per alcune configurazioni della macchina host, è necessario specificare la porta quando si esegue il flashing della scheda, come nell'esempio seguente.
>
> ```
> idf.py erase_flash flash monitor -p /dev/ttyUSB1
> ```

Crea ed esegui il flashing di FreerTOS con CMake

Oltre a utilizzare lo script idf.py fornito dall'IDF SDK per creare ed eseguire il codice, puoi anche creare il progetto con. CMake Attualmente supporta Unix Makefile e il sistema di build Ninja.

Per creare e aggiornare il progetto

1. In una finestra a riga di comando, vai alla radice della tua directory di download di FreerTOS.
2. Esegui lo script seguente per aggiungere gli strumenti ESP-IDF al PATH della tua shell.

Windows

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

Linux /macOS

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

3. Immettete il seguente comando per generare i file di build.

Con Unix Makefiles

```
cmake -DVENDOR=espressif -DBOARD=esp32_plus_ecc608a_devkitc -DCOMPILER=xtensa-esp32
 -S . -B ./YOUR_BUILD_DIRECTORY -DAFR_ENABLE_ALL_MODULES=1 -DAFR_ENABLE_TESTS=0
```

Con Ninja

```
cmake -DVENDOR=espressif -DBOARD=esp32_plus_ecc608a_devkitc -DCOMPILER=xtensa-esp32
 -S . -B ./YOUR_BUILD_DIRECTORY -DAFR_ENABLE_ALL_MODULES=1 -DAFR_ENABLE_TESTS=0 -
GNinja
```

4. Cancella il flash e poi fai lampeggiare la lavagna.

Con Unix Makefiles

```
make -C ./YOUR_BUILD_DIRECTORY erase_flash
```

```
make -C ./YOUR_BUILD_DIRECTORY flash
```

Con Ninja

```
ninja -C ./YOUR_BUILD_DIRECTORY erase_flash
```

```
ninja -C ./YOUR_BUILD_DIRECTORY flash
```

Monitoraggio dei messaggi MQTT sul cloud AWS

Puoi utilizzare il client MQTT nella AWS IoT console per monitorare i messaggi che il tuo dispositivo invia al AWS Cloud.

Per abbonarsi all'argomento MQTT con il client MQTT AWS IoT

1. Accedi alla AWS IoT console .
2. Nel riquadro di navigazione scegliere Test per aprire il client MQTT.
3. In Argomento sottoscrizione, digitare ***your-thing-name*/example/topic**, quindi scegliere Effettua sottoscrizione all'argomento.

Informazioni aggiuntive

Per ulteriori informazioni sull'utilizzo e la risoluzione dei problemi delle ESP32 schede Espressif, consultate i seguenti argomenti:

- Usare FreerTOS nel proprio progetto per CMake ESP32
- Risoluzione dei problemi
- Debug

Guida introduttiva al kit di connettività IoT Infineon XMC48 00

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Questo tutorial fornisce istruzioni per iniziare a usare il kit di connettività IoT Infineon XMC48 00. Se non disponi del kit di connettività IoT Infineon XMC48 00, visita il AWS Partner Device Catalog per acquistarne uno dal nostro partner.

Se desideri aprire una connessione seriale con la scheda per visualizzare le informazioni di registrazione e debug, hai bisogno di un USB/Serial converter, in addition to the XMC4800 IoT

Connectivity Kit. The CP2104 is a common USB/Serial convertitore da 3,3 V ampiamente disponibile in schede come 04 Friend di Adafruit. CP21

Prima di iniziare, devi configurare AWS IoT e scaricare FreerTOS per connettere il tuo dispositivo al Cloud. AWS Per istruzioni, consulta Fase iniziale. In questo tutorial, il percorso della directory di download di FreerTOS è indicato come. *`freertos`*

Panoramica

Questo tutorial contiene le istruzioni per i seguenti passaggi iniziali:

1. Installazione di software sul computer host per lo sviluppo e il debug di applicazioni integrate per la scheda a microcontroller.
2. Compilazione incrociata di un'applicazione demo FreerTOS con un'immagine binaria.
3. Caricamento dell'immagine binaria dell'applicazione sulla scheda in uso e successiva esecuzione dell'applicazione.
4. Interazione con l'applicazione in esecuzione sulla scheda attraverso una connessione seriale, per scopi di monitoraggio e debug.

Configurazione dell'ambiente di sviluppo

FreerTOS utilizza l'ambiente di sviluppo DAVE di Infineon per programmare 00. XMC48 Prima di iniziare, devi scaricare e installare DAVE e alcuni driver J-Link per comunicare con il debugger della scheda.

Installa DAVE

1. Andare alla pagina di download del software Infineon DAVE.
2. Scegliere il pacchetto DAVE per il proprio sistema operativo e inviare le informazioni di registrazione. Dopo la registrazione con Infineon, verrà inviata un'e-mail di conferma con un link per scaricare un file .zip.
3. Scaricare il file .zip del pacchetto DAVE (`DAVE_`*`version_os_date`*`.zip`) e decomprimerlo sul percorso in cui si desidera installare DAVE (ad esempio, `C:\DAVE4`).

> **Note**
>
> Alcuni utenti di Windows hanno riportato problemi con Windows Explorer per decomprimere il file. È consigliabile utilizzare un programma di terze parti, ad esempio 7-Zip.

4. Per avviare DAVE, eseguire il file eseguibile che si trova nella cartella decompressa `DAVE_version_os_date.zip`.

Per ulteriori informazioni, consulta la Guida rapida su DAVE.

Installa i driver Segger J-Link

Per comunicare con la sonda di debug integrata nella scheda XMC48 00 Relax EtherCAT, sono necessari i driver inclusi nel pacchetto software e documentazione J-Link. È possibile scaricare il pacchetto J-Link Software and Documentation dalla pagina di download del software J-Link di Segger.

Stabilire una connessione seriale

Stabilire una connessione seriale è facoltativo, ma consigliato. Una connessione seriale consente alla scheda di inviare le informazioni di accesso e di debug in un modulo che è possibile visualizzare sul computer di sviluppo.

L'applicazione demo XMC48 00 utilizza una connessione seriale UART sui pin P0.0 e P0.1, che sono etichettati sulla serigrafia della scheda 00 XMC48 Relax EtherCAT. Per stabilire una connessione seriale:

1. Collegare il connettore etichettato come "RX< P0.0" al connettore "TX" del convertitore USB/seriale.
2. Collegare il connettore etichettato come "TX>P0.1" al connettore "RX" del convertitore USB/seriale.
3. Collegare il perno di messa a terra del convertitore seriale a uno dei connettori etichettati come "GND" sulla scheda. I dispositivi devono condividere un terreno comune.

L'alimentazione è fornita dalla porta di debug USB, perciò non collegare il connettore di tensione positiva dell'adattatore seriale alla scheda.

> **Note**
>
> Alcuni cavi seriali utilizzano un livello di segnale da 5V. La scheda XMC48 00 e il modulo Wi-Fi Click richiedono una tensione di 3,3 V. Non utilizzare il cavo di collegamento IOREF della scheda per modificare i segnali della scheda su 5V.

Con il cavo collegato, è possibile aprire una connessione seriale su un emulatore di terminale, ad esempio GNU Screen. Il baud rate è impostato su 115200 per impostazione predefinita con 8 bit di dati, nessuna parità, 1 bit di arresto.

Crea ed esegui il progetto demo FreerTOS

Importa la demo di FreerTOS in DAVE

1. Avvia DAVE.
2. In DAVE, selezionare File, Import (Importa). Nella finestra Import (Importa), espandere la cartella Infineon, scegliere DAVE Project (Progetto DAVE) quindi scegliere Next (Successivo).
3. Nella finestra Importa progetti DAVE, scegli Seleziona la directory principale, scegli Sfoglia, quindi scegli il XMC48 progetto demo 00.

   Nella directory in cui hai decompresso il download di FreerTOS, si trova il progetto demo. `projects/infineon/xmc4800_iotkit/dave4/aws_demos`

   Assicurarsi che l'opzione Copy Projects Into Workspace (Copia progetti nel workspace) sia deselezionata.
4. Scegli Fine.

   Il progetto `aws_demos` deve essere importato nel workspace e attivato.
5. Dal menu Project (Progetto), scegliere Build Active Project (Crea progetto attivo).

   Assicurarsi che il progetto venga creato senza errori.

Esegui il progetto demo FreerTOS

1. Usa un cavo USB per collegare il tuo kit di connettività IoT XMC48 00 al computer. La scheda è dotata di due connettori microUSB. Utilizza quello etichettato come "X101", dove Debug vi è visualizzato accanto nella serigrafia della scheda.

2. Dal menu Project (Progetto) scegliere Rebuild Active Project (Ricrea progetto attivo) per ricreare `aws_demos` e accertarsi che le modifiche di configurazione siano selezionate.
3. Da Project Explorer, fare clic con il pulsante destro del mouse su `aws_demos`, scegliere Debug As (Debug come), e quindi scegliere DAVE C/C++ Application.
4. Fare doppio clic su GDB SEGGER J-Link Debugging (Debug di GDB SEGGER J-) per creare una conferma del debug. Scegliere Debug.
5. Quando il debugger si arresta sul punto di interruzione in `main()`, dal menu Run (Esegui), scegliere Resume (Riprendi).

Nella AWS IoT console, il client MQTT dei passaggi 4-5 dovrebbe visualizzare i messaggi MQTT inviati dal dispositivo. Se si utilizza la connessione seriale, l'output UART dovrebbe essere simile a questo:

```
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----
```

Crea la demo di FreerTOS con CMake

Se preferisci non utilizzare un IDE per lo sviluppo di FreerTOS, puoi in alternativa CMake utilizzarlo per creare ed eseguire le applicazioni demo o le applicazioni che hai sviluppato utilizzando editor di codice e strumenti di debug di terze parti.

> **Note**
>
> Questa sezione descrive l'utilizzo CMake su Windows con MingW come sistema di build nativo. Per ulteriori informazioni sull'utilizzo CMake con altri sistemi operativi e opzioni, vedereUtilizzo CMake con FreerTOS. (MinGW è un ambiente di sviluppo minimalista per applicazioni Microsoft Windows native).

Per creare la demo di FreerTOS con CMake

1. Configurare la suite di strumenti GNU ARM Embedded Toolchain.

   a. Scaricare una versione per Windows della toolchain dalla pagina di download di ARM Embedded Toolchain.

      > **Note**
      >
      > È consigliabile scaricare una versione diversa da "8-2018-q4-major", a causa di un bug rilevato mediante la utilità "objcopy" in tale versione.

   b. Aprire il programma di installazione della toolchain scaricata e seguire la procedura guidata di installazione per installare la toolchain.

> **Important**
>
> Nella pagina finale della procedura guidata di installazione, selezionare Add path to environment variable (Aggiungi percorso a variabili di ambiente) per aggiungere il percorso della toolchain alla variabile di ambiente del percorso di sistema.

2. Installa CMake e MinGW.

   Per istruzioni, consulta CMake Prerequisiti.

3. Crea una cartella per contenere i file di build generati (*build-folder*).

4. Cambia le directory nella tua directory di download di FreerTOS *freertos* () e usa il seguente comando per generare i file di build:

```
cmake -DVENDOR=infineon -DBOARD=xmc4800_iotkit -DCOMPILER=arm-gcc -S . -B build-folder -G "MinGW Makefiles" -DAFR_ENABLE_TESTS=0
```

5. Cambia le directory nella directory build (*build-folder*) e usa il seguente comando per creare il file binario:

```
cmake --build . --parallel 8
```

   Questo comando crea il file binario di output `aws_demos.hex` nella directory di compilazione.

6. Effettuare il flashing ed eseguire l'immagine con JLINK.

   a. Dalla directory build (*build-folder*), utilizzate i seguenti comandi per creare uno script flash:

```
echo loadfile aws_demos.hex > flash.jlink
```

```
echo r >> flash.jlink
```

```
echo g >> flash.jlink
```

```
echo q >> flash.jlink
```

   b. Effettuare il flashing dell'immagine utilizzando l'eseguibile JLNIK.

```
JLINK_PATH\JLink.exe  -device XMC4800-2048 -if SWD -speed auto -CommanderScript
 flash.jlink
```

I log dell'applicazione devono essere visibili mediante la connessione seriale stabilita con la scheda.

Monitoraggio dei messaggi MQTT in cloud

È possibile utilizzare il client MQTT nella AWS IoT console per monitorare i messaggi che il dispositivo invia al AWS Cloud.

Per sottoscrivere l'argomento MQTT con il AWS IoT client MQTT

1. Accedi alla console AWS IoT.
2. Nel riquadro di navigazione scegliere Test per aprire il client MQTT.
3. In Argomento sottoscrizione, digitare ***your-thing-name*/example/topic**, quindi scegliere Effettua sottoscrizione all'argomento.

Risoluzione dei problemi

Se non l'hai già fatto, assicurati di configurare AWS IoT e scaricare FreerTOS per connettere il tuo dispositivo al Cloud. AWS Per istruzioni, consulta Fase iniziale.

Per informazioni generali sulla risoluzione dei problemi su Getting Started with FreerTOS, consulta. Nozioni di base sulla risoluzione dei problemi

Guida introduttiva al kit di connettività XMC48 IoT Infineon OPTIGA Trust X e 00

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Questo tutorial fornisce istruzioni per iniziare a utilizzare il kit di connettività Infineon OPTIGA Trust X Secure Element e XMC48 00 IoT. Rispetto al tutorial Guida introduttiva al kit di connettività IoT Infineon XMC48 00, questa guida illustra come fornire credenziali sicure utilizzando Infineon OPTIGA Trust X Secure Element.

È necessario il seguente hardware:

1. Host MCU - Infineon 00 XMC48 IoT Connectivity Kit, visita il AWS Partner Device Catalog per acquistarne uno dal nostro partner.
2. Pacchetto di estensione di sicurezza:
   - Elemento di sicurezza - Infineon OPTIGA Trust X.

     Visita il AWS Partner Device Catalog per acquistarli dal nostro partner.
   - Scheda di personalizzazione - Scheda di personalizzazione Infineon OPTIGA.
   - Scheda adattatore - Infineon MyIo T Adapter.

Per seguire le fasi necessarie, è necessario aprire una connessione seriale con la scheda per visualizzare le informazioni di registrazione e di debug. (Uno dei passaggi richiede di copiare una chiave pubblica dall'output di debug seriale della scheda e incollarla in un file.) Per fare ciò, è necessario un convertitore USB/seriale da 3,3 V oltre allo 00 XMC48 IoT Connectivity Kit. È noto che il JBtek convertitore USB/seriale EL-PN-47310126 funziona bene per questa demo. Sono inoltre necessari tre cavi male-to-male jumper (per ricezione (RX), trasmissione (TX) e terra (GND)) per collegare il cavo seriale alla scheda adattatore Infineon T. MyIo

Prima di iniziare, devi configurare AWS IoT e scaricare FreerTOS per connettere il tuo dispositivo al Cloud. AWS Per istruzioni, consulta Opzione 2: Generazione di chiavi private integrate. In questo tutorial, il percorso della directory di download di FreerTOS è indicato come. *`freertos`*

Panoramica

Questa esercitazione contiene i seguenti passaggi:

1. Installazione del software sul computer host per lo sviluppo e il debug di applicazioni integrate per la scheda con microcontrollore.
2. Compila in modo incrociato un'applicazione demo FreerTOS su un'immagine binaria.
3. Caricamento dell'immagine binaria dell'applicazione sulla scheda in uso e successiva esecuzione dell'applicazione.

4. Interazione con l'applicazione in esecuzione sulla scheda attraverso una connessione seriale, per scopi di monitoraggio e debug.

Configurazione dell'ambiente di sviluppo

FreerTOS utilizza l'ambiente di sviluppo DAVE di Infineon per programmare 00. XMC48 Prima di iniziare, devi scaricare e installare DAVE e alcuni driver J-Link per comunicare con il debugger della scheda.

Installa DAVE

1. Andare alla pagina di download del software Infineon DAVE.
2. Scegliere il pacchetto DAVE per il proprio sistema operativo e inviare le informazioni di registrazione. Dopo la registrazione con Infineon, verrà inviata un'e-mail di conferma con un link per scaricare un file .zip.
3. Scaricare il file .zip del pacchetto DAVE (`DAVE_`*`version_os_date`*`.zip`) e decomprimerlo sul percorso in cui si desidera installare DAVE (ad esempio, `C:\DAVE4`).

   > **Note**
   >
   > Alcuni utenti di Windows hanno riportato problemi con Windows Explorer per decomprimere il file. È consigliabile utilizzare un programma di terze parti, ad esempio 7-Zip.

4. Per avviare DAVE, eseguire il file eseguibile che si trova nella cartella decompressa `DAVE_`*`version_os_date`*`.zip`.

Per ulteriori informazioni, consulta la Guida rapida su DAVE.

Installa i driver Segger J-Link

Per comunicare con la sonda di debug integrata del kit XMC48 00 IoT Connectivity, sono necessari i driver inclusi nel pacchetto software e documentazione J-Link. È possibile scaricare il pacchetto J-Link Software and Documentation dalla pagina di download del software J-Link di Segger.

Stabilire una connessione seriale

Collegare il cavo del convertitore USB/seriale all'adattatore Infineon Shield2Go. Questa operazione consente alla scheda di inviare le informazioni di accesso e di debug in un modulo che è possibile visualizzare sul computer di sviluppo. Per stabilire una connessione seriale:

1. Collegare il pin RX al pin TX del convertitore USB/seriale.
2. Collegare il pin TX al pin RX del convertitore USB/seriale.
3. Collegare il pin di messa a terra del convertitore seriale a uno dei pin "GND" sulla scheda. I dispositivi devono condividere un terreno comune.

L'alimentazione è fornita dalla porta di debug USB, perciò non collegare il connettore di tensione positiva dell'adattatore seriale alla scheda.

> **Note**
>
> Alcuni cavi seriali utilizzano un livello di segnale da 5V. La scheda XMC48 00 e il modulo Wi-Fi Click richiedono una tensione di 3,3 V. Non utilizzare il cavo di collegamento IOREF della scheda per modificare i segnali della scheda su 5V.

Con il cavo collegato, è possibile aprire una connessione seriale su un emulatore di terminale, ad esempio GNU Screen. Il baud rate è impostato su 115200 per impostazione predefinita con 8 bit di dati, nessuna parità, 1 bit di arresto.

Monitoraggio dei messaggi MQTT in cloud

Puoi utilizzare il client MQTT nella AWS IoT console per monitorare i messaggi che il tuo dispositivo invia al Cloud. AWS Si consiglia di effettuare queste impostazioni ora, prima che il dispositivo esegua il progetto demo.

Per abbonarsi all'argomento MQTT con il client MQTT AWS IoT

1. Accedi alla console AWS IoT.
2. Nel riquadro di navigazione scegliere Test per aprire il client MQTT.
3. In Argomento sottoscrizione, digitare ***your-thing-name*****/example/topic**, quindi scegliere Effettua sottoscrizione all'argomento.

Crea ed esegui il progetto demo FreerTOS

Importa la demo di FreerTOS in DAVE

1. Avvia DAVE.
2. In DAVE scegliere File, quindi Import (Importa). Espandere la cartella Infineon, scegliere DAVE Project (Progetto DAVE), quindi scegliere Next (Successivo).
3. Nella finestra Importa progetti DAVE, scegli Seleziona la directory principale, scegli Sfoglia, quindi scegli il XMC48 progetto demo 00.

   Nella directory in cui hai decompresso il download di FreerTOS, si trova il progetto demo. `projects/infineon/xmc4800_plus_optiga_trust_x/dave4/aws_demos/dave4`

   Assicurarsi che l'opzione Copy Projects Into Workspace (Copia progetti nel workspace) sia deselezionata.
4. Scegli Fine.

   Il progetto `aws_demos` deve essere importato nel workspace e attivato.
5. Dal menu Project (Progetto), scegliere Build Active Project (Crea progetto attivo).

   Assicurarsi che il progetto venga creato senza errori.

Esegui il progetto demo FreerTOS

1. Dal menu Project (Progetto) scegliere Rebuild Active Project (Ricrea progetto attivo) per ricreare `aws_demos` e accertarsi che le modifiche di configurazione siano selezionate.
2. Da Project Explorer, fare clic con il pulsante destro del mouse su `aws_demos`, scegliere Debug As (Debug come), e quindi scegliere DAVE C/C++ Application.
3. Fare doppio clic su GDB SEGGER J-Link Debugging (Debug di GDB SEGGER J-) per creare una conferma del debug. Scegliere Debug.
4. Quando il debugger si arresta sul punto di interruzione in `main()`, dal menu Run (Esegui), scegliere Resume (Riprendi).

A questo punto, continuare con la fase di estrazione della chiave pubblica in Opzione 2: Generazione di chiavi private integrate. Dopo aver completato tutti i passaggi, vai alla AWS IoT console. Il client MQTT configurato in precedenza dovrebbe visualizzare i messaggi MQTT inviati dal dispositivo.

Tramite la connessione seriale del dispositivo, si dovrebbe vedere qualcosa di simile sull'output 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----
```

Crea la demo di FreerTOS con CMake

Questa sezione descrive l'utilizzo CMake su Windows con MingW come sistema di build nativo. Per ulteriori informazioni sull'utilizzo CMake con altri sistemi operativi e opzioni, vedereUtilizzo CMake con FreerTOS. (MinGW è un ambiente di sviluppo minimalista per applicazioni Microsoft Windows native).

Se preferisci non utilizzare un IDE per lo sviluppo di FreerTOS, puoi CMake utilizzarlo per creare ed eseguire le applicazioni demo o le applicazioni che hai sviluppato utilizzando editor di codice e strumenti di debug di terze parti.

Per creare la demo di FreerTOS con CMake

1. Configurare la suite di strumenti GNU ARM Embedded Toolchain.

   a. Scaricare una versione per Windows della toolchain dalla pagina di download di ARM Embedded Toolchain.

      > **Note**
      >
      > È consigliabile scaricare una versione diversa da "8-2018-q4-major", a causa di un bug rilevato mediante l'utilità "objcopy" in tale versione.

   b. Aprire il programma di installazione toolchain scaricato e seguire le istruzioni nella procedura guidata.

   c. Nella pagina finale della procedura guidata di installazione, selezionare Add path to environment variable (Aggiungi percorso a variabili di ambiente) per aggiungere il percorso della toolchain alla variabile di ambiente del percorso di sistema.

2. Installa CMake e MinGW.

   Per istruzioni, consulta CMake Prerequisiti.

3. Crea una cartella per contenere i file di build generati (*`build-folder`*).

4. Cambia le directory nella tua directory di download di FreerTOS *`freertos`* () e usa il seguente comando per generare i file di build:

   ```
   cmake -DVENDOR=infineon -DBOARD=xmc4800_plus_optiga_trust_x -DCOMPILER=arm-gcc -S .
    -B build-folder -G "MinGW Makefiles" -DAFR_ENABLE_TESTS=0
   ```

5. Cambia le directory nella directory build (*build-folder*) e usa il seguente comando per creare il file binario:

   ```
   cmake --build . --parallel 8
   ```

   Questo comando crea il file binario di output `aws_demos.hex` nella directory di compilazione.

6. Effettuare il flashing ed eseguire l'immagine con JLINK.

   a. Dalla directory build (*build-folder*), utilizzate i seguenti comandi per creare uno script flash:

      ```
      echo loadfile aws_demos.hex > flash.jlink
      echo r >> flash.jlink
      echo g >> flash.jlink
      echo q >> flash.jlink
      ```

   b. Effettuare il flashing dell'immagine utilizzando l'eseguibile JLNIK.

      ```
      JLINK_PATH\JLink.exe  -device XMC4800-2048 -if SWD -speed auto -CommanderScript
       flash.jlink
      ```

      I log dell'applicazione devono essere visibili mediante la connessione seriale stabilita con la scheda. Continuare al passaggio di estrazione della chiave pubblica in Opzione 2: Generazione di chiavi private integrate. Dopo aver completato tutti i passaggi, vai alla AWS IoT console. Il client MQTT configurato in precedenza dovrebbe visualizzare i messaggi MQTT inviati dal dispositivo.

Risoluzione dei problemi

Per informazioni generiche sulla risoluzione dei problemi, consultare Nozioni di base sulla risoluzione dei problemi.

Guida introduttiva allo MW32x AWS IoT Starter Kit

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Lo AWS IoT Starter Kit è un kit di sviluppo basato sull'88 MW32 0/88MW322, l'ultimo microcontrollore Cortex M4 integrato di NXP, che integra il Wi-Fi 802.11b/g/n su un singolo chip di microcontrollore. Il kit di sviluppo è certificato FCC. Per ulteriori informazioni, consulta il AWS Partner Device Catalog per acquistarne uno dal nostro partner. I MW322 moduli 88 MW32 0/88 sono inoltre certificati FCC e sono disponibili per la personalizzazione e la vendita su larga scala.

Questa guida introduttiva mostra come compilare in modo incrociato l'applicazione e l'SDK su un computer host, quindi caricare il file binario generato sulla scheda utilizzando gli strumenti forniti con l'SDK. Quando l'applicazione inizia a funzionare sulla scheda, è possibile eseguire il debug o interagire con essa dalla console seriale sul computer host.

Ubuntu 16.04 è la piattaforma host supportata per lo sviluppo e il debug. Potresti essere in grado di utilizzare altre piattaforme, ma queste non sono supportate ufficialmente. È necessario disporre delle autorizzazioni per installare il software sulla piattaforma host. Per creare l'SDK sono necessari i seguenti strumenti esterni:

- Piattaforma host Ubuntu 16.04
- Toolchain ARM versione 4_9_2015q3
- IDE Eclipse 4.9.0

La toolchain ARM è necessaria per la compilazione incrociata dell'applicazione e dell'SDK. L'SDK sfrutta le ultime versioni della toolchain per ottimizzare l'ingombro dell'immagine e inserire più funzionalità in meno spazio. Questa guida presuppone che tu stia utilizzando la versione 4_9_2015q3 della toolchain. L'utilizzo di versioni precedenti della toolchain non è consigliato. Il kit di sviluppo è preinstallato con il firmware del progetto Wireless Microcontroller Demo.

Argomenti

- Configurazione dell'hardware

- Configurazione dell'ambiente di sviluppo
- Crea ed esegui il progetto demo FreerTOS
- Debug
- Risoluzione dei problemi

Configurazione dell'hardware

Connect la MW32x scheda al laptop utilizzando un cavo da mini-USB a USB. Connect il cavo mini-USB all'unico connettore mini-USB presente sulla scheda. Non è necessario cambiare il jumper.

Se la scheda è collegata a un laptop o desktop, non è necessario un alimentatore esterno.

Questa connessione USB fornisce quanto segue:

- Accesso da console alla scheda. Una porta tty/com virtuale è registrata presso l'host di sviluppo che può essere utilizzata per accedere alla console.
- Accesso JTAG alla scheda. Questo può essere usato per caricare o scaricare immagini del firmware nella RAM o nel flash della scheda o per scopi di debug.

Configurazione dell'ambiente di sviluppo

Per scopi di sviluppo, il requisito minimo è la toolchain ARM e gli strumenti forniti in bundle con l'SDK. Le seguenti sezioni forniscono dettagli sulla configurazione della toolchain ARM.

GNU Toolchain

L'SDK supporta ufficialmente la toolchain GCC Compiler. La toolchain cross-compiler per GNU ARM è disponibile su GNU Arm Embedded Toolchain 4.9-2015-q3-update.

Il sistema di compilazione è configurato per utilizzare la toolchain GNU di default. I Makefile presuppongono che i binari della toolchain del compilatore GNU siano disponibili sul PATH dell'utente e possano essere richiamati dai Makefile. I Makefile presuppongono inoltre che i nomi dei file binari della toolchain GNU abbiano il prefisso. `arm-none-eabi-`

La toolchain GCC può essere utilizzata con GDB per eseguire il debug con OpenOCD (fornito in bundle con l'SDK). Ciò fornisce il software che si interfaccia con JTAG.

Consigliamo la versione 4_9_2015q3 della toolchain. gcc-arm-embedded

Procedura di configurazione della toolchain Linux

Segui questi passaggi per configurare la toolchain GCC in Linux.

1. Scarica il tarball della toolchain disponibile su GNU Arm Embedded Toolchain 4.9-2015-q3-update. `gcc-arm-none-eabi-4_9-2015q3-20150921-linux.tar.bz2`Il file è.
2. Copiare il file in una cartella a scelta. Assicurati che non ci siano spazi nel nome della directory.
3. Utilizzate il seguente comando per decomprimere il file.

   ```
   tar -vxf filename
   ```

4. Aggiungi il percorso della toolchain installata al PATH del sistema. Ad esempio, aggiungi la riga seguente alla fine del `.profile` file che si trova nella `/home/user-name` directory.

   ```
   PATH=$PATH:path to gcc-arm-none-eabit-4_9_2015_q3/bin
   ```

> **Note**
>
> Le nuove distribuzioni di Ubuntu potrebbero includere una versione Debian del GCC Cross Compiler. In tal caso, è necessario rimuovere il Cross Compiler nativo e seguire la procedura di configurazione sopra riportata.

Lavorare con un host di sviluppo Linux

È possibile utilizzare qualsiasi distribuzione desktop Linux moderna come Ubuntu o Fedora. Tuttavia, ti consigliamo di eseguire l'aggiornamento alla versione più recente. È stato verificato che i seguenti passaggi funzionino su Ubuntu 16.04 e presupponiamo che tu stia utilizzando quella versione.

Installazione dei pacchetti

L'SDK include uno script per consentire la configurazione rapida dell'ambiente di sviluppo su una macchina Linux di nuova configurazione. Lo script tenta di rilevare automaticamente il tipo di macchina e installare il software appropriato, incluse le librerie C, la libreria USB, la libreria FTDI, ncurses, python e latex. In questa sezione, il nome generico della directory *amzsdk_bundle-x.y.z* indica la directory principale dell'SDK. AWS Il nome effettivo della directory potrebbe essere diverso. È necessario disporre dei privilegi di root.

- Vai alla *amzsdk_bundle-x.y.z/* directory ed esegui questo comando.

```
./lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/bin/installpkgs.sh
```

Evitare sudo

In questa guida, l'`flashprog`operazione utilizza `flashprog.py` lo script per eseguire il flashing della NAND della scheda, come spiegato di seguito. Analogamente, l'`ramload`operazione utilizza `ramload.py` lo script per copiare l'immagine del firmware dall'host direttamente nella RAM del microcontrollore, senza eseguire il flashing della NAND.

È possibile configurare l'host di sviluppo Linux per eseguire le `ramload` operazioni `flashprog` and senza richiedere il `sudo` comando ogni volta. Per farlo, esegui il comando seguente.

```
./lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/bin/perm_fix.sh
```

> **Note**
>
> È necessario configurare le autorizzazioni dell'host di sviluppo Linux in questo modo per garantire un'esperienza fluida con Eclipse IDE.

Configurazione della console seriale

Inserire il cavo USB nello slot USB dell'host Linux. Ciò attiva il rilevamento del dispositivo. Dovresti vedere messaggi come i seguenti nel `/var/log/messages` file o dopo aver eseguito il `dmesg` comando.

```
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
```

Verifica che siano stati creati due dispositivi ttyUSB. Il secondo ttyUSB è la console seriale. Nell'esempio precedente, si chiama «ttyUSB1».

In questa guida, utilizziamo minicom per vedere l'output della console seriale. Potresti anche usare altri programmi seriali come. `putty` Esegui il comando seguente per eseguire minicom in modalità di configurazione.

```
minicom –s
```

In minicom, vai a Serial Port Setup e acquisisci le seguenti impostazioni.

```
| 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
```

Puoi salvare questi messaggi su minicom per uso futuro. La finestra minicom mostra ora i messaggi provenienti dalla console seriale.

Scegli la finestra della console seriale e premi il tasto Invio. Viene visualizzato un cancelletto (#) sullo schermo.

> **Note**
>
> Le schede di sviluppo includono un dispositivo FTDI in silicio. Il dispositivo FTDI espone due interfacce USB per l'host. La prima interfaccia è associata alla funzionalità JTAG dell'MCU e la seconda interfaccia è associata alla porta fisica UARTx dell'MCU.

Installazione di OpenOCD

OpenOCD è un software che fornisce debug, programmazione interna al sistema e test di scansione dei confini per dispositivi di destinazione integrati.

La disponibilità di OpenOCD versione 0.9 è obbligatoria. È inoltre necessario per la funzionalità Eclipse. Se sul tuo host Linux è stata installata una versione precedente, ad esempio la versione 0.7, rimuovi tale repository con il comando appropriato per la distribuzione Linux che stai utilizzando attualmente.

Esegui il comando Linux standard per installare OpenOCD,

```
apt-get install openocd
```

Se il comando precedente non installa la versione 0.9 o successiva, utilizzate la seguente procedura per scaricare e compilare il codice sorgente di openocd.

Per installare OpenOCD

1. Esegui il seguente comando per installare libusb-1.0.

   ```
   sudo apt-get install libusb-1.0
   ```

2. Scaricate il codice sorgente di openocd 0.9.0 da http://openocd.org/.
3. Estrai openocd e vai alla directory in cui l'hai estratto.
4. Configura openocd con il seguente comando.

   ```
   ./configure --enable-ftdi --enable-jlink
   ```

5. Esegui l'utilità make per compilare opencd.

   ```
   make install
   ```

Configurazione di Eclipse

> **Note**
>
> Questa sezione presuppone che tu abbia completato i passaggi di Evitare sudo

Eclipse è l'IDE preferito per lo sviluppo e il debug di applicazioni. Fornisce un IDE ricco e intuitivo con supporto di debug integrato, incluso il debug basato sui thread. Questa sezione descrive la configurazione comune di Eclipse per tutti gli host di sviluppo supportati.

Per configurare Eclipse

1. Scaricate e installate Java Run Time Environment (JRE).

   Eclipse richiede l'installazione di JRE. Si consiglia di installarlo prima, anche se può essere installato dopo aver installato Eclipse. La versione JRE (32/64 bit) deve corrispondere alla versione di Eclipse (32/64 bit). È possibile scaricare JRE da Java SE Runtime Environment 8 Downloads sul sito Web di Oracle.
2. Scaricate e installate «Eclipse IDE for C/C++ Developers» da http://www.eclipse.org. È supportata la versione 4.9.0 o successiva di Eclipse. L'installazione richiede solo l'estrazione dell'archivio scaricato. Per avviare l'applicazione, è necessario eseguire l'eseguibile Eclipse specifico della piattaforma.

Crea ed esegui il progetto demo FreerTOS

Esistono due modi per eseguire il progetto demo FreerTOS:

- Usa la riga di comando.
- Usa l'IDE Eclipse.

Questo argomento copre entrambe le opzioni.

Provisioning

- A seconda che utilizzi l'applicazione di test o demo, imposta i dati di provisioning in uno dei seguenti file:
  - `./tests/common/include/aws_clientcredential.h`
  - `./demos/common/include/aws_clientcredential.h`

  Per esempio:

```
#define clientcredentialWIFI_SSID "Wi-Fi SSID"
#define clientcredentialWIFI_PASSWORD "Wi-Fi password"
#define clientcredentialWIFI_SECURITY "Wi-Fi security"
```

> **Note**
>
> È possibile inserire i seguenti valori di sicurezza Wi-Fi:
>
> - `eWiFiSecurityOpen`
> - `eWiFiSecurityWEP`
> - `eWiFiSecurityWPA`
> - `eWiFiSecurityWPA2`
>
> L'SSID e la password devono essere racchiusi tra virgolette doppie.

Compila ed esegui la demo di FreerTOS usando la riga di comando

1. Usa il seguente comando per iniziare a creare l'applicazione demo.

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

   Assicuratevi di ottenere lo stesso risultato mostrato nell'esempio seguente.

```
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. Vai alla directory di compilazione.

```
cd build
```

3. Esegui l'utilità make per creare l'applicazione.

```
make all -j4
```

Assicuratevi di ottenere lo stesso risultato mostrato nella figura seguente:

```
[ 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. Utilizzate i seguenti comandi per creare un'applicazione di test.

```
cmake -DVENDOR=marvell -DBOARD=mw320 -DCOMPILER=arm-gcc -S .  -Bbuild -
DAFR_ENABLE_TESTS=1
cd build
make all -j4
```

   Esegui il `cmake` comando ogni volta che passi da a`aws_tests project`. `aws_demos project`

5. Scrivi l'immagine del firmware sul flash della scheda di sviluppo. Il firmware verrà eseguito dopo il ripristino della scheda di sviluppo. È necessario creare l'SDK prima di eseguire il flashing dell'immagine sul microcontrollore.

   a. Prima di flashare l'immagine del firmware, preparate il flash della scheda di sviluppo con i componenti comuni Layout e Boot2. Utilizzate i seguenti comandi.

   ```
   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
   ```

   Il `flashprog` comando avvia quanto segue:

- Layout: all'utilità flashprog viene innanzitutto richiesto di scrivere un layout nel flash. Il layout è simile alle informazioni sulla partizione per il flash. Il layout predefinito si trova in`/lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/OpenOCD/mw300/layout.txt`.
- Boot2 — Questo è il bootloader utilizzato dal WMSDK. Il `flashprog` comando scrive anche un bootloader nel flash. È compito del bootloader caricare l'immagine del firmware del microcontrollore dopo il flashing. Assicuratevi di ottenere lo stesso risultato mostrato nella figura seguente.

```
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. Il firmware utilizza il chipset Wi-Fi per le sue funzionalità e il chipset Wi-Fi dispone di un proprio firmware che deve essere presente anche nel flash. `flashprog.py`L'utilità viene utilizzata per eseguire il flashing del firmware Wi-Fi nello stesso modo in cui si è fatto per eseguire il flashing del bootloader Boot2 e del firmware MCU. Utilizzate i seguenti comandi per eseguire il flashing del firmware 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
```

Assicuratevi che l'output del comando sia simile alla figura seguente.

```
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. Utilizzate i seguenti comandi per eseguire il flashing del firmware dell'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. Reimpostazione della scheda Dovreste vedere i log dell'app demo.

e. Per eseguire l'app di test, esegui il flashing del `aws_tests.bin` file binario che si trova nella stessa directory.

```
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
```

L'output del comando dovrebbe essere simile a quello mostrato nella figura seguente.

```
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. Dopo aver eseguito il flashing del firmware e ripristinato la scheda, l'app demo dovrebbe avviarsi come mostrato nella figura seguente.

```
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----
```

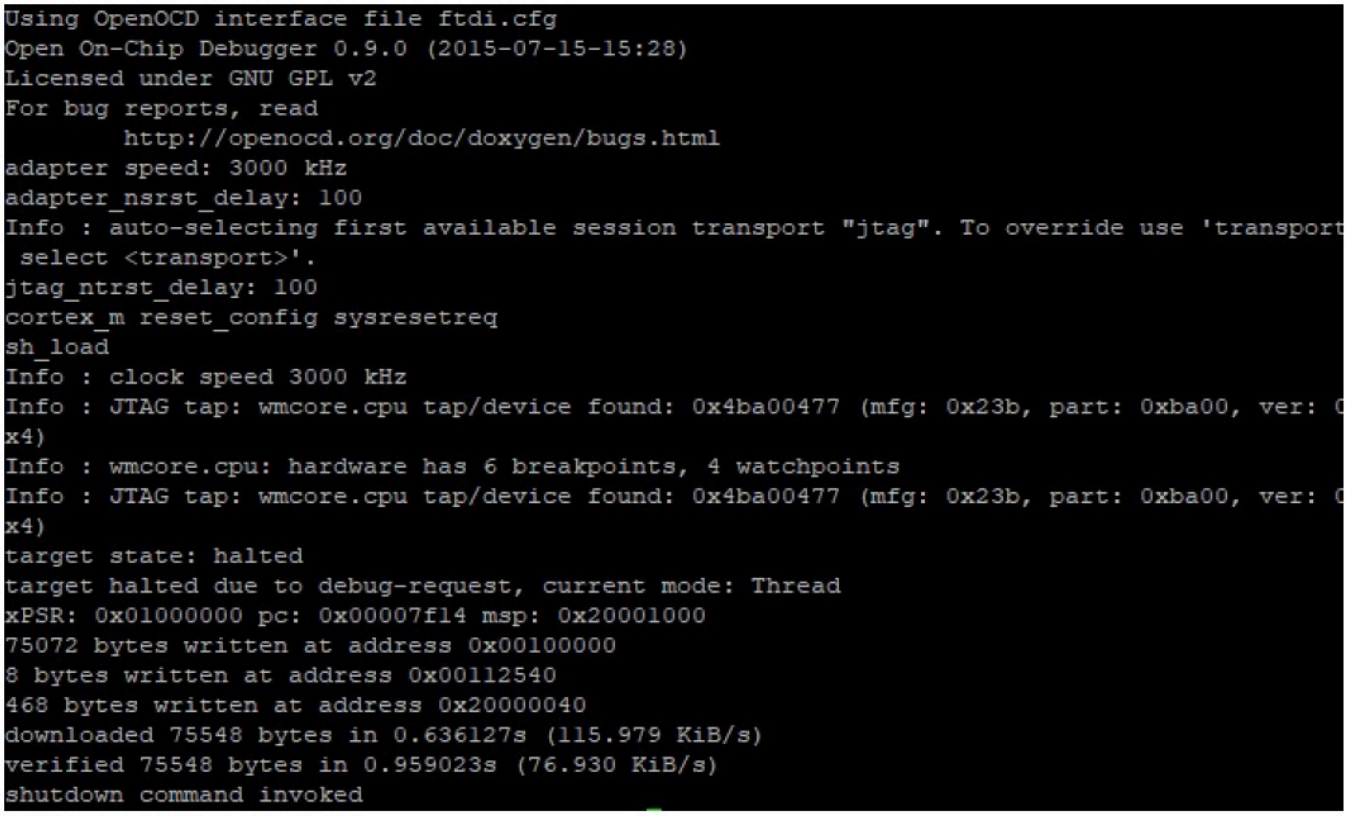
7. (Facoltativo) Come metodo alternativo per testare l'immagine, utilizzate l'utilità flashprog per copiare l'immagine del microcontrollore dall'host direttamente nella RAM del microcontrollore. L'immagine non viene copiata nel flash, quindi andrà persa dopo il riavvio del microcontrollore.

   Il caricamento dell'immagine del firmware nella SRAM è un'operazione più rapida perché avvia immediatamente il file di esecuzione. Questo metodo viene utilizzato principalmente per lo sviluppo iterativo.

   Usa i seguenti comandi per caricare il firmware nella 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
   ```

   L'output del comando è illustrato nella figura seguente.



Una volta completata l'esecuzione del comando, dovresti vedere i log dell'app demo.

Crea ed esegui la demo di FreerTOS utilizzando l'IDE Eclipse

1. Prima di configurare uno spazio di lavoro Eclipse, è necessario eseguire il comando. `cmake`

   Eseguite il comando seguente per lavorare con il progetto `aws_demos` Eclipse.

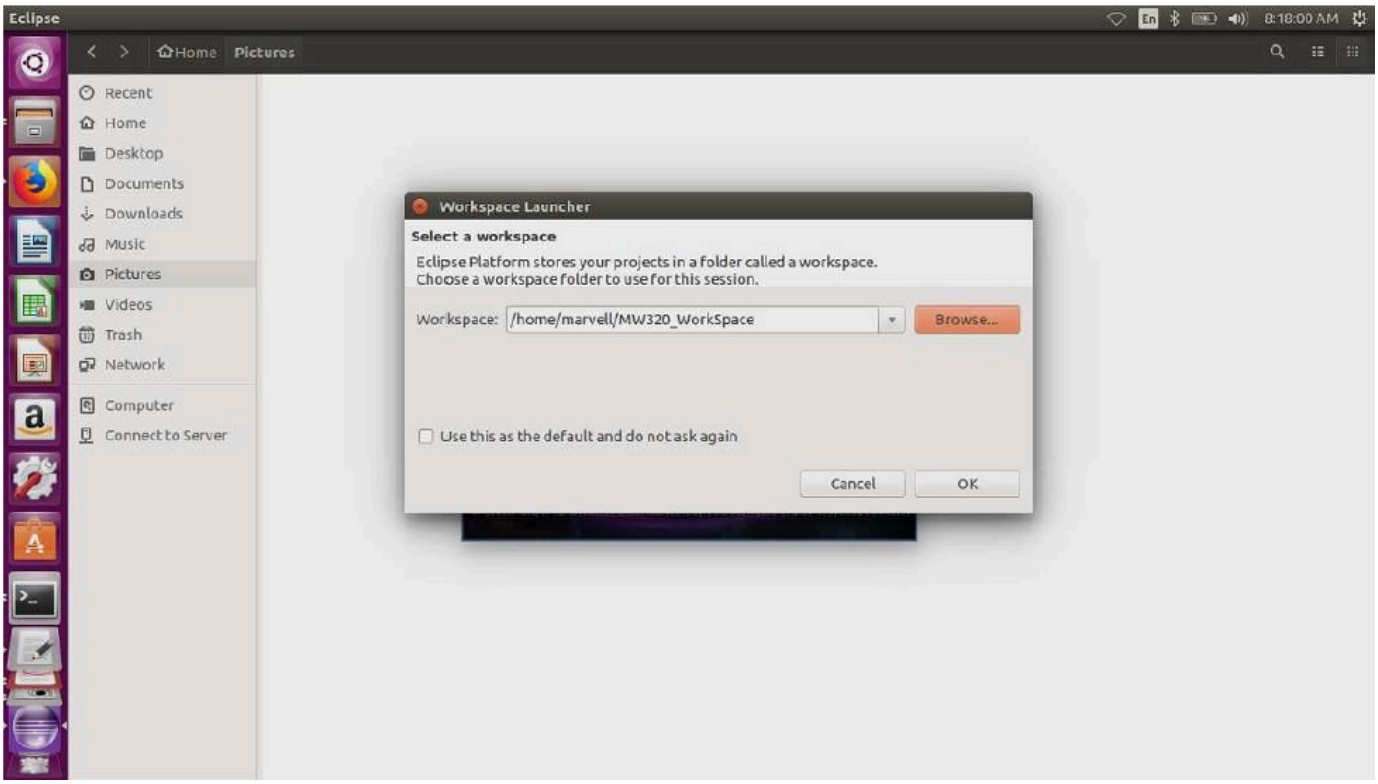
   ```
   cmake -DVENDOR=marvell -DBOARD=mw320 -DCOMPILER=arm-gcc -S .  -Bbuild -
   DAFR_ENABLE_TESTS=0
   ```

   Esegui il comando seguente per lavorare con il progetto `aws_tests` Eclipse.

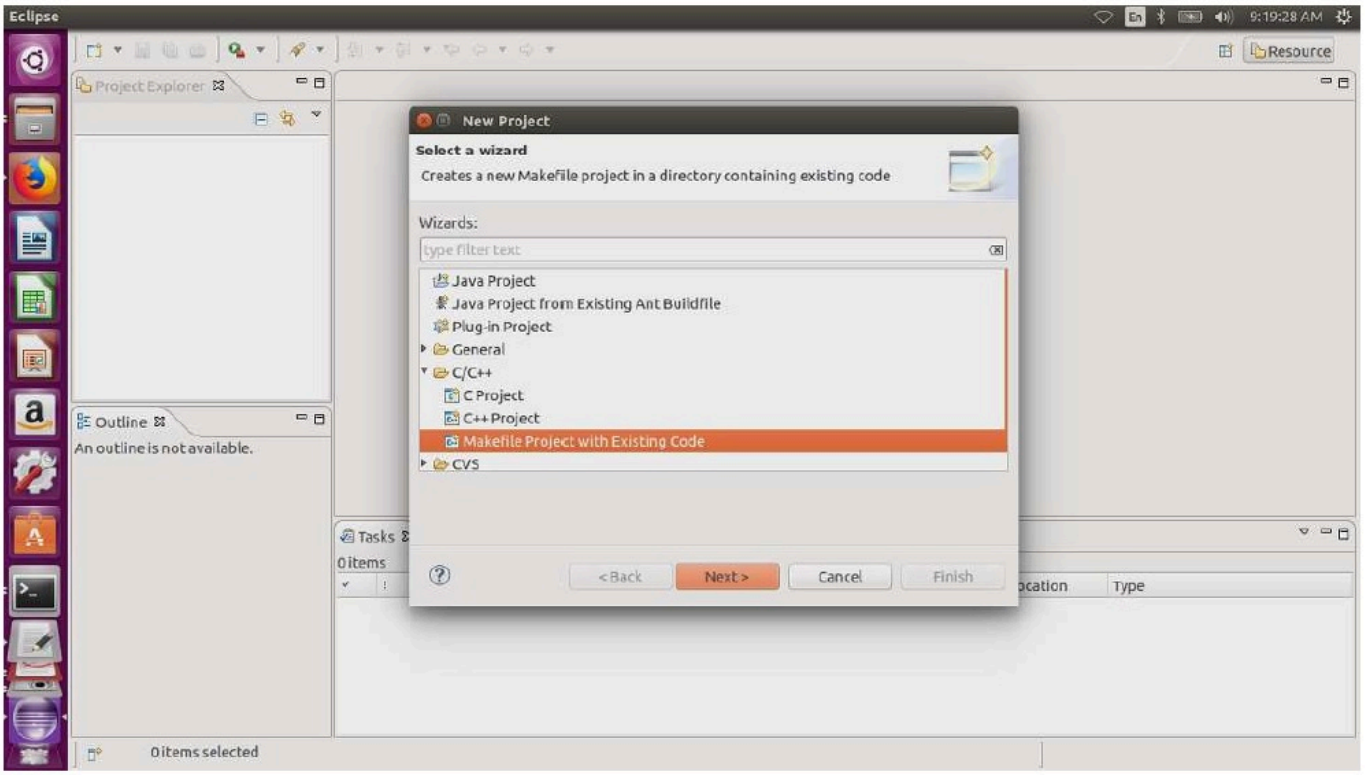
   ```
   cmake -DVENDOR=marvell -DBOARD=mw320 -DCOMPILER=arm-gcc -S .  -Bbuild -
   DAFR_ENABLE_TESTS=1
   ```

> **Tip**
>
> Esegui il `cmake` comando ogni volta che passi dal `aws_demos` progetto al `aws_tests` progetto.

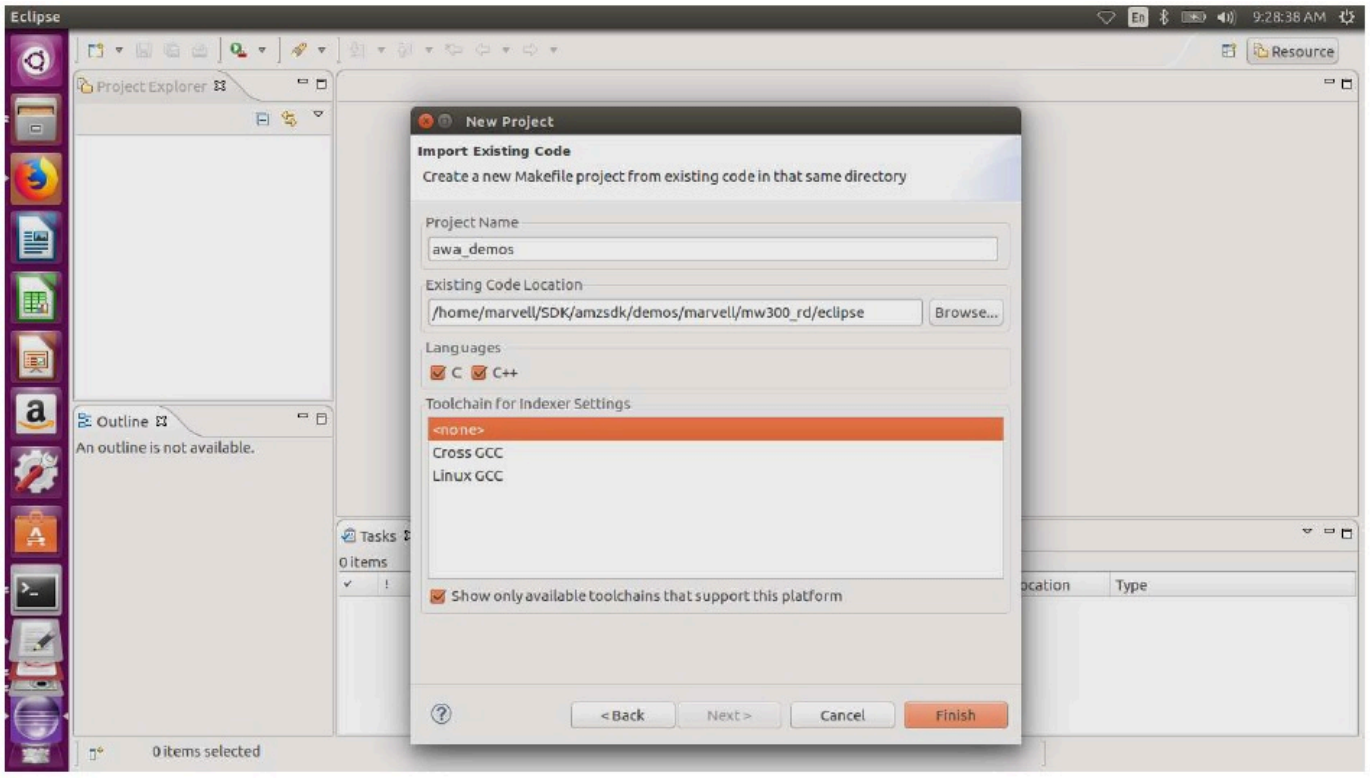
2. Apri Eclipse e, quando richiesto, scegli l'area di lavoro di Eclipse come mostrato nella figura seguente.



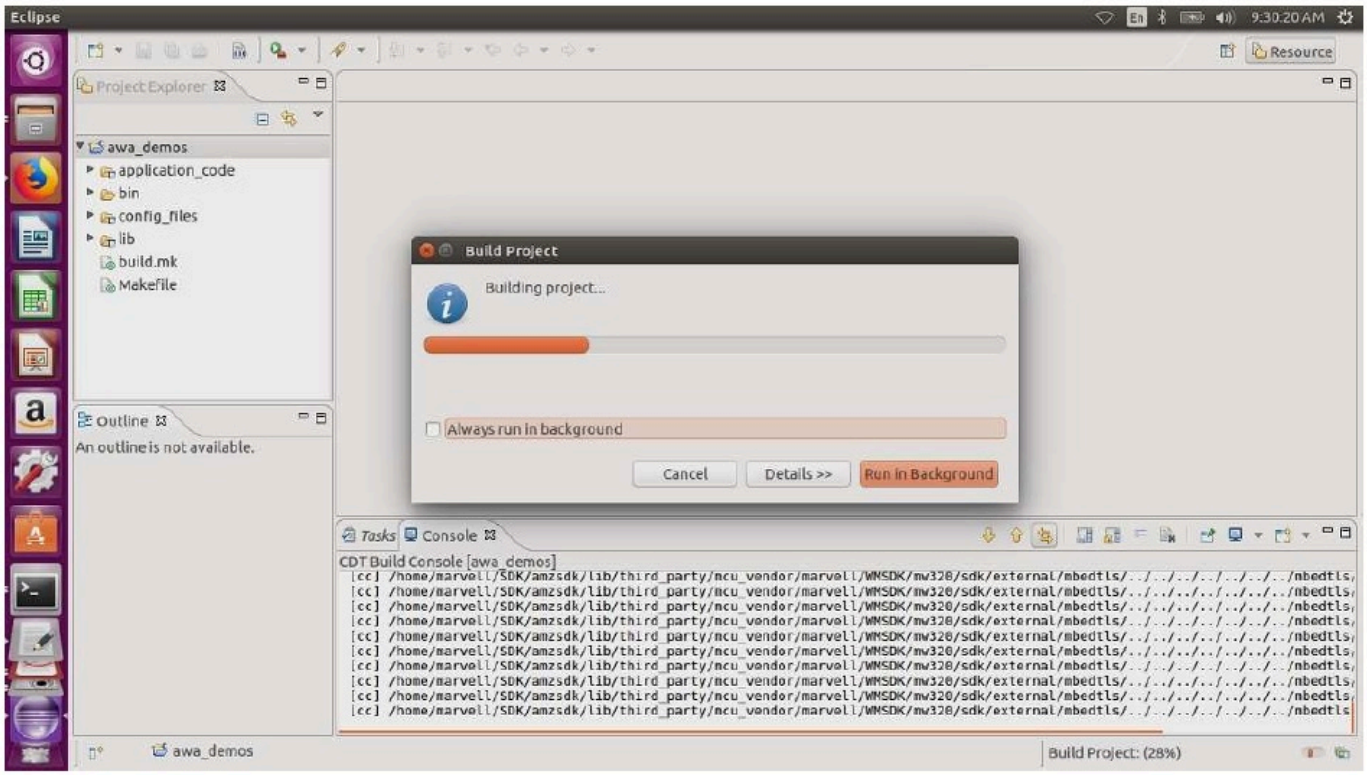
3. Scegli l'opzione per creare un progetto Makefile: con codice esistente, come mostrato nella figura seguente.



4. Scegliete Sfoglia, specificate la directory del codice esistente, quindi scegliete Fine.



5. Nel riquadro di navigazione, scegli aws_demos nel project explorer. Fai clic con il pulsante destro del mouse su aws_demos per aprire il menu, quindi scegli Build.



Se la build ha esito positivo, genera il file. `build/cmake/vendors/marvell/mw300_rd/aws_demos.bin`

6. Utilizzate gli strumenti della riga di comando per eseguire il flashing del file di layout (`layout.txt`), del binario Boot2 (`boot2.bin`), del binario del firmware MCU (`aws_demos.bin`) e del firmware Wi-Fi.

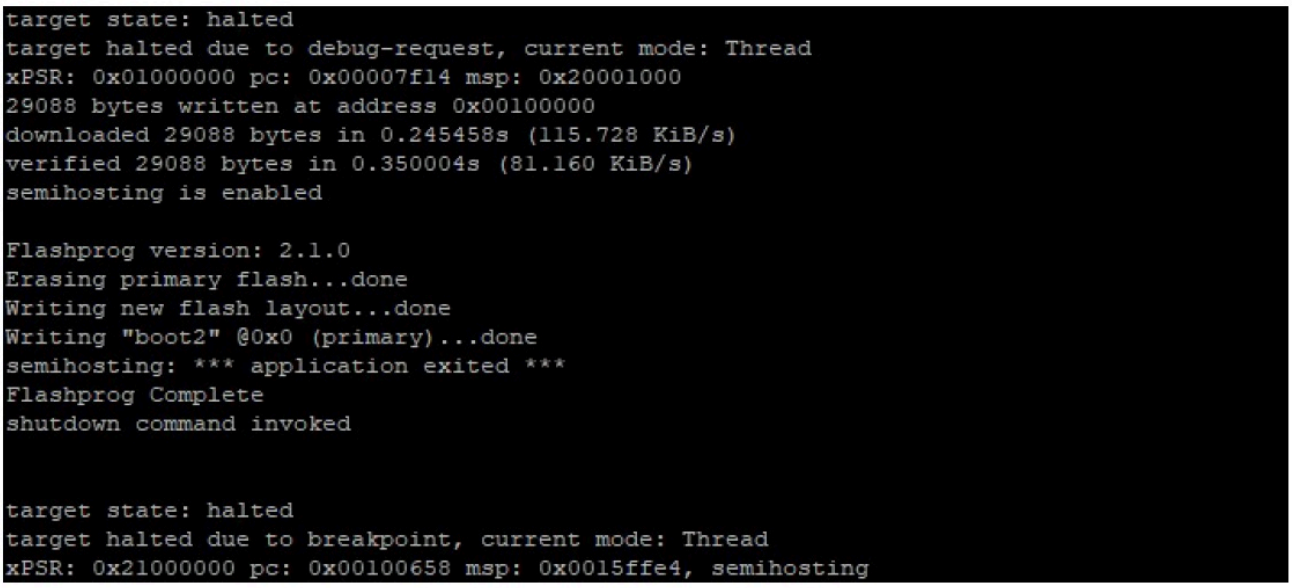
   a. Prima di eseguire il flashing dell'immagine del firmware, preparate il flash della scheda di sviluppo con i componenti comuni, Layout e Boot2. Utilizzate i seguenti comandi.

   ```
   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
   ```

   Il `flashprog` comando avvia quanto segue:

   - Layout: all'utilità flashprog viene innanzitutto richiesto di scrivere un layout nel flash. Il layout è simile alle informazioni sulla partizione per il flash. Il layout predefinito si trova in`/lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/OpenOCD/mw300/layout.txt`.

- Boot2 — Questo è il bootloader utilizzato dal WMSDK. Il comando flashprog scrive anche un bootloader nel flash. È compito del bootloader caricare l'immagine del firmware del microcontrollore dopo il flashing. Assicuratevi di ottenere lo stesso risultato mostrato nella figura seguente.



b. Il firmware utilizza il chipset Wi-Fi per le sue funzionalità e il chipset Wi-Fi dispone di un proprio firmware che deve essere presente anche nel flash. `flashprog.py`L'utilità viene utilizzata per eseguire il flashing del firmware Wi-Fi nello stesso modo in cui è stato fatto per eseguire il flashing del bootloader boot2 e del firmware MCU. Utilizzate i seguenti comandi per eseguire il flashing del firmware 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
```

Assicuratevi che l'output del comando sia simile alla figura seguente.

```
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. Utilizzate i seguenti comandi per eseguire il flashing del firmware dell'MCU.

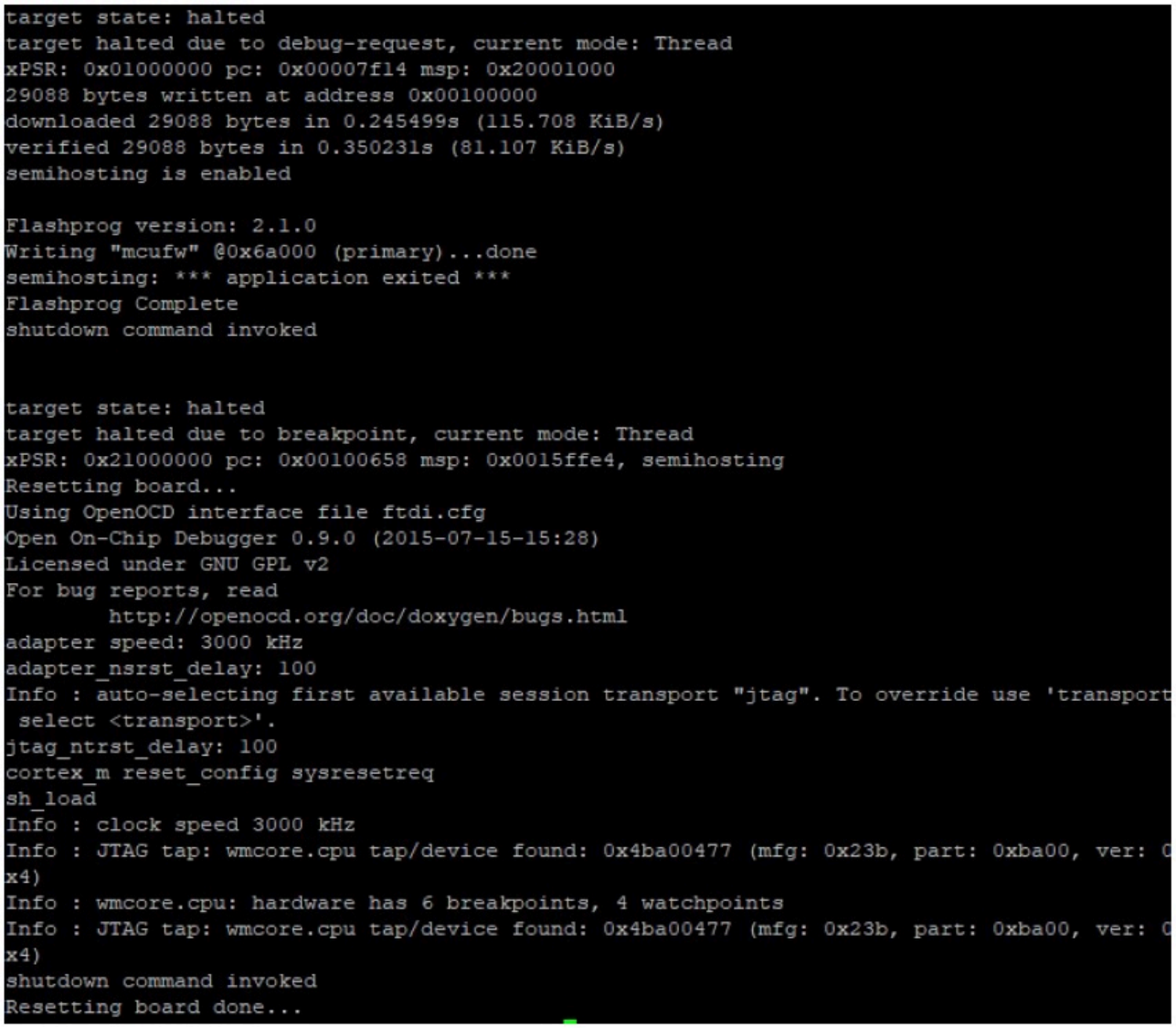
```
cd amzsdk_bundle-x.y.z
./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py --mcufw build/
cmake/vendors/marvell/mw300_rd/aws_demos.bin -r
```

d. Reimpostazione della scheda Dovreste vedere i log dell'app demo.

e. Per eseguire l'app di test, esegui il flashing del `aws_tests.bin` file binario che si trova nella stessa directory.

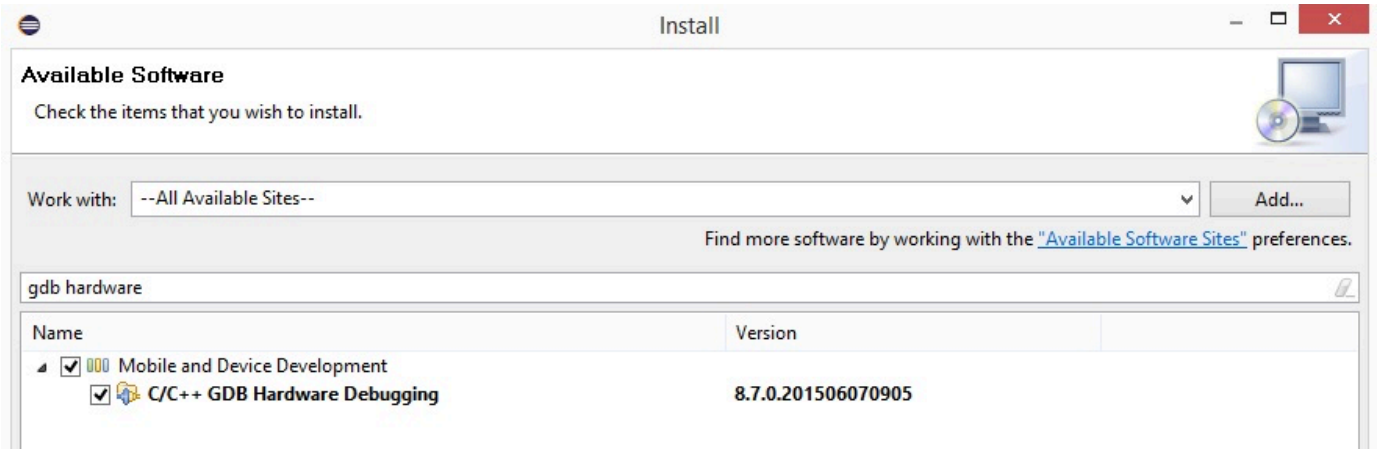
```
cd amzsdk_bundle-x.y.z
./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py --mcufw build/
cmake/vendors/marvell/mw300_rd/aws_tests.bin -r
```

L'output del comando dovrebbe essere simile a quello mostrato nella figura seguente.



Debug

- Avvia Eclipse e scegli Aiuto, quindi scegli Installa nuovo software. Nel menu Lavora con, scegli Tutti i siti disponibili. Inserisci il testo del filtroGDB `Hardware`. Seleziona l'opzione C/C++ GDB Hardware Debugging e installa il plugin.



Risoluzione dei problemi

Problemi di rete

Controlla le credenziali di rete. Vedere «Provisioning» in. Crea ed esegui il progetto demo FreerTOS

Abilitazione di registri aggiuntivi

- Abilita i log specifici della scheda.

  Abilita le chiamate alla `wmstdio_init(UART0_ID, 0)` funzione `prvMiscInitialization` nel `main.c` file per test o demo.

- Abilitazione dei registri Wi-Fi

  Abilita la macro `CONFIG_WLCMGR_DEBUG` nel *`freertos`*`/vendors/marvell/WMSDK/mw320/sdk/src/incl/autoconf.h` file.

Usare GDB

Ti consigliamo di utilizzare i file di gdb comando `arm-none-eabi-gdb` e inclusi nel pacchetto SDK. Passa alla directory .

```
cd freertos/lib/third_party/mcu_vendor/marvell/WMSDK/mw320
```

Esegui il seguente comando (su una sola riga) per connetterti a GDB.

```
arm-none-eabi-gdb -x ./sdk/tools/OpenOCD/gdbinit ../../../../../../build/cmake/
vendors/marvell/mw300 _rd/aws_demos.axf
```

Guida introduttiva al kit di sviluppo MediaTek MT7697 Hx

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Questo tutorial fornisce istruzioni per iniziare con l' MediaTek MT7697Hx Development Kit. Se non disponi dell' MediaTek MT7697Hx Development Kit, visita il AWS Partner Device Catalog per acquistarne uno dal nostro partner.

Prima di iniziare, devi configurare AWS IoT e scaricare FreerTOS per connettere il tuo dispositivo al Cloud. AWS Per istruzioni, consulta Fase iniziale. In questo tutorial, il percorso della directory di download di FreerTOS è indicato come. `freertos`

Panoramica

Questo tutorial contiene le istruzioni per i seguenti passaggi iniziali:

1. Installazione di software sul computer host per lo sviluppo e il debug di applicazioni integrate per la scheda a microcontroller.
2. Compilazione incrociata di un'applicazione demo FreerTOS con un'immagine binaria.
3. Caricamento dell'immagine binaria dell'applicazione sulla scheda in uso e successiva esecuzione dell'applicazione.
4. Interazione con l'applicazione in esecuzione sulla scheda attraverso una connessione seriale, per scopi di monitoraggio e debug.

Configurazione dell'ambiente di sviluppo

Prima di configurare l'ambiente, collega il computer alla porta USB dell' MediaTek MT7697Hx Development Kit.

Scarica e installa Keil MDK.

Puoi usare il Keil Microcontroller Development Kit (MDK) basato su GUI per configurare, creare ed eseguire progetti FreerTOS sulla tua scheda. Keil MDK include µVision IDE e µVision Debugger.

> **Note**
>
> Keil MDK è supportato su Windows 7, Windows 8 e Windows 10 solo per i computer a 64 bit.

Per scaricare e installare Keil MDK

1. Vai alla pagina delle nozioni di base di Keil MDK e scegli Download MDK-Core (Scarica MDK-core).
2. Inserisci e invia le informazioni per la registrazione a Keil.
3. Fai clic con il pulsante destro del mouse sull'eseguibile MDK e salva il programma di installazione Keil MDK sul computer.
4. Apri il programma di installazione Keil MDK e segui le fasi per completare l'operazione. Assicurati di installare il device pack (Series). MediaTek MT76x7

Stabilire una connessione seriale

Per stabilire una connessione seriale con l' MediaTek MT7697Hx Development Kit, è necessario installare il driver della porta seriale Windows Arm Mbed. Puoi scaricare il driver da Mbed. Segui i passaggi nella pagina del driver seriale di Windows per scaricare e installare il driver per l' MediaTek MT7697Hx Development Kit.

Dopo aver installato il driver, in Gestione dispositivi di Windows viene visualizzata una porta COM. Per il debug, puoi aprire una sessione sulla porta con uno strumento di utilità terminale come o. HyperTerminal TeraTerm

> **Note**
>
> In caso di problemi di connessione alla scheda dopo aver installato il driver, potrebbe essere necessario riavviare il computer.

Crea ed esegui il progetto demo FreerTOS con Keil MDK

Per creare il progetto demo FreerTOS in Keil µVision

1. Dal menu Start, apri Keil µVision 5.

2. Apri il file di progetto `projects/mediatek/mt7697hx-dev-kit/uvision/aws_demos/aws_demos.uvprojx`.
3. Dal menu, scegli Project (Progetto), quindi Build target (Destinazione compilazione).

   Dopo aver compilato il codice, il file eseguibile della demo viene visualizzato in `projects/mediatek/mt7697hx-dev-kit/uvision/aws_demos/out/Objects/aws_demo.axf`.

Per eseguire il progetto demo FreerTOS

1. Imposta l' MediaTek MT7697Hx Development Kit in modalità PROGRAM.

   Per impostare il kit per la modalità PROGRAM, premi e tieni premuto il pulsante PROG (PROGRAMMA). Con il pulsante PROG (PROGRAMMA) ancora premuto, premi e rilascia il pulsante RESET (REIMPOSTA), quindi rilascia il pulsante PROG (PROGRAMMA).
2. Dal menu, scegli Flash, quindi scegli Configure Flash Tools (Configura strumenti Flash).
3. In Opzioni per la destinazione '**`aws_demo`**', scegli la scheda Debug. Seleziona Use (Usa), imposta il debugger su CMSIS-DAP Debugger (Debugger CMSIS-DAP), quindi scegli OK.
4. Dal menu, scegli Flash, quindi Download (Scarica).

   µVision invia una notifica al completamento del download.
5. Utilizza un'utilità terminale per aprire la finestra della console seriale. Imposta la porta seriale a 115200 bps, nessuna parità, 8 bit e 1 bit di arresto.
6. Scegli il pulsante RESET sul tuo MediaTek MT7697 Hx Development Kit.

Monitoraggio dei messaggi MQTT in cloud

Puoi utilizzare il client MQTT nella AWS IoT console per monitorare i messaggi che il tuo dispositivo invia al AWS Cloud.

Per abbonarsi all'argomento MQTT con il client MQTT AWS IoT

1. Accedi alla console AWS IoT.
2. Nel riquadro di navigazione scegliere Test per aprire il client MQTT.
3. In Argomento sottoscrizione, digitare ***`your-thing-name`*`example/topic`**, quindi scegliere Effettua sottoscrizione all'argomento.

Risoluzione dei problemi

Eseguire il debug di progetti FreerTOS in Keil µVision

Attualmente, è necessario modificare il MediaTek pacchetto incluso in Keil µVision prima di poter eseguire il debug del progetto demo FreerTOS con Keil µVision. MediaTek

Per modificare il MediaTek pacchetto per il debug dei progetti FreerTOS

1. Trova e apri il file `Keil_v5\ARM\PACK\.Web\MediaTek.MTx.pdsc` nella cartella di installazione di Keil MDK.
2. Sostituisci tutte le istanze di `flag = Read32(0x20000000);`con `flag = Read32(0x0010FBFC);`.
3. Sostituisci tutte le istanze di `Write32(0x20000000, 0x76877697);`con `Write32(0x0010FBFC, 0x76877697);`.

Per avviare il debug del progetto

1. Dal menu, scegli Flash, quindi scegli Configure Flash Tools (Configura strumenti Flash).
2. Scegli la scheda Target (Destinazione), quindi scegli Read/Write Memory Areas (Aree di memoria in lettura/scrittura). Confermalo e sono entrambi selezionati IRAM1 . IRAM2
3. Scegli la scheda Debug, quindi scegli CMSIS-DAP Debugger (Debugger CMSIS-DAP).
4. Apri `vendors/mediatek/boards/mt7697hx-dev-kit/aws_demos/application_code/main.c` e imposta la macro `MTK_DEBUGGER` su `1`.
5. Ricostruisci il progetto demo in µVision.
6. Imposta l' MediaTek MT7697Hx Development Kit sulla modalità PROGRAM.

   Per impostare il kit per la modalità PROGRAM, premi e tieni premuto il pulsante PROG (PROGRAMMA). Con il pulsante PROG (PROGRAMMA) ancora premuto, premi e rilascia il pulsante RESET (REIMPOSTA), quindi rilascia il pulsante PROG (PROGRAMMA).
7. Dal menu, scegli Flash, quindi Download (Scarica).

   µVision invia una notifica al completamento del download.
8. Premi il pulsante RESET sul tuo MediaTek MT7697 Hx Development Kit.
9. Dal menu µVision, scegli Debug, quindi scegli Avvia/interrompi sessione di debug. La finestra Call Stack + Locals (Stack chiamate + Locali) si apre quando si avvia la sessione di debug.

10. Dal menu, scegliere Debug, quindi scegliere Stop (Interrompi) per sospendere l'esecuzione del codice. Il contatore del programma si arresta alla riga seguente:

```
{ volatile int wait_ice = 1 ; while ( wait_ice ) ; }
```

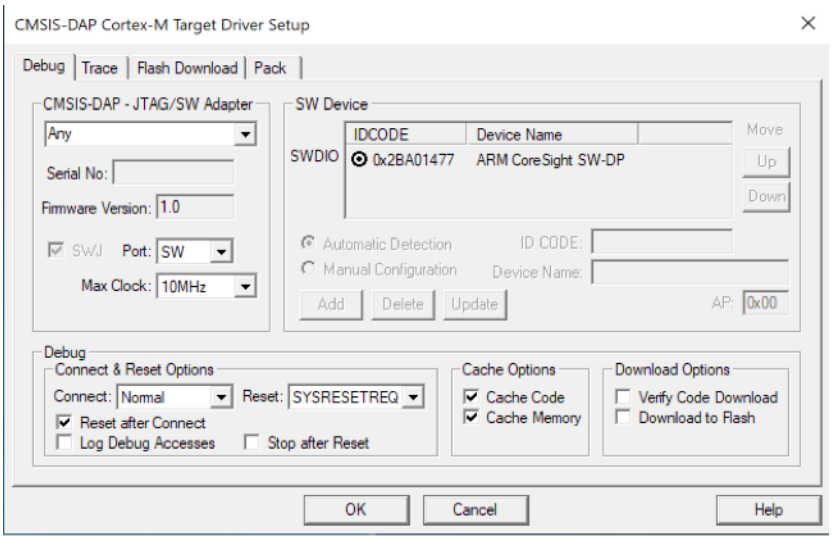
11. Nella finestra Call Stack + Locals (Stack chiamate + Locali), modifica il valore di `wait_ice` in `0`.
12. Imposta i punti di interruzione nel codice origine del progetto ed esegui il codice.

Risoluzione dei problemi relativi alle configurazioni del debugger dell'IDE

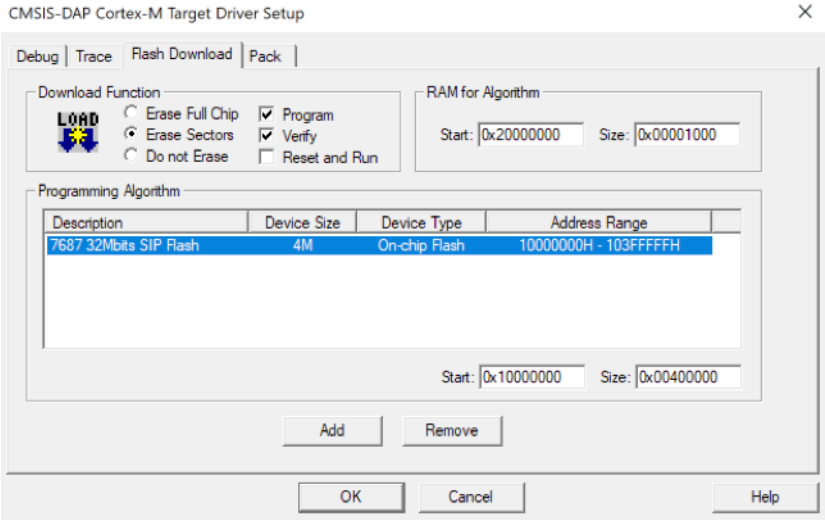
In caso di problemi durante il debug di un'applicazione, le impostazioni del debugger potrebbero essere errate.

Per verificare che le impostazioni del debugger siano corrette

1. Aprire Keil µVision.
2. Fare clic con il pulsante destro del mouse sul progetto `aws_demos`, scegliere Options (Opzioni), quindi nella scheda Utility (Utilità), scegliere Settings (Impostazioni) accanto a "-- Utilizza Debug Driver --".
3. Verificare che le impostazioni nella scheda Debug siano le seguenti:



4. Verificare che le impostazioni nella scheda Flash Download (Scarica Flash) siano le seguenti:



Per informazioni generali sulla risoluzione dei problemi su Getting Started with FreerTOS, consulta. Nozioni di base sulla risoluzione dei problemi

Guida introduttiva a Microchip Curiosity PIC32 MZ EF

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Questo tutorial fornisce istruzioni per iniziare con Microchip Curiosity PIC32 MZ EF. Se non disponi del pacchetto Microchip Curiosity PIC32 MZ EF, visita il AWS Partner Device Catalog per acquistarne uno dal nostro partner.

Il bundle include le seguenti voci:

- Scheda di sviluppo Curiosity PIC32 MZ EF
- MikroElectronika Clickboard USB UART
- MikroElectronika WiFi Scheda da 7 clic
- PIC32 LAN872Scheda figlia PHY 0

È inoltre necessario disporre delle seguenti voci per il debug:

- Debugger MPLAB Snap In-Circuit
- Kit di PICkit 3 cavi di programmazione (opzionale)

Prima di iniziare, devi configurare AWS IoT e scaricare FreerTOS per connettere il tuo dispositivo al Cloud. AWS Per istruzioni, consulta Fase iniziale.

> **Important**
>
> - In questo argomento, il percorso della directory di download di FreerTOS viene definito come. *`freertos`*
> - Gli spazi contenuti nel percorso *`freertos`* possono causare errori di compilazione. Quando si clona o si copia il repository, assicurarsi che il percorso creato non contenga spazi.
> - La lunghezza massima di un percorso di file su Microsoft Windows è di 260 caratteri. I lunghi percorsi delle directory di download di FreerTOS possono causare errori di compilazione.

Panoramica

Questo tutorial contiene le istruzioni per i seguenti passaggi iniziali:

1. Connessione della scheda a un computer host.
2. Installazione di software sul computer host per lo sviluppo e il debug di applicazioni integrate per la scheda a microcontroller.
3. Compilazione incrociata di un'applicazione demo FreerTOS con un'immagine binaria.
4. Caricamento dell'immagine binaria dell'applicazione sulla scheda in uso e successiva esecuzione dell'applicazione.
5. Interazione con l'applicazione in esecuzione sulla scheda attraverso una connessione seriale, per scopi di monitoraggio e debug.

Configura l'hardware Microchip Curiosity MZ EF PIC32

1. Collegare la MikroElectronika clickboard USB UART al connettore MicroBus 1 sul Microchip PIC32 Curiosity MZ EF.

2. Connect la scheda figlia PIC32 LAN872 0 PHY all'header J18 del Microchip Curiosity PIC32 MZ EF.
3. Connect la MikroElectronika clickboard USB UART al computer utilizzando un cavo da USB A a USB mini-B.
4. Per connettere la scheda a Internet, utilizzare una delle seguenti opzioni:

   - Per utilizzare il Wi-Fi, collega la click Board MikroElectronika Wi-Fi 7 al connettore MicroBus 2 sul Microchip Curiosity PIC32 MZ EF. Per informazioni, consulta Configurazione delle demo di FreerTOS.
   - Per utilizzare Ethernet per connettere la scheda Microchip Curiosity PIC32 MZ EF a Internet, collega la scheda figlia PIC32 LAN872 0 PHY all'intestazione J18 del Microchip Curiosity MZ EF. PIC32 Connect un'estremità di un cavo Ethernet alla scheda figlia LAN872 0 PHY. Collegare l'altra estremità al router o a un'altra porta Internet. È inoltre necessario definire la macro `PIC32_USE_ETHERNET` del preprocessore.
5. Se non l'hai già fatto, salda il connettore angolare all'intestazione ICSP del Microchip Curiosity MZ EF. PIC32
6. Collegare un'estremità del cavo ICSP dal kit di cavi di programmazione PICkit 3 al Microchip Curiosity PIC32 MZ EF.

   Se non disponi del kit PICkit 3 cavi di programmazione, puoi invece utilizzare i ponticelli M-F Dupont per cablare la connessione. Notare che il cerchio bianco indica la posizione del Pin 1.
7. Collegare l'altra estremità del cavo ICSP (o i ponticelli) al debugger snap MPLAB. Il pin 1 del connettore di programmazione SIL a 8 pin è contrassegnato dal triangolo nero in fondo a destra della scheda.

   Assicurati che qualsiasi cablaggio verso il Pin 1 del Microchip Curiosity PIC32 MZ EF, indicato dal cerchio bianco, sia allineato al Pin 1 del MPLAB Snap Debugger.

   Per ulteriori informazioni su MPLAB Snap Debugger, consulta la scheda informativa del debugger MPLAB Snap In-Circuit.

Configura l'hardware Microchip Curiosity MZ EF utilizzando On Board (PKOB) PIC32 PICkit

Consigliamo di seguire la procedura di configurazione descritta nella sezione precedente. Tuttavia, puoi valutare ed eseguire demo FreerTOS con debug di base utilizzando il programmatore/debugger PICkit integrato On Board (PKOB) seguendo questi passaggi.

1. Collegare la MikroElectronika clickboard USB UART al connettore MicroBus 1 sul Microchip PIC32 Curiosity MZ EF.
2. Per connettere la scheda a Internet, effettuare una delle seguenti operazioni:
   - Per utilizzare il Wi-Fi, collega la click Board MikroElectronika Wi-Fi 7 al connettore MicroBus 2 sul Microchip Curiosity PIC32 MZ EF. Attenersi alla procedura "Per configurare il Wi-Fi" in Configurazione delle demo di FreerTOS.
   - Per utilizzare Ethernet per connettere la scheda Microchip Curiosity PIC32 MZ EF a Internet, collega la scheda figlia PIC32 LAN872 0 PHY all'intestazione J18 del Microchip Curiosity MZ EF. PIC32 Connect un'estremità di un cavo Ethernet alla scheda figlia LAN872 0 PHY. Collegare l'altra estremità al router o a un'altra porta Internet. È inoltre necessario definire la macro `PIC32_USE_ETHERNET` del preprocessore.
3. Collegare la porta USB micro-B denominata «USB DEBUG» sulla scheda Microchip Curiosity PIC32 MZ EF al computer utilizzando un cavo da USB tipo A a USB Micro-B.
4. Connect la MikroElectronika clickboard USB UART al computer utilizzando un cavo da USB A a USB mini-B.

Configurazione dell'ambiente di sviluppo

> **Note**
>
> Il progetto FreerTOS per questo dispositivo è basato su MPLAB Harmony v2. Per creare il progetto, è necessario utilizzare versioni degli strumenti MPLAB compatibili con Harmony v2, come la v2.10 del XC32 compilatore MPLAB e le versioni 2.X.X di MPLAB Harmony Configurator (MHC).

1. Installare Python versione 3.x o versioni successive.
2. Installare MPLAB X IDE:

> **Note**
>
> AWS FreerTOS Reference Integrations v202007.00 è attualmente supportato solo nella versione 3.5. MPLabv5 Le versioni precedenti di AWS FreerTOS Reference Integrations sono supportate nella versione 4.0. MPLabv5

MPLabv53.5 download

- Ambiente di sviluppo integrato MPLAB X per Windows
- Ambiente di sviluppo integrato MPLAB X per macOS
- Ambiente di sviluppo integrato MPLAB X per Linux

Ultimi MPLab download (.40) MPLabv5

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

3. Installa il compilatore MPLAB: XC32

   - Compilatore MPLAB /32++ per Windows XC32
   - Compilatore XC32 MPLAB/32++ per macOS
   - Compilatore MPLAB XC32 /32++ per Linux

4. Avviare un emulatore di terminale UART e aprire una connessione con le impostazioni seguenti:

   - Velocità in baud: 115200
   - Dati: 8 bit
   - Parità: nessuna
   - Bit di stop: 1
   - Controllo di flusso: nessuno

Crea ed esegui il progetto demo FreerTOS

Apri la demo di FreerTOS nell'IDE MPLAB

1. Aprire MPLAB IDE. Se sono installate più versioni del compilatore, è necessario selezionare il compilatore che si desidera utilizzare dall'interno dell'IDE.
2. Dal menu File, scegliere Open Project (Apri progetto).
3. Individuare e aprire `projects/microchip/curiosity_pic32mzef/mplab/aws_demos`.
4. Scegliere Open project (Apri progetto).

> **Note**
>
> Quando si apre il progetto per la prima volta, è possibile che venga visualizzato un messaggio di errore relativo al compilatore. Nell'IDE, passare a Tools (Strumenti), Options (Opzioni), Embedded (Incorporato), quindi selezionare il compilatore che si desidera utilizzare per il progetto.

Per utilizzare Ethernet per la connessione, è necessario definire la macro del preprocessore. `PIC32_USE_ETHERNET`

Per utilizzare Ethernet per la connessione tramite l'IDE MPLAB

1. Nell'IDE MPLAB, fate clic con il pulsante destro del mouse sul progetto e scegliete Proprietà.
2. Nella finestra di dialogo Proprietà del progetto, scegliete ***`compiler-name`***(Opzioni globali) per espanderlo, quindi selezionate ***`compiler-name`*** -gcc.
3. Per le categorie Opzioni, scegliete Preelaborazione e messaggi, quindi aggiungete la **`PIC32_USE_ETHERNET`** stringa alle macro del preprocessore.

Esegui il progetto demo FreerTOS

1. Ricompilare il progetto.
2. Nella scheda Projects (Progetti), fare clic con il pulsante destro del mouse sulla cartella di livello superiore `aws_demos` e quindi scegliere Debug.
3. Quando il debugger si arresta sul punto di interruzione in `main()`, dal menu Run (Esegui), scegliere Resume (Riprendi).

Crea la demo di FreerTOS con CMake

Se preferisci non utilizzare un IDE per lo sviluppo di FreerTOS, puoi in alternativa CMake utilizzarlo per creare ed eseguire le applicazioni demo o le applicazioni che hai sviluppato utilizzando editor di codice e strumenti di debug di terze parti.

Per creare la demo di FreerTOS con CMake

1. Crea una directory per contenere i file di build generati, ad esempio. *build-directory*
2. Usa il seguente comando per generare file di build dal codice sorgente.

```
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**
>
> È necessario specificare i percorsi corretti per i file binari di Hexmate e della toolchain, come i `C:\Program Files (x86)\Microchip\MPLABX\v5.35\mplab_platform\bin` percorsi and. `C:\Program Files\Microchip\xc32\v2.40\bin`

3. Cambia le directory nella directory build (*build-directory*), quindi `make` esegui da quella directory.

Per ulteriori informazioni, consulta Utilizzo CMake con FreerTOS.

Per utilizzare Ethernet per la connessione, è necessario definire la macro del preprocessore. `PIC32_USE_ETHERNET`

Monitoraggio dei messaggi MQTT nel cloud

Puoi utilizzare il client MQTT nella AWS IoT console per monitorare i messaggi che il tuo dispositivo invia al AWS Cloud.

Per sottoscrivere l'argomento MQTT con il AWS IoT client MQTT

1. Accedere alla console AWS IoT.
2. Nel riquadro di navigazione scegliere Test per aprire il client MQTT.

3. In Argomento sottoscrizione, digitare ***your-thing-name*/example/topic**, quindi scegliere Effettua sottoscrizione all'argomento.

Risoluzione dei problemi

Per informazioni sulla risoluzione dei problemi, consulta Nozioni di base sulla risoluzione dei problemi.

Guida introduttiva al Nordic n 0-DK RF5284

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Questo tutorial fornisce istruzioni per iniziare con Nordic n 0-DK. RF5284 Se non possiedi il Nordic n RF5284 0-DK, visita il AWS Partner Device Catalog per acquistarne uno dal nostro partner.

Prima di iniziare, è necessario Configurazione AWS IoT e Amazon Cognito per FreerTOS Bluetooth Low Energy.

Per eseguire la demo FreerTOS Bluetooth Low Energy, è necessario anche un dispositivo mobile iOS o Android con funzionalità Bluetooth e Wi-Fi.

> **Note**
>
> Se utilizzi un dispositivo iOS, è richiesto Xcode per compilare l'applicazione demo per dispositivi mobili. Se utilizzi un dispositivo Android, puoi utilizzare Android Studio per compilare l'applicazione demo per dispositivi mobili.

Panoramica

Questo tutorial contiene le istruzioni per i seguenti passaggi iniziali:

1. Connessione della scheda a un computer host.

2. Installazione di software sul computer host per lo sviluppo e il debug di applicazioni integrate per la scheda a microcontroller.
3. Compilazione incrociata di un'applicazione demo FreerTOS con un'immagine binaria.
4. Caricamento dell'immagine binaria dell'applicazione sulla scheda in uso e successiva esecuzione dell'applicazione.
5. Interazione con l'applicazione in esecuzione sulla scheda attraverso una connessione seriale, per scopi di monitoraggio e debug.

Configurazione dell'hardware Nordic

Connect il computer host alla porta USB denominata J2, situata direttamente sopra il portabatterie a bottone della scheda Nordic n 0. RF5284

Per ulteriori informazioni sulla configurazione del Nordic n RF5284 0-DK, consulta la Guida per l'utente del kit di sviluppo n RF5284 0.

Configurazione dell'ambiente di sviluppo

Scaricare e installare Segger Embedded Studio

FreerTOS supporta Segger Embedded Studio come ambiente di sviluppo per Nordic n 0-DK. RF5284

Per configurare l'ambiente, è necessario scaricare e installare Segger Embedded Studio sul computer host.

Per scaricare e installare Segger Embedded Studio

1. Vai alla pagina Segger Embedded Studio Downloads e scegli l'opzione Embedded Studio for ARM per il sistema operativo in uso.
2. Eseguire il programma di installazione e seguire le istruzioni per completare l'operazione.

Configura l'applicazione demo FreerTOS Bluetooth Low Energy Mobile SDK

Per eseguire il progetto demo FreerTOS su Bluetooth Low Energy, devi eseguire l'applicazione demo FreerTOS Bluetooth Low Energy Mobile SDK sul tuo dispositivo mobile.

Per configurare l'applicazione demo FreerTOS Bluetooth Low Energy Mobile SDK

1. Seguire le istruzioni in Mobile SDKs per dispositivi Bluetooth FreerTOS per scaricare e installare l'SDK per la piattaforma mobile sul computer host.
2. Seguire le istruzioni in Applicazione dimostrativa FreerTOS Bluetooth Low Energy Mobile SDK per configurare l'applicazione demo mobile sul dispositivo mobile.

Stabilire una connessione seriale

Segger Embedded Studio include un emulatore di terminale che puoi utilizzare per ricevere i messaggi di log su una connessione seriale sulla scheda.

Per stabilire una connessione seriale con Segger Embedded Studio

1. Aprire Segger Embedded Studio.
2. Dal menu in alto, scegli Target (Destinazione), Connect J-Link (Connetti J-Link).
3. Dal menu in alto, scegli Tools (Strumenti), Terminal Emulator (Emulatore di terminale), Properties (Proprietà) e imposta le proprietà come indicato in Installazione di un emulatore di terminale.
4. Dal menu in alto, scegli Tools, Terminal Emulator, Connect ***port*** (115200, N,8,1).

> **Note**
>
> L'emulatore di terminale Segger Embedded Studio non supporta una funzionalità di input. Per questo, usa un emulatore di terminale come PuTTy, Tera Term o GNU Screen. Configura il terminale per la connessione alla scheda tramite una connessione seriale come indicato in Installazione di un emulatore di terminale.

Scarica e configura FreerTOS

Dopo aver configurato l'hardware e l'ambiente, puoi scaricare FreerTOS.

Scarica FreerTOS

Per scaricare FreerTOS for the Nordic RF5284 n 0-DK, vai alla pagina GitHub FreerTOS e clona il repository. Consultare il file README.md per le istruzioni.

Important

- In questo argomento, il percorso della directory di download di FreerTOS viene definito come. *freertos*
- Gli spazi contenuti nel percorso *freertos* possono causare errori di compilazione. Quando si clona o si copia il repository, assicurarsi che il percorso creato non contenga spazi.
- La lunghezza massima di un percorso di file su Microsoft Windows è di 260 caratteri. I lunghi percorsi delle directory di download di FreerTOS possono causare errori di compilazione.

Configurare il progetto

Per eseguire la demo, è necessario configurare il progetto su cui lavorare. AWS IoT Per configurare il progetto in modo che funzioni AWS IoT, il dispositivo deve essere registrato come AWS IoT oggetto. È necessario aver registrato il dispositivo in fase di Configurazione AWS IoT e Amazon Cognito per FreerTOS Bluetooth Low Energy.

Per configurare il tuo AWS IoT endpoint

1. Accedi alla console AWS IoT.
2. Nel pannello di navigazione scegli Impostazioni.

   L' AWS IoT endpoint viene visualizzato nella casella di testo Endpoint. L'URL dovrebbe essere del tipo *1234567890123*`-ats.iot.`*us-east-1*`.amazonaws.com`. Prendere nota di questo endpoint.
3. Nel riquadro di navigazione scegliere Manage (Gestisci), quindi Things (Oggetti). Prendi nota del nome dell' AWS IoT oggetto per il tuo dispositivo.
4. Con l' AWS IoT endpoint e il nome dell' AWS IoT oggetto a portata di mano, apri *freertos*`/demos/include/aws_clientcredential.h` nell'IDE e specifica i valori per le seguenti `#define` costanti:

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

Per abilitare la demo

1. Assicurarsi che la demo GATT Bluetooth Low Energy sia abilitata. Passare a `vendors/nordic/boards/nrf52840-dk/aws_demos/config_files/iot_ble_config.h` e aggiungere `#define IOT_BLE_ADD_CUSTOM_SERVICES ( 1 )` all'elenco delle istruzioni "define".
2. Aprire `vendors/nordic/boards/nrf52840-dk/aws_demos/config_files/aws_demos_config.h` e definire `CONFIG_MQTT_DEMO_ENABLED`.
3. Poiché il chip Nordic è dotato di poca RAM (250 KB), potrebbe essere necessario modificare la configurazione BLE per consentire voci di tabella GATT più grandi rispetto alle dimensioni di ciascun attributo. In questo modo è possibile regolare la quantità di memoria ottenuta dall'applicazione. A tale scopo, sovrascrivere le definizioni dei seguenti attributi nel file *`freertos`*`/vendors/nordic/boards/nrf52840-dk/aws_demos/config_files/sdk_config.h`:

   - NRF_SDH_BLE_VS_UUID_COUNT

     Il numero di specifiche del fornitore. UUIDs
   - NRF_SDH_BLE_GATTS_ATTR_TAB_SIZE

     Dimensione della tabella degli attributi in byte. La dimensione deve essere un multiplo di 4.

   Per i test, il percorso del file è *`freertos`*`/vendors/nordic/boards/nrf52840-dk/aws_tests/config_files/sdk_config.h`.

Crea ed esegui il progetto demo FreerTOS

Dopo aver scaricato FreerTOS e configurato il tuo progetto demo, sei pronto per creare ed eseguire il progetto demo sulla tua scheda.

> **Important**
>
> Se è la prima volta che esegui la demo sulla scheda, devi eseguire il flashing di un bootloader sulla scheda prima di poter eseguire la demo.
> Per compilare ed effettuare il flashing del bootloader, segui le istruzioni riportate di seguito e anziché utilizzare il file di progetto `projects/nordic/nrf52840-dk/ses/aws_demos/`

`aws_demos.emProject` usa `projects/nordic/nrf52840-dk/ses/aws_demos/bootloader/bootloader.emProject`.

Per creare ed eseguire la demo FreerTOS Bluetooth Low Energy di Segger Embedded Studio

1. Aprire Segger Embedded Studio. Nel menu in alto, scegliere File, Open Solution (Apri soluzione), quindi individuare il file di progetto `projects/nordic/nrf52840-dk/ses/aws_demos/aws_demos.emProject`
2. Se stai usando l'emulatore di terminale Segger Embedded Studio, scegli Tools (Strumenti) dal menu in alto, quindi scegli Terminal Emulator (Emulatore di terminale), Terminal Emulator (Emulatore di terminale) per visualizzare le informazioni della connessione seriale.

   Se stai usando un altro strumento di terminale, è possibile monitorarlo per l'output dalla connessione seriale.
3. Fare clic con il pulsante destro del mouse sul progetto demo `aws_demos` in Project Explorer (Esplora progetti) e scegliere Build (Crea).

   > **Note**
   >
   > Se è la prima volta che utilizzi Segger Embedded Studio, è possibile che venga visualizzato un avviso "No license for commercial use" (Nessuna licenza per uso commerciale). Segger Embedded Studio può essere utilizzato gratuitamente per dispositivi a semiconduttore Nordic. Scegli Activate Your Free License (Attiva licenza gratuita) e segui le istruzioni.
4. Scegliere Debug, quindi selezionare Go (Avvia).

   Dopo l'avvio della demo, viene eseguito l'accoppiamento con un dispositivo mobile in Bluetooth Low Energy.
5. Segui le istruzioni per l'applicazione demo MQTT via Bluetooth Low Energy per completare la demo con l'applicazione demo FreerTOS Bluetooth Low Energy Mobile SDK come proxy MQTT mobile.

Risoluzione dei problemi

Per informazioni generali sulla risoluzione dei problemi su Getting Started with FreerTOS, consulta. Nozioni di base sulla risoluzione dei problemi

Guida introduttiva al Nuvoton 487 NuMaker-IoT-M

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Questo tutorial fornisce istruzioni per iniziare con la scheda di sviluppo Nuvoton 487 NuMaker-IoT-M. La serie è un microcontrollore e include moduli RJ45 Ethernet e Wi-Fi integrati. Se non possiedi il Nuvoton NuMaker-IoT-M 487, visita il AWS Partner Device Catalog per acquistarne uno dal nostro partner.

Prima di iniziare, devi configurare AWS IoT il software FreerTOS per connettere la tua scheda di sviluppo al Cloud. AWS Per istruzioni, consulta Fase iniziale. In questo tutorial, il percorso della directory di download di FreerTOS è indicato come. *`freertos`*

Panoramica

Questo tutorial descrive le seguenti procedure:

1. Installazione di software sul computer host per lo sviluppo e il debug di applicazioni integrate per la scheda a microcontroller.
2. Compila in modo incrociato un'applicazione demo FreerTOS su un'immagine binaria.
3. Caricamento dell'immagine binaria dell'applicazione sulla scheda in uso e successiva esecuzione dell'applicazione.

Configurazione dell'ambiente di sviluppo

Keil MDK Nuvoton Edition è progettato per lo sviluppo e il debug di applicazioni per schede Nuvoton M487. La versione Keil MDK v5 Essential, Plus o Pro dovrebbe funzionare anche per il

microcontroller Nuvoton M487 (Cortex-M4 core). Puoi scaricare l'edizione Keil MDK Nuvoton con uno sconto sul prezzo per la serie Nuvoton Cortex-M4. MCUs Keil MDK è supportato solo su Windows.

Per installare lo NuMaker-IoT-M strumento di sviluppo per il 487

1. Scarica Keil MDK Nuvoton Edition dal sito Web di Keil MDK.
2. Installare Keil MDK sulla macchina host utilizzando la licenza. Keil MDK include Keil µVision IDE, una toolchain di compilazione C/C++ e il debugger di µVision.

   Se si verificano problemi durante l'installazione, contattare Nuvoton per ricevere assistenza.
3. Installa NU-Link_Keil_Driver_v3.06.7215r (o versione più recente), che si trova nella pagina Nuvoton Development Tool.

Crea ed esegui il progetto demo FreerTOS

Per creare il progetto demo FreerTOS

1. Aprire l'IDE Keil µVision.
2. Nel menu File scegliere Open (Apri). Nella finestra di dialogo Open file (Apri file), verificare che il selettore del tipo di file sia impostato su Project Files (File di progetto).
3. Scegli il progetto demo Wi-Fi o Ethernet da compilare.

   - Per aprire il progetto demo Wi-Fi, scegliere il progetto di destinazione `aws_demos.uvproj` nella directory *`freertos`*`\projects\nuvoton\numaker_iot_m487_wifi\uvision\aws_demos`.
   - Per aprire il progetto demo Ethernet, scegliere il progetto di destinazione `aws_demos_eth.uvproj` nella directory *`freertos`*`\projects\nuvoton\numaker_iot_m487_wifi\uvision\aws_demos_eth`.
4. Per assicurarti che le impostazioni siano corrette per la flash della scheda, fai clic con il pulsante destro del mouse sul progetto `aws_demo` nell'IDE, quindi scegli Options (Opzioni). Per ulteriori dettagli, consulta Risoluzione dei problemi.
5. Nella scheda Utilities (Utilità), verificare che l'opzione Use Target Driver for Flash Programming (Usa driver di destinazione per programmazione flash) sia selezionata e che Nuvoton Nu-Link Debugger sia impostato come driver di destinazione.
6. Nella scheda Debug accanto a Nuvoton Nu-Link Debugger, scegliere Settings (Impostazioni).
7. Verificare che chip Type (Tipo di chip) sia impostato su M480.

8. Nel riquadro di navigazione del Progetto Keil µVision IDE, scegliere il progetto `aws_demos`. Dal menu Project (Progetto), scegliere Build All (Crea tutti).

Puoi utilizzare il client MQTT nella AWS IoT console per monitorare i messaggi che il tuo dispositivo invia al Cloud. AWS

Per abbonarsi all'argomento MQTT con il client MQTT AWS IoT

1. Accedere alla console AWS IoT.
2. Nel riquadro di navigazione scegliere Test per aprire il client MQTT.
3. In Argomento sottoscrizione, digitare ***your-thing-name*****/example/topic**, quindi scegliere Effettua sottoscrizione all'argomento.

Per eseguire il progetto demo FreerTOS

1. Connect la scheda Numaker-IoT-M 487 alla macchina host (computer).
2. Ricreare il progetto.
3. Nel menu Flash dell'IDE Keil µVision, scegliere Download (Scarica).
4. Nel menu Debug scegliere Start/Stop Debug Session (Avvia/Arresta sessione di debug).
5. Quando il debugger si arresta nel punto di interruzione in `main()`, apri il menu Run (Esegui), quindi scegli Run (F5) (Esegui (F5).

   Dovreste vedere i messaggi MQTT inviati dal dispositivo nel client MQTT della console. AWS IoT

Utilizzo CMake con FreerTOS

Puoi anche usare CMake per creare ed eseguire le applicazioni demo FreerTOS o le applicazioni che hai sviluppato utilizzando editor di codice e strumenti di debug di terze parti.

Assicurati di aver installato il sistema di compilazione. CMake Segui le istruzioni che trovi su Utilizzo CMake con FreerTOS e quindi segui i passaggi descritti in questa sezione.

> **Note**
>
> Assicurati che il percorso alla posizione del compilatore (Keil) sia nella variabile di sistema Path (Percorso), ad esempio `C:\Keil_v5\ARM\ARMCC\bin`.

Puoi anche utilizzare il client MQTT nella AWS IoT console per monitorare i messaggi che il tuo dispositivo invia al AWS Cloud.

Per abbonarsi all'argomento MQTT con il client MQTT AWS IoT

1. Accedere alla console AWS IoT.
2. Nel riquadro di navigazione scegliere Test per aprire il client MQTT.
3. In Argomento sottoscrizione, digitare ***your-thing-name*/example/topic**, quindi scegliere Effettua sottoscrizione all'argomento.

Per generare file di compilazione dai file di origine ed eseguire il progetto demo

1. Sul computer host, aprire il prompt dei comandi e passare alla cartella *freertos*.
2. Creare una cartella per i file compilati generati. Faremo riferimento a questa cartella come a. *BUILD_FOLDER*
3. Genera i file della build per la dimostrazione Wi-Fi o Ethernet.

   - Per Wi-Fi:

     Passa alla directory che contiene i file sorgente per il progetto demo FreerTOS. Quindi generare i file della build eseguendo il comando seguente.

     ```
     cmake -DVENDOR=nuvoton -DBOARD=numaker_iot_m487_wifi -DCOMPILER=arm-keil -S . -
     B BUILD_FOLDER -G Ninja
     ```

   - Per Ethernet:

     Passa alla directory che contiene i file sorgente per il progetto demo FreerTOS. Quindi generare i file della build eseguendo il comando seguente.

     ```
     cmake -DVENDOR=nuvoton -DBOARD=numaker_iot_m487_wifi -DCOMPILER=arm-keil -
     DAFR_ENABLE_ETH=1 -S . -B BUILD_FOLDER -G Ninja
     ```

4. Generare il file binario nella memoria flash sull'M487 eseguendo il comando seguente.

   ```
   cmake --build BUILD_FOLDER
   ```

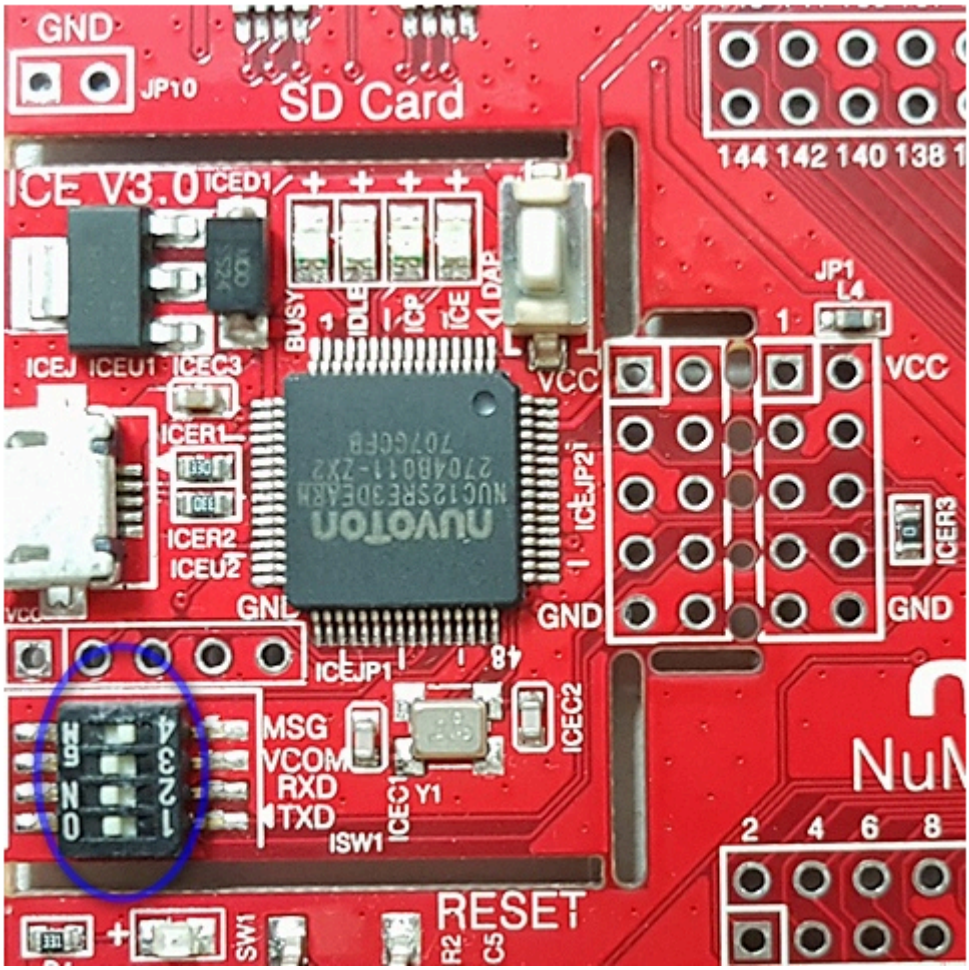
   A questo punto, il file binario `aws_demos.bin` deve trovarsi nella cartella *BUILD_FOLDER*`/vendors/Nuvoton/boards/numaker_iot_m487_wifi`.

5. Per configurare la scheda per la modalità flash, assicurati che l'interruttore MSG (n. 4 di ICE) sia ISW1 acceso. Quando si collega la scheda, verrà assegnata una finestra (e un'unità). Consultare Risoluzione dei problemi.
6. Aprire un emulatore di terminale per visualizzare i messaggi tramite UART. Segui le istruzioni riportate in Installazione di un emulatore di terminale.
7. Esegui il progetto demo copiando il file binario generato sul dispositivo.

   Se ti sei abbonato all'argomento MQTT con il client AWS IoT MQTT, dovresti vedere i messaggi MQTT inviati dal tuo dispositivo nella console. AWS IoT

Risoluzione dei problemi

- Se Windows non è in grado di riconoscere il dispositivo`VCOM`, installa il driver della porta seriale di NuMaker Windows dal link Nu-Link USB Driver v1.6.
- Se colleghi il tuo dispositivo a Keil MDK (IDE) tramite Nu-Link, assicurati che lo switch MSG (n. 4 di on ICE) sia OFF, come mostrato. ISW1



In caso di problemi durante la configurazione dell'ambiente di sviluppo o la connessione alla scheda, contatta Nuvoton.

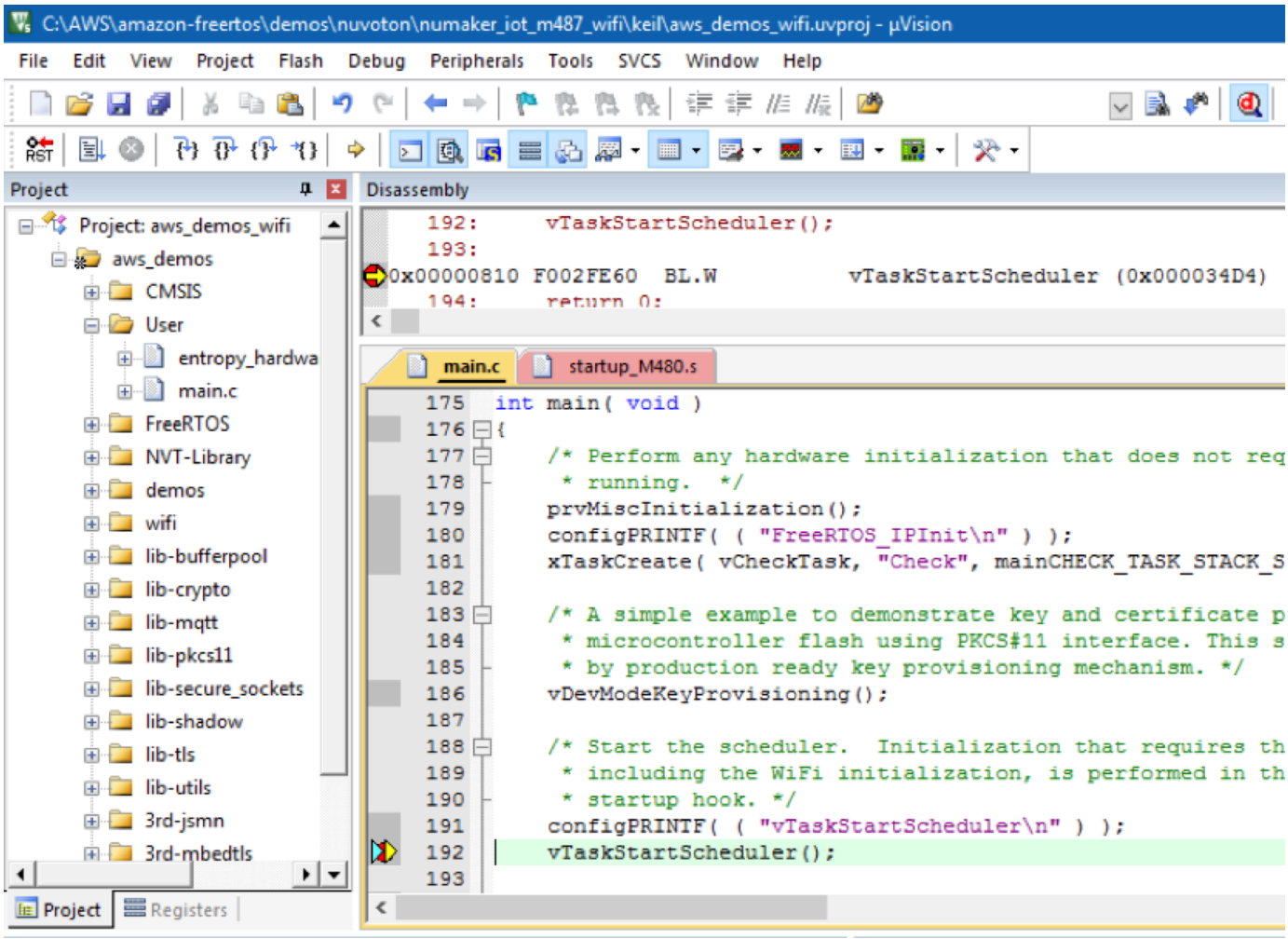
Eseguire il debug di progetti FreerTOS in Keil µVision

Per avviare una sessione di debug in Keil µVision

1. Aprire Keil µVision.
2. Segui i passaggi per creare il progetto demo FreerTOS. Crea ed esegui il progetto demo FreerTOS
3. Nel menu Debug scegliere Start/Stop Debug Session (Avvia/Arresta sessione di debug).

   La finestra Call Stack+Locals (Stack chiamate) viene visualizzata quando avvii una sessione di debug. µVision lampeggia la demo sulla scheda, esegue la demo e si arresta all'inizio della funzione `main()`.

4. Imposta i punti di interruzione nel codice origine del progetto ed esegui il codice. Il progetto deve essere simile al seguente:



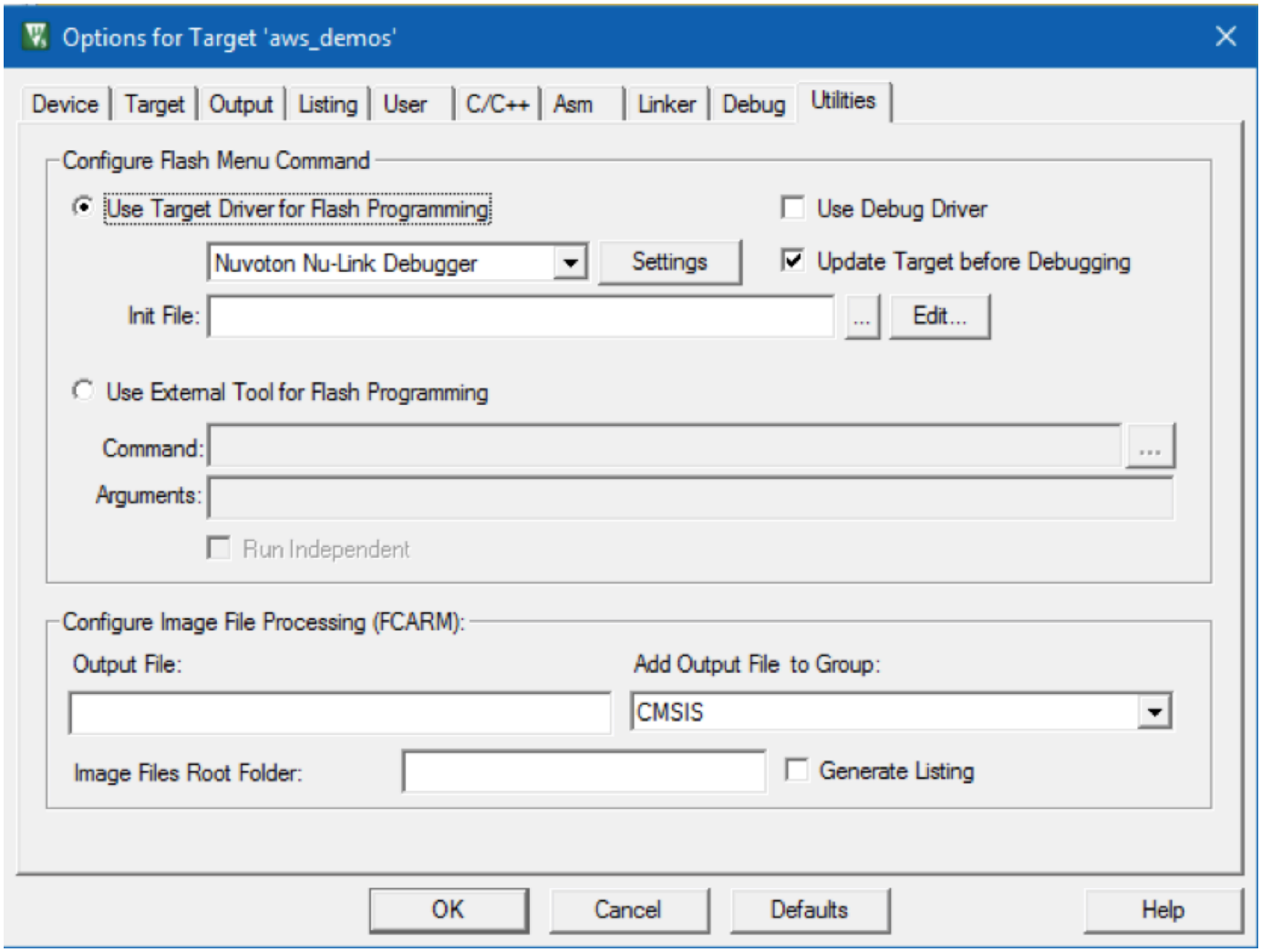
Risoluzione dei problemi delle impostazioni di debug di µVision

Se si verificano problemi durante il debug di un'applicazione, controllare che le impostazioni di debug siano impostate correttamente in Keil µVision.

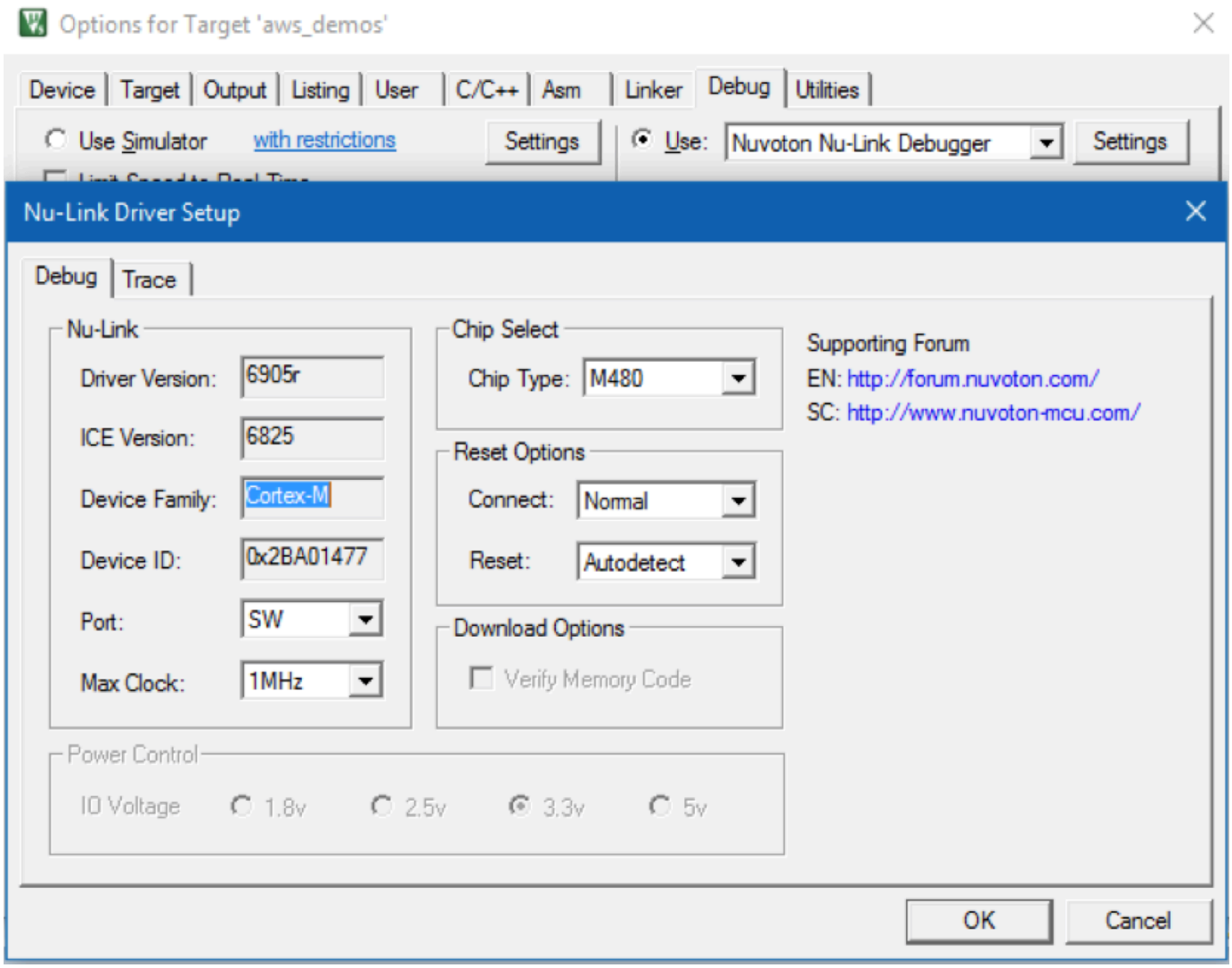
Per verificare che le impostazioni di debug di µVision siano corrette

1. Aprire Keil µVision.
2. Fare clic con il pulsante destro del mouse sul progetto `aws_demo` nell'IDE, quindi scegliere Options (Opzioni).

3. Nella scheda Utilities (Utilità), verificare che l'opzione Use Target Driver for Flash Programming (Usa driver di destinazione per programmazione flash) sia selezionata e che Nuvoton Nu-Link Debugger sia impostato come driver di destinazione.



4. Nella scheda Debug accanto a Nuvoton Nu-Link Debugger, scegliere Settings (Impostazioni).



5. Verificare che chip Type (Tipo di chip) sia impostato su M480.

Guida introduttiva al modulo LPC54 IoT NXP 018

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Questo tutorial fornisce istruzioni per iniziare con il modulo LPC54 IoT NXP 018. Se non disponi di un modulo LPC54 IoT NXP 018, visita il AWS Partner Device Catalog per acquistarne uno dal nostro partner. Usa un cavo USB per collegare il modulo LPC54 IoT NXP 018 al computer.

Prima di iniziare, devi configurare AWS IoT e scaricare FreerTOS per connettere il tuo dispositivo al Cloud. AWS Per istruzioni, consulta Fase iniziale. In questo tutorial, il percorso della directory di download di FreerTOS è indicato come. *freertos*

Panoramica

Questo tutorial contiene le istruzioni per i seguenti passaggi iniziali:

1. Connessione della scheda a un computer host.
2. Installazione di software sul computer host per lo sviluppo e il debug di applicazioni integrate per la scheda a microcontroller.
3. Compilazione incrociata di un'applicazione demo FreerTOS con un'immagine binaria.
4. Caricamento dell'immagine binaria dell'applicazione sulla scheda in uso e successiva esecuzione dell'applicazione.

Configurazione dell'hardware NXP

Per configurare NXP 018 LPC54

- Connect il computer alla porta USB dell'NXP LPC54 018.

Per configurare il debugger JTAG

È necessario un debugger JTAG per avviare ed eseguire il debug del codice in esecuzione sulla scheda NXP 018. LPC54 FreerTOS è stato testato utilizzando un modulo IoT OM4 0006. Per ulteriori informazioni sui debugger supportati, vedere il Manuale dell'utente per il modulo IoT NXP LPC54 018 disponibile dalla pagina del prodotto del modulo IoT OM4 0007 LPC54 018.

1. Se utilizzi un debugger del modulo IoT OM4 0006, usa un cavo convertitore per collegare il connettore a 20 pin dal debugger al connettore a 10 pin del modulo IoT NXP.
2. Connect NXP LPC54 018 e OM4 0006 IoT Module Debugger alle porte USB del computer utilizzando cavi da mini-USB a USB.

Configurazione dell'ambiente di sviluppo

FreerTOS ne supporta IDEs due per il modulo IoT NXP LPC54 018: IAR Embedded Workbench e. MCUXpresso

Prima di iniziare, installa uno di questi. IDEs

Per installare IAR Embedded Workbench per ARM

1. Accedi a IAR Embedded Workbench per ARM e scarica il software.

   > **Note**
   >
   > IAR Embedded Workbench per ARM richiede Microsoft Windows.

2. Esegui il programma di installazione e segui le istruzioni.
3. In License Wizard (Procedura guidata della licenza), scegliere Register with IAR Systems to get an evaluation license (Registra con IAR Systems per ottenere una licenza di valutazione).
4. Inserire il bootloader sul dispositivo prima di provare a eseguire qualsiasi demo.

Per eseguire l'installazione da NXP MCUXpresso

1. Scarica ed esegui il MCUXpresso programma di installazione da NXP.

   > **Note**
   >
   > Sono supportate le versioni 10.3.x e successive.

2. Vai a MCUXpresso SDK e scegli Crea il tuo SDK.

   > **Note**
   >
   > Sono supportate le versioni 2.5 e successive.

3. Scegliere Select Development Board (Seleziona scheda di sviluppo).
4. In Select Development Board (Seleziona scheda di sviluppo), in Search by Name (Cerca per nome), immettere **`LPC54018-IoT-Module`**.
5. In Boards, scegli LPC54018-IOT-Module.

6. Verifica i dettagli hardware, quindi scegli Build SDK. MCUXepresso
7. L'SDK per Windows che utilizza l' MCUXpresso IDE è già integrato. Scegliere Download SDK (Scarica SDK). Se si sta usando un altro sistema operativo, in Host OS (Sistema operativo dell'host), scegliere il sistema operativo e quindi scegliere Download SDK (Scarica SDK).
8. Avvia l' MCUXpresso IDE e scegli la SDKs scheda Installato.
9. Trascina e rilascia il file di archivio SDK scaricato nella SDKs finestra Installato.

In caso di problemi durante l'installazione, consulta Supporto NXP o Risorse per gli sviluppatori NXP.

Crea ed esegui il progetto FreerTOS Demo

Importa la demo di FreerTOS nel tuo IDE

Per importare il codice di esempio FreerTOS nell'IDE IAR Embedded Workbench

1. Aprire IAR Embedded Workbench e dal menu File scegliere Open Workspace (Apri Workspace).
2. Nella casella di testo search-directory, immettere `projects/nxp/lpc54018iotmodule/iar/aws_demos` e scegliere aws_demos.eww.
3. Dal menu Project (Progetto), scegliere Rebuild All (Ricrea tutti).

Per importare il codice di esempio di FreerTOS nell'IDE MCUXpresso

1. Apri MCUXpresso e dal menu File scegli Apri progetti dal file system.
2. Nella casella di testo Directory immettere `projects/nxp/lpc54018iotmodule/mcuxpresso/aws_demos` e scegliere Finish (Termina)
3. Dal menu Project (Progetto), scegliere Build All (Crea tutti).

Esegui il progetto demo FreerTOS

Per eseguire il progetto demo FreerTOS con l'IDE IAR Embedded Workbench

1. Nell'IDE, dal menu Project (Progetto), scegliere Make (Crea).
2. Dal menu Project (Progetto) scegliere Download and Debug (Scarica ed esegui il debug).
3. Dal menu Debug scegliere Start Debugging (Avvia debug).
4. Quando il debugger si arresta sul punto di interruzione in `main`, dal menu Debug, scegliere Go (Vai).

> **Note**
>
> Se si apre una finestra di dialogo J-Link Device Selection (Selezione dispositivi J-Link), scegliere OK per continuare. Nella finestra di dialogo Target Device Settings (Impostazioni del dispositivo target), scegliere Unspecified (Non specificato), scegliere Cortex-M4 e quindi scegliere OK. È sufficiente eseguire questa operazione una sola volta.

Per eseguire il progetto demo FreerTOS con l'IDE MCUxpresso

1. Nell'IDE, dal menu Project (Progetto), scegliere Build (Crea).
2. Se è la prima volta che si esegue il debug, scegliere il progetto `aws_demos` e dalla barra degli strumenti Debug, scegliere il pulsante di debug blu.
3. Vengono visualizzate tutte le sonde di debug rilevate. Scegliere la sonda che si desidera utilizzare e scegliere OK per iniziare il debug.

   > **Note**
   >
   > Quando il debugger si arresta sul punto di interruzione in `main()`, premere il pulsante di riavvio del debug
   >
   >
   >
   > una volta per reimpostare la sessione di debug. (Questo è necessario a causa di un bug con il MCUXpresso debugger per NXP54 il modulo 018-IoT).
4. Quando il debugger si arresta sul punto di interruzione in `main()`, dal menu Debug, scegliere Go (Vai).

Monitoraggio dei messaggi MQTT in cloud

Puoi utilizzare il client MQTT nella AWS IoT console per monitorare i messaggi che il tuo dispositivo invia al Cloud. AWS

Per abbonarsi all'argomento MQTT con il client MQTT AWS IoT

1. Accedi alla console AWS IoT.

2. Nel riquadro di navigazione scegliere Test per aprire il client MQTT.
3. In Argomento sottoscrizione, digitare ***your-thing-name*/example/topic**, quindi scegliere Effettua sottoscrizione all'argomento.

Risoluzione dei problemi

Per informazioni generali sulla risoluzione dei problemi su Getting Started with FreerTOS, consulta. Nozioni di base sulla risoluzione dei problemi

Guida introduttiva al Renesas Starter Kit+ per N-2MB RX65

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Questo tutorial fornisce istruzioni per iniziare con Renesas Starter Kit+ per N-2MB. RX65 Se non disponi del Renesas RSK+ per RX65 N-2 MB, visita il Partner Device Catalog e acquistane uno dai nostri partner. AWS

Prima di iniziare, devi configurare AWS IoT e scaricare FreerTOS per connettere il tuo dispositivo al Cloud. AWS Per istruzioni, consulta Fase iniziale. In questo tutorial, il percorso della directory di download di FreerTOS è indicato come. *freertos*

Panoramica

Questo tutorial contiene le istruzioni per i seguenti passaggi iniziali:

1. Connessione della scheda a un computer host.
2. Installazione di software sul computer host per lo sviluppo e il debug di applicazioni integrate per la scheda a microcontroller.
3. Compilazione incrociata di un'applicazione demo FreerTOS con un'immagine binaria.
4. Caricamento dell'immagine binaria dell'applicazione sulla scheda in uso e successiva esecuzione dell'applicazione.

Configurazione dell'hardware Renesas

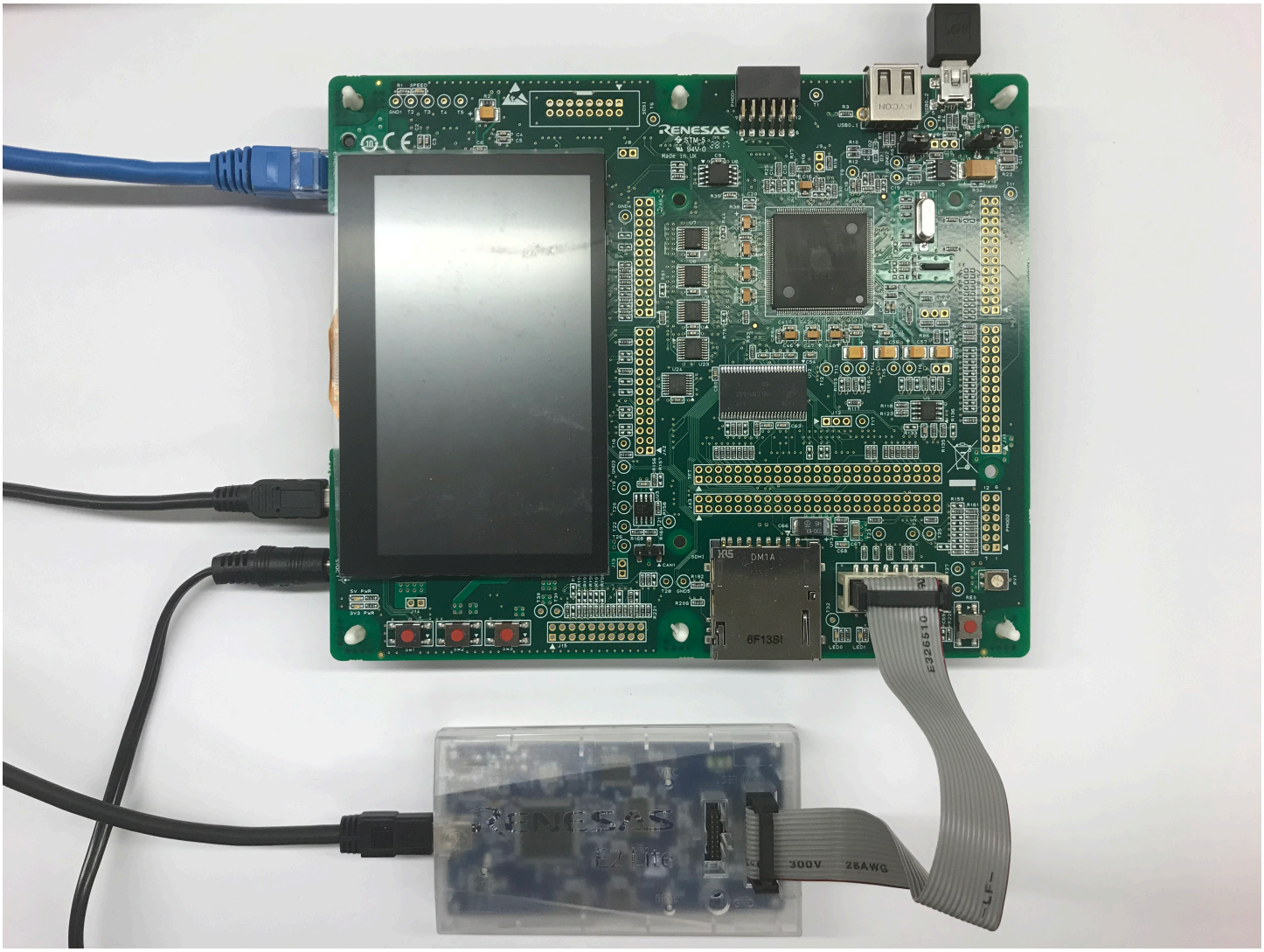
Per configurare RSK+ per N-2 MB RX65

1. Connect l'alimentatore positivo +5V al connettore PWR su RSK+ per N-2MB. RX65
2. Connect il computer alla porta USB2 3.0 FS su RSK+ per N-2 MB. RX65
3. Connect il computer alla USB-to-serial porta dell'RSK+ per RX65 N-2 MB.
4. Connect un router o una porta Ethernet connessa a Internet alla porta Ethernet dell'RSK+ per N-2 MB. RX65

Per configurare il modulo E2 Lite Debugger

1. Usa il cavo a nastro a 14 pin per collegare il modulo E2 Lite Debugger alla porta «E1/E2 Lite» su RSK+ per N-2 MB. RX65
2. Utilizzare un cavo USB per collegare il modulo E2 Lite debugger al computer host. Quando E2 Lite Debugger è connesso alla scheda e al computer, un LED verde "ACT" lampeggia sul debugger.
3. Dopo aver collegato il debugger al computer host e RSK+ per N-2MB, inizia l'installazione dei driver del debugger E2 Lite. RX65

   Si noti che sono necessari i privilegi di amministratore per l'installazione dei driver.



Configurazione dell'ambiente di sviluppo

Per configurare le configurazioni FreerTOS per RSK+ RX65 per N-2 MB, usa l'IDE di studio Renesas e 2 e il compilatore CC-RX.

> **Note**
>
> Renesas e^{2}studio IDE e il compilatore CC-RX sono supportati solo sui sistemi operativi Windows 7, 8 e 10.

Per scaricare e installare e^{2}studio

1. Vai alla pagina di download del programma di installazione di Renesas e 2 studio e scarica il programma di installazione offline.
2. L'utente verrà reindirizzato alla pagina di login di Renesas.

Se hai un account con Renesas, inserisci le tue credenziali di accesso e scegli Accedi.

Se non si dispone di un account, scegliere Register now (Registrati ora), e seguire le prime fasi della registrazione. Si riceverà un'e-mail con un collegamento per attivare l'account Renesas. Seguire questo link per completare la registrazione con Renesas, quindi effettuare l'accesso a Renesas.

3. Dopo aver effettuato l'accesso, scaricare il programma di installazione di e^{2}studio sul computer.
4. Aprire il programma di installazione e seguire le fasi per completare l'operazione.

Per ulteriori informazioni, consulta lo studio e ^{2} sul sito Web di Renesas.

Per scaricare e installare il pacchetto del compilatore RX Family C/C++

1. Vai alla pagina di download del pacchetto compilatore RX Family C/C++ e scarica il pacchetto V3.00.00.
2. Aprire l'eseguibile e installare il compilatore.

Per ulteriori informazioni, consultare Pacchetto compilatore C/C++ Package per RX Family sul sito Web di Renesas.

> **Note**
>
> Il compilatore è disponibile gratuitamente solo per la versione di prova con una validità di 60 giorni. Al 61° giorno è necessario ottenere una chiave di licenza. Per ulteriori informazioni, consulta Strumenti software di valutazione.

Crea ed esegui esempi di FreerTOS

Una volta configurato il progetto di demo, è possibile compilare ed eseguire il progetto demo sulla scheda.

Crea la demo di FreerTOS in e 2 studio

Per importare e creare la demo in e^{2}studio

1. Avviare e^{2}studio dal menu Start.

2. Nella finestra Select a directory as a workspace (Seleziona una directory come workspace), selezionare la cartella che si desidera utilizzare e scegliere Launch (Avvia).
3. La prima volta che si apre e^{2}studio, si apre la finestra Toolchain Registry (Registro Toolchain). Selezionare Renesas Toolchain (Toolchain Renesas) e confermare che è stato selezionato **`CC-RX v3.00.00`**. Scegliere Register (Registra), quindi selezionare OK.
4. Se si sta aprendo e^{2}studio per la prima volta, appare la finestraCode Generator Registration (Registrazione generatore codice). Scegli OK.
5. Appare la finestra Code Generator COM component register (Registro del componente COM del generatore di codice). In Riavvia e ^{2} studio per utilizzare Code Generator, scegli OK.
6. Viene visualizzata la finestra Restart e ^{2} studio. Scegli OK.
7. e^{2}studio si riavvia. Nella finestra Select a directory as a workspace (Seleziona una directory come workspace), scegliere Launch (Avvia).
8. Nella schermata di benvenuto di e ^{2} studio, scegli l'icona a forma di freccia Vai all'ambiente di lavoro e ^{2} studio.
9. Fare clic con il pulsante destro del mouse sulla finestra Project Explorer, quindi selezionare Import (Importa).
10. Nella procedura guidata di importazione selezionare General (Generale), Existing Projects into Workspace (Progetti esistenti in Workspace), quindi scegliere Next (Successivo).
11. Scegliere Browse (Sfoglia), individuare la directory `projects/renesas/rx65n-rsk/e2studio/aws_demos`, quindi scegliere Finish (Termina).
12. Dal menu Project (Progetto), scegliere Project (Progetto), Build All (Crea tutti).

    Nella console della build viene visualizzato un messaggio di avviso che License Manager non è installato. È possibile ignorare questo messaggio, a meno che non si disponga di una chiave di licenza per il compilatore CC-RX. Per installare License Manager, consulta la pagina di download di Licence Manager.

Esegui il progetto FreerTOS

Per eseguire il progetto in e^{2}studio

1. Conferma di aver collegato il modulo E2 Lite Debugger al tuo RSK+ per N-2 MB RX65
2. Nel menu in alto selezionare Run (Esegui), quindi Debug Configuration (Esegui debugging configurazione).
3. Espandi Renesas GDB Hardware Debugging e scegli aws_demos. HardwareDebug

4. Scegliere la scheda Debugger, quindi selezionare la scheda Connection Settings (Impostazioni di connessione). Verificare che le impostazioni di connessione siano corrette.
5. Scegliere Debug (Esegui debugging) per scaricare il codice nella scheda in uso e avviare il debugging.

   È possibile visualizzare un avviso del firewall per `e2-server-gdb.exe`. ControllarePrivate networks, such as my home or work network (Reti private, ad esempio la rete domestica o del lavoro), quindi scegliere Allow access (Consenti accesso).
6. e^{2}studio potrebbe chiedere di modificare in Renesas Debug Perspective. Scegliere Yes (Sì).

   Il LED verde "ACT" su E2 Lite Debugger si illumina.
7. Dopo aver scaricato il codice nella scheda, scegliere Resume (Riprendi) per eseguire il codice fino alla prima riga della funzione principale. Scegliere Resume (Riprendi) di nuovo per eseguire il resto del codice.

Monitoraggio dei messaggi MQTT in cloud

Puoi utilizzare il client MQTT nella AWS IoT console per monitorare i messaggi che il tuo dispositivo invia al Cloud. AWS

Per sottoscrivere l'argomento MQTT con il AWS IoT client MQTT

1. Accedi alla console AWS IoT.
2. Nel riquadro di navigazione scegliere Test per aprire il client MQTT.
3. In Argomento sottoscrizione, digitare ***your-thing-name*****/example/topic**, quindi scegliere Effettua sottoscrizione all'argomento.

Per gli ultimi progetti rilasciati da Renesas, consultate il `renesas-rx` fork del repository su. `amazon-freertos` GitHub

Risoluzione dei problemi

Per informazioni generali sulla risoluzione dei problemi su Getting Started with FreerTOS, consulta. Nozioni di base sulla risoluzione dei problemi

Guida introduttiva al nodo IoT STMicroelectronics STM32 L4 Discovery Kit

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Questo tutorial fornisce istruzioni per iniziare a utilizzare il nodo IoT STMicroelectronics STM32 L4 Discovery Kit. Se non disponi già del nodo IoT STMicroelectronics STM32 L4 Discovery Kit, visita il AWS Partner Device Catalog per acquistarne uno dal nostro partner.

Assicurati di aver installato il firmware Wi-Fi più recente. Per scaricare il firmware Wi-Fi più recente, vedi nodo IoT del kit STM32 L4 Discovery, wireless a basso consumo, Bluetooth Low Energy, NFC, GHz Sub, Wi-Fi. In Binary Resources (Risorse binarie), scegliere Inventek ISM 43362 Wi-Fi module firmware update (read the readme file for instructions) (Aggiornamento del firmware del modulo Wi-Fi di Inventek ISM 43362) (leggi il file readme per le istruzioni)).

Prima di iniziare, devi configurare AWS IoT e scaricare i FreerTOS per connettere il tuo dispositivo al Cloud. AWS Per istruzioni, consulta Fase iniziale. In questo tutorial, il percorso della directory di download di FreerTOS è indicato come. `freertos`

Panoramica

Questo tutorial contiene le istruzioni per i seguenti passaggi iniziali:

1. Installazione di software sul computer host per lo sviluppo e il debug di applicazioni integrate per la scheda a microcontroller.
2. Compilazione incrociata di un'applicazione demo FreerTOS con un'immagine binaria.
3. Caricamento dell'immagine binaria dell'applicazione sulla scheda in uso e successiva esecuzione dell'applicazione.

Configurazione dell'ambiente di sviluppo

Installa System Workbench per STM32

1. Accedere a STM32Open.org.

2. Effettua la registrazione sulla STM32 pagina web Open. È necessario effettuare l'accesso per scaricare System Workbench.
3. Accedere a System Workbench per il programma di STM32 installazione per scaricare e installare System Workbench.

Se riscontri problemi durante l'installazione, consulta il sito Web di System FAQs Workbench.

Crea ed esegui il progetto demo FreerTOS

Importa la demo di FreerTOS nel System Workbench STM32

1. Apri STM32 System Workbench e inserisci un nome per un nuovo spazio di lavoro.
2. Dal menu File scegliere Import (Importa). Espandere General (Generale), scegliere Existing Projects into Workspace (Progetti esistenti in Workspace), quindi scegliere Next (Successivo).
3. In Select Root Directory (Seleziona directory principale), immettere `projects/st/stm32l475_discovery/ac6/aws_demos`.
4. Il progetto `aws_demos` deve essere selezionato per impostazione predefinita.
5. Scegliete Fine per importare il progetto in STM32 System Workbench.
6. Dal menu Project (Progetto), scegliere Build All (Crea tutti). Confermare che il progetto è stato compilato senza errori.

Esegui il progetto demo FreerTOS

1. Usa un cavo USB per collegare il tuo nodo IoT STMicroelectronics STM32 L4 Discovery Kit al computer.
2. Da Project Explorer, fate clic con il pulsante destro del mouse`aws_demos`, scegliete Debug As, quindi scegliete Applicazione Ac6 C/C++ STM32 .

   Se si verifica un errore di debug la prima volta in cui viene avviata una sessione di debug, procedere nel seguente modo:

   1. In STM32 System Workbench, dal menu Esegui, scegli Debug Configurations.
   2. Scegliere aws_demos Debug. (Potrebbe essere necessario espandere STM32 Ac6 Debugging.)
   3. Selezionare la scheda Debugger.

4. In Configuration Script (Script di configurazione), scegliere Show Generator Options (Mostra opzioni di generatore).
5. In Mode Setup (Impostazione modalità), impostare Reset Mode (Modalità reset) per Software System Reset (Reset sistema software). Selezionare Apply (Applica), quindi selezionare Debug.

3. Quando il debugger si arresta sul punto di interruzione in `main()`, dal menu Run (Esegui), scegliere Resume (Riprendi).

Utilizzo CMake con FreerTOS

Se preferisci non utilizzare un IDE per lo sviluppo di FreerTOS, puoi in alternativa CMake utilizzarlo per creare ed eseguire le applicazioni demo o le applicazioni che hai sviluppato utilizzando editor di codice e strumenti di debug di terze parti.

Per prima cosa create una cartella che contenga i file di build generati (). *`build-folder`*

Utilizzare il comando seguente per generare i file compilati:

```
cmake -DVENDOR=st -DBOARD=stm32l475_discovery -DCOMPILER=arm-gcc -S freertos -B build-folder
```

Se non `arm-none-eabi-gcc` è nel percorso della shell, devi anche impostare la `AFR_TOOLCHAIN_PATH` CMake variabile. Per esempio:

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

Per ulteriori informazioni sull'utilizzo CMake con FreerTOS, consulta. Utilizzo CMake con FreerTOS

Monitoraggio dei messaggi MQTT in cloud

Puoi utilizzare il client MQTT nella AWS IoT console per monitorare i messaggi che il tuo dispositivo invia al Cloud. AWS

Per sottoscrivere l'argomento MQTT con il AWS IoT client MQTT

1. Accedi alla console AWS IoT.
2. Nel riquadro di navigazione scegliere Test per aprire il client MQTT.

3. In Argomento sottoscrizione, digitare ***your-thing-name*/example/topic**, quindi scegliere Effettua sottoscrizione all'argomento.

Risoluzione dei problemi

Se visualizzi le seguenti stringhe di testo nell'output UART dall'applicazione demo, devi aggiornare il firmware del modulo Wi-Fi:

```
[Tmr Svc] WiFi firmware version is: xxxxxxxxxxxx
[Tmr Svc] [WARN] WiFi firmware needs to be updated.
```

Per scaricare il firmware Wi-Fi più recente, vedi nodo IoT del kit STM32 L4 Discovery, wireless a basso consumo, Bluetooth Low Energy, NFC, GHz Sub, Wi-Fi. In Binary Resources (Risorse binarie), scegliere il link di download in Inventek ISM 43362 Wi-Fi module firmware update (Aggiornamento firmware del modulo Wi-Fi Inventek 43362 ISM).

Per informazioni generali sulla risoluzione dei problemi su Getting Started with FreerTOS, consulta. Nozioni di base sulla risoluzione dei problemi

Guida introduttiva a Texas Instruments CC322 0SF-LAUNCHXL

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Questo tutorial fornisce istruzioni per iniziare con Texas Instruments CC322 0SF-LAUNCHXL. Se non disponi del kit di sviluppo Texas Instruments (TI) CC322 0SF-LAUNCHXL, visita il Partner Device Catalog per acquistarne uno dal AWS nostro partner.

Prima di iniziare, devi configurare AWS IoT e scaricare FreerTOS per connettere il tuo dispositivo al Cloud. AWS Per istruzioni, consulta Fase iniziale. In questo tutorial, il percorso della directory di download di FreerTOS è indicato come. *freertos*

Panoramica

Questo tutorial contiene le istruzioni per i seguenti passaggi iniziali:

1. Installazione di software sul computer host per lo sviluppo e il debug di applicazioni integrate per la scheda a microcontroller.
2. Compilazione incrociata di un'applicazione demo FreerTOS con un'immagine binaria.
3. Caricamento dell'immagine binaria dell'applicazione sulla scheda in uso e successiva esecuzione dell'applicazione.

Configurazione dell'ambiente di sviluppo

Segui i passaggi seguenti per configurare il tuo ambiente di sviluppo e iniziare a usare FreerTOS.

Nota che FreerTOS ne supporta IDEs due per il kit di sviluppo CC322 TI 0SF-LAUNCHXL: Code Composer Studio e IAR Embedded Workbench versione 8.32. Per iniziare puoi utilizzare uno dei due IDE.

Installazione di Code Composer Studio

1. Passare a TI Code Composer Studio.
2. Scaricare il programma di installazione offline per la piattaforma del computer host (Windows, macOS o Linux a 64 bit).
3. Decomprimere ed eseguire il programma di installazione offline. Seguire le istruzioni.
4. Per le famiglie SimpleLink CC32xx di prodotti da MCUs installare, scegli Wi-Fi Wireless.
5. Nella pagina successiva accettare le impostazioni predefinite per il debug delle sonde e quindi scegliere Finish (Termina).

Se riscontri problemi durante l'installazione di Code Composer Studio, consulta TI Development Tools Support, Code Composer Studio FAQs e Troubleshooting CCS.

Installazione di IAR Embedded Workbench

1. Scaricare ed eseguire il programma di installazione di Windows per la versione 8.32 di IAR Embedded Workbench for ARM. In Debug probe driver (Effettua il debug del driver della sonda), accertarsi che l'opzione TI XDS sia selezionata:
2. Completare l'installazione e lanciare il programma. Nella pagina License Wizard (Procedura guidata della licenza), scegliere Register with IAR Systems to get an evaluation license (Registra con IAR Systems per ottenere una licenza di valutazione) oppure utilizzare la licenza di IAR.

Installa 0 SDK SimpleLink CC322

Installa l'SDK SimpleLink CC322 0. L'SDK SimpleLink Wi-Fi CC322 0 contiene i driver per l'MCU programmabile CC322 0SF, più di 40 applicazioni di esempio e la documentazione necessaria per utilizzare gli esempi.

Installazione di Uniflash

Installazione di Uniflash. CCS Uniflash è uno strumento autonomo utilizzato per programmare la memoria flash su chip su TI. MCUs Uniflash dispone di un'interfaccia utente grafica, a riga di comando e di scripting.

Installazione del Service Pack più recente

1. Sul tuo TI CC322 0SF-LAUNCHXL, posiziona il jumper SOP sul set centrale di pin (posizione = 1) e resetta la scheda.
2. Avviare Uniflash. Se la scheda CC322 0SF appare in Dispositivi rilevati, scegli Avvia LaunchPad . Se la tua scheda non viene rilevata, scegli CC3220SF-LAUNCHXL dall'elenco delle schede in Nuova configurazione, quindi scegli Avvia Image Creator.
3. Scegli Nuovo progetto.
4. Nella pagina Start new project (Inizia nuovo progetto) immettere un nome per il progetto. Per Tipo di dispositivo, scegli 0SF. CC322 Per Device Mode (Modalità dispositivo), scegliere Develop (Sviluppa), e quindi scegliere Create Project (Crea progetto).
5. A destra nella finestra dell'applicazione Uniflash scegliere Connect (Connetti).
6. Dalla colonna di sinistra, scegliere Advanced (Avanzato), Files (File), quindi Service Pack.
7. Scegli Sfoglia, quindi vai al punto in cui hai installato l'SDK CC322 SimpleLink 0SF. Il service pack si trova in `ti/simplelink_cc32xx_sdk_`*`VERSION`*`/tools/cc32xx_tools/servicepack-cc3x20/sp_`*`VERSION`*`.bin`.
8. Scegliere il pulsante Burn
   ( ),
   quindi scegliere Program Image (Create & Program) ((Immagine programma) (Crea e programma)) per installare il service pack. Ricordarsi ristabilire il cavo di collegamento SOP nella posizione 0 e di resettare la scheda.

Configurazione del provisioning Wi-Fi

Per configurare le impostazioni Wi-Fi per la scheda, procedi in uno dei seguenti modi:

- Configurare l'applicazione demo FreerTOS descritta in. Configurazione delle demo di FreerTOS
- Utilizzo SmartConfigda Texas Instruments.

Crea ed esegui il progetto demo FreerTOS

Crea ed esegui il progetto demo FreerTOS in TI Code Composer

Per importare la demo di FreerTOS in TI Code Composer

1. Aprire TI Code Composer e scegliere OK per accettare il nome predefinito del workspace.
2. Nella pagina Getting started (Nozioni di base) scegliere Import Project (Importa progetto).
3. In Select search-directory (Seleziona directory di ricerca), immettere `projects/ti/cc3220_launchpad/ccs/aws_demos`. Il progetto `aws_demos` deve essere selezionato per impostazione predefinita. Per importare il progetto in TI Code Composer, scegliere Finish (Termina).
4. In Project Explorer, fare doppio clic su aws_demos per rendere il progetto attivo.
5. Da Project (Progetto), scegliere Build Project (Progetto di compilazione) per accertarsi che il progetto venga compilato senza errori o avvisi.

Per eseguire la demo di FreerTOS in TI Code Composer

1. Assicurati che il jumper Sense On Power (SOP) sul tuo Texas Instruments CC322 0SF-LAUNCHXL sia in posizione 0. Per ulteriori informazioni, consulta SimpleLink Wi-Fi CC3x2 0, Guida per l'utente del processore di rete. CC3x3x
2. Usa un cavo USB per collegare Texas Instruments CC322 0SF-LAUNCHXL al computer.
3. Nell'utilità di esplorazione dei progetti, verificare che `CC3220SF.ccxml` sia selezionata come configurazione di destinazione attiva. Per renderla attiva, fare clic con il pulsante destro del mouse sul file e selezionare Set as active target configuration (Imposta come configurazione di destinazione attiva).
4. In TI Code Composer, da Run (Esegui), scegliere Debug (Debug).
5. Quando il debugger si arresta sul punto di interruzione in `main()`, passare al menu Run (Esegui) e scegliere Resume (Riprendi).

Crea ed esegui un progetto demo FreerTOS in IAR Embedded Workbench

Per importare la demo di FreerTOS in IAR Embedded Workbench

1. Aprire IAR Embedded Workbench, scegliere File e quindi scegliere Open Workspace (Apri Workspace).
2. Accedere a `projects/ti/cc3220_launchpad/iar/aws_demos`, scegliere aws_demos.eww e quindi selezionare OK.
3. Fare clic sul nome del progetto (`aws_demos`) con il pulsante destro del mouse e selezionare Make (Crea).

Per eseguire la demo di FreerTOS in IAR Embedded Workbench

1. Assicurati che il jumper Sense On Power (SOP) sul tuo Texas Instruments CC322 0SF-LAUNCHXL sia in posizione 0. Per ulteriori informazioni, consulta SimpleLink Wi-Fi CC3x2 0, Guida per l'utente del processore di rete. CC3x3x
2. Usa un cavo USB per collegare Texas Instruments CC322 0SF-LAUNCHXL al computer.
3. Ricompilare il progetto.

   Per ricostruire il progetto, dal menu Project (Progetto), scegliere Make (Crea).
4. Dal menu Project (Progetto) scegliere Download and Debug (Scarica ed esegui il debug). Puoi ignorare «Avviso: inizializzazione non riuscita» EnergyTrace, se viene visualizzato. Per ulteriori informazioni su EnergyTrace, consulta la sezione Tecnologia MSP. EnergyTrace
5. Quando il debugger si arresta sul punto di interruzione in `main()`, passare al menu Debug e scegliere Go (Procedi).

Utilizzo CMake con FreerTOS

Se preferisci non utilizzare un IDE per lo sviluppo di FreerTOS, puoi in alternativa CMake utilizzarlo per creare ed eseguire le applicazioni demo o le applicazioni che hai sviluppato utilizzando editor di codice e strumenti di debug di terze parti.

Per creare la demo di FreerTOS con CMake

1. Crea una cartella per contenere i file di build generati ()*`build-folder`*.

2. Assicurati che il percorso di ricerca (variabile di ambiente$PATH) contenga la cartella in cui si trova il binario del compilatore TI CGT (ad esempio `C:\ti\ccs910\ccs\tools\compiler\ti-cgt-arm_18.12.2.LTS\bin`).

   Se si sta utilizzando il compilatore TI ARM con la scheda TI, utilizzare il comando seguente per generare i file compilati dal codice sorgente:

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

Per ulteriori informazioni, consulta Utilizzo CMake con FreerTOS.

Monitoraggio dei messaggi MQTT in cloud

È possibile utilizzare il client MQTT nella AWS IoT console per monitorare i messaggi che il dispositivo invia al AWS Cloud.

Per abbonarsi all'argomento MQTT con il client MQTT AWS IoT

1. Accedi alla console AWS IoT.
2. Nel riquadro di navigazione scegliere Test per aprire il client MQTT.
3. In Argomento sottoscrizione, digitare ***your-thing-name*/example/topic**, quindi scegliere Effettua sottoscrizione all'argomento.

Risoluzione dei problemi

Se non vedi messaggi nel client MQTT della AWS IoT console, potresti dover configurare le impostazioni di debug per la scheda.

Per configurare le impostazioni di debug nelle schede TI

1. In Code Composer, su Project Explorer, scegliere aws_demos.
2. Nel menu Run (Esegui), scegliere Debug Configurations (Configurazioni di debug).
3. Nel riquadro di navigazione selezionare aws_demos.
4. Nella scheda Target (Destinazione), in Connection Options (Opzioni di connessione), scegliere Reset the target on a connect (Reimposta la destinazione su una connessione).
5. Seleziona Apply (Applica), quindi seleziona Close (Chiudi).

Se questa procedura non funziona, controlla l'output del programma nel terminale seriale. Dovresti vedere delle stringhe di testo che indicano l'origine del problema.

Per informazioni generali sulla risoluzione dei problemi su Getting Started with FreerTOS, consulta. Nozioni di base sulla risoluzione dei problemi

Nozioni di base su Windows Device Simulator

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Questo tutorial fornisce istruzioni per iniziare a usare FreerTOS Windows Device Simulator.

Prima di iniziare, devi configurare AWS IoT e scaricare FreerTOS per connettere il tuo dispositivo al Cloud. AWS Per istruzioni, consulta Fase iniziale. In questo tutorial, il percorso della directory di download di FreerTOS è indicato come. *`freertos`*

FreerTOS viene rilasciato come file zip che contiene le librerie FreerTOS e le applicazioni di esempio per la piattaforma specificata. Per eseguire gli esempi su un computer Windows, scarica le librerie e gli esempi trasferiti per l'esecuzione su Windows. Questo set di file è il simulatore FreeRTOS per Windows.

> **Note**
>
> Questo tutorial non può essere eseguito correttamente su istanze Amazon EC2 Windows.

Configurazione dell'ambiente di sviluppo

1. Installa l'ultima versione di Win PCap.
2. Installare Microsoft Visual Studio.

   Visual Studio versioni 2017 e 2019 funzionano. Sono supportate tutte le versioni di Visual Studio (Community, Professional o Enterprise).

Oltre all'ambiente di sviluppo integrato (IDE), installare il componente Sviluppo di applicazioni desktop conC++.

Installare la versione più recente di Windows 10 SDK. È possibile scegliere questa versione nella sezione Facoltativo del componente Sviluppo di applicazioni desktop conC++.

3. Verificare di disporre di una connessione Ethernet cablata.
4. (Facoltativo) Se desideri utilizzare il sistema di build CMake basato per creare i tuoi progetti FreerTOS, installa l'ultima versione di. CMake FreerTOS CMake richiede la versione 3.13 o successiva.

Crea ed esegui il progetto demo FreerTOS

Puoi usare Visual Studio o CMake creare progetti FreerTOS.

Creazione ed esecuzione del progetto demo FreerTOS con l'IDE di Visual Studio

1. Caricare il progetto in Visual Studio.

   In Visual Studio, dal menu File scegliere Open (Apri). Scegliere File/Solution (File/Soluzione), accedere al file `projects/pc/windows/visual_studio/aws_demos/aws_demos.sln` e quindi scegliere Open (Apri).
2. Definire una nuova destinazione per il progetto demo.

   Il progetto demo fornito dipende da Windows SDK, ma non ha una versione di Windows SDK specifica. Per impostazione predefinita, l'IDE potrebbe tentare di creare il progetto demo con una versione dell'SDK non presente sul computer in uso. Per impostare la versione di Windows SDK, fare clic con il pulsante destro del mouse su `aws_demos` e quindi scegliere Retarget Projects (Ridestina progetti). Viene visualizzata la finestra Review Solution Actions (Esamina azioni della soluzione). Scegliere una versione di Windows SDK è presente sul computer in uso (è possibile scegliere il valore iniziale nell'elenco a discesa), quindi scegliere OK.
3. Creare ed eseguire il progetto.

   Dal menu Build (Crea), scegliere Build Solution (Crea soluzione) e verificare che la soluzione venga creata senza errori o avvisi. Scegliere Debug, Start Debugging (Avvia debug) per eseguire il progetto. Durante la prima esecuzione è necessario selezionare un'interfaccia di rete.

Creazione ed esecuzione del progetto demo FreerTOS con CMake

Ti consigliamo di utilizzare la CMake GUI anziché lo strumento da CMake riga di comando per creare il progetto demo per Windows Simulator.

Dopo l'installazione CMake, apri la GUI. CMake Su Windows, puoi trovarla dal menu Start sotto CMake, CMake (cmake-gui).

1. Imposta la directory del codice sorgente di FreerTOS.

   Nella GUI, imposta la directory del codice sorgente di FreerTOS *`freertos`* () per Dove è il codice sorgente.

   Impostare *`freertos`*`/build` per Where to build the binaries (Posizione di creazione file binari).
2. Configura il progetto. CMake

   Nella CMake GUI, scegli Aggiungi voce e, nella finestra Aggiungi voce nella cache, imposta i seguenti valori:

   Nome

   AFR_BOARD

   Tipo

   STRING

   Valore

   pc.windows

   Descrizione

   (Facoltativo)
3. Scegli Configura. Se CMake ti chiede di creare la directory di compilazione, scegli Sì, quindi seleziona un generatore in Specificare il generatore per questo progetto. Ti consigliamo di usare Visual Studio come generatore. È tuttavia supportato anche Ninja. Tieni presente che quando utilizzi Visual Studio 2019, la piattaforma deve essere impostata su Win32 al posto di quella predefinita. Lasciare invariate le altre opzioni relative al generatore, quindi scegliere Finish (Fine).
4. Genera e apri il progetto. CMake

Dopo aver configurato il progetto, la CMake GUI mostra tutte le opzioni disponibili per il progetto generato. Ai fini di questo tutorial, è possibile lasciare invariati i valori predefiniti delle opzioni.

Scegliere Generate (Genera) per creare una soluzione Visual Studio e quindi scegliere Open Project (Apri progetto) per aprire il progetto in Visual Studio.

In Visual Studio, fai clic con il pulsante destro del mouse sul `aws_demos` progetto e scegli Imposta come StartUp progetto. In questo modo è possibile creare ed eseguire il progetto. Durante la prima esecuzione è necessario selezionare un'interfaccia di rete.

Per ulteriori informazioni sull'utilizzo CMake con FreerTOS, consulta. Utilizzo CMake con FreerTOS

Configurazione dell'interfaccia di rete

Durante la prima esecuzione del progetto demo, è necessario selezionare l'interfaccia di rete da utilizzare. Il programma enumera le interfacce di rete. Trovare il numero per l'interfaccia Ethernet cablata. L'output dovrebbe essere simile al seguente:

```
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.
```

Dopo aver identificato il numero dell'interfaccia Ethernet cablata, chiudere la finestra dell'applicazione. Nell'esempio, il numero da usare è 1.

Aprire `FreeRTOSConfig.h` e impostare `configNETWORK_INTERFACE_TO_USE` sul numero che corrisponde all'interfaccia di rete cablata.

> **Important**
>
> Sono supportate solo le interfacce Ethernet. La connessione Wi-Fi non è supportata. Per ulteriori informazioni, consulta la voce delle WinPcap domande frequenti Q-16: Da quali adattatori di rete sono supportati? WinPcap .

Monitoraggio dei messaggi MQTT in cloud

Puoi utilizzare il client MQTT nella AWS IoT console per monitorare i messaggi che il tuo dispositivo invia al AWS Cloud.

Per abbonarsi all'argomento MQTT con il client MQTT AWS IoT

1. Accedi alla console AWS IoT.
2. Nel riquadro di navigazione scegliere Test per aprire il client MQTT.
3. In Argomento sottoscrizione, digitare ***your-thing-name*****/example/topic**, quindi scegliere Effettua sottoscrizione all'argomento.

Risoluzione dei problemi

Risoluzione dei problemi comuni in Windows

È possibile che venga generato il seguente errore durante il tentativo di creazione del progetto demo con Visual Studio:

```
Error "The Windows SDK version X.Y was not found" when building the provided Visual
 Studio solution.
```

Il progetto deve basarsi su una versione di Windows SDK presente sul computer in uso.

Per informazioni generali sulla risoluzione dei problemi su Getting Started with FreerTOS, consulta. Nozioni di base sulla risoluzione dei problemi

Guida introduttiva allo Xilinx Avnet MicroZed Industrial IoT Kit

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Questo tutorial fornisce istruzioni per iniziare con lo Xilinx Avnet MicroZed Industrial IoT Kit. Se non disponi del kit Xilinx Avnet MicroZed Industrial IoT, visita AWS il Partner Device Catalog per acquistarne uno dal nostro partner.

Prima di iniziare, devi configurare AWS IoT e scaricare FreerTOS per connettere il tuo dispositivo al Cloud. AWS Per istruzioni, consulta Fase iniziale. In questo tutorial, il percorso della directory di download di FreerTOS è indicato come. *`freertos`*

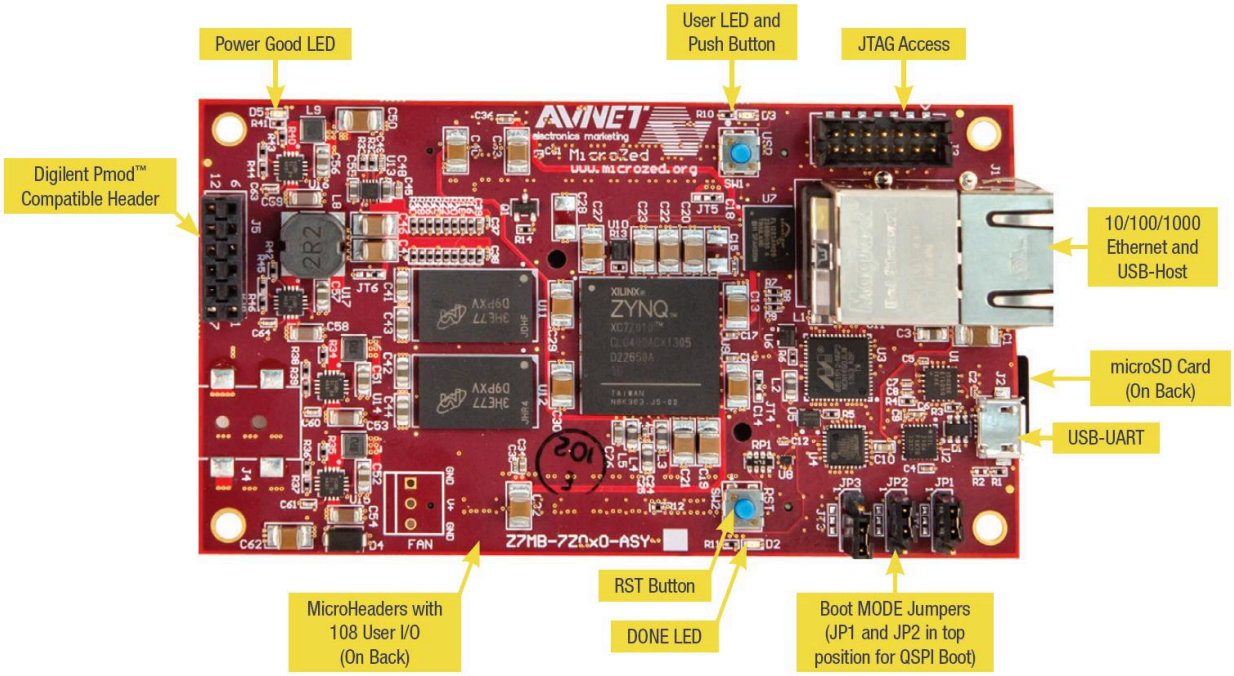
Panoramica

Questo tutorial contiene le istruzioni per i seguenti passaggi iniziali:

1. Connessione della scheda a un computer host.
2. Installazione di software sul computer host per lo sviluppo e il debug di applicazioni integrate per la scheda a microcontroller.
3. Compilazione incrociata di un'applicazione demo FreerTOS con un'immagine binaria.
4. Caricamento dell'immagine binaria dell'applicazione sulla scheda in uso e successiva esecuzione dell'applicazione.

Configura l'hardware MicroZed

Il diagramma seguente può essere utile per la configurazione dell' MicroZed hardware:



Per configurare la scheda MicroZed

1. Connect il computer alla porta USB-UART della scheda. MicroZed
2. Connect il computer alla porta JTAG Access sulla MicroZed scheda.
3. Collega un router o una porta Ethernet connessa a Internet alla porta Ethernet e USB-host sulla scheda. MicroZed

Configurazione dell'ambiente di sviluppo

Per configurare le configurazioni FreerTOS per MicroZed il kit, è necessario utilizzare lo Xilinx Software Development Kit (XSDK). XSDK è supportato su Windows e Linux.

Download e installazione di XSDK

Per installare il software Xilinx, occorre un account Xilinx gratuito.

Per scaricare XSDK

1. Vai alla pagina di download del Software Development Kit Standalone Client. WebInstall
2. Scegliere l'opzione appropriata per il sistema operativo in uso.
3. Viene visualizzata la pagina di accesso Xilinx.

   Se hai un account con Xilinx, inserisci le tue credenziali di accesso e poi scegli Accedi.

Se non si dispone di un account, scegliere Create your account (Crea account). Dopo la registrazione, si riceverà un'e-mail con un collegamento per attivare l'account Xilinx.

4. Nella pagina Name and Address Verification (Verifica di nome e indirizzo), immettere le proprie informazioni e scegliere Next (Avanti). Il download è pronto per iniziare.
5. Salvare il file `Xilinx_SDK_`*`version`*`_`*`os`*.

Per installare XSDK

1. Apri il file `Xilinx_SDK_`*`version`*`_`*`os`*.
2. In Select Edition to Install (Seleziona edizione da installare), scegliere Xilinx Software Development Kit (XSDK) e quindi selezionare Next (Avanti).
3. Nella pagina successiva dell'installazione guidata, in Installation Options (Opzioni di installazione), selezionare Install Cable Drivers (Installa driver del cavo) e quindi scegliere Next (Avanti).

Se il computer non rileva la connessione USB-UART, MicroZed installa manualmente i driver 0x Bridge VCP. CP21 USB-to-UART Per istruzioni, consulta la Guida all'installazione di Silicon Labs 0x. CP21 USB-to-UART

Per ulteriori informazioni su XSDK, consulta la pagina Getting Started with Xilinx SDK sul sito Web Xilink.

Crea ed esegui il progetto demo FreerTOS

Apri la demo di FreerTOS nell'IDE XSDK

1. Avviare l'IDE XSDK con la directory del workspace impostata su *`freertos`*`/projects/xilinx/microzed/xsdk`.
2. Chiudere la pagina di benvenuto. Dal menu, scegliere Project (Progetto), quindi deselezionare Build Automatically (Compila automaticamente).
3. Da menu, scegliere File, quindi selezionare Import (Importa).
4. Nella pagina Select (Seleziona), espandere General (Generale), scegliere Existing Projects into Workspace (Progetti esistenti nel workspace), quindi selezionare Next (Avanti).

5. Nella pagina Importa progetti, scegliere Seleziona directory root, quindi immettere la directory root del progetto demo: *freertos*/projects/xilinx/microzed/xsdk/aws_demos. Per individuare la directory, scegliere Browse (Sfoglia).

   Dopo aver specificato una directory root, i progetti nella directory vengono visualizzati nella pagina Import Projects (Importa progetti). Tutti i progetti disponibili sono selezionati per impostazione predefinita.

   > **Note**
   >
   > Se viene visualizzato un avviso nella parte superiore della pagina Import Projects (Importa progetti) ("Some projects cannot be imported because they already exist in the workspace" (Impossibile importare alcuni progetti perché esistono già nel workspace)), puoi ignorarlo.

6. Con tutti i progetti selezionati, scegliere Finish (Fine).
7. Se non vengono visualizzati i progetti aws_bsp, fsbl e MicroZed_hw_platform_0 nel riquadro dei progetti, ripetere i passaggi precedenti a partire da #3 ma con la directory principale impostata su *freertos*/vendors/xilinx e importare aws_bsp, fsbl e MicroZed_hw_platform_0.
8. Da menu, scegliere Window (Finestra), quindi selezionare Preferences (Preferenze).
9. Nel riquadro di navigazione, espandere Run/Debug (Esegui/Debug), scegliere String Substitution (Sostituzione stringa), quindi selezionare New (Nuovo).
10. In New String Substitution Variable (Nuova variabile di sostituzione stringa), per Name (Nome), immettere **AFR_ROOT**. Per Valore, immettere il percorso root della *freertos*/projects/xilinx/microzed/xsdk/aws_demos. Scegliere OK, quindi selezionare OK per salvare la variabile e chiudere Preferences (Preferenze).

Crea il progetto demo FreerTOS

1. Nell'IDE XSDK, dal menu, scegliere Project (Progetto), quindi selezionare Clean (Pulisci).
2. In Clean (Pulisci), lasciare i valori predefinite delle opzioni, quindi scegliere OK. XSDK pulisce e compila tutti i progetti, quindi genera file .elf.

> **Note**
>
> Per compilare tutti i progetti senza pulirli, scegli Project (Progetto), quindi seleziona Build All (Compila tutto).
> Per compilare singoli progetti, seleziona il progetto da compilare, scegli Project (Progetto), quindi seleziona Build Project (Compila progetto).

Genera l'immagine di avvio per il progetto demo FreerTOS

1. Nell'IDE XSDK, fare clic con il pulsante destro del mouse su aws_demos, quindi scegliere Create Boot Image (Crea immagine di avvio).
2. In Create Boot Image (Crea immagine di avvio), scegliere Create new BIF file (Crea nuovo file BIF).
3. Accanto a Output BIF file path (Percorso file BIF di output), scegliere Browse (Sfoglia), quindi selezionare `aws_demos.bif` situato in *`<freertos>`*`/vendors/xilinx/microzed/aws_demos/aws_demos.bif`.
4. Scegli Aggiungi.
5. In Add new boot image partition (Aggiungi nuova partizione immagine di avvio), accanto a File path (Percorso file), scegliere Browse (Sfoglia), quindi selezionare `fsbl.elf`, situato in `vendors/xilinx/fsbl/Debug/fsbl.elf`.
6. In Partition type (Tipo di partizione), scegliere bootloader, quindi selezionare OK.
7. In Create Boot Image (Crea immagine di avvio), scegliere Create Image (Crea immagine). In Override Files (Ignora file), scegliere OK per sovrascrivere il file `aws_demos.bif` esistente e generare il file `BOOT.bin` in `projects/xilinx/microzed/xsdk/aws_demos/BOOT.bin`.

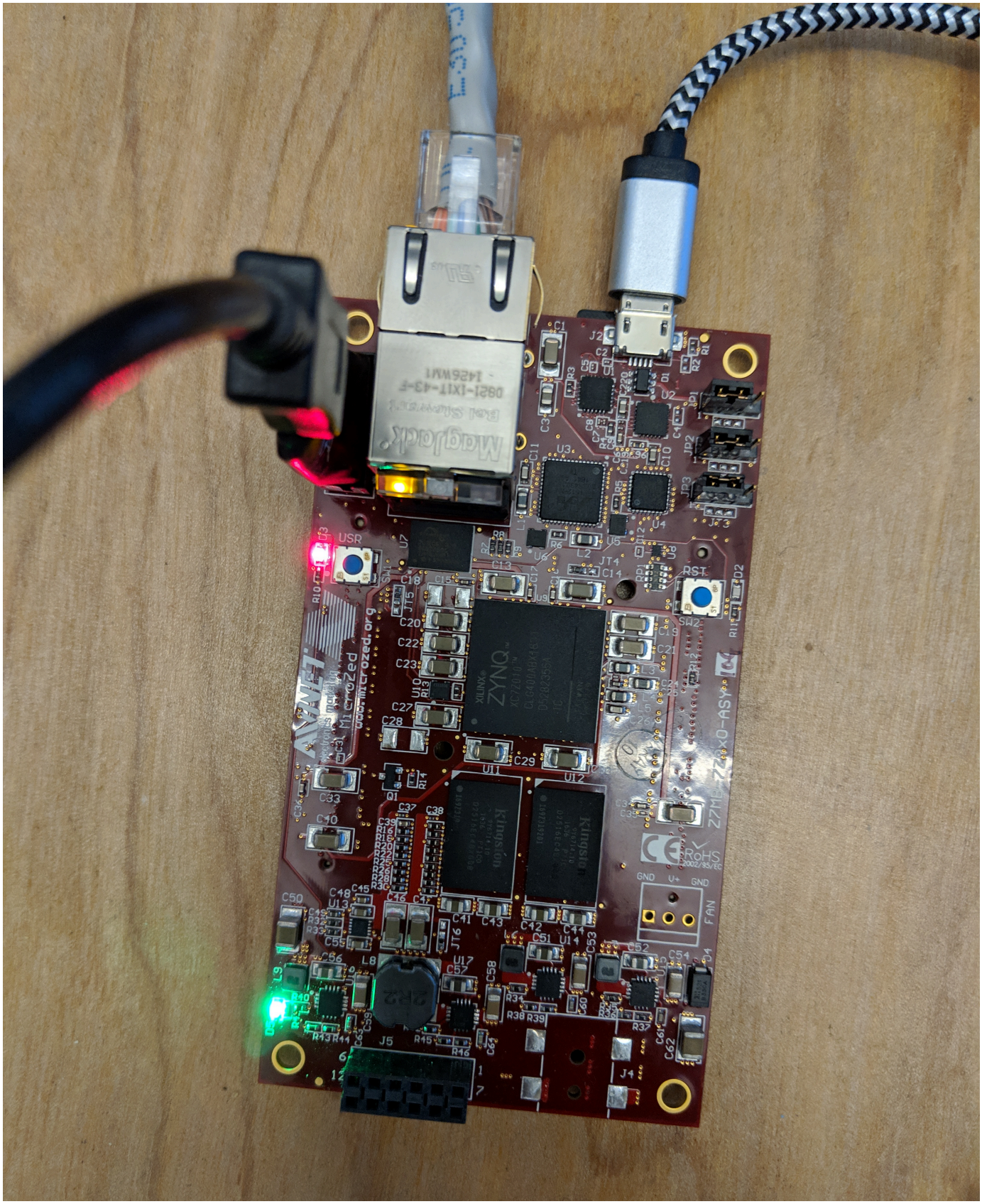
Debug di JTAG

1. Imposta i jumper della modalità di avvio della MicroZed scheda sulla modalità di avvio JTAG.

2. Inserire la scheda MicroSD nello slot della scheda MicroSD che si trova subito sotto la porta USB-UART.

> **Note**
>
> Prima di eseguire il debug, assicurarsi di eseguire il backup dei contenuti disponibili sulla scheda MicroSD.

L'aspetto della scheda è simile al seguente:


USR
RST
ZYNQ
XILINX
Kingston
GND
V+
GND
FAN
RoHS

3. Nell'IDE XSDK, fare clic con il pulsante destro del mouse su aws_demos, scegliere Debug As (Esegui debug come), quindi selezionare 1 Launch on System Hardware (System Debugger) (1 Avvia su hardware di sistema (Debugger di sistema)).
4. Quando il debugger si arresta in corrispondenza del punto di interruzione in `main()`, dal menu, scegliere Run (Esegui), quindi selezionare Resume (Riprendi).

> **Note**
>
> La prima volta che esegui l'applicazione, una nuova coppia certificato-chiave viene importata in memoria non volatile. Per esecuzioni successive, è possibile commentare `vDevModeKeyProvisioning()` nel file `main.c` prima di ricompilare le immagini e il file `BOOT.bin`. In questo modo si impedisce che i certificati e la chiave vengano copiati nello storage ad ogni esecuzione.

Puoi scegliere di avviare la MicroZed scheda da una scheda microSD o da QSPI Flash per eseguire il progetto demo FreerTOS. Per istruzioni, consulta Genera l'immagine di avvio per il progetto demo FreerTOS e Esegui il progetto demo FreerTOS.

Esegui il progetto demo FreerTOS

Per eseguire il progetto demo FreerTOS, puoi avviare MicroZed la scheda da una scheda microSD o da un flash QSPI.

Mentre configuri la MicroZed scheda per l'esecuzione del progetto demo FreerTOS, fai riferimento allo schema in. Configura l'hardware MicroZed Assicurati di aver collegato la MicroZed scheda al computer.

Avvia il progetto FreerTOS da una scheda microSD

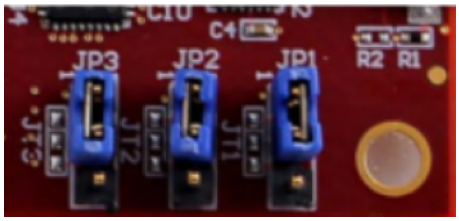
Formatta la scheda microSD fornita con lo MicroZed Xilinx Industrial IoT Kit.

1. Copiare il file `BOOT.bin` nella scheda MicroSD.
2. Inserire la scheda nello slot della scheda MicroSD direttamente sotto la porta USB-UART.
3. Imposta i jumper della modalità di MicroZed avvio sulla modalità di avvio SD.

4. Premere il pulsante RST per ripristinare il dispositivo e avviare la procedura di avvio dell'applicazione. È anche possibile scollegare il cavo USB-UART dalla porta USB-UART e quindi reinserire il cavo.

Avvia il progetto demo FreerTOS da QSPI flash

1. Imposta i jumper della modalità di avvio della MicroZed scheda sulla modalità di avvio JTAG.



2. Verificare che il computer sia connesso alle porte USB-UART e JTAG Access. Il LED Power Good verde deve essere acceso.
3. Nell'IDE XSDK, dal menu, scegliere Xilinx, quindi selezionare Program Flash (Programma flash).
4. In Program Flash Memory (Programma memoria flash), la piattaforma hardware deve essere compilata automaticamente. Per Connection, scegli il tuo server MicroZed hardware per connettere la scheda al computer host.

> **Note**
>
> Se si sta utilizzando il cavo Xilinx Smart Lync JTAG, è necessario creare un server hardware in XSDK IDE. Scegliere New (Nuovo), quindi definire il server.

5. In Image File (File di immagine), immettere il percorso di directory al file di immagine `BOOT.bin`. Scegliere Browse (Sfoglia) per individuare il file.
6. In Offset (Scostamento), immettere **`0x0`**.
7. In FSBL File (File FSBL), immettere il percorso di directory al file `fsbl.elf`. Scegliere Browse (Sfoglia) per individuare il file.
8. Scegliere Program (Programma) per programmare la scheda.

9. Al termine della programmazione QSPI, rimuovere il cavo USB-UART per spegnere la scheda.
10. Imposta i jumper della modalità di avvio della MicroZed scheda sulla modalità di avvio QSPI.
11. Inserire la scheda nello slot della scheda MicroSD che si trova subito sotto la porta USB-UART.

> **Note**
>
> Assicurarsi di eseguire il backup dei contenuti disponibili sulla scheda MicroSD.

12. Premere il pulsante RST per ripristinare il dispositivo e avviare la procedura di avvio dell'applicazione. È anche possibile scollegare il cavo USB-UART dalla porta USB-UART e quindi reinserire il cavo.

Monitoraggio dei messaggi MQTT in cloud

È possibile utilizzare il client MQTT nella AWS IoT console per monitorare i messaggi che il dispositivo invia al Cloud. AWS

Per abbonarsi all'argomento MQTT con il client MQTT AWS IoT

1. Accedi alla console AWS IoT.
2. Nel riquadro di navigazione scegliere Test per aprire il client MQTT.
3. In Argomento sottoscrizione, digitare ***your-thing-name*****/example/topic**, quindi scegliere Effettua sottoscrizione all'argomento.

Risoluzione dei problemi

Se si verificano errori di compilazione correlati a percorsi errati, prova a pulire e ricompilare il progetto, come descritto in Crea il progetto demo FreerTOS.

Se utilizzi il sistema operativo Windows, assicurati di utilizzare le barre quando imposti le variabili di sostituzione stringa nell'IDE XSDK Windows.

Per informazioni generali sulla risoluzione dei problemi su Getting Started with FreerTOS, consulta. Nozioni di base sulla risoluzione dei problemi

Aggiornamenti FreerTOS Over-the-Air

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Over-the-air Gli aggiornamenti (OTA) consentono di distribuire gli aggiornamenti del firmware su uno o più dispositivi del parco dispositivi. Anche se gli aggiornamenti OTA sono stati progettati per aggiornare il firmware dei dispositivi, è possibile utilizzarli per l'invio di qualsiasi file a uno o più dispositivi registrati con AWS IoT. Quando invii aggiornamenti over the air, ti consigliamo di applicare la firma digitale in modo che i dispositivi che li ricevono possano verificare che non siano stati alterati durante il trasferimento.

È possibile utilizzare Code Signing for AWS IoT per firmare i file oppure è possibile firmare i file con i propri strumenti di firma del codice.

Quando crei un aggiornamento OTA, Servizio OTA Update Manager crea un processo AWS IoT per notificare ai dispositivi la disponibilità di un aggiornamento. L'applicazione demo OTA viene eseguita sul dispositivo e crea un'attività FreerTOS che sottoscrive gli argomenti di notifica AWS IoT per i lavori e ascolta i messaggi di aggiornamento. Quando è disponibile un aggiornamento, l'agente OTA pubblica le richieste AWS IoT e riceve gli aggiornamenti utilizzando il protocollo HTTP o MQTT, a seconda delle impostazioni scelte. L'agente OTA controlla la firma digitale dei file scaricati e, se valida, installa l'aggiornamento del firmware. Se non utilizzi l'applicazione demo FreerTOS OTA Update, devi integrarla nella tua applicazione per ottenere Libreria dell'agente OTA la funzionalità di aggiornamento del firmware.

Gli aggiornamenti di over-the-air FreerTOS ti consentono di:

- Aggiungere una firma digitale al firmware prima della distribuzione.
- Distribuire le nuove immagini del firmware a un solo dispositivo, un gruppo di dispositivi oppure all'intero parco istanze.
- Distribuire il firmware ai dispositivi quando vengono aggiunti ai gruppi, reimpostati o sottoposti a nuovo provisioning.

- Verificare l'autenticità e l'integrità del nuovo firmware dopo che è stato distribuito ai dispositivi.
- Monitorare l'avanzamento di una distribuzione.
- Eseguire il debug di una distribuzione non riuscita.

Tagging delle risorse OTA

Per semplificare la gestione delle risorse OTA puoi decidere di assegnare metadati personalizzati ad aggiornamenti e flussi sotto forma di tag. I tag consentono di classificare le AWS IoT risorse in diversi modi (ad esempio, per scopo, proprietario o ambiente). Questa funzione è utile quando si dispone di numerose risorse dello stesso tipo. Puoi identificare velocemente una risorsa in base ai tag a questa assegnati;

Per ulteriori informazioni, consulta Tagging delle risorse AWS IoT.

Prerequisiti per l'aggiornamento OTA

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Per utilizzare gli aggiornamenti over-the-air (OTA), procedi come segue:

- Controllare i Prerequisiti per gli aggiornamenti OTA mediante HTTP o i Prerequisiti per gli aggiornamenti OTA mediante MQTT.
- Crea un bucket Amazon S3 per archiviare l'aggiornamento.
- Creazione di un ruolo del servizio per gli aggiornamenti OTA.
- Creazione di una policy utente OTA.
- Creazione di un certificato di firma del codice.
- Se utilizzi Code Signing per AWS IoT,Concedi l'accesso alla firma del codice per AWS IoT.
- Scarica FreerTOS con la libreria OTA.

Crea un bucket Amazon S3 per archiviare l'aggiornamento

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

I file di aggiornamento OTA sono archiviati in bucket Amazon S3.

Se utilizzi Code Signing for AWS IoT, il comando che usi per creare un processo di firma del codice richiede un bucket di origine (in cui si trova l'immagine del firmware non firmata) e un bucket di destinazione (in cui viene scritta l'immagine del firmware firmata). È possibile specificare lo stesso bucket per l'origine e per la destinazione. I nomi dei file vengono modificati in GUIDs modo che i file originali non vengano sovrascritti.

Come creare un bucket Amazon S3.

1. Accedi alla console Amazon S3 all'indirizzo. https://console.aws.amazon.com/s3/
2. Seleziona Crea bucket.
3. Inserisci il nome del bucket.
4. In Impostazioni Bucket per Blocca accesso pubblico, tieni selezionata l'opzione Blocca tutto l'accesso pubblico per accettare le autorizzazioni predefinite.
5. In Bucket Versioning, seleziona Abilita per mantenere tutte le versioni nello stesso bucket.
6. Seleziona Crea bucket.

Per ulteriori informazioni su Amazon S3, consulta la Guida per l'utente di Amazon Simple Storage Service.

Creazione di un ruolo del servizio per gli aggiornamenti OTA

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Il servizio aggiornamenti OTA assume questo ruolo per creare e gestire i processi di aggiornamento OTA a tuo nome.

Per creare un ruolo del servizio OTA

1. Accedi alla https://console.aws.amazon.com/iam/.
2. Nel riquadro di navigazione scegliere Roles (Ruoli).
3. Scegliere Crea ruolo.
4. In Select type of trusted entity (Seleziona tipo di entità attendibile), scegli AWS Service.
5. Scegli IoT dall'elenco dei AWS servizi.
6. In Select your use case (Seleziona il tuo caso d'uso) seleziona IoT.
7. Scegliere Next: Permissions (Successivo: Autorizzazioni).
8. Scegliere Next: Tags (Successivo: Tag).
9. Scegliere Next:Review (Successivo: Rivedi).
10. Digitare un nome e una descrizione per il ruolo, quindi scegliere Create role (Crea ruolo).

Per ulteriori informazioni sui ruoli IAM, consulta IAM Roles.

Per aggiungere autorizzazioni per gli aggiornamenti OTA al ruolo del servizio OTA

1. Nella casella di ricerca nella pagina della console IAM, inserisci il nome del tuo ruolo, quindi selezionalo dall'elenco.
2. Scegli Collega policy.
3. Nella casella di ricerca, inserisci "AmazonFreeRTOSOTAUpdate«, seleziona `AmazonFreeRTOSOTAUpdate` dall'elenco delle politiche filtrate, quindi scegli Allega policy per allegare la policy al tuo ruolo di servizio.

Per aggiungere le autorizzazioni IAM richieste al tuo ruolo di servizio OTA

1. Nella casella di ricerca sulla pagina della console IAM, inserisci il nome del tuo ruolo, quindi selezionalo dall'elenco.
2. Scegliere Add inline policy (Aggiungi policy inline).
3. Scegliere la scheda JSON.
4. Copiare e incollare il documento della policy seguente nella casella di testo:

```
{
    "Version": "2012-10-17",
    "Statement": [
      {
            "Effect": "Allow",
            "Action": [
                "iam:GetRole",
                "iam:PassRole"
            ],
            "Resource": "arn:aws:iam::your_account_id:role/your_role_name"
      }
    ]
}
```

   Assicurati di sostituirlo *your_account_id* con l'ID AWS del tuo account e *your_role_name* con il nome del ruolo del servizio OTA.
5. Scegli Verifica policy.
6. Immettere un nome per la policy e scegliere Create policy (Crea policy).

> **Note**
>
> La procedura seguente non è richiesta se il nome del bucket Amazon S3 inizia con «afr-ota». In caso affermativo, la policy AWS gestita include `AmazonFreeRTOSOTAUpdate` già le autorizzazioni richieste.

Per aggiungere le autorizzazioni Amazon S3 richieste al tuo ruolo di servizio OTA

1. Nella casella di ricerca sulla pagina della console IAM, inserisci il nome del tuo ruolo, quindi selezionalo dall'elenco.

2. Scegliere Add inline policy (Aggiungi policy inline).
3. Scegliere la scheda JSON.
4. Copiare e incollare il documento della policy seguente nella casella.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObjectVersion",
                "s3:GetObject",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::example-bucket/*"
            ]
        }
    ]
}
```

   Questa politica concede al ruolo di servizio OTA l'autorizzazione a leggere oggetti Amazon S3. Assicurati di sostituirlo *example-bucket* con il nome del tuo bucket.
5. Scegli Verifica policy.
6. Immettere un nome per la policy e scegliere Create policy (Crea policy).

Creazione di una policy utente OTA

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Devi concedere al tuo utente IAM l'autorizzazione per eseguire over-the-air gli aggiornamenti. L'utente deve essere dotato delle autorizzazioni per:

- Accedere al bucket S3 in cui sono archiviati gli aggiornamenti del firmware.
- Accedi ai certificati archiviati in AWS Certificate Manager.
- Accedi alla funzionalità di consegna dei file AWS IoT basata su MQTT.
- Accedi agli aggiornamenti OTA di FreerTOS.
- Accedi ai lavori AWS IoT .
- Accedi a IAM.
- Firma del codice di accesso per AWS IoT. Consultare Concedi l'accesso alla firma del codice per AWS IoT.
- Elenca le piattaforme hardware FreerTOS.
- Etichetta e rimuovi i tag. AWS IoT

Per fornire l'accesso, aggiungi autorizzazioni agli utenti, gruppi o ruoli:

- Utenti e gruppi in AWS IAM Identity Center:

  Crea un set di autorizzazioni. Segui le istruzioni riportate nella pagina Create a permission set (Creazione di un set di autorizzazioni) nella Guida per l'utente di AWS IAM Identity Center .
- Utenti gestiti in IAM tramite un provider di identità:

  Crea un ruolo per la federazione delle identità. Segui le istruzioni riportate nella pagina Create a role for a third-party identity provider (federation) della Guida per l'utente IAM.
- Utenti IAM:
  - Crea un ruolo che l'utente possa assumere. Segui le istruzioni riportate nella pagina Create a role for an IAM user della Guida per l'utente IAM.
  - (Non consigliato) Collega una policy direttamente a un utente o aggiungi un utente a un gruppo di utenti. Segui le istruzioni riportate nella pagina Aggiunta di autorizzazioni a un utente (console) nella Guida per l'utente IAM.

Creazione di un certificato di firma del codice

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Per aggiungere una firma digitale alle immagini del firmware, sono necessari una chiave privata e del certificato di firma del codice. A scopo di test, puoi creare un certificato autofirmato e una chiave privata. Per gli ambienti di produzione, acquista un certificato tramite una nota autorità di certificazione (CA).

Piattaforme diverse richiedono tipi diversi di certificati di firma del codice. Le sezioni seguenti descrivono come creare certificati di firma del codice per diverse piattaforme qualificate per FreeRTOS.

Argomenti

- Creazione di un certificato di firma del codice per Texas Instruments 0SF-LAUNCHXL CC322
- Creazione di un certificato di firma del codice per Microchip Curiosity MZEF PIC32
- Creazione di un certificato di firma del codice per Espressif ESP32
- Creazione di un certificato di firma del codice per Nordic nrf52840-dk
- Creazione di un certificato di firma del codice per il simulatore Windows FreerTOS
- Creazione di un certificato di firma del codice per l'hardware personalizzato

Creazione di un certificato di firma del codice per Texas Instruments 0SF-LAUNCHXL CC322

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Il kit di sviluppo SimpleLink Wi-Fi CC322 0SF Wireless Microcontroller Launchpad supporta due catene di certificati per la firma del codice del firmware:

- Produzione (certificato-catalogo)

  Per utilizzare la catena di certificati di produzione, è necessario acquistare un certificato di firma del codice commerciale e utilizzare lo strumento TI Uniflash per impostare la scheda sulla modalità di produzione.

- Test e sviluppo (certificato-prodotti)

  La catena di certificati Playground ti consente di provare gli aggiornamenti OTA con un certificato di firma del codice autofirmato.

Utilizzalo AWS Command Line Interface per importare il certificato di firma del codice, la chiave privata e la catena di certificati. AWS Certificate Manager Per ulteriori informazioni, consulta Installazione di AWS CLI nella Guida per l'AWS Command Line Interface utente.

Scarica e installa l'ultima versione di SimpleLink CC3220 SDK. Per impostazione predefinita, i file necessari si trovano qui:

`C:\ti\simplelink_cc32xx_sdk_version\tools\cc32xx_tools\certificate-playground` (Windows)

`/Applications/Ti/simplelink_cc32xx_version/tools/cc32xx_tools/certificate-playground` (macOS)

I certificati in SimpleLink CC322 0 SDK sono in formato DER. Per creare un certificato di firma del codice autofirmato, è necessario convertirlo in formato PEM.

Segui questi passaggi per creare un certificato di firma del codice che sia collegato alla gerarchia dei certificati di Texas Instruments Playground e soddisfi i criteri di Code Signing. AWS Certificate Manager AWS IoT

> **Note**
>
> Per creare un certificato di firma del codice, installare OpenSSL sul computer. Dopo aver installato OpenSSL, verifica che `openssl` sia assegnato al file eseguibile OpenSSL nel prompt dei comandi o nell'ambiente del terminale.

Per creare un certificato di firma del codice autofirmato

1. Utilizzando le autorizzazioni di amministratore, aprire un prompt dei comandi o un terminale.
2. Nella directory di lavoro, utilizzare il testo seguente per creare un file denominato `cert_config.txt`. *`test_signer@amazon.com`*Sostituiscilo con il tuo indirizzo email.

   ```
   [ req ]
   prompt             = no
   distinguished_name = my dn

   [ my dn ]
   commonName = test_signer@amazon.com

   [ my_exts ]
   keyUsage         = digitalSignature
   extendedKeyUsage = codeSigning
   ```

3. Creare una chiave privata e una richiesta di firma del certificato (CSR):

   ```
   openssl req -config cert_config.txt -extensions my_exts -nodes -days 365 -newkey
    rsa:2048 -keyout tisigner.key -out tisigner.csr
   ```

4. Convertire la chiave privata CA radice dei prodotti Texas Instruments dal formato DER al formato PEM.

   La chiave privata CA radice dei prodotti TI si trova qui:

   `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. Convertire il certificato CA radice dei prodotti Texas Instruments dal formato DER al formato PEM.

   Il certificato CA radice dei prodotti TI si trova qui:

`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. Firmare la CSR con l'autorità di certificazione radice per 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. Convertire il certificato di firma del codice (`tisigner.crt.pem`) nel formato DER:

```
openssl x509 -in tisigner.crt.pem -out tisigner.crt.der -outform DER
```

> **Note**
>
> È possibile scrivere il certificato `tisigner.crt.der` sulla scheda di sviluppo TI successivamente.

8. Importa il certificato di firma del codice, la chiave privata e la catena di certificati in: 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
```

Questo comando visualizza un ARN per il certificato. L'ARN sarà necessario al momento della creazione di un processo di aggiornamento OTA.

> **Note**
>
> Questo passaggio è stato scritto partendo dal presupposto che utilizzerai Code Signing per firmare AWS IoT le immagini del firmware. Sebbene sia consigliato l'uso di Code Signing for, AWS IoT è possibile firmare le immagini del firmware manualmente.

Creazione di un certificato di firma del codice per Microchip Curiosity MZEF PIC32

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Il Microchip Curiosity PIC32 MZEF supporta un certificato di firma del codice SHA256 autofirmato con ECDSA.

> **Note**
>
> Per creare un certificato di firma del codice, installare OpenSSL sul computer. Dopo aver installato OpenSSL, verifica che `openssl` sia assegnato al file eseguibile OpenSSL nel prompt dei comandi o nell'ambiente del terminale.
> Utilizzalo AWS Command Line Interface per importare il certificato di firma del codice, la chiave privata e la catena di certificati. AWS Certificate Manager Per informazioni sull'installazione di AWS CLI, vedere Installazione di. AWS CLI

1. Nella directory di lavoro, utilizzare il testo seguente per creare un file denominato `cert_config.txt`. Sostituisci *`test_signer@amazon.com`* con il tuo indirizzo email:

```
[ req ]
prompt             = no
distinguished_name = my_dn

[ my_dn ]
commonName = test_signer@amazon.com

[ my_exts ]
keyUsage         = digitalSignature
extendedKeyUsage = codeSigning
```

2. Creare una chiave privata con firma del codice ECDSA:

```
openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-256 -pkeyopt
 ec_param_enc:named_curve -outform PEM -out ecdsasigner.key
```

3. Creare un certificato di firma del codice ECDSA:

```
openssl req -new -x509 -config cert_config.txt -extensions my_exts -nodes -days 365
 -key ecdsasigner.key -out ecdsasigner.crt
```

4. Importa il certificato di firma del codice, la chiave privata e la catena di certificati in: AWS Certificate Manager

```
aws acm import-certificate --certificate fileb://ecdsasigner.crt --private-key
 fileb://ecdsasigner.key
```

Questo comando visualizza un ARN per il certificato. L'ARN sarà necessario al momento della creazione di un processo di aggiornamento OTA.

> **Note**
>
> Questo passaggio è stato scritto partendo dal presupposto che utilizzerai Code Signing per firmare AWS IoT le immagini del firmware. Sebbene sia consigliato l'uso di Code Signing for, AWS IoT è possibile firmare le immagini del firmware manualmente.

Creazione di un certificato di firma del codice per Espressif ESP32

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Le ESP32 schede Espressif supportano un certificato di firma del codice SHA-256 autofirmato con ECDSA.

> **Note**
>
> Per creare un certificato di firma del codice, installare OpenSSL sul computer. Dopo aver installato OpenSSL, verifica che `openssl` sia assegnato al file eseguibile OpenSSL nel prompt dei comandi o nell'ambiente del terminale.
> Utilizzalo per AWS Command Line Interface importare il certificato di firma del codice, la chiave privata e la catena di certificati. AWS Certificate Manager Per informazioni sull'installazione di AWS CLI, vedere Installazione di. AWS CLI

1. Nella directory di lavoro, utilizzare il testo seguente per creare un file denominato `cert_config.txt`. Sostituisci *`test_signer@amazon.com`* con il tuo indirizzo email:

```
[ req ]
prompt             = no
distinguished_name = my_dn

[ my_dn ]
commonName = test_signer@amazon.com

[ my_exts ]
keyUsage         = digitalSignature
extendedKeyUsage = codeSigning
```

2. Creare una chiave privata con firma del codice ECDSA:

```
openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-256 -pkeyopt
 ec_param_enc:named_curve -outform PEM -out ecdsasigner.key
```

3. Creare un certificato di firma del codice ECDSA:

```
openssl req -new -x509 -config cert_config.txt -extensions my_exts -nodes -days 365
 -key ecdsasigner.key -out ecdsasigner.crt
```

4. Importa il certificato di firma del codice, la chiave privata e la catena di certificati in: AWS Certificate Manager

```
aws acm import-certificate --certificate fileb://ecdsasigner.crt --private-key
 fileb://ecdsasigner.key
```

Questo comando visualizza un ARN per il certificato. L'ARN sarà necessario al momento della creazione di un processo di aggiornamento OTA.

> **Note**
>
> Questo passaggio è stato scritto partendo dal presupposto che utilizzerai Code Signing per firmare AWS IoT le immagini del firmware. Sebbene sia consigliato l'uso di Code Signing for, AWS IoT è possibile firmare le immagini del firmware manualmente.

Creazione di un certificato di firma del codice per Nordic nrf52840-dk

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Il Nordic nrf52840-dk supporta un certificato di firma del codice autofirmato con ECDSA. SHA256

> **Note**
>
> Per creare un certificato di firma del codice, installare OpenSSL sul computer. Dopo aver installato OpenSSL, verifica che `openssl` sia assegnato al file eseguibile OpenSSL nel prompt dei comandi o nell'ambiente del terminale.
> Utilizzalo per importare il certificato di firma del codice, la AWS Command Line Interface chiave privata e la catena di certificati. AWS Certificate Manager Per informazioni sull'installazione di AWS CLI, vedere Installazione di. AWS CLI

1. Nella directory di lavoro, utilizzare il testo seguente per creare un file denominato `cert_config.txt`. Sostituisci *`test_signer@amazon.com`* con il tuo indirizzo email:

```
[ req ]
prompt             = no
distinguished_name = my_dn
```

```
[ my_dn ]
commonName = test_signer@amazon.com

[ my_exts ]
keyUsage         = digitalSignature
extendedKeyUsage = codeSigning
```

2. Creare una chiave privata con firma del codice ECDSA:

```
openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-256 -pkeyopt
 ec_param_enc:named_curve -outform PEM -out ecdsasigner.key
```

3. Creare un certificato di firma del codice ECDSA:

```
openssl req -new -x509 -config cert_config.txt -extensions my_exts -nodes -days 365
 -key ecdsasigner.key -out ecdsasigner.crt
```

4. Importa il certificato di firma del codice, la chiave privata e la catena di certificati in: AWS Certificate Manager

```
aws acm import-certificate --certificate fileb://ecdsasigner.crt --private-key
 fileb://ecdsasigner.key
```

Questo comando visualizza un ARN per il certificato. L'ARN sarà necessario al momento della creazione di un processo di aggiornamento OTA.

> **Note**
>
> Questo passaggio è stato scritto partendo dal presupposto che utilizzerai Code Signing per firmare AWS IoT le immagini del firmware. Sebbene sia consigliato l'uso di Code Signing for, AWS IoT è possibile firmare le immagini del firmware manualmente.

Creazione di un certificato di firma del codice per il simulatore Windows FreerTOS

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Il simulatore Windows FreerTOS richiede un certificato di firma del codice con una chiave ECDSA P-256 e hash SHA-256 per eseguire gli aggiornamenti OTA. Se non disponi di un certificato di firma del codice, segui queste fasi per crearne uno.

> **Note**
>
> Per creare un certificato di firma del codice, installare OpenSSL sul computer. Dopo aver installato OpenSSL, verifica che `openssl` sia assegnato al file eseguibile OpenSSL nel prompt dei comandi o nell'ambiente del terminale.
> Usa il per importare il certificato di firma del codice AWS Command Line Interface , la chiave privata e la catena di certificati in. AWS Certificate Manager Per informazioni sull'installazione di AWS CLI, vedere Installazione di. AWS CLI

1. Nella directory di lavoro, utilizzare il testo seguente per creare un file denominato `cert_config.txt`. Sostituisci *`test_signer@amazon.com`* con il tuo indirizzo email:

```
[ req ]
prompt             = no
distinguished_name = my_dn

[ my_dn ]
commonName = test_signer@amazon.com

[ my_exts ]
keyUsage         = digitalSignature
extendedKeyUsage = codeSigning
```

2. Creare una chiave privata con firma del codice ECDSA:

```
openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-256 -pkeyopt
 ec_param_enc:named_curve -outform PEM -out ecdsasigner.key
```

3. Creare un certificato di firma del codice ECDSA:

```
openssl req -new -x509 -config cert_config.txt -extensions my_exts -nodes -days 365
 -key ecdsasigner.key -out ecdsasigner.crt
```

4. Importa il certificato di firma del codice, la chiave privata e la catena di certificati in: AWS Certificate Manager

```
aws acm import-certificate --certificate fileb://ecdsasigner.crt --private-key
 fileb://ecdsasigner.key
```

Questo comando visualizza un ARN per il certificato. L'ARN sarà necessario al momento della creazione di un processo di aggiornamento OTA.

> **Note**
>
> Questo passaggio è stato scritto partendo dal presupposto che utilizzerai Code Signing per firmare AWS IoT le immagini del firmware. Sebbene sia consigliato l'uso di Code Signing for, AWS IoT è possibile firmare le immagini del firmware manualmente.

Creazione di un certificato di firma del codice per l'hardware personalizzato

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Utilizzando un set di strumenti appropriato, è possibile creare una chiave privata e del certificato autofirmato per l'hardware.

Utilizzalo AWS Command Line Interface per importare il certificato di firma del codice, la chiave privata e la catena di certificati. AWS Certificate Manager Per informazioni sull'installazione di AWS CLI, vedere Installazione di. AWS CLI

Dopo aver creato il certificato di firma del codice, puoi utilizzarlo AWS CLI per importarlo in ACM:

```
aws acm import-certificate --certificate fileb://code-sign.crt --private-key fileb://
code-sign.key
```

L'output di questo comando visualizza un ARN per il certificato. L'ARN sarà necessario al momento della creazione di un processo di aggiornamento OTA.

ACM richiede certificati per utilizzare algoritmi e dimensioni di chiave specifici. Per ulteriori informazioni, consulta la sezione Prerequisiti per l'importazione di certificati. Per ulteriori informazioni su ACM, consulta Importazione di certificati in. AWS Certificate Manager

Devi copiare, incollare e formattare il contenuto del tuo certificato di firma del codice nel `aws_ota_codesigner_certificate.h` file che fa parte del codice FreerTOS che scaricherai in seguito.

Concedi l'accesso alla firma del codice per AWS IoT

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Negli ambienti di produzione è necessario aggiungere una firma digitale all'aggiornamento del firmware per garantire l'autenticità e l'integrità dell'aggiornamento. Puoi firmare l'aggiornamento manualmente oppure utilizzare Code Signing AWS IoT per firmare il codice. Per utilizzare Code Signing for FreerTOS, devi concedere al tuo account utente IAM l'accesso a Code Signing for FreerTOS.

Per fornire l'accesso, aggiungi autorizzazioni agli utenti, gruppi o ruoli:

- Utenti e gruppi in: AWS IAM Identity Center

Crea un set di autorizzazioni. Segui le istruzioni riportate nella pagina Create a permission set (Creazione di un set di autorizzazioni) nella Guida per l'utente di AWS IAM Identity Center .

- Utenti gestiti in IAM tramite un provider di identità:

  Crea un ruolo per la federazione delle identità. Segui le istruzioni riportate nella pagina Create a role for a third-party identity provider (federation) della Guida per l'utente IAM.

- Utenti IAM:
  - Crea un ruolo che l'utente possa assumere. Segui le istruzioni riportate nella pagina Create a role for an IAM user della Guida per l'utente IAM.
  - (Non consigliato) Collega una policy direttamente a un utente o aggiungi un utente a un gruppo di utenti. Segui le istruzioni riportate nella pagina Aggiunta di autorizzazioni a un utente (console) nella Guida per l'utente IAM.

Scarica FreerTOS con la libreria OTA

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Puoi clonare o scaricare FreerTOS da. GitHub Consultare il file README.md per le istruzioni.

Per ulteriori informazioni sulla configurazione e l'esecuzione dell'applicazione demo OTA, consulta Over-the-air aggiorna l'applicazione demo.

Prerequisiti per gli aggiornamenti OTA mediante MQTT

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Questa sezione descrive i requisiti generali per l'utilizzo di MQTT to Perform over-the-air (aggiornamenti OTA).

Per fornire l'accesso, aggiungi autorizzazioni agli utenti, gruppi o ruoli:

- Utenti e gruppi in AWS IAM Identity Center:

  Crea un set di autorizzazioni. Segui le istruzioni riportate nella pagina Create a permission set (Creazione di un set di autorizzazioni) nella Guida per l'utente di AWS IAM Identity Center .
- Utenti gestiti in IAM tramite un provider di identità:

  Crea un ruolo per la federazione delle identità. Segui le istruzioni riportate nella pagina Create a role for a third-party identity provider (federation) della Guida per l'utente IAM.
- Utenti IAM:
  - Crea un ruolo che l'utente possa assumere. Segui le istruzioni riportate nella pagina Create a role for an IAM user della Guida per l'utente IAM.
  - (Non consigliato) Collega una policy direttamente a un utente o aggiungi un utente a un gruppo di utenti. Segui le istruzioni riportate nella pagina Aggiunta di autorizzazioni a un utente (console) nella Guida per l'utente IAM.

Requisiti minimi

- Il firmware del dispositivo deve includere le librerie FreerTOS necessarie (CoreMQTT, OTA Agent e le relative dipendenze).
- È richiesta la versione 1.4.0 o successiva di FreerTOS. Tuttavia, si consiglia di utilizzare la versione più recente quando possibile.

Configurazioni

A partire dalla versione 201912.00, FreerTOS OTA può utilizzare il protocollo HTTP o MQTT per trasferire le immagini di aggiornamento del firmware dai dispositivi. AWS IoT Se si specificano entrambi i protocolli quando si crea un aggiornamento OTA in FreerTOS, ogni dispositivo determinerà il protocollo utilizzato per trasferire l'immagine. Per ulteriori informazioni, consulta Prerequisiti per gli aggiornamenti OTA mediante HTTP.

Per impostazione predefinita, la configurazione dei protocolli OTA in `aws_ota_agent_config.h` consiste nell'utilizzare il protocollo 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 )
```

Configurazioni specifiche del dispositivo

Nessuna.

Utilizzo della memoria

Quando MQTT viene utilizzato per il trasferimento dei dati, non è necessaria alcuna memoria aggiuntiva per la connessione MQTT perché è condivisa tra operazioni di controllo e dati.

Policy dei dispositivi

Ogni dispositivo che riceve un aggiornamento OTA tramite MQTT deve essere registrato come oggetto in AWS IoT e deve avere una politica allegata come quella elencata qui. Ulteriori informazioni sugli elementi degli oggetti `"Action"` e `"Resource"` sono disponibili in Operazioni di policy AWS IoT Core e Risorse per operazioni AWS IoT Core.

```
{
    "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/*"
            ]
        }
    ]
}
```

Note

- Le `iot:Connect` autorizzazioni consentono al dispositivo di connettersi tramite MQTT AWS IoT .
- Le `iot:Publish` autorizzazioni `iot:Subscribe` e sugli argomenti dei AWS IoT lavori (`.../jobs/*`) consentono al dispositivo connesso di ricevere notifiche e documenti relativi ai lavori e di pubblicare lo stato di completamento dell'esecuzione di un lavoro.
- Le `iot:Publish` autorizzazioni `iot:Subscribe` e le autorizzazioni sugli argomenti degli stream AWS IoT OTA (`.../streams/*`) consentono al dispositivo connesso di recuperare i dati di aggiornamento OTA da. AWS IoT Queste autorizzazioni sono necessarie per eseguire gli aggiornamenti del firmware tramite MQTT.
- Le `iot:Receive` autorizzazioni consentono di AWS IoT Core pubblicare messaggi su tali argomenti sul dispositivo connesso. Questa autorizzazione viene controllata ad ogni recapito di un messaggio MQTT. È possibile utilizzare questa autorizzazione per revocare l'accesso ai client attualmente sottoscritti a un argomento.

Prerequisiti per gli aggiornamenti OTA mediante HTTP

Important

Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Questa sezione descrive i requisiti generali per l'utilizzo di HTTP to perform updates over-the-air (OTA). A partire dalla versione 201912.00, FreerTOS OTA può utilizzare il protocollo HTTP o MQTT per trasferire le immagini di aggiornamento del firmware dai dispositivi. AWS IoT

Note

- Sebbene il protocollo HTTP possa essere utilizzato per trasferire l'immagine del firmware, la libreria CoreMQTT è comunque necessaria perché altre interazioni con la libreria CoreMQTT, tra cui AWS IoT Core l'invio o la ricezione di notifiche di esecuzione del lavoro, documenti di lavoro e aggiornamenti dello stato di esecuzione.
- Quando si specificano entrambi i protocolli MQTT e HTTP per i processi attività di aggiornamento OTA, l'installazione del software dell'agente OTA su ogni singolo dispositivo

determina il protocollo utilizzato per trasferire l'immagine del firmware. Per modificare l'agente OTA dal metodo di protocollo predefinito MQTT al protocollo HTTP, puoi cambiare i file di intestazione utilizzati per compilare il codice sorgente FreeRTOS per il dispositivo.

Requisiti minimi

- Il firmware del dispositivo deve includere le librerie FreerTOS necessarie (CoreMQTT, HTTP, OTA Agent e le relative dipendenze).
- È necessaria la versione FreerTOS 201912.00 o successiva per modificare la configurazione dei protocolli OTA per abilitare il trasferimento di dati OTA tramite HTTP.

Configurazioni

Vedi la seguente configurazione dei protocolli OTA nel file `\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 )
```

Per abilitare il trasferimento dei dati OTA su HTTP

1. Passare da `configENABLED_DATA_PROTOCOLS` a `OTA_DATA_OVER_HTTP`.
2. Quando OTA esegue l'aggiornamento, è possibile specificare entrambi i protocolli in modo che sia possibile utilizzare il protocollo MQTT o HTTP. È possibile impostare il protocollo principale utilizzato dal dispositivo su HTTP modificando `configOTA_PRIMARY_DATA_PROTOCOL` in `OTA_DATA_OVER_HTTP`.

> **Note**
>
> HTTP è supportato solo per le operazioni di dati OTA. Per le operazioni di controllo, è necessario utilizzare MQTT.

Configurazioni specifiche del dispositivo

ESP32

A causa di una quantità limitata di RAM, è necessario disattivare BLE quando si attiva HTTP come protocollo per i dati OTA. Nel file `vendors/espressif/boards/esp32/aws_demos/config_files/aws_iot_network_config.h`, modificare `configENABLED_NETWORKS` solo in `AWSIOT_NETWORK_TYPE_WIFI`.

```
/**
     * @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 )
```

Utilizzo della memoria

Quando MQTT viene utilizzato per il trasferimento dei dati, non è necessaria alcuna memoria heap aggiuntiva per la connessione MQTT perché è condivisa tra operazioni di controllo e dati. Tuttavia, l'attivazione dei dati su HTTP richiede memoria heap aggiuntiva. Di seguito sono riportati i dati sull'utilizzo della memoria heap per tutte le piattaforme supportate, calcolati utilizzando l'API FreerTOS`xPortGetFreeHeapSize`. È necessario assicurarsi che ci sia RAM sufficiente per utilizzare la libreria OTA.

Texas Instruments 0SF-LAUNCHXL CC322

- Operazioni di controllo (MQTT): 12 KB
- Operazioni dei dati (HTTP): 10 KB

> **Note**
>
> TI utilizza significativamente meno RAM perché esegue SSL sull'hardware, quindi non utilizza la libreria mbedtls.

Microchip PIC32 Curiosity MZEF

- Operazioni di controllo (MQTT): 65 KB
- Operazioni dei dati (HTTP): 43 KB

Espresso ESP32

- Operazioni di controllo (MQTT): 65 KB
- Operazioni dei dati (HTTP): 45 KB

> **Note**
>
> BLE on ESP32 richiede circa 87 KB di RAM. La RAM non è sufficiente per abilitarli tutti, in base alle indicazioni precedenti delle configurazioni specifiche del dispositivo.

Simulatore Windows

Operazioni di controllo (MQTT): 82 KB

Operazioni dei dati (HTTP): 63 KB

Nordic nrf52840-dk

HTTP non è supportato.

Policy dei dispositivi

Questa policy consente di utilizzare MQTT o HTTP per gli aggiornamenti OTA.

Ogni dispositivo che riceve un aggiornamento OTA utilizzando HTTP deve essere registrato come una cosa in AWS IoT e la cosa deve avere una policy collegata come indicato qui. Ulteriori informazioni sugli elementi degli oggetti `"Action"` e `"Resource"` sono disponibili in Operazioni di policy AWS IoT Core e Risorse per operazioni AWS IoT Core.

```
{
    "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/*"
            ]
        }
    ]
}
```

Note

- Le autorizzazioni `iot:Connect` consentono al dispositivo di connettersi a AWS IoT tramite MQTT.
- Le `iot:Publish` autorizzazioni `iot:Subscribe` e le autorizzazioni relative agli argomenti dei AWS IoT lavori (`.../jobs/*`) consentono al dispositivo connesso di ricevere notifiche e documenti relativi ai lavori e di pubblicare lo stato di completamento dell'esecuzione di un lavoro.
- Le `iot:Receive` autorizzazioni consentono AWS IoT Core di pubblicare messaggi su tali argomenti sul dispositivo attualmente connesso. Questa autorizzazione viene controllata ad ogni recapito di un messaggio MQTT. È possibile utilizzare questa autorizzazione per revocare l'accesso ai client attualmente sottoscritti a un argomento.

Tutorial OTA

Important

Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Questa sezione contiene un tutorial per l'aggiornamento del firmware sui dispositivi che eseguono FreerTOS utilizzando gli aggiornamenti OTA. Oltre alle immagini del firmware, è possibile utilizzare un aggiornamento OTA per inviare qualsiasi tipo di file a un dispositivo collegato AWS IoT.

È possibile utilizzare la AWS IoT console o AWS CLI creare un aggiornamento OTA. La console è il modo più semplice per iniziare a utilizzare il sistema OTA perché gran parte del lavoro viene eseguito automaticamente. AWS CLI È utile quando si automatizzano i lavori di aggiornamento OTA, si lavora con un gran numero di dispositivi o si utilizzano dispositivi che non sono stati qualificati per FreerTOS. Per ulteriori informazioni sui dispositivi idonei per FreerTOS, consulta il sito web FreeRTOS Partners.

Per creare un aggiornamento OTA.

1. Distribuire una versione iniziale del firmware in uno o più dispositivi.
2. Verificare che il firmware venga eseguito correttamente.
3. Quando viene richiesto un aggiornamento del firmware, apportare le modifiche al codice e creare la nuova immagine.
4. Se firmi manualmente il firmware, firma e carica l'immagine del firmware firmata nel tuo bucket Amazon S3. Se utilizzi Code Signing for AWS IoT, carica l'immagine del firmware non firmata in un bucket Amazon S3.
5. Creare un aggiornamento OTA.

Quando si crea un aggiornamento OTA, si specifica il protocollo di recapito immagini (MQTT o HTTP) o si specificano entrambi per consentire al dispositivo di scegliere. L'agente OTA FreerTOS sul dispositivo riceve l'immagine del firmware aggiornata e verifica la firma digitale, il checksum e il numero di versione della nuova immagine. Se l'aggiornamento del firmware viene verificato, il dispositivo viene reimpostato e, in base alla logica definita dall'applicazione, esegue il commit dell'aggiornamento. Se i tuoi dispositivi non eseguono FreerTOS, devi implementare un agente OTA che funzioni sui tuoi dispositivi.

Installazione del firmware iniziale

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Per aggiornare il firmware, è necessario installare una versione iniziale del firmware che utilizza la libreria dell'agente OTA per restare in ascolto dei processi di aggiornamento OTA. Se non

stai usando FreerTOS, salta questo passaggio. È invece necessario copiare l'implementazione dell'agente OTA sui dispositivi.

Argomenti

- Installa la versione iniziale del firmware su Texas Instruments 0SF-LAUNCHXL CC322
- Installa la versione iniziale del firmware su Microchip PIC32 Curiosity MZEF
- Installa la versione iniziale del firmware su Espressif ESP32
- Installa la versione iniziale del firmware su Nordic n RF5284 0 DK
- Firmware iniziale sul simulatore Windows
- Installare la versione iniziale del firmware su una scheda personalizzata

Installa la versione iniziale del firmware su Texas Instruments 0SF-LAUNCHXL CC322

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Queste fasi presuppongono che sia già stato creato il progetto `aws_demos`, come descritto in Scarica, crea, esegui il flashing ed esegui la demo OTA di FreerTOS su Texas Instruments 0SF-LAUNCHXL CC322.

1. Sul tuo Texas Instruments CC322 0SF-LAUNCHXL, posiziona il jumper SOP sul set centrale di pin (posizione = 1) e resetta la scheda.
2. Scaricare e installare lo strumento Uniflash di TI.
3. Avviare Uniflash. Dall'elenco delle configurazioni, scegli 0SF-LAUNCHXL, quindi scegli CC322 Avvia Image Creator.
4. Scegli Nuovo progetto.
5. Nella pagina Start new project (Inizia nuovo progetto) immettere un nome per il progetto. Per Tipo di dispositivo, scegli 0SF. CC322 Per Device Mode (Modalità dispositivo), scegliere Develop (Sviluppo). Scegli Crea progetto.
6. Disconnettere l'emulatore di terminale.
7. A destra nella finestra dell'applicazione Uniflash scegliere Connect (Connetti).

8. In Advanced (Avanzato), Files (File), selezionare User Files (File utente).
9. Nel riquadro del selettore File scegliere l'icona Add File (Aggiungi file) .
10. Passare alla directory `/Applications/Ti/simplelink_cc32xx_sdk_`*`version`*`/tools/cc32xx_tools/certificate-playground`, selezionare `dummy-root-ca-cert`, scegliere Open (Apri), quindi scegliere Write (Scrivi).
11. Nel riquadro del selettore File scegliere l'icona Add File (Aggiungi file) .
12. Passare alla directory di lavoro in cui sono stati creati la chiave privata e del certificato di firma del codice, scegliere `tisigner.crt.der`, scegliere Open (Apri), quindi scegliere Write (Scrivi).
13. Nell'elenco a discesa Action (Operazione) scegliere Select MCU Image (Seleziona immagine MCU), quindi scegliere Browse (Sfoglia) per scegliere l'immagine del firmware da scrivere nel dispositivo (aws_demos.bin). Questo file si trova nella directory *`freertos`*`/vendors/ti/boards/cc3220_launchpad/aws_demos/ccs/Debug`. Seleziona Apri.
    a. Nella finestra di dialogo, verificare che il nome file sia impostato su `mcuflashimg.bin`.
    b. Selezionare la casella di controllo Vendor (Fornitore).
    c. In File Token (Token file) digitare **`1952007250`**.
    d. In Private Key File Name (Nome file chiave privata) scegliere Browse (Sfoglia), quindi selezionare `tisigner.key` nella directory di lavoro in cui sono stati creati la chiave privata e il certificato di firma del codice.
    e. In Certification File Name (Nome file certificato) scegliere `tisigner.crt.der`.
    f. Scegliere Write (Scrivi).
14. Nel riquadro sinistro, in Files (File), scegliere Service Pack.
15. In Service Pack File Name (Nome file Service Pack) selezionare Browse (Sfoglia), passare a `simplelink_cc32x_sdk_`*`version`*`/tools/cc32xx_tools/servicepack-cc3x20`, scegliere `sp_3.7.0.1_2.0.0.0_2.2.0.6.bin`, quindi selezionare Open (Apri).
16. Nel riquadro sinistro, in Files (File), scegliere Trusted Root-Certificate Catalog (Radice attendibile-Catalogo certificati).
17. Deselezionare la casella di controllo Use default Trusted Root-Certificate Catalog (Usa radice attendibile predefinita-Catalogo certificati).
18. In File di origine, scegli Sfoglia, scegli ***`version`***simplelink_cc32xx_sdk_/20160911.lst, quindi scegli Apri. tools/cc32xx_tools/certificate-playground/certcatalogPlayGround

19. In Signature Source File, scegliete Sfoglia, scegliete simplelink_cc32xx_sdk_/20160911.lst.signed_3220.bin, quindi scegliete Apri. ***version*** tools/cc32xx_tools/certificate-playground/certcatalogPlayGround
20. Scegliere il pulsante

    per salvare il progetto.
21. Scegliere il pulsante

    .
22. Scegliere Program Image (Create and Program) (Immagine programma) ((Crea e programma)).
23. Una volta completato il processo di programmazione, posizionare il jumper SOP nel primo set di pin (posizione = 0), reimpostare la scheda e riconnettere l'emulatore di terminale per verificare che l'output sia lo stesso di quando è stato eseguito il debug della demo con Code Composer Studio. Prendere nota del numero di versione dell'applicazione nell'output del terminale. Servirà in seguito per verificare che il firmware sia stato aggiornato da un aggiornamento OTA.

    L'emulatore di terminale dovrebbe visualizzare un output simile al seguente:

```
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
```

Installa la versione iniziale del firmware su Microchip PIC32 Curiosity MZEF

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Queste fasi presuppongono che sia già stato creato il progetto `aws_demos`, come descritto in Scarica, crea, esegui il flashing ed esegui la demo OTA di FreerTOS su Microchip Curiosity MZEF PIC32.

Per masterizzare l'applicazione dimostrativa nella scheda

1. Creare nuovamente il progetto `aws_demos` e assicurarsi che l'operazione venga completata senza errori.
2. Sulla barra degli strumenti scegliere

   .
3. Al termine del processo di programmazione, scollegare il debugger ICD 4 e reimpostare la scheda. Riconnettere l'emulatore di terminale per accertarsi che l'output sia lo stesso di quando è stato eseguito il debug della demo con il modulo MPLAB X IDE.

   L'emulatore di terminale dovrebbe visualizzare un output simile al seguente:

   ```
   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
```

La procedura seguente consente di creare un file esadecimale unificato o un'immagine di fabbrica costituita da un bootloader di riferimento e da un'applicazione con una firma crittografata. Il bootloader consente di verificare la firma crittografata dell'applicazione all'avvio e supporta gli aggiornamenti OTA.

Per creare e memorizzare nella flash un'immagine di fabbrica

1. Assicurati di avere SRecord gli strumenti installati da Source Forge. Verificare che la directory contenente i programmi `srec_cat` e `srec_info` si trovi nel percorso di sistema.
2. Aggiornare il numero di sequenza OTA e la versione dell'applicazione per l'immagine di fabbrica.
3. Compilare il progetto `aws_demos`.
4. Eseguire lo script `factory_image_generator.py` per generare l'immagine di fabbrica.

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

   Questo comando accetta i parametri seguenti:

   - `mplab.production.bin`: file binario dell'applicazione.
   - `MCHP-Curiosity-PIC32MZEF`: nome della piattaforma.
   - `private_key.pem`: chiave privata di firma del codice.
   - `aws_bootloader.X.production.hex`: file esadecimale del bootloader.

Quando viene creato il progetto `aws_demos`, l'immagine binaria dell'applicazione e il file esadecimale del bootloader vengono creati come parte del processo. Ogni progetto nella directory `vendors/microchip/boards/curiosity_pic32mzef/aws_demos/` contiene una directory `dist/pic32mz_ef_curiosity/production/` che include questi file. Il file esadecimale unificato generato è denominato `mplab.production.factory.unified.hex`.

5. Utilizzate lo strumento MPLab IPE per programmare il file esadecimale generato sul dispositivo.
6. È possibile verificare che l'immagine di fabbrica funzioni guardando l'output UART della scheda quando l'immagine viene caricata. Se le impostazioni sono state eseguite correttamente, dovrebbe essere possibile avviare l'immagine:

```
[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 i certificati non vengono configurati correttamente, o se un'immagine OTA non viene firmata in modo appropriato, è possibile che vengano visualizzati messaggi come i seguenti prima che il bootloader del chip cancelli l'aggiornamento non valido. Verificare che i certificati di firma del codice siano coerenti e rivedere con attenzione le fasi precedenti.

```
[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
```

Installa la versione iniziale del firmware su Espressif ESP32

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Questa guida è stata scritta partendo dal presupposto che tu abbia già eseguito i passaggi descritti in Getting Started with Espressif ESP32 - DevKit C e and Update Prerequisiti. ESP-WROVER-KIT Over-the-Air Prima di provare un aggiornamento OTA, potresti voler eseguire il progetto dimostrativo MQTT descritto in Guida introduttiva a FreerTOS per assicurarti che la scheda e la catena di strumenti siano configurate correttamente.

Per eseguire il flashing dell'immagine produttore computer sulla scheda

1. Aprire *freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h, commentare `#define CONFIG_MQTT_DEMO_ENABLED` e definire `CONFIG_OTA_UPDATE_DEMO_ENABLED`.
2. Copiare il certificato di firma del codice in formato PEM SHA-256/ECDSA generato in Prerequisiti per l'aggiornamento OTA in demos/include/aws_ota_codesigner_certificate.h. Deve essere formattato nel modo seguente:

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

3. Con la demo di OTA Update selezionata, segui gli stessi passaggi descritti in Getting Started with ESP32 per creare e flashare l'immagine. Se precedentemente il progetto è stato creato e memorizzato nella flash, potrebbe essere necessario eseguire prima `make clean`. Dopo avere eseguito `make flash monitor`, dovrebbe essere visualizzato un output simile al seguente. L'ordine di alcuni messaggi potrebbe variare, in quanto l'applicazione dimostrativa esegue contemporaneamente più task:

   ```
   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. Il ESP32 consiglio sta ora ascoltando gli aggiornamenti OTA. Il monitor di ESP-IDF viene avviato dal comando `make flash monitor`. È possibile premere Ctrl+] per uscire. È anche possibile usare un programma terminale TTY preferito (ad esempio PuTTY, Tera Term o GNU Screen) per rimanere in ascolto dell'output seriale della scheda. La connessione alla porta seriale della scheda potrebbe comportarne il riavvio.

Installa la versione iniziale del firmware su Nordic n RF5284 0 DK

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Questa guida è stata scritta partendo dal presupposto che tu abbia già eseguito i passaggi indicati in Guida introduttiva al Nordic n 0-DK RF5284 e Over-the-Air Update Prerequisites. Prima di provare un aggiornamento OTA, potresti voler eseguire il progetto dimostrativo MQTT descritto in Guida introduttiva a FreerTOS per assicurarti che la scheda e la toolchain siano configurate correttamente.

Per eseguire il flashing dell'immagine produttore computer sulla scheda

1. Aprire *freertos*/vendors/nordic/boards/nrf52840-dk/aws_demos/config_files/aws_demo_config.h.
2. Sostituisci `#define CONFIG_MQTT_DEMO_ENABLED` con `#define democonfigOTA_UPDATE_DEMO_ENABLED`.
3. Con la demo dell'aggiornamento OTA selezionata, seguire le stesse fasi descritte nella pagina Guida introduttiva al Nordic n 0-DK RF5284 per compilare ed eseguire il flashing dell'immagine.

   Verrà visualizzato un output simile al seguente.

```
9 1285 [OTA Task] [prvSubscribeToJobNotificationTopics] OK: $aws/things/your-thing-name/jobs/notify-next
10 1286 [OTA Task] [OTA_CheckForUpdate] Request #0
11 1289 [OTA Task] [prvParseJSONbyModel] Extracted parameter [ clientToken: 0:your-thing-name ]
12 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: execution
13 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: jobId
14 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: jobDocument
15 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: afr_ota
16 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: streamname
17 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: files
18 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: filepath
19 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: filesize
20 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: fileid
21 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: certfile
22 1289 [OTA Task] [prvParseJSONbyModel] parameter not present: sig-sha256-ecdsa
23 1289 [OTA Task] [prvParseJobDoc] Ignoring job without ID.
24 1289 [OTA Task] [prvOTA_Close] Context->0x3ffbb4a8
25 1290 [OTA] [OTA_AgentInit] Ready.
26 1390 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
27 1490 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
28 1590 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
29 1690 [OTA] State: Ready Received: 1 Queued: 1 Processed: 1 Dropped: 0
```

   La scheda ora rimane in ascolto di aggiornamenti OTA.

Firmware iniziale sul simulatore Windows

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Quando si utilizza il simulatore Windows, non è necessario memorizzare nella flash una versione iniziale del firmware. Il simulatore di Windows fa parte dell'applicazione `aws_demos`, che include anche il firmware.

Installare la versione iniziale del firmware su una scheda personalizzata

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Utilizzando il proprio IDE, creare il progetto `aws_demos`, assicurandosi di includere la libreria OTA. Per ulteriori informazioni sulla struttura del codice sorgente di FreerTOS, vedere. Demo FreerTOS

Assicurati di includere il certificato di firma del codice, la chiave privata e la catena di fiducia dei certificati nel progetto FreerTOS o sul tuo dispositivo.

Utilizzando lo strumento appropriato, masterizzare l'applicazione nella scheda e assicurarsi che venga eseguita correttamente.

Aggiornare la versione del firmware

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

L'agente OTA incluso in FreerTOS controlla la versione di qualsiasi aggiornamento e lo installa solo se è più recente della versione del firmware esistente. La procedura seguente mostra come incrementare la versione del firmware dell'applicazione dimostrativa OTA.

1. Aprire il progetto `aws_demos` nell'IDE.
2. Aprire `demos/include/aws_application_version.h` e incrementare il valore del token `APP_VERSION_BUILD`.
3. Se utilizzi Microchip Curiosity PIC32 MZEF, incrementa il numero di sequenza OTA. `vendors/microchip/boards/curiosity_pic32mzef/bootloader/bootloader/utility/user-config/ota-descriptor.config` Il numero di sequenza OTA deve essere incrementato per ogni nuova immagine OTA generata.
4. Ricreare il progetto.

È necessario copiare l'aggiornamento del firmware nel bucket Amazon S3 creato come descritto in. Crea un bucket Amazon S3 per archiviare l'aggiornamento Il nome del file da copiare su Amazon S3 dipende dalla piattaforma hardware utilizzata:

- Texas Instruments CC322 0SF-LAUNCHXL: `vendors/ti/boards/cc3220_launchpad/aws_demos/ccs/debug/aws_demos.bin`
- Microchip PIC32 Curiosity MZEF: `vendors/microchip/boards/curiosity_pic32mzef/aws_demos/mplab/dist/pic32mz_ef_curiosity/production/mplab.production.ota.bin`
- ESP32Espresso: `vendors/espressif/boards/esp32/aws_demos/make/build/aws_demos.bin`

Creazione di un aggiornamento OTA (console)AWS IoT

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

1. Nel pannello di navigazione della AWS IoT console, scegli Gestisci, quindi scegli Jobs.

2. Scegli Create (Crea).
3. In Crea un processo di aggiornamento Over-the-Air FreerTOS (OTA), scegli Crea processo di aggiornamento OTA.
4. È possibile distribuire un aggiornamento OTA in un singolo dispositivo o in un gruppo di dispositivi. In Seleziona i dispositivi da aggiornare scegliere Seleziona. Per aggiornare un singolo dispositivo, scegliere la scheda Oggetti. Per aggiornare un gruppo di dispositivi, scegliere la scheda Gruppi di oggetti.
5. Se si intende aggiornare un singolo dispositivo, selezionare la casella di controllo accanto all'oggetto IoT associato al dispositivo. Se si intende aggiornare un gruppo di dispositivi, selezionare la casella di controllo accanto al gruppo di oggetti associato ai dispositivi. Scegli Next (Successivo).
6. In Select the protocol for firmware image transfer (Seleziona il protocollo per il trasferimento dell'immagine del firmware), scegliere HTTP, MQTT, o scegliere entrambi per consentire a ciascun dispositivo di determinare il protocollo da utilizzare.
7. In Selezione e firma dell'immagine del firmware scegliere Firma nuova immagine firmware automaticamente.
8. In Code signing profile (Profilo di firma del codice) scegliere Create (Crea).
9. In Create a code signing profile (Crea profilo di firma del codice) immettere un nome per il profilo di firma del codice.

    a. In Piattaforma hardware dispositivo scegliere la piattaforma hardware.

> **Note**
>
> In questo elenco vengono visualizzate solo le piattaforme hardware che sono state qualificate per FreerTOS. Se si intende eseguire test di una piattaforma non qualificata e si utilizza il pacchetto di crittografia ECDSA P-256 SHA-256 per la firma, si può scegliere il profilo di firma del codice del simulatore Windows per produrre una firma compatibile. Se si utilizza una piattaforma non qualificata e si utilizza una suite di crittografia diversa da ECDSA P-256 SHA-256 per la firma, è possibile utilizzare Code Signing per, oppure è possibile firmare personalmente l'aggiornamento del firmware. AWS IoT Per ulteriori informazioni, consulta Aggiungere una firma digitale all'aggiornamento del firmware.

b. In Code signing certificate (Certificato di firma del codice), scegliere Select (Seleziona) per selezionare un certificato importato precedentemente o Import (Importa) per importare un nuovo certificato.

c. In Pathname of code signing certificate on device (Nome di percorso del certificato della firma di codice), immettere il nome del percorso completo al certificato di firma del codice nel dispositivo. Il percorso del certificato varia in base alla piattaforma. Si tratta del percorso in cui si posiziona il certificato di firma del codice seguendo le istruzioni in Installazione del firmware iniziale.

> **Important**
>
> Su Texas Instruments CC322 0SF-LAUNCHXL, non includete una barra anteriore (**/**) davanti al nome del file se il certificato di firma del codice esiste nella radice del file system. In caso contrario, l'aggiornamento OTA non riesce durante l'autenticazione, con l'errore `file not found`.

10. In Select your firmware image in S3 or upload it (Seleziona l'immagine del firmware in S3 o caricala) scegliere Seleziona. Viene visualizzato un elenco dei bucket Amazon S3. Scegliere il bucket contenente l'aggiornamento del firmware, quindi scegliere l'aggiornamento del firmware nel bucket.

> **Note**
>
> I progetti dimostrativi di Microchip Curiosity PIC32 MZEF producono due immagini binarie con nomi predefiniti di e. `mplab.production.bin` `mplab.production.ota.bin` Utilizzare il secondo file quando si carica un'immagine per l'aggiornamento OTA.

11. In Nome di percorso dell'immagine firmware sul dispositivo immettere il nome del percorso completo nel dispositivo in cui l'immagine del firmware verrà copiata dal processo OTA. Questo percorso varia in base alla piattaforma.

> **Important**
>
> Sul Texas Instruments CC322 0SF-LAUNCHXL, a causa delle restrizioni di sicurezza, il nome del percorso dell'immagine del firmware deve essere. `/sys/mcuflashimg.bin`

12. In Tipo di file, inserisci un valore intero compreso tra 0 e 255. Il tipo di file immesso verrà aggiunto al documento Job che viene consegnato all'MCU. Lo sviluppatore del firmware/software MCU ha la piena responsabilità su cosa fare con questo valore. Gli scenari possibili includono un MCU dotato di un processore secondario il cui firmware può essere aggiornato indipendentemente dal processore principale. Quando il dispositivo riceve un processo di aggiornamento OTA, può utilizzare il tipo di file per identificare a quale processore è destinato l'aggiornamento.
13. In IAM role for OTA update job (Ruolo IAM per il processo di aggiornamento OTA), scegliere un ruolo in base alle istruzioni in Creazione di un ruolo del servizio per gli aggiornamenti OTA.
14. Scegli Next (Successivo).
15. Immetti un ID e una descrizione per il processo di aggiornamento OTA.
16. In Tipo di lavoro, scegli Il tuo lavoro verrà completato dopo la distribuzione nella cartella selezionata devices/groups (snapshot).
17. Scegli le configurazioni facoltative appropriate per il processo (rollout esecuzioni processi, interruzione processo, timeout esecuzioni processi e tag).
18. Scegli Create (Crea).

Per utilizzare un'immagine del firmware firmata precedentemente

1. In Select and sign your firmware image (Selezione e firma dell'immagine del firmware), scegliere Select a previously signed firmware image (Seleziona un'immagine firmware firmata precedentemente).
2. In Nome di percorso dell'immagine firmware sul dispositivo immettere il nome del percorso completo nel dispositivo in cui l'immagine del firmware verrà copiata dal processo OTA. Questo percorso varia in base alla piattaforma.
3. In Previous code signing job (Processo di firma di codice precedente), scegliere Select (Seleziona), quindi scegliere il processo di firma del codice precedente utilizzato per firmare l'immagine del firmware utilizzata per l'aggiornamento OTA.

Utilizzo di un'immagine del firmware firmata personalizzata

1. In Select and sign your firmware image (Selezione e firma dell'immagine del firmware), scegliere Use my custom signed firmware image (Usa la mia immagine firmware firmata personalizzata).

2. In Pathname of code signing certificate on device (Nome di percorso del certificato della firma di codice), immettere il nome del percorso completo al certificato di firma del codice nel dispositivo. Il nome di questo percorso varia in base alla piattaforma.
3. In Nome di percorso dell'immagine firmware sul dispositivo immettere il nome del percorso completo nel dispositivo in cui l'immagine del firmware verrà copiata dal processo OTA. Questo percorso varia in base alla piattaforma.
4. In Signature (Firma), incollare la firma in formato PEM.
5. In Original hash algorithm (Algoritmo hash originale), scegliere l'algoritmo hash utilizzato durante la creazione della firma del file.
6. In Original encryption algorithm (Algoritmo di crittografia originale), scegliere l'algoritmo hash utilizzato durante la creazione della firma del file.
7. In Seleziona l'immagine del firmware in Amazon S3, scegli il bucket Amazon S3 e l'immagine del firmware firmata nel bucket Amazon S3.

Dopo aver specificato le informazioni di firma del codice, specificare il tipo di processo di aggiornamento OTA, il ruolo del servizio e un ID per l'aggiornamento.

> **Note**
>
> Non utilizzare informazioni personali nell'ID processo per l'aggiornamento OTA. Esempi di informazioni personali includono:
>
> - Nomi.
> - Indirizzi IP.
> - Indirizzi e-mail.
> - Posizioni
> - Dati bancari.
> - Informazioni mediche.

1. In Job type (Tipo di processo), scegliere Your job will complete after deploying to the selected devices/groups (snapshot) (Il processo sarà completo dopo la distribuzione nei dispositivi/gruppi selezionati (snapshot)).
2. In IAM role for OTA update job (Ruolo IAM per il processo di aggiornamento OTA), scegliere il ruolo del servizio OTA.

3. Immettere un ID alfanumerico per il processo, quindi scegliere Create (Crea).

Il processo viene visualizzato nella AWS IoT console con lo stato IN CORSO.

> **Note**
>
> - La AWS IoT console non aggiorna automaticamente lo stato dei lavori. Aggiornare il browser per vedere gli aggiornamenti.

Connettere il terminale UART seriale al dispositivo. Dovrebbe essere visualizzato l'output che indica che il dispositivo sta scaricando il firmware aggiornato.

Al termine del download, il dispositivo viene riavviato, quindi viene installato il firmware aggiornato. Nel terminale UART si può vedere cosa sta accadendo.

Per un tutorial che illustra come utilizzare la console per creare un aggiornamento OTA, consulta Over-the-air aggiorna l'applicazione demo.

Creazione di un aggiornamento OTA con AWS CLI

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Quando usi il AWS CLI per creare un aggiornamento OTA, tu:

1. Aggiungere una firma digitale all'immagine del firmware.
2. Creare un flusso dell'immagine del firmware con firma digitale.
3. Avviare un processo di aggiornamento OTA.

Aggiungere una firma digitale all'aggiornamento del firmware

Quando si utilizza il AWS CLI per eseguire gli aggiornamenti OTA, è possibile utilizzare Code Signing for AWS IoT oppure è possibile firmare personalmente l'aggiornamento del firmware. Per un

elenco degli algoritmi di firma e hashing crittografici supportati da Code Signing for AWS IoT, vedi. SigningConfigurationOverrides Se desideri utilizzare un algoritmo crittografico non supportato da Code Signing for AWS IoT, devi firmare il file binario del firmware prima di caricarlo su Amazon S3.

Firma dell'immagine del firmware con Code Signing per AWS IoT

Per firmare l'immagine del firmware utilizzando Code Signing for AWS IoT, puoi utilizzare uno dei nostri AWS SDKs strumenti a riga di comando. Per ulteriori informazioni su Code Signing for AWS IoT, consulta Code Signing for AWS IoT.

Dopo aver installato e configurato gli strumenti di firma del codice, copia l'immagine del firmware non firmata nel bucket Amazon S3 e avvia un processo di firma del codice con i seguenti comandi. AWS CLI Il comando put-signing-profile crea un profilo di firma del codice riutilizzabile. Il comando start-signing-job avvia il processo di firma.

```
aws signer put-signing-profile \
    --profile-name your_profile_name \
    --signing-material certificateArn=arn:aws:acm::your-region:your-aws-account-
id:certificate/your-certificate-id \
    --platform your-hardware-platform \
    --signing-parameters certname=your_certificate_path_on_device
```

```
aws signer start-signing-job \
    --source
 's3={bucketName=your_s3_bucket,key=your_s3_object_key,version=your_s3_object_version_id}'
 \
    --destination 's3={bucketName=your_destination_bucket}' \
    --profile-name your_profile_name
```

> **Note**
>
> *your-source-bucket-name*e *your-destination-bucket-name* può essere lo stesso bucket Amazon S3.

Questi sono i parametri per i comandi put-signing-profile e start-signing-job:

`source`

Specifica la posizione del firmware senza firma in un bucket S3.

- `bucketName`: nome del bucket S3.
- `key`: chiave (nome file) del firmware nel bucket S3.
- `version`: la versione S3 del firmware nel bucket S3. È una cosa diversa rispetto alla versione del firmware. Puoi trovarlo accedendo alla console Amazon S3, scegliendo il tuo bucket e nella parte superiore della pagina, accanto a Versioni, selezionando Mostra.

`destination`

La destinazione sul dispositivo in cui verrà copiato il firmware firmato nel bucket S3. Il formato di questo parametro è lo stesso del parametro `source`.

`signing-material`

L'ARN del certificato di firma del codice. Questo ARN viene generato quando importi il certificato in ACM.

`signing-parameters`

Una mappa di coppie chiave-valore per la firma. Può includere tutte le informazioni utili durante la procedura di registrazione.

> **Note**
>
> Questo parametro è obbligatorio quando si crea un profilo di firma del codice per la firma degli aggiornamenti OTA con Code Signing for AWS IoT.

`platform`

Il codice `platformId` della piattaforma hardware in cui vuoi distribuire l'aggiornamento OTA.

Per ottenere un elenco delle piattaforme disponibili e dei relativi valori `platformId`, utilizza il comando aws signer list-signing-platforms.

Il processo di firma viene avviato e scrive l'immagine del firmware firmata nel bucket Amazon S3 di destinazione. Il nome file per l'immagine del firmware con firma è un GUID. Questo nome file ti servirà per creare un flusso. Puoi trovare il nome del file accedendo alla console Amazon S3 e scegliendo il tuo bucket. Se non viene visualizzato un file con un nome di file GUID, aggiorna il browser.

Il comando visualizza un ARN e un ID processo. Questi valori serviranno in seguito. Per ulteriori informazioni su Code Signing for AWS IoT, consulta Code Signing for. AWS IoT

Aggiungere una firma all'immagine del firmware in modo manuale

Firma digitalmente l'immagine del firmware e carica l'immagine del firmware firmata nel tuo bucket Amazon S3.

Creazione di un flusso dell'aggiornamento del firmware

Un flusso è un'interfaccia astratta ai dati che possono essere consumati da un dispositivo. Un flusso può nascondere la complessità dell'accesso ai dati archiviati in posizioni diverse o servizi basati su cloud diversi. Il servizio OTA Update Manager consente di utilizzare più dati, archiviati in varie posizioni in Amazon S3, per eseguire un aggiornamento OTA.

Quando crei un aggiornamento AWS IoT OTA, puoi anche creare uno stream che contiene l'aggiornamento del firmware firmato. Creare un file JSON (`stream.json`) che identifichi l'immagine del firmware con firma. Il file JSON deve contenere il testo seguente:

```
[
  {
    "fileId":"your_file_id",
    "s3Location":{
      "bucket":"your_bucket_name",
      "key":"your_s3_object_key"
    }
  }
]
```

Questi sono gli attributi nel file JSON:

`fileId`

Un numero intero arbitrario compreso tra 0 e 255 che identifica l'immagine del firmware.

`s3Location`

Il bucket e la chiave per il firmware di cui eseguire lo streaming.

`bucket`

Il bucket Amazon S3 in cui è archiviata l'immagine del firmware non firmata.

`key`

Il nome del file dell'immagine del firmware firmata nel bucket Amazon S3. Puoi trovare questo valore nella console Amazon S3 esaminando il contenuto del tuo bucket.

Se utilizzi Code Signing for AWS IoT, il nome del file è un GUID generato da Code Signing for. AWS IoT

Utilizzare il comando create-stream AWS CLI per creare un flusso.

```
aws iot create-stream \
    --stream-id your_stream_id \
    --description your_description \
    --files file://stream.json \
    --role-arn your_role_arn
```

Questi sono gli argomenti del create-stream AWS CLI comando:

`stream-id`

Stringa arbitraria per identificare il flusso.

`description`

Descrizione facoltativa del flusso.

`files`

Uno o più riferimenti ai file JSON che contengono i dati sulle immagini del firmware di cui eseguire lo streaming. Il file JSON deve contenere gli attributi seguenti:

`fileId`

Un ID file arbitrario.

`s3Location`

Il nome del bucket in cui l'immagine del firmware con firma viene archiviata e la chiave (nome file) dell'immagine del firmware con firma.

`bucket`

Il bucket Amazon S3 in cui è archiviata l'immagine del firmware firmata.

`key`

La chiave (nome file) del l'immagine del firmware con firma.

Quando usi Code Signing for AWS IoT, questa chiave è un GUID.

Di seguito è riportato un esempio del file `stream.json`.

```
[
    {
        "fileId":123,
        "s3Location": {
            "bucket":"codesign-ota-bucket",
            "key":"48c67f3c-63bb-4f92-a98a-4ee0fbc2bef6"
        }
    }
]
```

`role-arn`

Il ruolo del servizio OTA che consente anche l'accesso al bucket Amazon S3 in cui è archiviata l'immagine del firmware.

Per trovare la chiave oggetto Amazon S3 dell'immagine del firmware firmata, usa il aws signer describe-signing-job --job-id ***my-job-id*** comando where `my-job-id` is the job ID visualizzato dal create-signing-job AWS CLI comando. L'output del comando describe-signing-job contiene la chiave dell'immagine firmware con firma.

```
... text deleted for brevity ...
  "signedObject": {
    "s3": {
      "bucketName": "ota-bucket",
      "key": "7309da2c-9111-48ac-8ee4-5a4262af4429"
    }
  }
... text deleted for brevity ...
```

Creazione di un aggiornamento OTA

Usa il create-ota-update AWS CLI comando per creare un processo di aggiornamento OTA.

> **Note**
>
> Non utilizzare informazioni personali (PII) nell'ID processo per l'aggiornamento OTA. Esempi di informazioni personali includono:
>
> - Nomi.
> - Indirizzi IP.
> - Indirizzi e-mail.
> - Posizioni
> - Dati bancari.
> - Informazioni mediche.

```
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]
```

`cli-input-json` formato

```
{
  "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"
    }
  ]
```

```
}
```

`cli-input-json` campi

Nome	Tipo	Descrizione
`otaUpdateId`	stringa (max:128 min:1)	ID dell'aggiornamento OTA da creare.
`description`	string (max:2028)	Descrizione dell'aggi ornamento OTA.
`targets`	elenco	Dispositivi destinati alla ricezione di aggiornamenti OTA.
`protocols`	elenco	Il protocollo utilizzato per trasferire l'immagine di aggiornamento OTA. I valori validi sono [HTTP], [MQTT], [HTTP, MQTT]. Quando vengono specificati sia HTTP che MQTT, il dispositivo di destinazione può scegliere il protocollo.
`targetSelection`	string	Specifica se l'esecuzione dell'aggiornamento continuerà (CONTINUOUS) o se l'aggiorn amento verrà completat o dopo che tutti gli oggetti specificati come target avranno completato l'aggiorn amento (SNAPSHOT). Se è continuo, l'aggiornamento può anche essere eseguito in un oggetto quando viene rilevata

Nome	Tipo	Descrizione
		una modifica in un target. Ad esempio, un aggiornam ento verrà eseguito in un oggetto quando l'oggetto viene aggiunto a un gruppo target, anche dopo che l'aggiorn amento è stato completato da tutti gli oggetti originariamente nel gruppo. Valori validi: CONTINUOUS | SNAPSHOT. Enumerazione: CONTINUOU S | SNAPSHOT
`awsJobExecutionsRo lloutConfig`		Configurazione per l'impleme ntazione degli aggiornamenti OTA.
`maximumPerMinute`	integer (max:1000 min:1)	Numero massimo di esecuzion i del processo di aggiornam ento OTA avviate al minuto.
`exponentialRate`		La velocità di aumento di un rollout di processo. Questo parametro consente di definire un aumento di velocità esponenziale per un rollout di processo.
`baseRatePerMinute`	integer (max:1000 min:1)	Il numero minimo di oggetti che riceveranno una notifica di un processo in sospeso, ogni minuto all'inizio del rollout di processo. Questa è la velocità iniziale del rollout.

Nome	Tipo	Descrizione
`rateIncreaseCriteria`		I criteri per avviare l'aumento della velocità di rollout per un processo. AWS IoT supporta fino a una cifra dopo il decimale (ad esempio, 1,5, ma non 1,55).
`numberOfNotifiedThings`	integer (min:1)	Quando questo numero di elementi viene notificato, inizia l'aumento della velocità di rollout.
`numberOfSucceededThings`	integer (min:1)	Quando questo numero di elementi viene completato nell'esecuzione del processo, inizia l'aumento della velocità di rollout.
`awsJobPresignedUrlConfig`		Informazioni di configurazione per pre-signed. URLs
`expiresInSec`	Long	Per quanto tempo (in secondi) i prefirmati URLs sono validi. I valori validi sono compresi tra 60 e 3600 e il valore predefinito è 1800 secondi. I prefirmati URLs vengono generati quando viene ricevuta una richiesta per il documento di lavoro.
`awsJobAbortConfig`		I criteri che determinano quando e come avviene l'interruzione di un processo.

Nome	Tipo	Descrizione
`abortCriteriaList`	elenco	Elenco dei criteri che determinano quando e come interrompere il processo.
`failureType`	string	Tipo di errori di esecuzione dei processi che possono avviare un processo di interruzione. enum: FAILED | REJECTED | TIMED_OUT | ALL
`action`	string	Tipo di operazione del processo da eseguire per avviare l'interruzione del processo. enum: CANCEL
`minNumberOfExecutedThings`	integer (min:1)	Il numero minimo di elementi che devono ricevere le notifiche di esecuzione del processo prima che il processo possa essere interrotto.

Nome	Tipo	Descrizione
`awsJobTimeoutConfig`		Specifica l'intervallo di tempo a disposizione di ciascun dispositivo per terminare l'esecuzione del processo. Viene avviato un timer quando imposti lo stato di esecuzione del processo su `IN_PROGRESS` . Se lo stato di esecuzione del processo non è impostato su un altro stato terminale prima del timeout, verrà automaticamente impostato su `TIMED_OUT` .
`inProgressTimeoutInMinutes`	Long	Specifica l'intervallo di tempo, in minuti, a disposizione di ciascun dispositivo per terminare l'esecuzione di questo processo. L'intervallo di timeout può essere compreso fra 1 minuto e 7 giorni (da 1 a 10080 minuti). Il timer in corso non può essere aggiornato e verrà applicato a tutte le esecuzioni del processo. Se l'esecuzione del processo resta nello stato IN_PROGRESS per un periodo di tempo superiore a quello consentito dall'intervallo, l'esecuzione del processo non andrà a buon fine e verrà impostato lo stato `TIMED_OUT` terminale.

Nome	Tipo	Descrizione
`files`	elenco	File da distribuire in streaming dall'aggiornamento OTA.
`fileName`	string	Nome del file.
`fileType`	integer Intervallo – Max: 255, min.: 0	Un valore intero che puoi includere nel documento di lavoro per consentire ai tuoi dispositivi di identificare il tipo di file ricevuto dal cloud.
`fileVersion`	string	Versione del file.
`fileLocation`		Percorso del firmware aggiornato.
`stream`		Flusso contenente l'aggiorn amento OTA.
`streamId`	string (max:128 min:1)	L'ID del flusso.
`fileId`	integer (max:255 min:0)	ID di un file associato a un flusso.
`s3Location`		Percorso del firmware aggiornato in S3.
`bucket`	string (min:1)	Bucket S3.
`key`	string (min:1)	La chiave S3.
`version`	string	Versione del bucket S3

Nome	Tipo	Descrizione
codeSigning		Metodo di firma del codice del file.
awsSignerJobId	string	L'ID del AWSSigner Job creato per firmare il file.
startSigningJobParameter		Descrive il processo di firma del codice.
signingProfileParameter		Descrive il profilo di firma del codice.
certificateArn	string	ARN del certificato.
platform	string	Piattaforma hardware del tuo dispositivo.
certificatePathOnDevice	string	Percorso del certificato di firma del codice sul tuo dispositivo.
signingProfileName	string	Nome del profilo di firma del codice.
destination		Percorso in cui scrivere il file firmato del codice.
s3Destination		Descrive il percorso del firmware aggiornato in S3.
bucket	string (min:1)	Bucket S3 che contiene il firmware aggiornato.
prefix	string	Prefisso S3.
customCodeSigning		Metodo personalizzato per la firma del codice di un file.

Nome	Tipo	Descrizione
`signature`		Firma per il file.
`inlineDocument`	blob	Rappresentazione binaria codificata in base64 della firma del codice.
`certificateChain`		Catena di certificati.
`certificateName`	string	Nome del certificato.
`inlineDocument`	string	Rappresentazione binaria codificata in base64 della catena di certificati di firma del codice.
`hashAlgorithm`	string	Algoritmo hash usato per firmare il codice del file.
`signatureAlgorithm`	string	Algoritmo di firma usato per firmare il codice del file.
`attributes`	map	Un elenco di name/attribute coppie.
`roleArn`	string (max:2048 min:20)	Il ruolo IAM che garantisce AWS IoT l'accesso ad Amazon S3 AWS IoT , ai job AWS e alle risorse di Code Signing per creare un processo di aggiornamento OTA.
`additionalParameters`	map	Elenco di parametri aggiuntivi per l'aggiornamento OTA che sono coppie nome/valore.
`tags`	elenco	Metadati utilizzabili per la gestione degli aggiornamenti.

Nome	Tipo	Descrizione
`Key`	string (max:128 min:1)	La chiave del tag.
`Value`	string (max:256 min:1)	Il valore del tag.

Output

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

AWS CLI campi di output

Nome	Tipo	Descrizione
`otaUpdateId`	stringa (max:128 min:1)	ID aggiornamento OTA.
`awsIotJobId`	string	L'ID del AWS IoT lavoro associato all'aggiornamento OTA.
`otaUpdateArn`	string	ARN dell'aggiornamento OTA.
`awsIotJobArn`	string	L'ARN del AWS IoT lavoro associato all'aggiornamento OTA.
`otaUpdateStatus`	string	Stato dell'aggiornamento OTA.

Nome	Tipo	Descrizione
		Enumerazione: CREATE_PENDING | CREATE_IN_PROGRESS | CREATE_COMPLETE | CREATE_FAILED

Di seguito è riportato un esempio di file JSON passato al create-ota-update comando che utilizza Code Signing for. AWS IoT

```
[
  {
    "fileName": "firmware.bin",
    "fileType": 1,
    "fileLocation": {
      "stream": {
        "streamId": "004",
        "fileId":123
      }
    },
    "codeSigning": {
      "awsSignerJobId": "48c67f3c-63bb-4f92-a98a-4ee0fbc2bef6"
    }
  }
]
```

Di seguito è riportato un esempio di file JSON passato al create-ota-update AWS CLI comando che utilizza un file in linea per fornire materiale personalizzato per la firma del codice.

```
[
  {
    "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"
      }
    }
  }
]
```

Di seguito è riportato un esempio di file JSON passato al create-ota-update AWS CLI comando che consente a FreerTOS OTA di avviare un processo di firma del codice e creare un profilo e uno stream di firma del codice.

```
[
  {
    "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"
          }
        }
```

```
      }
    }
  }
]
```

Di seguito è riportato un esempio di file JSON passato al create-ota-update AWS CLI comando che crea un aggiornamento OTA che avvia un processo di firma del codice con un profilo esistente e utilizza il flusso specificato.

```
[
  {
    "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"
          }
        }
      }
    }
  }
]
```

Di seguito è riportato un esempio di file JSON passato al create-ota-update AWS CLI comando che consente a FreerTOS OTA di creare uno stream con un ID lavoro di firma del codice esistente.

```
[
  {
    "fileName": "your_firmware_path_on_device",
    "fileType": 1,
    "fileVersion": "1",
```

```
    "codeSigning":{
      "awsSignerJobId": "your_signer_job_id"
    }
  }
]
```

Di seguito è riportato un esempio di file JSON passato al create-ota-update AWS CLI comando che crea un aggiornamento OTA. L'aggiornamento crea un flusso dall'oggetto S3 specificato e usa la firma del codice personalizzata.

```
[
  {
    "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"
      }
    }
  }
]
```

Creazione di un elenco degli aggiornamenti OTA

È possibile utilizzare il list-ota-updates AWS CLI comando per ottenere un elenco di tutti gli aggiornamenti OTA.

```
aws iot list-ota-updates
```

L'output del comando list-ota-updates è simile al seguente.

```
{
  "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
    }
  ]
}
```

Recupero di informazioni su un aggiornamento OTA

È possibile utilizzare il get-ota-update AWS CLI comando per ottenere lo stato di creazione o cancellazione di un aggiornamento OTA.

```
aws iot get-ota-update --ota-update-id your-ota-update-id
```

L'output del comando get-ota-update è simile al seguente.

```
{
    "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"
    }
}
```

I valori restituiti per `otaUpdateStatus` includono i seguenti:

`CREATE_PENDING`

La creazione di un aggiornamento OTA è in sospeso.

`CREATE_IN_PROGRESS`

Un aggiornamento OTA è in fase di creazione.

`CREATE_COMPLETE`

Un aggiornamento OTA è stato creato.

`CREATE_FAILED`

La creazione di un aggiornamento OTA non è riuscita.

`DELETE_IN_PROGRESS`

Un aggiornamento OTA è in fase di eliminazione.

`DELETE_FAILED`

L'eliminazione di un aggiornamento OTA non è riuscita.

> **Note**
>
> Se desideri ottenere lo stato di esecuzione di un aggiornamento OTA dopo che è stato creato, devi utilizzare il comando describe-job-execution. Per ulteriori informazioni, vedere Descrivi Job Execution.

Eliminazione di dati correlati al sistema OTA

Al momento, non è possibile utilizzare la AWS IoT console per eliminare stream o aggiornamenti OTA. È possibile utilizzare il AWS CLI per eliminare gli stream, gli aggiornamenti OTA e i AWS IoT lavori creati durante un aggiornamento OTA.

Eliminazione di un flusso OTA

Quando si crea un aggiornamento OTA che utilizza MQTT, è possibile utilizzare la riga di comando o la AWS IoT console per creare uno stream per suddividere il firmware in blocchi in modo che possa essere inviato tramite MQTT. È possibile eliminare questo stream con il delete-stream AWS CLI comando, come illustrato nell'esempio seguente.

```
aws iot delete-stream --stream-id your_stream_id
```

Eliminazione di un aggiornamento OTA

Durante la creazione di un aggiornamento OTA, viene creato quanto segue:

- Una voce nel database dei processi di aggiornamento OTA.
- Un AWS IoT lavoro per eseguire l'aggiornamento.
- Un'esecuzione AWS IoT di lavoro per ogni dispositivo in fase di aggiornamento.

Il comando delete-ota-update elimina solo la voce nel database dei processi di aggiornamento OTA. È necessario utilizzare il comando delete-job per eliminare il processo AWS IoT .

Usa il comando delete-ota-update per eliminare un aggiornamento OTA:

```
aws iot delete-ota-update --ota-update-id your_ota_update_id
```

`ota-update-id`

ID dell'aggiornamento OTA da eliminare.

`delete-stream`

Elimina il flusso associato all'aggiornamento OTA.

`force-delete-aws-job`

Elimina il AWS IoT lavoro associato all'aggiornamento OTA. Se questo flag non è impostato e il processo si trova nello stato `In_Progress`, il processo non viene eliminato.

Eliminazione di un processo IoT creato per un aggiornamento OTA

FreerTOS crea AWS IoT un lavoro quando si crea un aggiornamento OTA. Viene inoltre creata l'esecuzione di un processo per ogni dispositivo che elabora il processo. È possibile utilizzare il delete-job AWS CLI comando per eliminare un lavoro e le relative esecuzioni di lavoro associate.

```
aws iot delete-job --job-id your-job-id --no-force
```

Il parametro `no-force` specifica che possono essere eliminati solo i processi che si trovano in uno stato terminale (COMPLETED o CANCELLED). È possibile eliminare un processo che si trova in uno stato non terminale passando il parametro `force`. Per ulteriori informazioni, consulta API DeleteJob .

> **Note**
>
> L'eliminazione di un processo con stato IN_PROGRESS interrompe qualsiasi esecuzione di processo con stato IN_PROGRESS nei dispositivi e può lasciare un dispositivo in uno stato non deterministico. Assicurati che per ogni dispositivo che esegue un processo che è stato eliminato venga ripristinato uno stato noto.

Il tempo necessario per eliminare un processo dipende dal numero di esecuzioni di processo create per il processo e da altri fattori. Mentre il processo viene eliminato, il suo stato è DELETION_IN_PROGRESS. Il tentativo di eliminare o annullare un processo il cui stato è già DELETION_IN_PROGRESS restituirà un errore.

Per eliminare l'esecuzione di un processo, puoi utilizzare delete-job-execution. È possibile eliminare l'esecuzione di un processo quando un esiguo numero di dispositivi non riesce a elaborare un processo. In questo modo viene eliminata l'esecuzione del processo per un singolo dispositivo, come mostrato nell'esempio seguente.

```
aws iot delete-job-execution --job-id your-job-id --thing-name
                    your-thing-name --execution-number your-job-execution-number --no-
force
```

Analogamente al delete-job AWS CLI comando, è possibile passare il `--force` parametro a per delete-job-execution forzare l'eliminazione di un'esecuzione di lavoro. Per ulteriori informazioni, consulta DeleteJobExecutionAPI.

> **Note**
>
> L'eliminazione di un'esecuzione di processo con stato IN_PROGRESS interrompe qualsiasi esecuzione di processo con stato IN_PROGRESS nei dispositivi e può lasciare un dispositivo in uno stato non deterministico. Assicurati che per ogni dispositivo che esegue un processo che è stato eliminato venga ripristinato uno stato noto.

Per ulteriori informazioni sull'utilizzo dell'applicazione dimostrativa dell'aggiornamento OTA, consulta Over-the-air aggiorna l'applicazione demo.

Servizio OTA Update Manager

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Il servizio over-the-air (OTA) Update Manager offre un modo per:

- Crea un aggiornamento OTA e le risorse che utilizza, tra cui un AWS IoT processo, uno AWS IoT stream e la firma del codice.
- Recuperare informazioni su un aggiornamento OTA.
- Elenca tutti gli aggiornamenti OTA associati al tuo AWS account.
- Elimina un aggiornamento OTA.

Un aggiornamento OTA è una struttura di dati gestita dal servizio OTA Update Manager. Contiene:

- Un ID aggiornamento OTA.
- Una descrizione facoltativa dell'aggiornamento OTA.
- Un elenco dei dispositivi da aggiornare (destinazioni).
- Il tipo di aggiornamento OTA: CONTINUOUS o SNAPSHOT. Consulta la sezione Jobs della AWS IoT Developer Guide per una discussione sul tipo di aggiornamento di cui hai bisogno.
- Protocollo utilizzato per eseguire l'aggiornamento OTA: [MQTT], [HTTP] o [MQTT, HTTP]. Quando si specifica MQTT e HTTP, la configurazione del dispositivo determina il protocollo utilizzato.
- Un elenco di file da inviare ai dispositivi di destinazione.
- Il ruolo IAM che garantisce AWS IoT l'accesso ad Amazon S3 AWS IoT , ai job AWS e alle risorse di Code Signing per creare un processo di aggiornamento OTA.
- Elenco facoltativo di coppie nome-valore definite dall'utente.

Gli aggiornamenti OTA sono stati progettati per aggiornare il firmware del dispositivo, ma puoi utilizzarli per inviare qualsiasi file che desideri a uno o più dispositivi con cui sei registrato. AWS IoT Quando invii aggiornamenti del firmware over the air, ti consigliamo di applicare la firma digitale in modo che i dispositivi che li ricevono possano verificare che non siano stati alterati durante il trasferimento.

È possibile inviare immagini firmware aggiornate utilizzando il protocollo HTTP o MQTT, a seconda delle impostazioni scelte. Puoi firmare gli aggiornamenti del firmware con Code Signing for FreerTOS oppure utilizzare i tuoi strumenti di firma del codice.

Per un maggiore controllo sul processo, puoi utilizzare l' CreateStreamAPI per creare uno stream quando invii aggiornamenti tramite MQTT. In alcuni casi, puoi modificare il codice FreerTOS Agent per regolare la dimensione dei blocchi che invii e ricevi.

Quando crei un aggiornamento OTA, il servizio OTA Manager crea un'attivitàAWS IoT per notificare ai dispositivi che è disponibile un aggiornamento. L'agente OTA FreerTOS viene eseguito sui tuoi dispositivi e ascolta i messaggi di aggiornamento. Quando è disponibile un aggiornamento, questo richiede l'immagine di aggiornamento del firmware su HTTP o MQTT e memorizza i file localmente. Controlla la firma digitale dei file scaricati e se valida, installa l'aggiornamento del firmware. Se non utilizzi FreerTOS, devi implementare il tuo agente OTA per ascoltare e scaricare gli aggiornamenti ed eseguire qualsiasi operazione di installazione.

Integrazione dell'agente OTA nell'applicazione

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

L'agente over-the-air (OTA) è progettato per semplificare la quantità di codice da scrivere per aggiungere la funzionalità di aggiornamento OTA al prodotto. Questo lavoro di integrazione è costituito principalmente dall'inizializzazione dell'agente OTA e, facoltativamente, dalla creazione di una funzione di callback personalizzata per rispondere ai messaggi di evento relativi al completamento OTA.

> **Note**
>
> Anche se l'integrazione della funzionalità di aggiornamento OTA nell'applicazione è piuttosto semplice, il sistema di aggiornamento OTA richiede qualcosa in più della semplice conoscenza dei principi di integrazione del codice dei dispositivi. Per acquisire familiarità con la configurazione dell' AWS account con AWS IoT elementi, credenziali, certificati di firma del codice, dispositivi di provisioning e processi di aggiornamento OTA, consulta Prerequisiti di FreerTOS.

Gestione delle connessioni

L'agente OTA utilizza il protocollo MQTT per tutte le operazioni di comunicazione di controllo che coinvolgono i AWS IoT servizi, ma non gestisce la connessione MQTT. Per garantire che l'agente OTA non interferisca con la policy di gestione della connessione dell'applicazione, la connessione MQTT, incluse la disconnessione e qualsiasi funzionalità di riconnessione, deve essere gestita dall'applicazione utente principale. Il file può essere scaricato tramite il protocollo MQTT o HTTP. È possibile scegliere il protocollo quando si crea l'attività OTA. Se si sceglie MQTT, l'agente OTA utilizza la stessa connessione per le operazioni di controllo e per il download dei file. Se si sceglie HTTP, l'agente OTA gestisce le connessioni HTTP.

Demo OTA semplice

Di seguito è riportato un estratto di una semplice demo OTA che mostra il modo in cui l'agente si connette al broker MQTT e inizializza l'agente OTA. In questo esempio, configuriamo la demo per utilizzare il callback di completamento OTA predefinito e per restituire alcune statistiche una volta al secondo. Per brevità, nella demo tralasceremo alcuni dettagli.

La demo OTA dimostra anche la gestione della connessione MQTT monitorando il callback di disconnessione e ristabilendo la connessione. Quando si verifica una disconnessione, la demo sospende prima le operazioni dell'agente OTA e poi tenta di ristabilire la connessione MQTT. I tentativi di riconnessione MQTT vengono ritardati di un periodo di tempo che viene aumentato esponenzialmente fino a un valore massimo e viene aggiunto anche un jitter. Se la connessione viene ristabilita, l'agente OTA continua le sue operazioni.

Per un esempio funzionante che utilizza il broker AWS IoT MQTT, consulta il codice demo OTA nella `demos/ota` directory.

Poiché l'agente OTA è costituito dai propri task, il ritardo intenzionale di un secondo in questo esempio interessa solo questa applicazione. Non ha alcun impatto sulle prestazioni dell'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();
    }
}
```

Questo è il flusso generale di questa applicazione dimostrativa:

- Creare un contesto per l'agente MQTT.
- Connettersi all'endpoint AWS IoT .
- Inizializzare l'agente OTA.
- Loop che consente un processo di aggiornamento OTA e genera statistiche una volta al secondo.
- Se MQTT si disconnette, sospendi le operazioni dell'agente OTA.
- Prova a connetterti di nuovo con ritardo e tremolio esponenziali.
- Se riconnesso, riprendi le operazioni dell'agente OTA.
- Se l'agente si arresta, ritarda un secondo, quindi prova a riconnetterti.

Utilizzo di un callback personalizzato per gli eventi di completamento OTA

L'esempio precedente utilizzato `App_OTACompleteCallback` come gestore di callback per gli eventi di completamento OTA. (Vedi il terzo parametro della chiamata `OTA_AgentInit` API.) `App_OTACompleteCallback`è definito in*`freertos`*`/demos/ota/aws_iot_ota_update_demo.c`. Se si desidera implementare la gestione personalizzata degli eventi di completamento, è necessario passare l'indirizzo della funzione del gestore di callback all'API `OTA_AgentInit`. Durante il processo OTA, l'agente può inviare uno dei seguenti enum di evento al gestore di callback. Sarà lo sviluppatore dell'applicazione a decidere come e quando gestire tali eventi.

```
/**
* @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;
```

L'agente OTA può ricevere un aggiornamento in background durante l'elaborazione attiva dell'applicazione principale. Lo scopo di rendere disponibili questi eventi è quello di consentire all'applicazione di decidere se l'operazione può essere intrapresa immediatamente o se deve essere rimandata fino al completamento di un'altra elaborazione specifica dell'applicazione. In questo modo si impedisce l'interruzione imprevista del dispositivo durante l'elaborazione attiva (ad esempio la rimozione dei dati) che potrebbe essere causata da una reimpostazione dopo un aggiornamento del firmware. Questi sono gli eventi di processo ricevuti dal gestore di callback:

`eOTA_JobEvent_Activate event`

Quando questo evento viene ricevuto dal gestore di callback, è possibile reimpostare il dispositivo immediatamente o programmare una chiamata per reimpostare il dispositivo più tardi. In questo modo è possibile rinviare la fase si reimpostazione del dispositivo e di self-test, se necessario.

`eOTA_JobEvent_Fail event`

Quando questo evento viene ricevuto dal gestore di callback, l'aggiornamento non è andato a buon fine. Non è necessario fare nulla in questo caso. È possibile eseguire l'output di un messaggio di log o di eseguire un'operazione specifica dell'applicazione.

`eOTA_JobEvent_StartTest event`

La fase di self-test ha lo scopo di consentire l'esecuzione e il self-test del firmware appena aggiornato per stabilire se funziona correttamente ed eseguirne il commit in modo che sia

l'immagine dell'applicazione permanente più recente. Quando viene ricevuto e autenticato un nuovo aggiornamento e il dispositivo è stato ripristinato, l'agente OTA invia l'evento `eOTA_JobEvent_StartTest` alla funzione di callback quando è pronto per il test. Gli sviluppatori possono aggiungere i test richiesti per determinare se il firmware del dispositivo funziona correttamente dopo l'aggiornamento. Quando il firmware del dispositivo è ritenuto affidabile dai self-test, il codice deve usare il firmware come la nuova immagine permanente chiamando la funzione `OTA_SetImageState( eOTA_ImageState_Accepted )`.

Se il dispositivo non prevede hardware o meccanismi speciali che devono essere testati, è possibile utilizzare il gestore di callback predefinito. Dopo aver ricevuto l'evento `eOTA_JobEvent_Activate`, il gestore predefinito reimposta il dispositivo immediatamente.

Sicurezza OTA

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Di seguito sono riportati tre aspetti della sicurezza over-the-air (OTA):

Sicurezza della connessione

Il servizio OTA Update Manager si basa sui meccanismi di sicurezza esistenti, come l'autenticazione reciproca Transport Layer Security (TLS), utilizzati da AWS IoT. Il traffico di aggiornamento OTA passa attraverso il gateway del AWS IoT dispositivo e utilizza meccanismi AWS IoT di sicurezza. Ogni messaggio HTTP o MQTT in entrata e in uscita attraverso il gateway del dispositivo viene sottoposto a procedure rigorose di autenticazione e autorizzazione.

Autenticità e integrità degli aggiornamenti OTA

È possibile aggiungere una firma digitale al firmware prima di un aggiornamento OTA per garantire che provenga da una fonte affidabile e che non sia stato manomesso.

Il servizio FreerTOS OTA Update Manager utilizza Code Signing AWS IoT per firmare automaticamente il firmware. Per ulteriori informazioni, consulta Code Signing for AWS IoT.

L'agente OTA, che viene eseguito sui dispositivi, esegue i controlli di integrità sul firmware che raggiunge il dispositivo.

Sicurezza dell'operatore

Ogni chiamata API effettuata tramite l'API del piano di controllo è sottoposta all'autenticazione e all'autorizzazione standard IAM Signature versione 4. Per creare una distribuzione, è necessario disporre delle autorizzazioni necessarie per richiamare il comando`CreateDeployment`, `CreateJob` e. `CreateStream` APIs Inoltre, nella tua policy o ACL di Amazon S3, devi concedere le autorizzazioni di lettura al responsabile del AWS IoT servizio in modo che l'aggiornamento del firmware archiviato in Amazon S3 sia accessibile durante lo streaming.

Firma del codice per AWS IoT

La AWS IoT console utilizza Code Signing AWS IoT per firmare automaticamente l'immagine del firmware per qualsiasi dispositivo supportato da AWS IoT.

Code Signing for AWS IoT utilizza un certificato e una chiave privata importati in ACM. Puoi utilizzare un certificato autofirmato per i test, ma ti consigliamo di ottenere un certificato da un'autorità di certificazione (CA) commerciale nota.

I certificati di firma del codice utilizzano la versione 3 e le estensioni X.509. `Key Usage` `Extended Key Usage` L'estensione `Key Usage` è impostata su `Digital Signature` e l'estensione `Extended Key Usage` è impostata su `Code Signing`. Per ulteriori informazioni sulla firma dell'immagine del codice, consulta la Code Signing for AWS IoT Developer Guide e il Code Signing for API Reference. AWS IoT

> **Note**
>
> Puoi scaricare Code Signing for AWS IoT SDK da Tools for Amazon Web Services.

Risoluzione dei problemi di OTA

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Le sezioni seguenti contengono informazioni che possono essere utili per risolvere i problemi con gli aggiornamenti OTA.

Argomenti

- Configura i CloudWatch registri per gli aggiornamenti OTA
- Registra le AWS IoT chiamate API OTA con AWS CloudTrail
- Ottieni i dettagli sull'OTAUpdate errore di Create utilizzando il AWS CLI
- Ottieni i codici di errore OTA con AWS CLI
- Risoluzione dei problemi relativi agli aggiornamenti OTA di più dispositivi
- Risolvi i problemi relativi agli aggiornamenti OTA con il Launchpad 0SF di Texas Instruments CC322

Configura i CloudWatch registri per gli aggiornamenti OTA

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Il servizio OTA Update supporta la registrazione con Amazon CloudWatch. Puoi utilizzare la AWS IoT console per abilitare e configurare la CloudWatch registrazione di Amazon per gli aggiornamenti OTA. Per ulteriori informazioni, consulta Cloudwatch Logs.

Per abilitare la registrazione, devi creare un ruolo IAM e configurare la registrazione degli aggiornamenti OTA.

> **Note**
>
> Prima di abilitare la registrazione degli aggiornamenti OTA, assicurati di aver compreso le autorizzazioni di CloudWatch accesso ai registri. Gli utenti con accesso ai CloudWatch registri possono visualizzare le informazioni di debug. Per informazioni, consulta Autenticazione e controllo degli accessi per Amazon CloudWatch Logs.

Creare un ruolo di logging e abilitare il logging

Usa la console AWS IoT per creare un ruolo di logging e abilitare il logging.

1. Nel riquadro di navigazione scegliere Impostazioni.
2. Under Log scegliere Modifica.
3. In Livello di dettaglio scegliere Debug.
4. In Imposta ruolo, scegli Crea nuovo per creare un ruolo IAM per la registrazione.
5. In Nome immettere un nome univoco per il ruolo. Il ruolo verrà creato con tutte le autorizzazioni necessarie.
6. Scegli Aggiorna.

Log degli aggiornamenti OTA

Il servizio di aggiornamento OTA pubblica i log nel tuo account quando si verifica uno dei seguenti eventi:

- Viene creato un aggiornamento OTA.
- Viene completato un aggiornamento OTA.
- Viene creato un processo di firma del codice.
- Viene completato un processo di firma del codice.
- Viene creato un AWS IoT lavoro.
- Un AWS IoT lavoro è completato.
- Viene creato un flusso.

Puoi visualizzare i tuoi log nella console CloudWatch .

Per visualizzare un aggiornamento OTA nei CloudWatch registri

1. Nel riquadro di navigazione scegliere Log.
2. In Log Groups, scegli AWSIoTLogsV2.

I log degli aggiornamenti OTA possono contenere le seguenti proprietà:

accountId

L'ID AWS dell'account in cui è stato generato il registro.

actionType

L'operazione che ha generato il log. Può essere uno dei seguenti valori:

- `CreateOTAUpdate`: un aggiornamento OTA è stato creato.
- `DeleteOTAUpdate`: un aggiornamento OTA è stato eliminato.
- `StartCodeSigning`: un processo di firma del codice è stato avviato.
- `CreateAWSJob`: è stato creato un AWS IoT lavoro.
- `CreateStream`: un flusso è stato creato.
- `GetStream`: è stata inviata una richiesta di streaming alla funzionalità di consegna dei file AWS IoT basata su MQTT.
- `DescribeStream`: Una richiesta di informazioni su uno stream è stata inviata alla funzionalità di distribuzione dei AWS IoT file basata su MQTT.

awsJobId

L'ID del AWS IoT lavoro che ha generato il registro.

clientId

L'ID del client MQTT che ha effettuato la richiesta che ha generato il log.

clientToken

Il token del client associato alla richiesta che ha generato il log.

details

Informazioni aggiuntive sull'operazione che ha generato il log.

logLevel

Il livello di registrazione del log. Per i log degli aggiornamenti OTA, l'impostazione è sempre `DEBUG`.

otaUpdateId

L'ID dell'aggiornamento OTA che ha generato il log.

protocol

Il protocollo utilizzato per effettuare la richiesta che ha generato il log.

status

Lo stato dell'operazione che ha generato il log. I valori validi sono:

- Riuscito
- Errore

streamId

L'ID dello AWS IoT stream che ha generato il log.

timestamp

L'orario in cui è stato generato il log.

topicName

Argomento MQTT utilizzato per effettuare la richiesta che ha generato il log.

Log di esempio

Di seguito è riportato un esempio di log generato quando viene avviato un processo di firma del codice:

```
{
    "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."
}
```

Di seguito è riportato un esempio di log generato quando viene creato un AWS IoT lavoro:

```
{
    "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."
}
```

Di seguito è riportato un esempio di log generato quando viene creato un aggiornamento OTA:

```
{
    "timestamp": "2018-07-23 22:59:45.413",
    "logLevel": "DEBUG",
    "accountId": "123456789012",
    "status": "Success",
    "actionType": "CreateOTAUpdate",
    "otaUpdateId": "08957b03-eea3-448a-87fe-743e6891ca3a",
    "details": "OTAUpdate creation complete. The request status is SUCCESS."
}
```

Di seguito è riportato un esempio di log generato quando viene creato un flusso:

```
{
    "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."
}
```

Di seguito è riportato un esempio di log generato quando viene eliminato un aggiornamento OTA:

```
{
```

```
    "timestamp": "2018-07-23 23:03:09.505",
    "logLevel": "DEBUG",
    "accountId": "123456789012",
    "status": "Success",
    "actionType": "DeleteOTAUpdate",
    "otaUpdateId": "9bdd78fb-f113-4001-9675-1b595982292f",
    "details": "Delete OTA Update. The request status is SUCCESS."
}
```

Di seguito è riportato un esempio di log generato quando un dispositivo richiede uno stream dalla funzionalità di consegna dei file basata su 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."
}
```

Di seguito è riportato un esempio di log generato quando un dispositivo chiama l'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."
```

```
}
```

Registra le AWS IoT chiamate API OTA con AWS CloudTrail

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

FreerTOS è integrato CloudTrail con, un servizio che AWS IoT acquisisce le chiamate API OTA e consegna i file di registro a un bucket Amazon S3 specificato dall'utente. CloudTrail acquisisce le chiamate API dal tuo codice all'OTA. AWS IoT APIs Utilizzando le informazioni raccolte da CloudTrail, è possibile determinare la richiesta effettuata a AWS IoT OTA, l'indirizzo IP di origine da cui è stata effettuata la richiesta, chi ha effettuato la richiesta, quando è stata effettuata e così via.

Per ulteriori informazioni CloudTrail, incluse le modalità di configurazione e attivazione, consulta la Guida per l'AWS CloudTrail utente.

Informazioni su FreerTOS in CloudTrail

Quando CloudTrail la registrazione è abilitata nel tuo AWS account, le chiamate API effettuate alle azioni AWS IoT OTA vengono tracciate nei file di CloudTrail registro dove vengono scritte insieme ad altri record di servizio. AWS CloudTrail determina quando creare e scrivere su un nuovo file in base al periodo di tempo e alle dimensioni del file.

Le seguenti azioni del piano di controllo AWS IoT OTA vengono registrate da CloudTrail:

- CreateStream
- DescribeStream
- ListStreams
- UpdateStream
- DeleteStream
- CreaOTAUpdate
- Ottenere OTAUpdate
- ElencoOTAUpdates

- EliminaOTAUpdate

> **Note**
>
> AWS IoT Le azioni del piano dati OTA (lato dispositivo) non vengono registrate da CloudTrail. CloudWatch Utilizzatelo per monitorarle.

Ogni voce di log contiene informazioni sull'utente che ha generato la richiesta. Le informazioni sull'identità dell'utente nella voce di log ti permettono di determinare quanto segue:

- Se la richiesta è stata effettuata con le credenziali dell'utente IAM o root.
- Se la richiesta è stata effettuata con le credenziali di sicurezza temporanee per un ruolo o un utente federato.
- Se la richiesta è stata effettuata da un altro AWS servizio.

Per ulteriori informazioni, vedete l'elemento CloudTrail userIdentity. AWS IoT Le azioni OTA sono documentate nell'AWS IoT OTA API Reference.

Puoi archiviare i file di log nel tuo bucket Amazon S3 per tutto il tempo che desideri, ma puoi anche definire regole del ciclo di vita di Amazon S3 per archiviare o eliminare automaticamente i file di registro. Per impostazione predefinita, i file di log sono crittografati mediante la crittografia lato server (SSE) di Amazon S3.

Se desideri ricevere una notifica quando i file di log vengono consegnati, puoi CloudTrail configurare la pubblicazione delle notifiche di Amazon SNS. Per ulteriori informazioni, consulta Configurazione delle notifiche di Amazon SNS per. CloudTrail

Puoi anche aggregare i file di log AWS IoT OTA da più AWS regioni e più AWS account in un unico bucket Amazon S3.

Per ulteriori informazioni, consulta Ricezione di file di CloudTrail registro da più regioni e Ricezione di file di CloudTrail registro da più account.

Comprendere le voci dei file di registro di FreerTOS

CloudTrail i file di registro possono contenere una o più voci di registro. Ogni voce elenca più eventi in formato JSON. Una voce di log rappresenta una singola richiesta inviata da un'origine e include

informazioni sull'operazione richiesta, la data e l'ora dell'operazione, i parametri della richiesta e così via. Le voci di log non sono una traccia di stack ordinata delle chiamate API pubbliche, pertanto non vengono visualizzate in un ordine specifico.

L'esempio seguente mostra una voce di CloudTrail registro che illustra il log di un invito all'`CreateOTAUpdate`azione.

```
{
    "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"
}
```

Ottieni i dettagli sull'OTAUpdate errore di Create utilizzando il AWS CLI

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Se il processo di creazione di un processo di aggiornamento OTA fallisce, è possibile che sia possibile intraprendere delle azioni per risolvere il problema. Quando crei un processo di aggiornamento OTA, il servizio OTA manager crea un processo IoT e lo pianifica per i dispositivi di destinazione, e questo processo crea o utilizza anche altri tipi di AWS risorse nel tuo account (un processo di firma del codice, uno AWS IoT stream, un oggetto Amazon S3). Qualsiasi errore riscontrato può causare il fallimento del processo senza creare un lavoro. AWS IoT In questa sezione sulla risoluzione dei problemi forniamo istruzioni su come recuperare i dettagli dell'errore.

1. Istalla e configura la AWS CLI.
2. Esegui aws configure e inserisci le seguenti informazioni.

```
$ aws configure
```

```
AWS Access Key ID [None]: AccessID
AWS Secret Access Key [None]: AccessKey
Default region name [None]: Region
Default output format [None]: json
```

Per ulteriori informazioni, vedere Configurazione rapida con aws configure.

3. Esegui:

```
aws iot get-ota-update --ota-update-id ota_update_job_001
```

*ota_update_job_001*Dov'è l'ID che hai fornito all'aggiornamento OTA quando lo hai creato.

4. L'output apparirà come segue:

```
{
    "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 la creazione non è riuscita, il `otaUpdateStatus` campo nell'output del comando conterrà `CREATE_FAILED` e il `errorInfo` campo conterrà i dettagli dell'errore.

Ottieni i codici di errore OTA con AWS CLI

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

1. Istalla e configura la AWS CLI.
2. Esegui aws configure e inserisci le seguenti informazioni.

```
$ aws configure
AWS Access Key ID [None]: AccessID
```

```
AWS Secret Access Key [None]: AccessKey
Default region name [None]: Region
Default output format [None]: json
```

Per ulteriori informazioni, vedere Configurazione rapida con aws configure.

3. Esegui:

```
aws iot describe-job-execution --job-id JobID --thing-name ThingName
```

*JobID*Dov'è la stringa dell'ID del lavoro completo per il lavoro di cui vogliamo ottenere lo stato (era associato al processo di aggiornamento OTA quando è stato creato) ed *ThingName* è il nome dell' AWS IoT oggetto con cui il dispositivo è registrato in AWS IoT

4. L'output apparirà come segue:

```
{
    "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
    }
}
```

In questo output di esempio, "`reason`" in "`detailsmap`" comprende due campi: il campo visualizzato come "0xEEEEEEEE" contiene il codice di errore generico dall'agente OTA, il campo visualizzato come "0xffffffff" contiene il codice secondario. I codici di errore generici sono elencati in https://docs.aws.amazon.com/freertos/latest/lib-ref/html1/aws__ota__agent_8h.html. Consultare i codici di errore con il prefisso "`kOTA_Err_`". Il codice secondario può essere un codice specifico della piattaforma o fornire ulteriori dettagli sull'errore generico.

Risoluzione dei problemi relativi agli aggiornamenti OTA di più dispositivi

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Per eseguire OTAs su più dispositivi (cose) utilizzando la stessa immagine del firmware, implementa una funzione (ad esempio`getThingName()`) che recuperi `clientcredentialIOT_THING_NAME` dalla memoria non volatile. Assicurarsi che questa funzione legga il nome dell'oggetto da una parte della memoria non volatile che non sia sovrascritta dall'OTA e che venga effettuato il provisioning del nome dell'oggetto prima dell'esecuzione della prima attività. Se si utilizza il flusso JITP, è possibile leggere il nome dell'oggetto dal nome comune del certificato del dispositivo.

Risolvi i problemi relativi agli aggiornamenti OTA con il Launchpad 0SF di Texas Instruments CC322

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

La piattaforma CC322 0SF Launchpad fornisce un meccanismo di rilevamento delle manomissioni software. Utilizza un contatore degli avvisi di sicurezza che viene incrementato ogni volta che si verifica una violazione dell'integrità. Il dispositivo viene bloccato quando il contatore degli avvisi di sicurezza raggiunge una determinata soglia (il valore predefinito è 15) e l'host riceve l'evento asincrono `SL_ERROR_DEVICE_LOCKED_SECURITY_ALERT`. Il dispositivo bloccato quindi ha accessibilità limitata. Per ripristinare il dispositivo, è possibile riprogrammarlo o utilizzare la procedura per ripristinare l' restore-to-factoryimmagine di fabbrica. È consigliabile programmare il comportamento desiderato aggiornando il gestore eventi asincroni in `network_if.c`.

Librerie FreerTOS

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Le librerie FreerTOS forniscono funzionalità aggiuntive al kernel FreerTOS e alle sue librerie interne. È possibile utilizzare le librerie FreerTOS per il networking e la sicurezza nelle applicazioni integrate. Le librerie FreerTOS consentono inoltre alle applicazioni di interagire con i servizi. AWS IoT FreerTOS include librerie che consentono di:

- Connetti in modo sicuro i dispositivi al AWS IoT Cloud utilizzando MQTT e le ombre dei dispositivi.
- Scopri e connettiti ai core. AWS IoT Greengrass
- Gestire connessioni Wi-Fi.
- Ascolta ed elabora Aggiornamenti FreerTOS Over-the-Air.

La `libraries` directory contiene il codice sorgente delle librerie FreerTOS. Ci sono funzioni helper che consentono di implementare la funzionalità libreria. Non è consigliabile modificare le funzioni helper.

Librerie di porting FreerTOS

Le seguenti librerie di porting sono incluse nelle configurazioni di FreerTOS che sono disponibili per il download sulla console FreerTOS. Queste librerie sono dipendenti dalla piattaforma. I loro contenuti variano in base alla piattaforma hardware. Per informazioni sul porting di queste librerie su un dispositivo, consulta la FreerTOS Porting Guide.

Librerie di porting FreerTOS

Libreria	Documentazione di riferimento API	Descrizione
Bluetooth Low Energy	Documentazione di riferimento dell'API Bluetooth Low Energy	Utilizzando la libreria FreerTOS Bluetooth Low Energy, il microcont

Libreria	Documentazione di riferimento API	Descrizione
		rollore può comunicare con AWS IoT il broker MQTT tramite un dispositivo gateway. Per ulteriori informazioni, consulta Libreria Bluetooth Low Energy.
Over-the-Air Aggiornamenti	Riferimento all'API dell'agente OTA	La libreria AWS IoT Over-the-Air FreerTOS (OTA) Agent collega il tuo dispositivo FreerTOS all'agente OTA. AWS IoT Per ulteriori informazioni, consulta Libreria dell'agente OTA.
FreeRTOS+POSIX	Documentazione di riferimento all'API FreeRTOS+POSIX	Puoi usare la libreria FreeRTOS +POSIX per portare applicazioni conformi a POSIX nell'ecosistema FreerTOS. Per ulteriori informazioni, consulta FreeRTOS+POSIX.
Secure Sockets	Riferimento all'API Secure Sockets	Per ulteriori informazioni, consulta Libreria Secure Sockets.
FreeRTOS+TCP	Documentazione di riferimento all'API FreeRTOS+TCP	FreeRTOS+TCP è uno stack TCP/IP per FreeRTOS scalabile, open source e thread safe. Per ulteriori informazioni, consulta FreeRTOS+TCP.

Libreria	Documentazione di riferimento API	Descrizione
Wi-Fi	Guida di riferimento delle API di Wi-Fi	La libreria Wi-Fi FreerTOS consente di interfacciarsi con lo stack wireless di livello inferiore del microcont rollore. Per ulteriori informazioni, consulta Libreria Wi-Fi.
core PKCS11		La PKCS11 libreria principale è un'implementazione di riferimen to del Public Key Cryptography Standard #11, per supportare il provisioning e l'autenticazione del client TLS. Per ulteriori informazioni, consulta PKCS11 libreria principale.
TLS		Per ulteriori informazioni, consulta Transport Layer Security.
I/O comuni	Riferimento comune all'API I/O	Per ulteriori informazioni, consulta I/O comuni.

Librerie di applicazioni FreerTOS

Opzionalmente puoi includere le seguenti librerie di applicazioni autonome nella tua configurazione FreerTOS per interagire con i servizi sul cloud. AWS IoT

> **Note**
>
> Alcune librerie di applicazioni hanno le stesse APIs librerie del AWS IoT Device SDK for Embedded C. Per queste librerie, consulta il AWS IoT Device SDK C API Reference. Per ulteriori informazioni su AWS IoT Device SDK for Embedded C, consulta. AWS IoT SDK del dispositivo per C integrato

Librerie di applicazioni FreerTOS

Libreria	Documentazione di riferimento API	Descrizione
AWS IoT Device Defender	Riferimento all'API SDK di Device Defender C	La libreria FreerTOS collega il tuo dispositivo AWS IoT Device Defender FreerTOS a. AWS IoT Device Defender Per ulteriori informazioni, consulta AWS IoT Device Defender bibliotec a.
AWS IoT Greengrass	Riferimento all'API Greengrass	La libreria FreerTOS collega il tuo dispositivo AWS IoT Greengrass FreerTOS a. AWS IoT Greengrass Per ulteriori informazioni, consulta AWS IoT Greengrass Libreria Discovery.
MQTT	Riferimento all'API della libreria MQTT (v1.x.x) Documentazione di riferimento dell'API dell'agente MQTT (v1) Riferimento all'API SDK MQTT (v2.x.xx)	La libreria CoreMQTT fornisce un client per il dispositivo FreerTOS per pubblicare e sottoscrivere argomenti MQTT. MQTT è il protocollo con cui i dispositivi interagiscono. AWS IoT Per ulteriori informazioni sulla versione 3.0.0 della libreria CoreMQTT, vedere. libreria CoreMQTT
AWS IoT Device Shadow	Documentazione di riferimento delle API SDK C di Device Shadow	La libreria AWS IoT Device Shadow consente al dispositivo FreerTOS di interagire con AWS IoT le ombre del dispositivo. Per ulteriori informazioni, consulta AWS IoT Libreria Device Shadow.

Librerie comuni FreerTOS

Le seguenti librerie comuni estendono la funzionalità di kernel con strutture di dati e funzioni aggiuntive per lo sviluppo di applicazioni integrate. Queste librerie dipendono spesso dalle librerie di porting e applicazioni FreerTOS.

> **Note**
>
> Il AWS IoT Device SDK for Embedded C include librerie comuni con funzionalità APIs identiche a queste librerie. Per informazioni di riferimento sull'API, consulta la Documentazione di riferimento sull'API SDK C per dispositivi AWS IoT.

Librerie comuni FreerTOS

Libreria	Documentazione di riferimento API	Descrizione
Operazioni atomiche	Documentazione di riferimento dell'API SDK C per l'atomicità delle operazioni	Per ulteriori informazioni, consulta Operazioni atomiche.
Container lineari	Documentazione di riferimento delle API SDK C per i container lineari	Per ulteriori informazioni, consulta Libreria Linear Container.
Registrazione	Documentazione di riferimento delle API SDK C per la registrazione	Per ulteriori informazioni, consulta Libreria registrazione dei log.
Memoria statica	Documentazione di riferimento delle API SDK C per la memoria statica	Per ulteriori informazioni, consulta Libreria della memoria statica.
Pool di attività	Documentazione di riferimento dell'API SDK C per i pool di attività	Per ulteriori informazioni, consulta Libreria Pool di attività.

Configurazione delle librerie FreerTOS

Le impostazioni di configurazione per FreerTOS e AWS IoT Device SDK for Embedded C sono definite come costanti del preprocessore C. È possibile definire le impostazioni di configurazione mediante un file di configurazione globale o utilizzando un'opzione del compilatore, ad esempio `-D` in

`gcc`. Poiché le impostazioni di configurazione sono definite come costanti della fase di compilazione, una libreria deve essere ricompilata se un'impostazione di configurazione viene modificata.

Se vuoi utilizzare un file di configurazione globale per impostare le opzioni di configurazione, crea e salva il file denominato `iot_config.h` e posizionalo nel percorso di inclusione. All'interno del file, usa `#define` le direttive per configurare le librerie, le demo e i test FreerTOS.

Per ulteriori informazioni sulle opzioni di configurazione globali supportate, consulta Documentazione di riferimento sui file di configurazione globali.

Librerie comuni

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

FreerTOS include alcune librerie comuni che estendono la funzionalità del kernel con strutture di dati e funzioni aggiuntive per lo sviluppo di applicazioni integrate. Queste librerie sono spesso dipendenze di altre librerie FreerTOS.

Argomenti

- Operazioni atomiche
- Libreria Linear Container
- Libreria registrazione dei log
- Libreria della memoria statica
- Libreria Pool di attività

Operazioni atomiche

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Panoramica

Le operazioni atomiche garantiscono una sincronizzazione non bloccante nella programmazione concorrente. È possibile utilizzare le operazioni atomiche per risolvere problemi di prestazioni causati da operazioni asincrone che agiscono sulla memoria condivisa. FreerTOS supporta le operazioni atomiche, come implementato nel file header. `iot_atomic.h`

Il file di intestazione `iot_atomic.h` include due implementazioni per le operazioni atomiche:

- Disabilitazione dell'interrupt a livello globale.

  Questa implementazione è disponibile per tutte le piattaforme FreerTOS.
- Supporto atomico nativo ISA.

  Questa implementazione è disponibile solo per le piattaforme di compilazione basate su GCC, versione 4.7.0 e successive e con il supporto atomico ISA. Per ulteriori informazioni sulle funzioni GCC integrate, consulta la sezione relativa alle funzioni integrate per le operazioni atomiche basate sul modello di memoria.

Inizializzazione

Prima di utilizzare FreerTOS Atomic Operations, devi scegliere quale implementazione di operazioni atomiche vuoi usare.

1. Aprire file di configurazione `FreeRTOSConfig.h` da modificare.
2. Per l'implementazione del supporto atomico nativo ISA, definire e impostare la variabile `configUSE_ATOMIC_INSTRUCTION` su 1.

   Per disabilitare l'implementazione degli interrupt globale, annullare la definizione di `configUSE_ATOMIC_INSTRUCTION` o cancellarla.

Riferimento API

Per informazioni di riferimento complete sull'API, consulta Documentazione di riferimento sulle API dell'SDK C per le operazioni atomiche.

Libreria Linear Container

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

La libreria FreerTOS Linear Containers definisce strutture di dati lineari, inclusi elenchi e code, da utilizzare durante lo sviluppo di applicazioni integrate.

Riferimento API

Per informazioni di riferimento complete sull'API, consulta Documentazione di riferimento sull'API SDK C per container lineari.

Libreria registrazione dei log

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

La libreria di registrazione FreerTOS consente alle librerie FreeRTOS di stampare messaggi nel registro per il debug.

Riferimento API

Per un riferimento completo all'API, vedi Logging C SDKAPI Reference.

Libreria della memoria statica

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

La libreria FreerTOS Static Memory definisce alcune funzioni per la gestione dei buffer statici. Con questa libreria puoi utilizzare un componente della memoria statica per fornire buffer allocati staticamente anziché utilizzare l'allocazione della memoria dinamica.

Riferimento API

Per informazioni di riferimento complete sull'API, consulta Documentazione di riferimento sull'API SDK C per la memoria statica.

Libreria Pool di attività

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Panoramica

FreerTOS supporta la gestione delle attività con la libreria FreerTOS Task Pool. Oltre a consentire di pianificare le attività in background, la libreria Pool di attività supporta la pianificazione e l'annullamento delle attività in modalità asincrona e sicura. Utilizzando il Task Pool APIs, è possibile configurare le attività dell'applicazione per ottimizzare il compromesso tra prestazioni e ingombro di memoria.

La libreria Pool attività si basa su due strutture di dati fondamentali, ovvero il pool delle attività e i processi del pool delle attività.

Pool delle attività (**`IotTaskPool_t`**)

Il pool delle attività include una coda di distribuzione che gestisce l'esecuzione della coda dei processi, nonché i thread dei processi di esecuzione.

Processi del pool delle attività (**`IotTaskPoolJob_t`**)

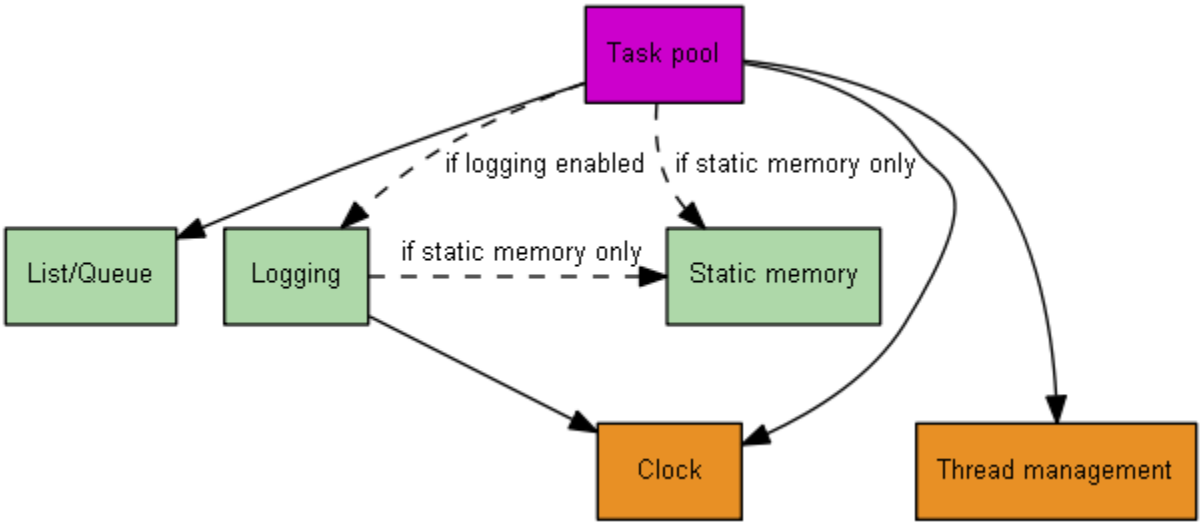
I processi del pool delle attività possono essere eseguiti come processi in background oppure come processi in background programmati. I lavori in background vengono avviati in First-In-First-Out ordine e non hanno vincoli di tempo. I processi programmati sono caratterizzati da un'esecuzione in background pianificata in base a un timer.

> **Note**
>
> I pool delle attività possono solo garantire che un processo pianificato venga eseguito al termine di un intervallo di timeout e non all'interno di una finestra temporale specifica.

Dipendenze e requisiti

La libreria Pool di attività dispone delle seguenti dipendenze:

- Libreria di Linear Container (elenchi/code) per la gestione delle strutture dati per le operazioni dei pool delle attività pianificate e in corso.
- La libreria di log (se l'impostazione di configurazione `IOT_LOG_LEVEL_TASKPOOL` non è `IOT_LOG_NONE`).
- Il livello piattaforma che fornisce un'interfaccia al sistema operativo per la gestione dei thread, i timer, le funzioni di clock e così via.



Funzionalità

Utilizzando la libreria Task Pool APIs, è possibile effettuare le seguenti operazioni:

- Pianificare processi immediati e differiti mediante le funzioni API non bloccanti della libreria.
- Creare processi allocati a livello statico e dinamico.
- Configurare le impostazioni della libreria per garantire la scalabilità a livello di prestazioni e footprint, in base alle risorse del sistema.
- Personalizzare la memorizzazione nella cache in caso di sovraccarico causato da memoria insufficiente durante la creazione dinamica di processi.

Risoluzione dei problemi

Le funzioni della libreria Pool di attività restituiscono codici di errore sotto forma di valori enumerati `IotTaskPoolError_t`. Per ulteriori informazioni su ciascun codice di errore, consulta la documentazione di riferimento per il tipo di dati enumerati `IotTaskPoolError_t` nella Documentazione di riferimento sull'API C SDK per i pool di attività.

Limitazioni d'uso

La libreria Pool di attività non può essere utilizzata dall'interno di una routine del servizio di interrupt (IRS).

È consigliabile evitare l'esecuzione di callback a livello di utente del pool delle attività che eseguono operazioni di blocco, in particolar modo operazioni di blocco a tempo indefinito. Le operazioni di

blocco di lungo termine intercettano infatti il thread di un pool di attività e pertanto generano potenziali situazioni di blocco critico (deadlock) o uso eccessivo delle risorse.

Inizializzazione

Prima di poter utilizzare il pool di attività, un'applicazione deve chiamare `IotTaskPool_CreateSystemTaskPool` per inizializzare un'istanza di un pool di attività di sistema. L'applicazione deve accertarsi che il pool di attività a livello di sistema venga inizializzato all'inizio della sequenza di avvio prima che qualsiasi libreria possa utilizzare il pool di attività e prima che il codice dell'applicazione pubblichi un processo nel pool di attività. Poco dopo l'avvio, il sistema inizializza il pool di attività a livello di sistema per tutte le librerie da condividere. Dopo l'inizializzazione, l'handle del pool di attività può essere recuperato per essere usato con l'API `IOT_SYSTEM_TASKPOOL`.

> **Note**
>
> La chiamata a `IotTaskPool_CreateSystemTaskPool` non alloca la memoria per la memorizzazione di strutture e stati dei dati del pool di attività, ma potrebbe allocare memoria per la memorizzazione di entità e strutture di dati dipendenti, ad esempio i thread del pool di dati.

Riferimento API

Per informazioni di riferimento complete sull'API, consulta Documentazione di riferimento sull'API SDK C del pool di attività.

Esempio di utilizzo

Supponiamo di dover pianificare una raccolta ricorrente di AWS IoT Device Defender metriche e di decidere di utilizzare un timer per pianificare la raccolta con le chiamate alla connessione, alla sottoscrizione e alla pubblicazione di MQTT. APIs Il codice seguente definisce una funzione di callback per accettare AWS IoT Device Defender metriche su MQTT, con un callback di disconnessione che si disconnette dalla connessione 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 );
}
```

Libreria BackOffAlgorithm

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Introduzione

La libreria BackOffAlgorithm è una libreria di utilità che viene utilizzata per distanziare le ritrasmissioni ripetute dello stesso blocco di dati, per evitare la congestione della rete. Questa libreria calcola il periodo di backoff necessario per riprovare le operazioni di rete (ad esempio una connessione di rete fallita con il server) utilizzando un algoritmo di backoff esponenziale con jitter.

Il backoff esponenziale con jitter viene in genere utilizzato quando si ritenta una connessione o una richiesta di rete non riuscita a un server a causa della congestione della rete o di carichi elevati sul server. Viene utilizzato per suddividere la tempistica delle richieste di nuovi tentativi create da più dispositivi che tentano di connettersi alla rete contemporaneamente. In un ambiente con scarsa connettività, un client può disconnettersi in qualsiasi momento; quindi una strategia di backoff aiuta anche il client a risparmiare la batteria evitando di tentare ripetutamente le riconnessioni quando è improbabile che abbiano successo.

La libreria è scritta in C e progettata per essere conforme a ISO C90 e MISRA C:2012. La libreria non dipende da alcuna libreria aggiuntiva diversa dalla libreria C standard e non ha alcuna allocazione

di heap, il che la rende adatta per i microcontrollori IoT, ma anche completamente portabile su altre piattaforme.

Questa libreria può essere utilizzata liberamente ed è distribuita con la licenza open source MIT.

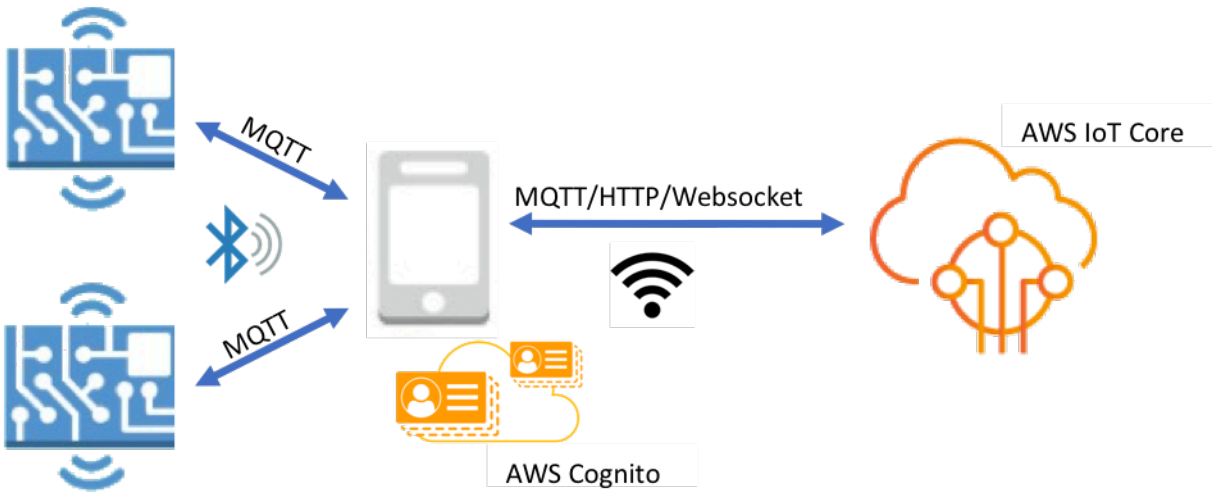
Dimensione del codice di BackOffAlgorithm (esempio generato con GCC per ARM Cortex-M)		
File	Con ottimizzazione -O1	Con ottimizzazione -Os
backoff_algorithm.c	0,1 K	0,1 K
Stime totali	0,1 K	0,1 K

Libreria Bluetooth Low Energy

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Panoramica

FreerTOS supporta la pubblicazione e la sottoscrizione di argomenti MQTT tramite Bluetooth Low Energy tramite un dispositivo proxy, come un telefono cellulare. Con la libreria FreerTOS Bluetooth Low Energy, il microcontrollore può comunicare in modo sicuro con il broker MQTT. AWS IoT

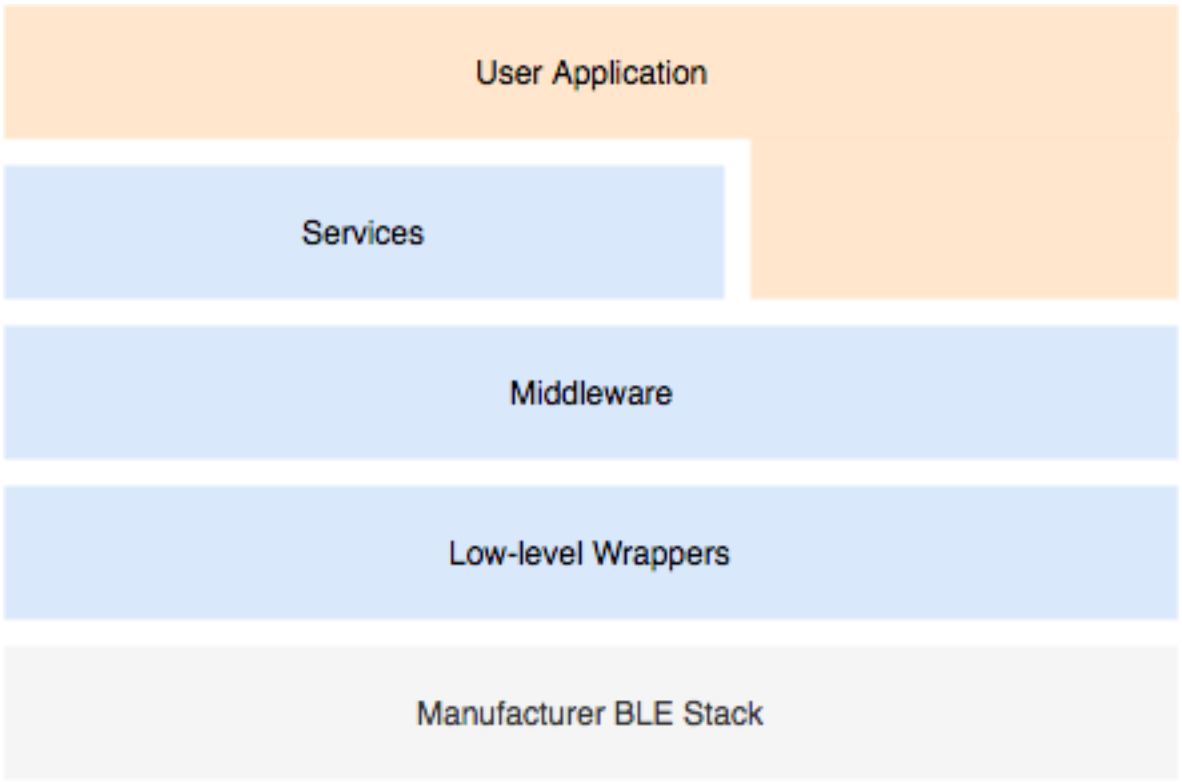


Utilizzando Mobile SDKs for FreeRTOS Bluetooth Devices, puoi scrivere applicazioni mobili native che comunicano con le applicazioni integrate sul tuo microcontrollore tramite Bluetooth Low Energy. Per ulteriori informazioni sul cellulare SDKs, consulta. Mobile SDKs per dispositivi Bluetooth FreerTOS

La libreria FreerTOS Bluetooth Low Energy include servizi per la configurazione di reti Wi-Fi, il trasferimento di grandi quantità di dati e la fornitura di astrazioni di rete tramite Bluetooth Low Energy. La libreria FreerTOS Bluetooth Low Energy include anche middleware e di APIs livello inferiore per un controllo più diretto sullo stack Bluetooth Low Energy.

Architettura

Tre livelli costituiscono la libreria FreerTOS Bluetooth Low Energy: servizi, middleware e wrapper di basso livello.



Servizi

Il livello di servizi FreerTOS Bluetooth Low Energy è costituito da quattro servizi Generic Attributes (GATT) che sfruttano il APIs middleware: Device Information, Wi-Fi Provisioning, Network Abstraction e Large Object Transfer.

Informazioni sul dispositivo

Il servizio Informazioni dispositivo raccoglie le informazioni sul microcontroller, incluse:

- La versione di FreerTOS utilizzata dal tuo dispositivo.
- L' AWS IoT endpoint dell'account per il quale è registrato il dispositivo.
- Unità di trasmissione massima (MTU) Bluetooth Low Energy

Provisioning Wi-Fi

Il servizio di provisioning Wi-Fi consente ai microcontroller con funzionalità Wi-Fi di eseguire le seguenti operazioni:

- Elencare reti nell'intervallo.
- Salvare reti e credenziali di rete nella memoria flash.

- Impostare priorità di rete.
- Eliminare reti e credenziali di rete dalla memoria flash.

Astrazione di rete

Il servizio Astrazione di rete estrae il tipo di connessione di rete per le applicazioni. Un'API comune interagisce con lo stack hardware dei servizi Wi-Fi, Ethernet e Bluetooth Low Energy del dispositivo in modo da consentire la compatibilità di un'applicazione con più tipi di connessione.

Trasferimento di oggetti di grandi dimensioni

Il servizio Trasferimento di oggetti di grandi dimensioni invia e riceve dati a livello di client. Altri servizi, ad esempio Provisioning Wi-Fi e Astrazione di rete, utilizzano il servizio Trasferimento di oggetti di grandi dimensioni per inviare e ricevere dati. È inoltre possibile utilizzare l'API di trasferimento di oggetti di grandi dimensioni per l'interazione con il servizio direttamente.

Middleware

Il middleware FreerTOS Bluetooth Low Energy è un'astrazione dal livello inferiore. APIs Il middleware costituisce un'interfaccia più intuitiva APIs per lo stack Bluetooth Low Energy.

Utilizzando il middleware APIs, è possibile registrare più callback, su più livelli, su un singolo evento. L'inizializzazione del middleware Bluetooth Low Energy comporta l'inizializzazione dei servizi e l'avvio della pubblicità.

Sottoscrizione ai callback flessibile

Supponiamo che l'hardware Bluetooth Low Energy si disconnetta e che il servizio MQTT su Bluetooth Low Energy debba rilevare la disconnessione. Anche un'applicazione scritta dall'utente potrebbe dover rilevare lo stesso evento di disconnessione. Il middleware Bluetooth Low Energy è in grado di instradare l'evento a diverse parti del codice in cui hai registrato i callback, senza che i livelli più elevati competano tra loro per risorse di basso livello.

Wrapper di basso livello

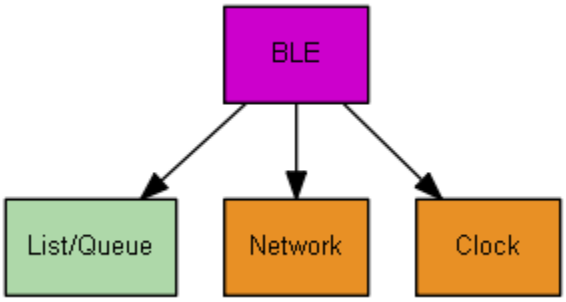
I wrapper FreerTOS Bluetooth Low Energy di basso livello sono un'astrazione dello stack Bluetooth Low Energy del produttore. I wrapper di basso livello offrono un set comune per il controllo diretto dell'hardware. APIs I livelli bassi APIs ottimizzano l'utilizzo della RAM, ma hanno funzionalità limitate.

Utilizza il servizio Bluetooth Low Energy APIs per interagire con i servizi Bluetooth Low Energy. Il servizio APIs richiede più risorse rispetto a quello di basso livello APIs.

Dipendenze e requisiti

La libreria Bluetooth Low Energy è caratterizzata dalle seguenti dipendenze dirette:

- Libreria Linear Container
- Una piattaforma che si interfaccia con il sistema operativo per la gestione dei thread, i timer, le funzioni di clock e l'accesso di rete.



Solo il servizio Wi-Fi Provisioning ha dipendenze dalla libreria FreerTOS:

Servizio GATT	Dipendenza
Provisioning Wi-Fi	Libreria Wi-Fi

Per comunicare con il broker AWS IoT MQTT, è necessario disporre di un AWS account e registrare i dispositivi come oggetti. AWS IoT Per ulteriori informazioni sulla configurazione, consulta la Guida per gli sviluppatori di AWS IoT.

FreerTOS Bluetooth Low Energy utilizza Amazon Cognito per l'autenticazione degli utenti sul tuo dispositivo mobile. Per utilizzare i servizi proxy MQTT, devi creare un'identità e pool di utenti Amazon Cognito. A ogni Amazon Cognito Identity deve essere associata la policy appropriata. Per ulteriori informazioni, consulta la Guida per sviluppatori di Amazon Cognito .

File di configurazione della libreria

Le applicazioni che utilizzano il servizio FreerTOS MQTT over Bluetooth Low Energy devono fornire `iot_ble_config.h` un file di intestazione, in cui sono definiti i parametri di configurazione. I parametri di configurazione non definiti assumono i valori predefiniti specificati in `iot_ble_config_defaults.h`.

Alcune parametri di configurazione importanti includono:

`IOT_BLE_ADD_CUSTOM_SERVICES`

Consente agli utenti di creare i propri servizi.

`IOT_BLE_SET_CUSTOM_ADVERTISEMENT_MSG`

Consente agli utenti di personalizzare le pubblicità e analizzare i messaggi di risposta.

Per ulteriori informazioni, consulta la Documentazione di riferimento sull'API Bluetooth Low Energy.

Ottimizzazione

Durante l'ottimizzazione delle prestazioni della scheda, considera quanto segue:

- Il basso livello APIs utilizza meno RAM, ma offre funzionalità limitate.
- Puoi impostare il parametro `bleconfigMAX_NETWORK` nel file di intestazione `iot_ble_config.h` su un valore più basso per ridurre la quantità di stack utilizzata.
- Puoi incrementare le dimensioni MTU fino al valore massimo per limitare il buffering dei messaggi e consentire un'esecuzione del codice più veloce e meno consumo di RAM.

Limitazioni d'uso

Per impostazione predefinita, la libreria FreerTOS Bluetooth Low Energy imposta `eBTpropertySecureConnectionOnly` la proprietà su TRUE, che colloca il dispositivo in modalità Solo connessioni sicure. Come specificato dal Bluetooth Core Specification v5.0, Vol 3, Part C, 10.2.4, quando un dispositivo è in modalità Secure Connections Only (Solo connessioni sicure), è richiesto il livello di modalità di sicurezza 1 LE massimo, livello 4, per accedere a qualsiasi attributo che dispone di autorizzazioni superiori a quelle del livello modalità di sicurezza 1 LE minimo, livello 1. Al livello 4 della modalità di sicurezza 1 LE, un dispositivo deve disporre di funzionalità di input e output per confronti numerici.

Di seguito sono riportate le modalità supportate e le relative proprietà associate:

Modalità 1, livello 1 (nessuna sicurezza)

```
/* 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 )
```

Modalità 1, livello 2 (associazione non autenticata con crittografia)

```
#define IOT_BLE_ENABLE_NUMERIC_COMPARISON       ( 0 )
#define IOT_BLE_ENABLE_SECURE_CONNECTION        ( 0 )
#define IOT_BLE_INPUT_OUTPUT                    ( eBTIONone )
```

Modalità 1, livello 3 (associazione autenticata con crittografia)

Questa modalità non è supportata.

Modalità 1, livello 4 (associazione autenticata delle connessioni sicure LE con crittografia)

Questa modalità è supportata per impostazione predefinita.

Per informazioni sulle modalità di sicurezza LE, consulta il documento Bluetooth Core Specification v5.0, Vol 3, Part C, 10.2.1.

Inizializzazione

Se l'applicazione interagisce con lo stack Bluetooth Low Energy tramite middleware, occorre solo inizializzare il middleware. Il middleware si occupa dell'inizializzazione dei livelli inferiori dello stack.

Middleware

Per inizializzare il middleware

1. Prima di eseguire una chiamata all'API del middleware Bluetooth Low Energy, occorre inizializzare qualsiasi driver dell'hardware Bluetooth Low Energy.
2. Abilitare Bluetooth Low Energy.
3. Inizializzare il middleware mediante `IotBLE_Init()`.

> **Note**
>
> Questo passaggio di inizializzazione non è necessario se si eseguono le demo. AWS L'inizializzazione delle demo viene gestita dal network manager, disponibile in *freertos*`/demos/network_manager`.

Di basso livello APIs

Se non desideri utilizzare i servizi FreerTOS Bluetooth Low Energy GATT, puoi bypassare il middleware e interagire direttamente con il livello basso per risparmiare risorse. APIs

Per inizializzare il livello basso APIs

1. Inizializza tutti i driver hardware Bluetooth Low Energy prima di chiamare il. APIs L'inizializzazione del driver non fa parte del livello Bluetooth Low Energy di basso livello. APIs
2. L'API Bluetooth Low Energy di basso livello fornisce una chiamata di abilitazione/disabilitazione allo stack Bluetooth Low Energy per l'ottimizzazione di alimentazione e risorse. Prima di chiamare il APIs, è necessario abilitare Bluetooth Low Energy.

   ```
   const BTInterface_t * pxIface = BTGetBluetoothInterface();
   xStatus = pxIface->pxEnable( 0 );
   ```

3. Il gestore Bluetooth contiene elementi comuni APIs sia a Bluetooth Low Energy che a Bluetooth classic. I callback per il responsabile comune devono essere inizializzati per secondi.

   ```
   xStatus = xBTInterface.pxBTInterface->pxBtManagerInit( &xBTManagerCb );
   ```

4. L'adattatore Bluetooth Low Energy si inserisce sopra l'API comune. È necessario inizializzare i relativi callback come è stata inizializzata l'API comune.

   ```
   xBTInterface.pxBTLeAdapterInterface = ( BTBleAdapter_t * )
    xBTInterface.pxBTInterface->pxGetLeAdapter();
   xStatus = xBTInterface.pxBTLeAdapterInterface-
   >pxBleAdapterInit( &xBTBleAdapterCb );
   ```

5. Registrare la nuova applicazione utente.

   ```
   xBTInterface.pxBTLeAdapterInterface->pxRegisterBleApp( pxAppUuid );
   ```

6. Inizializzare i callback ai server GATT.

```
xBTInterface.pxGattServerInterface = ( BTGattServerInterface_t * )
 xBTInterface.pxBTLeAdapterInterface->ppvGetGattServerInterface();
xBTInterface.pxGattServerInterface->pxGattServerInit( &xBTGattServerCb );
```

Dopo aver inizializzato l'adattatore Bluetooth Low Energy, è possibile aggiungere un server GATT. Puoi registrare un solo server GATT per volta.

```
xStatus = xBTInterface.pxGattServerInterface->pxRegisterServer( pxAppUuid );
```

7. Impostare le proprietà dell'applicazione come Secure Connection Only (Solo connessione sicura) e dimensioni MTU.

```
xStatus = xBTInterface.pxBTInterface-
>pxSetDeviceProperty( &pxProperty[ usIndex ] );
```

Riferimento API

Per informazioni di riferimento complete sull'API, consulta la Documentazione di riferimento sull'API Bluetooth Low Energy.

Esempio di utilizzo

Gli esempi riportati di seguito illustrano come utilizzare la libreria Bluetooth Low Energy per la pubblicità e la creazione di nuovi servizi. Per le applicazioni demo complete di FreerTOS Bluetooth Low Energy, vedi Applicazioni demo Bluetooth Low Energy.

Pubblicità

1. Nell'applicazione impostare l'UUID della pubblicità:

```
static const BTUuid_t _advUUID =
{
    .uu.uu128 = IOT_BLE_ADVERTISING_UUID,
    .ucType   = eBTuuidType128
};
```

2. Definire la funzione di callback `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;
}
```

La funzione di callback invia l'UUID nel messaggio pubblicitario e il nome completo nella risposta di analisi (scan response).

3. Aprire `vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/iot_ble_config.h` e impostare `IOT_BLE_SET_CUSTOM_ADVERTISEMENT_MSG` su `1`. In questo modo viene attivata la funzione di callback `IotBle_SetCustomAdvCb`.

Aggiunta di un nuovo servizio

Per esempi di servizi completi, consulta *`freertos`*`/.../ble/services`.

1. Crea UUIDs per le caratteristiche e i descrittori del servizio:

```
#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. Creare un buffer per registrare gli handle delle caratteristiche e dei descrittori:

```
static uint16_t usHandlesBuffer[egattDemoNbAttributes];
```

3. Creare l'attributo tabella. Per salvare RAM, definire la tabella come `const`.

> **Important**
>
> Creare sempre gli attributi nell'ordine indicato, con il servizio come primo attributo.

```
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. Creare un array di funzioni di callback. L'array di funzioni di callback deve avere lo stesso ordine dell'array di tabelle definito in precedenza.

   Ad esempio, se la funzione `vReadCounter` viene attivata quando si accede alla funzione `xCharCounterUUID_TYPE` e la funzione `vWriteCommand` viene attivata quando si accede alla funzione `xCharControlUUID_TYPE`, definire l'array come segue:

```
static const IotBleAttributeEventCallback_t pxCallBackArray[egattDemoNbAttributes]
 =
    {
  NULL,
  vReadCounter,
  vEnableNotification,
  vWriteCommand
};
```

5. Creare il servizio.

```
static const BTService_t xGattDemoService =
{
  .xNumberOfAttributes = egattDemoNbAttributes,
  .ucInstId = 0,
  .xType = eBTServiceTypePrimary,
  .pusHandlesBuffer = usHandlesBuffer,
  .pxBLEAttributes = (BTAttribute_t *)pxAttributeTable
};
```

6. Richiamare l'API `IotBle_CreateService` con la struttura creata nella fase precedente. Il middleware sincronizza la creazione di tutti i servizi. Pertanto, i nuovi servizi devono già essere definiti quando viene attivata la funzione di callback `IotBle_AddCustomServicesCb`.

a. Impostare `IOT_BLE_ADD_CUSTOM_SERVICES` su 1 in `vendors/vendor/boards/board/aws_demos/config_files/iot_ble_config.h`.

b. Crea IotBle _ AddCustomServicesCb nella tua applicazione:

```
void IotBle_AddCustomServicesCb(void)
{
    BTStatus_t xStatus;
    /* Select the handle buffer. */
    xStatus = IotBle_CreateService( (BTService_t *)&xGattDemoService,
 (IotBleAttributeEventCallback_t *)pxCallBackArray );
}
```

Portabilità

Periferica di input e output utente

Una connessione sicura richiede input e output per confronti numerici. L'evento `eBLENumericComparisonCallback` può essere registrato utilizzando la gestione eventi:

```
xEventCb.pxNumericComparisonCb = &prvNumericComparisonCb;
xStatus = BLE_RegisterEventCb( eBLENumericComparisonCallback, xEventCb );
```

La periferica deve visualizzare la passkey numerica e prendere il risultato del confronto come un input.

Implementazione delle API per la portabilità

Per portare FreerTOS su un nuovo target, è necessario implementarne APIs alcuni per il servizio Wi-Fi Provisioning e la funzionalità Bluetooth Low Energy.

Bluetooth Low Energy APIs

Per utilizzare il middleware FreerTOS Bluetooth Low Energy, è necessario implementarne alcuni. APIs

APIs comune tra GAP per Bluetooth Classic e GAP per Bluetooth Low Energy

- `pxBtManagerInit`
- `pxEnable`

- pxDisable
- pxGetDeviceProperty
- pxSetDeviceProperty (Tutte le opzioni sono obbligatorie tranne eBTpropertyRemoteRssi ed eBTpropertyRemoteVersionInfo)
- pxPair
- pxRemoveBond
- pxGetConnectionState
- pxPinReply
- pxSspReply
- pxGetTxpower
- pxGetLeAdapter
- pxDeviceStateChangedCb
- pxAdapterPropertiesCb
- pxSspRequestCb
- pxPairingStateChangedCb
- pxTxPowerCb

APIs specifico per GAP per Bluetooth Low Energy

- pxRegisterBleApp
- pxUnregisterBleApp
- pxBleAdapterInit
- pxStartAdv
- pxStopAdv
- pxSetAdvData
- pxConnParameterUpdateRequest
- pxRegisterBleAdapterCb
- pxAdvStartCb
- pxSetAdvDataCb
- pxConnParameterUpdateRequestCb
- pxCongestionCb

Server 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`

Per ulteriori informazioni sul porting della libreria Bluetooth Low Energy FreerTOS sulla tua piattaforma, consulta Porting the Bluetooth Low Energy Library nella FreerTOS Porting Guide.

Mobile SDKs per dispositivi Bluetooth FreerTOS

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Puoi utilizzare Mobile SDKs for FreeRTOS Bluetooth Devices per creare applicazioni mobili che interagiscono con il tuo microcontrollore tramite Bluetooth Low Energy. The Mobile SDKs può anche comunicare con AWS i servizi, utilizzando Amazon Cognito per l'autenticazione degli utenti.

Android SDK per dispositivi Bluetooth FreerTOS

Usa Android SDK per dispositivi Bluetooth FreerTOS per creare applicazioni mobili Android che interagiscono con il tuo microcontrollore tramite Bluetooth Low Energy. L'SDK è disponibile su. GitHub

Per installare l'SDK Android per i dispositivi Bluetooth FreerTOS, segui le istruzioni per «Configurare l'SDK» nel file README.md del progetto.

Per informazioni sulla configurazione e l'esecuzione della demo dell'applicazione per dispositivi mobili inclusa con l'SDK, consulta Prerequisiti e Applicazione dimostrativa FreerTOS Bluetooth Low Energy Mobile SDK.

SDK iOS per dispositivi Bluetooth FreerTOS

Usa l'SDK iOS per dispositivi Bluetooth FreerTOS per creare applicazioni mobili iOS che interagiscono con il tuo microcontrollore tramite Bluetooth Low Energy. L'SDK è disponibile su. GitHub

Per installare l'SDK iOS

1. Installare CocoaPods:

```
$ gem install cocoapods
$ pod setup
```

> **Note**
>
> Potrebbe essere necessario utilizzare `sudo` per l'installazione CocoaPods.

2. Installa l'SDK con CocoaPods (aggiungilo al tuo podfile):

```
$ pod 'FreeRTOS', :git => 'https://github.com/aws/amazon-freertos-ble-ios-sdk.git'
```

Per informazioni sulla configurazione e l'esecuzione della demo dell'applicazione per dispositivi mobili inclusa con l'SDK, consulta Prerequisiti e Applicazione dimostrativa FreerTOS Bluetooth Low Energy Mobile SDK.

I/O comuni

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Panoramica

In generale, i driver dei dispositivi sono indipendenti dal sistema operativo sottostante e sono specifici per una determinata configurazione hardware. Un HAL (Hardware Abstraction Layer) fornisce un'interfaccia comune tra driver e codice applicativo di livello superiore. L'HAL estrae i dettagli del funzionamento di un driver specifico e fornisce un'API uniforme per controllare tali dispositivi. È possibile utilizzarla APIs per accedere a vari driver di dispositivo su più schede di riferimento basate su microcontrollori (MCU).

L'I/O comune di FreerTOS funge da livello di astrazione hardware. Fornisce un set di standard APIs per l'accesso ai dispositivi seriali comuni sulle schede di riferimento supportate. Questi comuni APIs comunicano e interagiscono con queste periferiche e consentono al codice di funzionare su più

piattaforme. Senza I/O comune, la scrittura di codice per funzionare con dispositivi di basso livello è specifico dei fornitori di componenti.

Periferiche supportate

- UART
- SPI
- I2C

Funzionalità supportate

- Lettura/scrittura sincrona: la funzione non ritorna finché non viene trasferita la quantità di dati richiesta.
- Lettura/scrittura asincrona: la funzione ritorna immediatamente e il trasferimento dei dati avviene in modo asincrono. Al termine dell'operazione, viene richiamato un callback utente registrato.

Specifico delle periferiche

- I2C: combina più operazioni in un'unica transazione. Utilizzato per eseguire operazioni di scrittura e quindi di lettura in una transazione.
- SPI: trasferimento di dati tra primario e secondario, il che significa che la scrittura e la lettura avvengono contemporaneamente.

Portabilità

Per ulteriori informazioni, consulta la FreerTOS Porting Guide.

AWS IoT Device Defender biblioteca

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Introduzione

Puoi utilizzare la AWS IoT Device Defender libreria per inviare metriche di sicurezza dai tuoi dispositivi IoT a AWS IoT Device Defender. Puoi utilizzarla AWS IoT Device Defender per monitorare continuamente queste metriche di sicurezza dai dispositivi per rilevare eventuali deviazioni da quello che hai definito come comportamento appropriato per ciascun dispositivo. Se qualcosa non sembra corretto, AWS IoT Device Defender invia un avviso in modo che tu possa intervenire per risolvere il problema. Le interazioni con AWS IoT Device Defender utilizzano MQTT, un protocollo di pubblicazione e sottoscrizione leggero. Questa libreria fornisce un'API per comporre e riconoscere le stringhe di argomenti MQTT utilizzate da. AWS IoT Device Defender

Per ulteriori informazioni, consulta la sezione AWS IoT Device Defender nella Guida per gli sviluppatori di AWS IoT .

La libreria è scritta in C e progettata per essere conforme a ISO C90 e MISRA C:2012. La libreria non ha dipendenze da altre librerie diverse dalla libreria C standard. Inoltre non ha alcuna dipendenza dalla piattaforma, come il threading o la sincronizzazione. Può essere utilizzato con qualsiasi libreria MQTT e qualsiasi libreria JSON o CBOR. La libreria dispone di prove che dimostrano l'uso sicuro della memoria e l'assenza di allocazione di heap, il che la rende adatta ai microcontrollori IoT, ma anche completamente portabile su altre piattaforme.

La AWS IoT Device Defender libreria può essere utilizzata liberamente ed è distribuita con la licenza open source MIT.

Dimensione del codice di AWS IoT Device Defender (esempio generato con GCC per ARM Cortex-M)		
File	Con ottimizzazione -O1	Con ottimizzazione -Os
defender.c	1,1 K	0,6 K
Stime totali	1,1 K	0,6 K

AWS IoT Greengrass Libreria Discovery

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Panoramica

La libreria AWS IoT Greengrass Discovery viene utilizzata dai dispositivi a microcontrollore per scoprire un core Greengrass sulla rete. Utilizzando AWS IoT Greengrass Discovery APIs, il dispositivo può inviare messaggi a un core Greengrass dopo aver trovato l'endpoint del core.

Dipendenze e requisiti

Per utilizzare la libreria Greengrass Discovery, è necessario creare un oggetto in AWS IoT, inclusi un certificato e una politica. Per ulteriori informazioni, consulta Nozioni di base su AWS IoT.

È necessario impostare i valori per le seguenti costanti nel file *`freertos`*`/demos/include/aws_clientcredential.h`:

`clientcredentialMQTT_BROKER_ENDPOINT`

Il tuo AWS IoT endpoint.

`clientcredentialIOT_THING_NAME`

Nome dell'oggetto IoT.

`clientcredentialWIFI_SSID`

Il SSID della rete Wi-Fi.

`clientcredentialWIFI_PASSWORD`

Password del Wi-Fi.

`clientcredentialWIFI_SECURITY`

Il tipo di sicurezza utilizzato dalla rete Wi-Fi.

È anche necessario impostare i valori per le seguenti costanti nel file *freertos*/demos/include/aws_clientcredential_keys.h:

`keyCLIENT_CERTIFICATE_PEM`

Il file PEM del certificato associato all'oggetto.

`keyCLIENT_PRIVATE_KEY_PEM`

Il file PEM della chiave privata associato all'oggetto.

È necessario che nella console siano configurati un dispositivo core e un gruppo Greengrass. Per ulteriori informazioni, consulta Guida introduttiva a AWS IoT Greengrass.

Sebbene la libreria CoreMQTT non sia richiesta per la connettività Greengrass, consigliamo vivamente di installarla. La libreria può essere utilizzata per comunicare con il core Greengrass dopo che è stato individuato.

Riferimento API

Per informazioni di riferimento complete sull'API, consulta Documentazione di riferimento sull'API Greengrass.

Esempio di utilizzo

Flusso di lavoro di Greengrass

Il dispositivo MCU avvia il processo di scoperta richiedendo AWS IoT un file JSON che contiene i parametri di connettività di base di Greengrass. Sono disponibili due metodi per il recupero dei parametri di connettività del core Greengrass dal file JSON:

- La selezione automatica scorre tutti i core Greengrass elencati nel file JSON e si connette al primo disponibile.
- La selezione manuale utilizza le informazioni in `aws_ggd_config.h` per connettersi al core Greengrass specificato.

Come usare l'API Greengrass

Tutte le opzioni di configurazione predefinite per l'API Greengrass sono definite in `aws_ggd_config_defaults.h`.

Se è presente un solo core Greengrass, chiamare `GGD_GetGGCIPandCertificate` per richiedere al file JSON informazioni di connettività del core Greengrass. Quando `GGD_GetGGCIPandCertificate` viene restituito, il parametro `pcBuffer` contiene il testo del file JSON. Il parametro `pxHostAddressData` contiene l'indirizzo IP e la porta del core Greengrass a cui è possibile connettersi.

Per ulteriori opzioni di personalizzazione, come l'allocazione dinamica dei certificati, è necessario chiamare quanto segue: APIs

`GGD_JSONRequestStart`

Effettua una richiesta HTTP GET AWS IoT per avviare la richiesta di rilevamento per scoprire un core Greengrass. `GD_SecureConnect_Send`viene utilizzato per inviare la richiesta a. AWS IoT

`GGD_JSONRequestGetSize`

Ottiene le dimensioni del file JSON dalla risposta HTTP.

`GGD_JSONRequestGetFile`

Ottiene l'oggetto stringa JSON. `GGD_JSONRequestGetSize` e `GGD_JSONRequestGetFile` utilizzano `GGD_SecureConnect_Read` per ottenere i dati JSON dal socket. È necessario chiamare `GGD_JSONRequestStart`, `GGD_SecureConnect_Send` e `GGD_JSONRequestGetSize` per ricevere i dati JSON da AWS IoT.

`GGD_GetIPandCertificateFromJSON`

Estrae l'indirizzo IP e il certificato core Greengrass dai dati JSON. È possibile attivare la selezione automatica impostando `xAutoSelectFlag` su `True`. La selezione automatica trova il primo dispositivo core a cui il dispositivo FreeRTOS può connettersi. Per connettersi a un core Greengrass, chiamare la funzione `GGD_SecureConnect_Connect` trasmettendo l'indirizzo IP, la porta e il certificato del dispositivo core. Per utilizzare la selezione manuale, impostare i campi seguenti del parametro `HostParameters_t`:

`pcGroupName`

L'ID del gruppo Greengrass a cui appartiene il core. Puoi utilizzare il comando `aws greengrass list-groups` dell'interfaccia a riga di comando per trovare l'ID dei tuoi gruppi Greengrass.

`pcCoreAddress`

ARN del core Greengrass a cui ti stai connettendo.

Libreria CoreHTTP

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Libreria client HTTP C per piccoli dispositivi IoT (MCU o MPU di piccole dimensioni)

Introduzione

La libreria CoreHTTP è un'implementazione client di un sottoinsieme dello standard HTTP/1.1. Lo standard HTTP fornisce un protocollo stateless che funziona su TCP/IP e viene spesso utilizzato in sistemi informativi ipertestuali distribuiti e collaborativi.

La libreria CoreHTTP implementa un sottoinsieme dello standard del protocollo HTTP/1.1. Questa libreria è stata ottimizzata per un basso ingombro di memoria. La libreria fornisce un'API completamente sincrona in modo che le applicazioni possano gestire completamente la loro concorrenza. Utilizza solo buffer fissi, in modo che le applicazioni abbiano il controllo completo della loro strategia di allocazione della memoria.

La libreria è scritta in C e progettata per essere conforme a ISO C90 e MISRA C:2012. Le uniche dipendenze della libreria sono la libreria C standard e la versione LTS (v12.19.1) del parser http-parser di Node.js. La libreria dispone di prove che dimostrano l'uso sicuro della memoria e l'assenza di allocazione di heap, il che la rende adatta ai microcontrollori IoT, ma anche completamente portabile su altre piattaforme.

Quando si utilizzano connessioni HTTP nelle applicazioni IoT, si consiglia di utilizzare un'interfaccia di trasporto sicura, ad esempio un'interfaccia che utilizza il protocollo TLS, come dimostrato nella demo CoreHTTP (autenticazione reciproca).

Questa libreria può essere utilizzata liberamente ed è distribuita con la licenza open source MIT.

Dimensione del codice di CoreHTTP (esempio generato con GCC per ARM Cortex-M)		
File	Con ottimizzazione -O1	Con ottimizzazione -Os
core_http_client.c	3,2 K	2,6 K
http_parser.c (parser http)	15,7 K	13,0 K
Stime totali	18,9 K	15,6 K

libreria CoreJSON

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Introduzione

JSON (JavaScript Object Notation) è un formato di serializzazione dei dati leggibile dall'uomo. È ampiamente utilizzato per lo scambio di dati, ad esempio con il servizio AWS IoT Device Shadow, e fa parte di molti altri APIs, come l'API GitHub REST. JSON è mantenuto come standard da Ecma International.

La libreria CoreJSON fornisce un parser che supporta le ricerche di chiavi applicando rigorosamente la sintassi JSON Data Interchange standard ECMA-404. La libreria è scritta in C e progettata per essere conforme a ISO C90 e MISRA C:2012. Ha prove che dimostrano l'uso sicuro della memoria e l'assenza di allocazione dell'heap, il che lo rende adatto ai microcontrollori IoT, ma anche completamente portatile su altre piattaforme.

Uso della memoria

La libreria CoreJSON utilizza uno stack interno per tracciare le strutture annidate in un documento JSON. Lo stack esiste per la durata di una singola chiamata di funzione; non viene mantenuto. La dimensione dello stack può essere specificata definendo la macro`JSON_MAX_DEPTH`, che per impostazione predefinita è 32 livelli. Ogni livello consuma un singolo byte.

Dimensione del codice di CoreJSON (esempio generato con GCC per ARM Cortex-M)		
File	Con ottimizzazione -O1	Con ottimizzazione -Os
core_json.c	2,9K	2,4K
Stime totali	2,9 K	2,4K

libreria CoreMQTT

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Introduzione

La libreria CoreMQTT è un'implementazione client dello standard MQTT (Message Queue Telemetry Transport). Lo standard MQTT fornisce un protocollo di messaggistica di pubblicazione/sottoscrizione (o PubSub) leggero che funziona su TCP/IP e viene spesso utilizzato nei casi d'uso Machine to Machine (M2M) e Internet of Things (IoT).

La libreria CoreMQTT è conforme allo standard del protocollo MQTT 3.1.1. Questa libreria è stata ottimizzata per un basso ingombro di memoria. Il design di questa libreria abbraccia diversi casi d'uso, che vanno da piattaforme con risorse limitate che utilizzano solo messaggi QoS 0 MQTT PUBLISH a piattaforme ricche di risorse che utilizzano connessioni QoS 2 MQTT PUBLISH su TLS (Transport Layer Security). La libreria offre un menu di funzioni componibili, che possono essere scelte e combinate per adattarsi con precisione alle esigenze di un particolare caso d'uso.

La libreria è scritta in C e progettata per essere conforme a ISO C90 e MISRA C:2012. Questa libreria MQTT non ha dipendenze da alcuna libreria aggiuntiva ad eccezione delle seguenti:

- La libreria C standard
- Un'interfaccia di trasporto di rete implementata dal cliente
- (Opzionale) Una funzione temporale della piattaforma implementata dall'utente

La libreria è disaccoppiata dai driver di rete sottostanti mediante la fornitura di una semplice specifica dell'interfaccia di trasporto di invio e ricezione. L'autore dell'applicazione può selezionare un'interfaccia di trasporto esistente o implementarne una propria in base alle esigenze dell'applicazione.

La libreria fornisce un'API di alto livello per connettersi a un broker MQTT, iscriversi/annullare l'iscrizione a un argomento, pubblicare un messaggio su un argomento e ricevere messaggi in arrivo. Questa API utilizza l'interfaccia di trasporto sopra descritta come parametro e la utilizza per inviare e ricevere messaggi da e verso il broker MQTT.

La libreria presenta anche 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 di basso livello che possono essere utilizzate insieme a qualsiasi API di livello di trasporto disponibile, come i socket, per inviare e ricevere messaggi da e verso il broker.

Quando si utilizzano connessioni MQTT nelle applicazioni IoT, si consiglia di utilizzare un'interfaccia di trasporto sicura, ad esempio una che utilizza il protocollo TLS.

Questa libreria MQTT non ha dipendenze dalla piattaforma, come il threading o la sincronizzazione. Questa libreria dispone di prove che dimostrano l'uso sicuro della memoria e l'assenza di allocazione di heap, il che la rende adatta ai microcontrollori IoT, ma anche completamente portabile su altre piattaforme. Può essere utilizzata liberamente ed è distribuita con la licenza open source MIT.

Dimensione del codice di CoreMQTT (esempio generato con GCC per ARM Cortex-M)		
File	Con ottimizzazione -O1	Con ottimizzazione -Os
core_mqtt.c	3,0 K	2,6K
core_mqtt_state.c	1,4K	1,1 K
core_mqtt_serializer.c	2,5K	2,0K
Stime totali	6,9 K	5,7K

Libreria dell'agente OTA

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Panoramica

Libreria dell'agente OTAConsente di gestire la notifica, il download e la verifica degli aggiornamenti del firmware per i dispositivi FreerTOS utilizzando HTTP o MQTT come protocollo. Utilizzando la libreria dell'agente OTA, è possibile separare logicamente gli aggiornamenti firmware e l'applicazione in esecuzione sul dispositivo. L'agente OTA può condividere una connessione di rete con l'applicazione. Mediante la condivisione di una connessione di rete, è possibile ottenere un potenziale risparmio di una notevole quantità di RAM. Inoltre, la libreria dell'agente OTA consente di definire la logica specifica dell'applicazione per il testing, il commit o il rollback di un aggiornamento del firmware.

Per ulteriori informazioni sulla configurazione degli aggiornamenti OTA con FreerTOS, vedere. Aggiornamenti FreerTOS Over-the-Air

Funzionalità

Di seguito è riportata l'interfaccia completa dell'agente OTA:

`OTA_AgentInit`

Inizializza l'agente OTA. Il chiamante fornisce il contesto del protocollo di messaggistica, un callback opzionale e un timeout.

`OTA_AgentShutdown`

Libera le risorse dopo l'utilizzo dell'agente OTA.

`OTA_GetAgentState`

Ritorna lo stato corrente dell'agente OTA.

`OTA_ActivateNewImage`

Attiva l'immagine del firmware più recente del microcontroller ricevuta tramite OTA. (Lo stato del processo dettagliato ora dovrebbe essere self-test).

`OTA_SetImageState`

Imposta lo stato della convalida dell'immagine del firmware del microcontroller attualmente in esecuzione (testing, accettato o rifiutato).

`OTA_GetImageState`

Ottiene lo stato della convalida dell'immagine del firmware del microcontroller attualmente in esecuzione (testing, accettato o rifiutato).

`OTA_CheckForUpdate`

Richiede l'aggiornamento OTA successivo disponibile dal servizio aggiornamenti OTA.

`OTA_Suspend`

Sospendere tutte le operazioni dell'agente OTA.

`OTA_Resume`

Riprendi le operazioni dell'agente OTA.

Riferimento API

Per ulteriori informazioni, consulta il riferimento all'API di OTA Agent.

Esempio di utilizzo

Un'applicazione per un dispositivi compatibili con OTA che utilizza il protocollo MQTT consente di gestire l'agente OTA utilizzando la seguente sequenza di chiamate API.

1. Connect al broker AWS IoT MQTT. Per ulteriori informazioni, consulta libreria CoreMQTT.
2. Inizializzare l'agente OTA richiamando `OTA_AgentInit`. L'applicazione potrebbe definire una funzione di callback OTA personalizzata oppure utilizzare il callback predefinito specificando un puntatore NULL alla funzione di callback. È necessario anche fornire un timeout di inizializzazione.

Il callback implementa la logica specifica dell'applicazione che viene eseguita dopo il completamento di un processo di aggiornamento OTA. Il timeout definisce il tempo di attesa per il completamento dell'inizializzazione.

3. Se `OTA_AgentInit` ha raggiunto il timeout prima che l'agente fosse pronto, puoi richiamare `OTA_GetAgentState` per verificare che l'agente sia stato inizializzato e che stia funzionando come previsto.
4. Quando l'aggiornamento OTA è completo, FreerTOS richiama il callback di completamento del lavoro con uno dei seguenti eventi`accepted`:, o. `rejected` `self test`
5. Se la nuova immagine del firmware è stata rifiutata (ad esempio a causa di un errore di convalida), l'applicazione può in genere ignorare la notifica e attendere l'aggiornamento successivo.
6. Se l'aggiornamento è valido ed è stato contrassegnato come accettato, chiama `OTA_ActivateNewImage` per ripristinare il dispositivo e avviare la nuova immagine del firmware.

Portabilità

Per informazioni sulla portabilità della funzionalità OTA sulla tua piattaforma, consulta Porting the OTA Library nella FreerTOS Porting Guide.

Uso della memoria

Dimensione del codice di AWS IoT OTA (esempio generato con GCC per ARM Cortex-M)		
File	Con ottimizzazione -O1	Con ottimizzazione -Os
ota.c	8,0K	7,2K
ota_interface.	0,1 K	0,1 K
ota_base64.c	0,6K	0,6 K
ota_mqtt.c	2,3K	2,2 K
ota_cbor.c	0,8K	0,6 K
ota_http.c	0,3K	0,3 K

Dimensione del codice di AWS IoT OTA (esempio generato con GCC per ARM Cortex-M)		
File	Con ottimizzazione -O1	Con ottimizzazione -Os
Stime totali	12,1 K	11,0 K

PKCS11 libreria principale

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Panoramica

Il Public Key Cryptography Standard #11 definisce un'API indipendente dalla piattaforma per gestire e utilizzare i token crittografici. PKCS #11 si riferisce all'API definita dallo standard e allo standard stesso. L'API crittografica PKCS #11 astrae l'archiviazione delle chiavi, le proprietà get/set per gli oggetti crittografici e la semantica delle sessioni. È ampiamente utilizzata per manipolare oggetti crittografici comuni ed è importante perché le funzioni che specifica consentono al software applicativo di utilizzare, creare, modificare ed eliminare oggetti crittografici, senza mai esporre tali oggetti alla memoria dell'applicazione. Ad esempio, le integrazioni di riferimento AWS FreerTOS utilizzano un piccolo sottoinsieme dell'API PKCS #11 per accedere alla chiave segreta (privata) necessaria per creare una connessione di rete autenticata e protetta dal protocollo Transport Layer Security (TLS) senza che l'applicazione «veda» la chiave.

La PKCS11 libreria principale contiene un'implementazione fittizia basata su software dell'interfaccia (API) PKCS #11 che utilizza la funzionalità crittografica fornita da Mbed TLS. L'utilizzo di un software mock consente uno sviluppo e una flessibilità rapidi, ma è previsto che il mock venga sostituito con un'implementazione specifica per l'archiviazione sicura delle chiavi utilizzata nei dispositivi di produzione. In genere, i fornitori di criptoprocessori sicuri, come Trusted Platform Module (TPM), Hardware Security Module (HSM), Secure Element o qualsiasi altro tipo di enclave hardware sicura, distribuiscono un'implementazione PKCS #11 con l'hardware. Lo scopo della mock library core PKCS11 software only è quindi quello di fornire un'implementazione PKCS #11 non specifica per

l'hardware che consenta la prototipazione e lo sviluppo rapidi prima di passare a un'implementazione PKCS #11 specifica per criptoprocessori nei dispositivi di produzione.

Viene implementato solo un sottoinsieme dello standard PKCS #11, con particolare attenzione alle operazioni che coinvolgono chiavi asimmetriche, generazione di numeri casuali e hashing. I casi d'uso mirati includono la gestione di certificati e chiavi per l'autenticazione TLS e la verifica della firma con firma in codice su piccoli dispositivi integrati. Guarda il file `pkcs11.h` (ottenuto da OASIS, il corpo standard) nel repository del codice sorgente di FreerTOS. Nell'implementazione di riferimento di FreerTOS, le chiamate API PKCS #11 vengono effettuate dall'interfaccia helper TLS per eseguire l'autenticazione del client TLS durante. `SOCKETS_Connect` Le chiamate API PKCS #11 vengono effettuate anche dal nostro flusso di lavoro monouso di provisioning per gli sviluppatori per importare un certificato client TLS e una chiave privata per l'autenticazione nel broker MQTT. AWS IoT Questi due casi d'uso, il provisioning e l'autenticazione del client TLS, richiedono l'implementazione solo di un piccolo sottoinsieme dello standard di interfaccia PKCS #11.

Funzionalità

Viene utilizzato il seguente sottoinsieme di PKCS #11. L'ordine dell'elenco è pressappoco quello con cui le routine vengono chiamate per il provisioning, l'autenticazione client TLS e la pulizia. Per descrizioni dettagliate delle funzioni, consultate la documentazione PKCS #11 fornita dal comitato standard.

Configurazione generale e API di disattivazione

- `C_Initialize`
- `C_Finalize`
- `C_GetFunctionList`
- `C_GetSlotList`
- `C_GetTokenInfo`
- `C_OpenSession`
- `C_CloseSession`
- `C_Login`

API di provisioning

- `C_CreateObject CKO_PRIVATE_KEY` (per chiave privata dispositivo)

- `C_CreateObject CKO_CERTIFICATE` (per certificato dispositivo e certificato di verifica del codice)
- `C_GenerateKeyPair`
- `C_DestroyObject`

Autenticazione client

- `C_GetAttributeValue`
- `C_FindObjectsInit`
- `C_FindObjects`
- `C_FindObjectsFinal`
- `C_GenerateRandom`
- `C_SignInit`
- `C_Sign`
- `C_VerifyInit`
- `C_Verify`
- `C_DigestInit`
- `C_DigestUpdate`
- `C_DigestFinal`

Supporto per sistema di crittografia asimmetrica

L'implementazione di riferimento di FreerTOS utilizza PKCS #11 RSA a 2048 bit (solo firma) ed ECDSA con la curva NIST P-256. Le seguenti istruzioni descrivono come creare un oggetto basato su un certificato client P-256. AWS IoT

Assicurati di utilizzare le seguenti (o più recenti) versioni di AWS CLI e 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
```

La procedura seguente presuppone che sia stato utilizzato il aws configure comando per configurare. AWS CLI Per ulteriori informazioni, vedere Configurazione rapida con aws configure nella Guida per l'AWS Command Line Interface utente.

Per creare AWS IoT qualcosa basato su un certificato client P-256

1. Crea qualsiasi AWS IoT cosa.

```
aws iot create-thing --thing-name thing-name
```

2. Utilizzare OpenSSL per la creazione di una chiave 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. Creare una richiesta di registrazione dei certificati firmata dalla chiave creata nella fase 2.

```
openssl req -new -nodes -days 365 -key thing-name.key -out thing-name.req
```

4. Invia la richiesta di iscrizione al certificato a. AWS IoT

```
aws iot create-certificate-from-csr  \
  --certificate-signing-request file://thing-name.req --set-as-active  \
  --certificate-pem-outfile thing-name.crt
```

5. Collegare il certificato (cui fa riferimento l'output ARN mediante il comando precedente) all'oggetto.

```
aws iot attach-thing-principal --thing-name thing-name \
  --principal "arn:aws:iot:us-
east-1:123456789012:cert/
86e41339a6d1bbc67abf31faf455092cdebf8f21ffbc67c4d238d1326c7de729"
```

6. Creazione di una policy. (Questa politica è troppo permissiva. Dovrebbe essere usato solo per scopi di sviluppo.)

```
aws iot create-policy --policy-name FullControl --policy-document file://
policy.json
```

Di seguito è riportato un elenco relativo al file policy.json specificato nel comando `create-policy`. Puoi omettere l'`greengrass:*`azione se non vuoi eseguire la demo di FreerTOS per la connettività e il rilevamento di Greengrass.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:*",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "greengrass:*",
      "Resource": "*"
    }
  ]
}
```

7. Collegare l'entità principale (certificato) e la policy all'oggetto.

```
aws iot attach-principal-policy --policy-name FullControl \
  --principal "arn:aws:iot:us-
east-1:123456789012:cert/
86e41339a6d1bbc67abf31faf455092cdebf8f21ffbc67c4d238d1326c7de729"
```

Ora seguire le fasi illustrate nella sezione Nozioni di base su AWS IoT di questa guida. Non dimenticare di copiare la chiave privata e del certificato creati nel file `aws_clientcredential_keys.h`. Copiare il nome dell'oggetto in `aws_clientcredential.h`.

> **Note**
>
> Il certificato e la chiave privata sono hardcoded solo a scopo dimostrativo. Le applicazioni a livello di produzione devono archiviare questi file in un percorso sicuro.

Portabilità

Per informazioni sul porting della PKCS11 libreria principale sulla tua piattaforma, consulta Porting the core PKCS11 Library nella FreerTOS Porting Guide.

Uso della memoria

Dimensione del codice del core PKCS11 (esempio generato con GCC per ARM Cortex-M)		
File	Con ottimizzazione -O1	Con ottimizzazione -Os
core_pkcs11.c	0,8K	0,8 K
core_pki_utils.c	0,5K	0,3 K
core_pkcs11_mbedtls.c	8,7K	7,3K
Stime totali	10,0 K	8,4K

Libreria Secure Sockets

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Panoramica

Puoi utilizzare la libreria FreerTOS Secure Sockets per creare applicazioni integrate che comunicano in modo sicuro. La libreria è progettata per agevolare l'onboarding di sviluppatori di software con background di programmazione di rete diversi.

La libreria FreerTOS Secure Sockets si basa sull'interfaccia Berkeley Sockets, con un'opzione di comunicazione sicura aggiuntiva tramite protocollo TLS. Per informazioni sulle differenze tra la libreria FreerTOS Secure Sockets e l'interfaccia Berkeley sockets, `SOCKETS_SetSockOpt` vedere nel Secure Sockets API Reference.

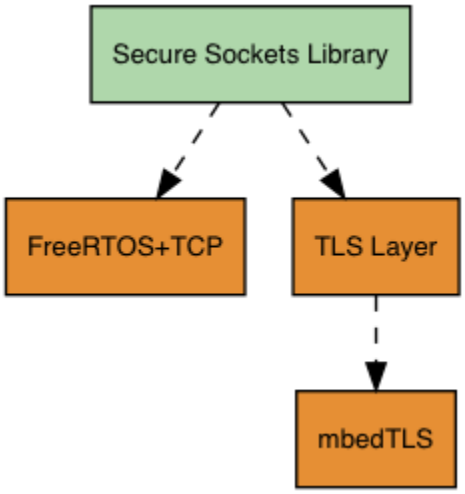
> **Note**
>
> Attualmente, solo il client APIs, più un'implementazione IP leggera (LWiP) dell'API `Bind` lato server, sono supportati per FreerTOS Secure Sockets.

Dipendenze e requisiti

La libreria FreerTOS Secure Sockets dipende da uno stack TCP/IP e da un'implementazione TLS. Le porte per FreerTOS soddisfano queste dipendenze in tre modi:

- Un'implementazione personalizzata sia di TCP/IP sia di TLS
- Un'implementazione personalizzata di TCP/IP e il livello TLS FreerTOS con mbedTLS
- FreerTOS+TCP e il livello FreerTOS TLS con mbedTLS

Il diagramma delle dipendenze seguente mostra l'implementazione di riferimento inclusa nella libreria FreerTOS Secure Sockets. L'implementazione di riferimento supporta TLS e TCP/IP su Ethernet e Wi-Fi con FreeRTOS+TCP e mbedTLS come dipendenze. Per ulteriori informazioni sul layer TLS FreerTOS, vedere. Transport Layer Security



Funzionalità

Le funzionalità della libreria FreerTOS Secure Sockets includono:

- Un'interfaccia standard basata sui socket di Berkeley
- Thread-safe per l'invio e la ricezione di dati APIs

- Easy-to-enable TLS

Risoluzione dei problemi

Codici di errore

I codici di errore restituiti dalla libreria FreerTOS Secure Sockets sono valori negativi. Per ulteriori informazioni su ciascun codice di errore, consulta Codici di errore Secure Sockets nella Documentazione di riferimento sull'API Secure Sockets.

> **Note**
>
> Se l'API FreerTOS Secure Sockets restituisce un codice di errore,, libreria CoreMQTT che dipende dalla libreria FreerTOS Secure Sockets, restituisce il codice di errore. `AWS_IOT_MQTT_SEND_ERROR`

Supporto per gli sviluppatori

La libreria FreerTOS Secure Sockets include due macro di supporto per la gestione degli indirizzi IP:

`SOCKETS_inet_addr_quick`

Questa macro converte un indirizzo IP espresso con quattro ottetti numerici separati in un indirizzo IP espresso con un numero a 32 bit nell'ordine dei byte di rete.

`SOCKETS_inet_ntoa`

Questa macro converte un indirizzo IP espresso con un numero a 32 bit nell'ordine dei byte di rete in una stringa con notazione decimale puntata.

Limitazioni d'uso

Solo i socket TCP sono supportati dalla libreria FreerTOS Secure Sockets. I socket UDP non sono supportati.

APIs I server non sono supportati dalla libreria FreerTOS Secure Sockets, ad eccezione di un'implementazione IP leggera (LWiP) dell'API lato server. `Bind` I client APIs sono supportati.

Inizializzazione

Per utilizzare la libreria FreerTOS Secure Sockets, è necessario inizializzare la libreria e le sue dipendenze. Per inizializzare la libreria Secure Sockets, utilizza il codice seguente nell'applicazione:

```
BaseType_t xResult = pdPASS;
xResult = SOCKETS_Init();
```

Le librerie dipendenti devono essere inizializzate separatamente. Ad esempio, se FreeRTOS+TCP è una dipendenza, nella tua applicazione devi invocare anche `FreeRTOS_IPInit`.

Riferimento API

Per informazioni di riferimento complete sull'API, consulta Documentazione di riferimento sull'API Secure Sockets.

Esempio di utilizzo

Il codice seguente collega un client a un server.

```
#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 );
}
```

Per un esempio completo, consulta Demo Client Echo Secure Sockets.

Portabilità

FreerTOS Secure Sockets dipende da uno stack TCP/IP e da un'implementazione TLS. A seconda del tuo stack, per effettuare il porting della libreria Secure Sockets, potresti dover trasferire alcuni dei seguenti elementi:

- Lo stack TCP/IP FreeRTOS+TCP
- Il PKCS11 libreria principale
- Il Transport Layer Security

Per ulteriori informazioni sul porting, consulta Porting the Secure Sockets Library nella FreerTOS Porting Guide.

AWS IoT Libreria Device Shadow

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Introduzione

È possibile utilizzare la libreria AWS IoT Device Shadow per archiviare e recuperare lo stato corrente (l'ombra) di ogni dispositivo registrato. L'ombra del dispositivo è una rappresentazione virtuale persistente del dispositivo con cui è possibile interagire nelle applicazioni Web anche se il dispositivo è offline. Lo stato del dispositivo viene catturato come ombra in un documento JSON. È possibile inviare comandi al servizio AWS IoT Device Shadow tramite MQTT o HTTP per interrogare l'ultimo stato noto del dispositivo o per modificarlo. L'ombra di ogni dispositivo è identificata in modo univoco dal nome dell'elemento corrispondente, una rappresentazione di uno specifico dispositivo o entità logica sul AWS Cloud. Per ulteriori informazioni, consulta Gestione dei dispositivi con AWS IoT. Maggiori dettagli sulle ombre sono disponibili nella AWS IoT documentazione.

La libreria AWS IoT Device Shadow non ha dipendenze da librerie aggiuntive diverse dalla libreria C standard. Inoltre non ha alcuna dipendenza dalla piattaforma, come il threading o la sincronizzazione. Può essere utilizzato con qualsiasi libreria MQTT e qualsiasi libreria JSON.

Questa libreria può essere utilizzata liberamente ed è distribuita sotto la licenza open source MIT.

Dimensione del codice di AWS IoT Device Shadow (esempio generato con GCC per ARM Cortex-M)		
File	Con ottimizzazione -O1	Con ottimizzazione -Os
shadow.c	1,2 K	0,9 K
Stime totali	1,2 K	0,9 K

AWS IoT Libreria di lavori

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Introduzione

AWS IoT Jobs è un servizio che notifica a uno o più dispositivi connessi un lavoro in sospeso. Puoi utilizzare un job per gestire la tua flotta di dispositivi, aggiornare il firmware e i certificati di sicurezza sui tuoi dispositivi o eseguire attività amministrative come il riavvio dei dispositivi e l'esecuzione di diagnosi. Per ulteriori informazioni, consulta Jobs nella Developer Guide.AWS IoT Le interazioni con il servizio AWS IoT Jobs utilizzano MQTT, un protocollo di pubblicazione e sottoscrizione leggero. Questa libreria fornisce un'API per comporre e riconoscere le stringhe di argomenti MQTT utilizzate dal servizio Jobs. AWS IoT

La libreria AWS IoT Jobs è scritta in C e progettata per essere conforme a ISO C90 e MISRA C:2012. La libreria non ha dipendenze da altre librerie diverse dalla libreria C standard. Può essere utilizzata con qualsiasi libreria MQTT e qualsiasi libreria JSON. La libreria dispone di prove che dimostrano l'uso sicuro della memoria e l'assenza di allocazione di heap, il che la rende adatta ai microcontrollori IoT, ma anche completamente portabile su altre piattaforme.

Questa libreria può essere utilizzata liberamente ed è distribuita con la licenza open source MIT.

Code Size of AWS IoT Jobs (esempio generato con GCC per ARM Cortex-M)		
File	Con ottimizzazione -O1	Con ottimizzazione -Os
jobs.c	1,8K	1,5 K
Stime totali	1,8 K	1,5 K

Transport Layer Security

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

L'interfaccia FreerTOS Transport Layer Security (TLS) è un wrapper sottile e opzionale utilizzato per astrarre i dettagli dell'implementazione crittografica lontano dall'interfaccia Secure Sockets Layer (SSL) che la sovrasta nello stack del protocollo. Scopo dell'interfaccia TLS è rendere la libreria di crittografia software corrente, TLS mbed, facilmente sostituibile con un'implementazione alternativa per le primitive di crittografia e la negoziazione del protocollo TLS. L'interfaccia TLS può essere scambiata senza che sia necessaria alcuna modifica all'interfaccia SSL. Vedi `iot_tls.h` nel repository del codice sorgente di FreerTOS.

L'interfaccia TLS è opzionale perché puoi scegliere di interfacciarti direttamente da SSL in una libreria di crittografia. L'interfaccia non viene utilizzata per le soluzioni MCU che includono un'implementazione offload dell'intero stack del trasporto di rete e TLS.

Per ulteriori informazioni sul porting dell'interfaccia TLS, consulta Porting the TLS Library nella FreerTOS Porting Guide.

Libreria Wi-Fi

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Panoramica

La libreria Wi-Fi FreeRTOS riassume le implementazioni Wi-Fi specifiche delle porte in un'API comune che semplifica lo sviluppo e il porting delle applicazioni per tutte le schede qualificate

FreerTOS con funzionalità Wi-Fi. Utilizzando queste API comuni, le applicazioni possono comunicare con il loro stack wireless di livello inferiore attraverso un'interfaccia comune.

Dipendenze e requisiti

La libreria Wi-Fi FreeRTOS richiede il core FreeRTOS+TCP.

Funzionalità

La libreria Wi-Fi include le seguenti caratteristiche:

- Support per WEP, WPA e WPA2 autenticazione WPA3
- Scansione del punto di accesso
- Risparmio energetico
- Profiling di rete

Ulteriori informazioni sulle caratteristiche della libreria Wi-Fi sono riportate di seguito.

Modalità Wi-Fi

I dispositivi Wi-Fi possono essere in una di tre modalità: Station, Access Point o P2P. È possibile ottenere la modalità corrente di un dispositivo Wi-Fi chiamando `WIFI_GetMode`. È possibile impostare la modalità Wi-Fi di un dispositivo chiamando `WIFI_SetMode`. Cambiando le modalità chiamando `WIFI_SetMode`, si disconnette il dispositivo, se connesso a una rete.

Modalità Station

Imposta il dispositivo in modalità Station per collegare la scheda a un punto di accesso esistente.

Modalità Access Point (AP)

Imposta il dispositivo in modalità AP per renderlo un punto di accesso a cui altri dispositivi possano connettersi. Quando il dispositivo è in modalità AP, puoi connettere un altro dispositivo al tuo dispositivo FreeRTOS e configurare le nuove credenziali Wi-Fi. Per configurare la modalità del punto di accesso, chiamare `WIFI_ConfigureAP`. Per inserire il dispositivo in modalità AP, chiama `WIFI_StartAP`. Per disattivare la modalità AP, chiama `WIFI_StopAP`.

> **Note**
>
> Le librerie FreerTOS non forniscono il provisioning Wi-Fi in modalità AP. È necessario fornire le funzionalità aggiuntive, incluse le funzionalità del server DHCP e HTTP, per ottenere il supporto completo della modalità AP.

Modalità P2P

Imposta il dispositivo in modalità P2P per consentire a più dispositivi di connettersi l'un l'altro direttamente, senza un punto di accesso.

Sicurezza

L'API Wi-Fi supporta WEP, WPA e tipi di sicurezza. WPA2 WPA3 Se un dispositivo è in modalità Station, è necessario specificare il tipo di sicurezza di rete quando si chiama la funzione `WIFI_ConnectAP`. Se è in modalità AP, un dispositivo può essere configurato per l'utilizzo di qualsiasi tipo di sicurezza supportato:

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

Scansione e connessione

Per eseguire la scansione di punti di accesso nelle vicinanze, imposta il dispositivo in modalità Station e chiama la funzione `WIFI_Scan`. Se la scansione trova la rete a cui intendi connetterti, puoi effettuare la connessione chiamando `WIFI_ConnectAP` e fornendo le credenziali di rete. Puoi disconnettere un dispositivo Wi-Fi dalla rete chiamando `WIFI_Disconnect`. Per ulteriori informazioni sulla scansione e la connessione, consulta Esempio di utilizzo e Riferimento API.

Risparmio energetico

Dispositivi Wi-Fi diversi prevedono requisiti di alimentazione diversi, a seconda dell'applicazione e delle sorgenti di alimentazione disponibili. Un dispositivo potrebbe essere sempre acceso per ridurre

la latenza oppure potrebbe essere connesso a intermittenza e passare a una modalità a basso consumo quando non è richiesta una connessione Wi-Fi. L'API dell'interfaccia supporta vari metodi di gestione dell'alimentazione, ad esempio sempre acceso, modalità a basso consumo e modalità normale. È possibile impostare la modalità di alimentazione per un dispositivo utilizzando la funzione `WIFI_SetPMMode`. È possibile conoscere la modalità di alimentazione attuale di un dispositivo chiamando la funzione `WIFI_GetPMMode`.

Profili di rete

La libreria Wi-Fi consente di salvare i profili di rete nella memoria non volatile dei tuoi dispositivi. In questo modo è possibile salvare le impostazioni di rete in modo che possano essere recuperate quando un dispositivo si riconnette a una rete Wi-Fi, eliminando la necessità di effettuare nuovamente il provisioning dei dispositivi dopo che sono stati connessi a una rete. `WIFI_NetworkAdd` aggiunge un profilo di rete. `WIFI_NetworkGet` recupera un profilo di rete. `WIFI_NetworkDel` elimina un profilo di rete. Il numero di profili che è possibile salvare dipende dalla piattaforma.

Configurazione

Per usare la libreria Wi-Fi, è necessario definire diversi identificatori in un file di configurazione. Per ulteriori informazioni su questi identificatori, consulta la Riferimento API.

> **Note**
>
> La libreria non include il file di configurazione necessario. Devi crearne uno. Quando crei il tuo file di configurazione, assicurati di includere qualsiasi identificatore di configurazione specificatamente richiesto dalla scheda.

Inizializzazione

Prima di utilizzare la libreria Wi-Fi, oltre ai componenti FreeRTOS è necessario inizializzare alcuni componenti specifici per la scheda. Utilizzando il file `vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/application_code/main.c` come modello per l'inizializzazione, procedi come segue:

1. Se la tua applicazione è in grado di gestire le connessioni Wi-Fi, rimuovi la logica di connessione Wi-Fi di esempio in `main.c`. Sostituisci la seguente chiamata di funzione `DEMO_RUNNER_RunDemos()`:

```
if( SYSTEM_Init() == pdPASS )
    {
    ...
        DEMO_RUNNER_RunDemos();
    ...
        }
```

Con una chiamata alla tua applicazione:

```
if( SYSTEM_Init() == pdPASS )
    {
    ...
        // This function should create any tasks
        // that your application requires to run.
        YOUR_APP_FUNCTION();
    ...
        }
```

2. Chiama `WIFI_On()` per inizializzare e accendere il chip Wi-Fi.

> **Note**
>
> Alcune schede potrebbero richiedere un'inizializzazione hardware aggiuntiva.

3. Passa una struttura `WIFINetworkParams_t` configurata a `WIFI_ConnectAP()` per connettere la scheda a una rete Wi-Fi disponibile. Per ulteriori informazioni sulla struttura `WIFINetworkParams_t`, consulta Esempio di utilizzo e Riferimento API.

Riferimento API

Per informazioni di riferimento complete sull'API, consulta Documentazione di riferimento sull'API Wi-Fi.

Esempio di utilizzo

Connessione a un punto di accesso noto

```
#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
}
```

Scansione nelle vicinanze 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);
}
```

Portabilità

L'implementazione `iot_wifi.c` deve introdurre le funzioni definite in `iot_wifi.h`. Come minimo, l'implementazione deve restituire `eWiFiNotSupported` per qualsiasi funzione non essenziale o non supportata.

Per ulteriori informazioni sul porting della libreria Wi-Fi, consulta Porting the Wi-Fi Library nella FreerTOS Porting Guide.

Demo FreerTOS

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

FreeRTOS include alcune applicazioni demo nella cartella, nella directory principale `demos` di FreeRTOS. Tutti gli esempi che possono essere eseguiti da FreerTOS appaiono nella cartella, `common` sotto. `demos` C'è anche una cartella per ogni piattaforma qualificata da FreeRTOS all'interno della cartella. `demos`

Prima di provare le applicazioni demo, si consiglia di completare il tutorial in Guida introduttiva a FreerTOS. Mostra come configurare ed eseguire la demo di autenticazione reciproca di CoreMQTT.

Esecuzione delle demo di FreerTOS

I seguenti argomenti mostrano come configurare ed eseguire le demo di FreerTOS:

- Applicazioni demo Bluetooth Low Energy
- Bootloader dimostrativo per Microchip Curiosity MZEF PIC32
- AWS IoT Device Defender demo
- AWS IoT Greengrass applicazione demo discovery
- Demo CoreHTTP
- AWS IoT Demo della libreria Jobs
- Demo CoreMQTT
- Over-the-air aggiorna l'applicazione demo
- Demo Client Echo Secure Sockets
- AWS IoT Applicazione demo Device Shadow

La `DEMO_RUNNER_RunDemos` funzione, situata nel *`freertos`*`/demos/demo_runner/iot_demo_runner.c` file, inizializza un thread separato su cui viene eseguita una singola

applicazione demo. Per impostazione predefinita, chiama e avvia `DEMO_RUNNER_RunDemos` solo la demo di autenticazione reciproca di CoreMQTT. A seconda della configurazione selezionata quando hai scaricato FreerTOS e da dove hai scaricato FreerTOS, le altre funzioni runner di esempio potrebbero avviarsi per impostazione predefinita. Per abilitare un'applicazione demo, apri il *`freertos`*`/vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/aws_demo_config.h` file e definisci la demo che desideri eseguire.

> **Note**
>
> Non tutte le combinazioni di esempi funzionano insieme. A seconda della combinazione, il software potrebbe non funzionare sulla destinazione selezionata a causa di limiti di memoria. È consigliabile eseguire una demo alla volta.

Configurazione delle demo

Le demo sono state configurate per consentire agli utenti di iniziare velocemente. Potresti avere la necessità di modificare alcune delle configurazioni per creare una versione da eseguire sulla tua piattaforma. Puoi trovare i file di configurazione in `vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files`.

Applicazioni demo Bluetooth Low Energy

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Panoramica

FreerTOS Bluetooth Low Energy include tre applicazioni demo:

- Demo MQTT su Bluetooth Low Energy

  Questa applicazione dimostra come utilizzare il servizio MQTT su Bluetooth Low Energy.

- Demo Provisioning Wi-Fi

  Questa applicazione dimostra come utilizzare il servizio Provisioning Wi-Fi di Bluetooth Low Energy.

- Demo Server Generic Attributes

  Questa applicazione dimostra come utilizzare il APIs middleware FreerTOS Bluetooth Low Energy per creare un semplice server GATT.

> **Note**
>
> Per configurare ed eseguire le demo di FreerTOS, segui i passaggi indicati. Guida introduttiva a FreerTOS

Prerequisiti

Per proseguire con queste demo, occorre un microcontroller con funzionalità Bluetooth Low Energy. È inoltre necessario disporre di SDK iOS per dispositivi Bluetooth FreerTOS o Android SDK per dispositivi Bluetooth FreerTOS.

Configurazione AWS IoT e Amazon Cognito per FreerTOS Bluetooth Low Energy

Per connettere i tuoi AWS IoT dispositivi a più MQTT, devi configurare Amazon AWS IoT Cognito.

Per configurare AWS IoT

1. Configura un AWS account su https://aws.amazon.com/.
2. Aprire la console AWS IoT e dal riquadro di navigazione scegliere Manage (Gestisci), quindi scegliere Things (Oggetti).
3. Scegliere Create (Crea), quindi scegliere Create a single thing (Crea un singolo oggetto).
4. Immettere un nome per il dispositivo, quindi scegliere Next (Avanti).
5. Se si sta collegando il microcontroller al cloud mediante un dispositivo mobile, scegliere Create thing without certificate (Crea oggetto senza certificato). Poiché i dispositivi mobili SDKs utilizzano Amazon Cognito per l'autenticazione dei dispositivi, non è necessario creare un certificato del dispositivo per le demo che utilizzano Bluetooth Low Energy.

Se si sta collegando il microcontroller al cloud direttamente tramite Wi-Fi, scegliere Create certificate (Crea certificato), scegliere Activate (Attiva), quindi scaricare il certificato, la chiave pubblica e la chiave privata dell'oggetto.

6. Scegliere l'oggetto appena creato dall'elenco degli oggetti registrati e quindi scegliere Interact (Interazione) dalla pagina dell'oggetto. Prendi nota dell'endpoint dell'API AWS IoT REST.

Per ulteriori informazioni sulla configurazione, consulta la Guida introduttiva con AWS IoT.

Per creare un pool di utenti Amazon Cognito

1. Apri la console Amazon Cognito e scegli Gestisci pool di utenti.
2. Scegli Create a User Pool (Crea un bacino d'utenza).
3. Fornire al pool di utenti un nome, quindi scegliere Review defaults (Esamina impostazioni predefinite).
4. Nel riquadro di navigazione scegliere App clients (Client app) e quindi scegliere Add an app (Aggiungi un'app).
5. Immettere il nome del client dell'app, quindi scegliere Create app client (Crea client dell'app).
6. Dal riquadro di navigazione, scegliere Review (Verifica) e quindi selezionare Create pool (Crea pool).

   Annotare l'ID del pool visualizzato nella pagina General Settings (Impostazioni generali) del pool di utenti.
7. Nel riquadro di navigazione, scegliere App clients (Client app) e quindi scegliere Show details (Mostra dettagli). Annotare l'ID e il segreto del client dell'app.

Per creare un pool di identità Amazon Cognito

1. Apri la console Amazon Cognito e scegli Gestisci pool di identità.
2. Immettere un nome per il pool di identità.
3. Espandere Authentication providers (Provider di autenticazione), scegliere la scheda Cognito, quindi immettere l'ID del pool di utenti e l'ID del client dell'app.
4. Seleziona Create Pool (Crea pool).

5. Espandere View Details (Visualizza dettagli) e annotare i due nomi di ruolo IAM. Scegli Consenti per creare i ruoli IAM per identità autenticate e non autenticate per accedere ad Amazon Cognito.
6. Scegli Modifica pool di identità. Annotare l'ID del pool di identità. Il formato è simile al seguente `us-west-2:12345678-1234-1234-1234-123456789012`.

Per ulteriori informazioni sulla configurazione di Amazon Cognito, consulta la Guida introduttiva ad Amazon Cognito.

Per creare e allegare una policy IAM all'identità autenticata

1. Apri la console IAM e dal pannello di navigazione, scegli Ruoli.
2. Trovare e scegliere il ruolo dell'identità autenticata, scegliere Attach policies (Allega policy), quindi scegliere Add inline policy (Aggiungi policy inline).
3. Scegliere la scheda JSON e incollare il file JSON seguente:

```
{
   "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. Scegliere Review policy (Esamina policy), immettere un nome per la policy, quindi scegliere Create policy (Crea policy).

Tieni a portata di AWS IoT mano le tue informazioni e quelle di Amazon Cognito. Hai bisogno dell'endpoint e IDs dell'autenticazione della tua applicazione mobile con il cloud. AWS

Configura il tuo ambiente FreerTOS per Bluetooth Low Energy

Per configurare il tuo ambiente, devi scaricare FreeRTOS con Libreria Bluetooth Low Energy il microcontrollore e scaricare e configurare il Mobile SDK per i dispositivi Bluetooth FreerTOS sul tuo dispositivo mobile.

Per configurare l'ambiente del tuo microcontrollore con FreerTOS Bluetooth Low Energy

1. Scarica o clona FreerTOS da. GitHub Consultare il file README.md per le istruzioni.
2. Configura FreerTOS sul tuo microcontrollore.

   Per informazioni su come iniziare a usare FreerTOS su un microcontrollore certificato FreeRTOS, consulta la guida per la tua scheda in Getting Started with FreerTOS.

   > **Note**
   >
   > Puoi eseguire le demo su qualsiasi microcontrollore abilitato a Bluetooth Low Energy con FreeRTOS e librerie FreeRTOS Bluetooth Low Energy con porte. Attualmente, il progetto demo MQTT su Bluetooth Low Energy FreerTOS è completamente adattato ai seguenti dispositivi Bluetooth Low Energy:
   >
   > - Espressif - C e ESP32 DevKit ESP-WROVER-KIT
   > - Nordic in 0-DK RF5284

Componenti comuni

Le applicazioni demo FreerTOS hanno due componenti comuni:

- Network Manager
- Applicazione demo Bluetooth Low Energy Mobile SDK

Network Manager

Network Manager consente di gestire la connessione di rete del microcontroller. Si trova nella tua directory FreerTOS all'indirizzo. `demos/network_manager/aws_iot_network_manager.c` Se il Network Manager è abilitato per Wi-Fi e Bluetooth Low Energy, le demo iniziano con Bluetooth Low Energy per impostazione predefinita. Se la connessione Bluetooth Low Energy viene interrotta e la scheda lo è Wi-Fi-enabled, Network Manager passa a una connessione Wi-Fi disponibile per impedire la disconnessione dalla rete.

Per abilitare un tipo di connessione di rete con Network Manager, aggiungi il tipo di connessione di rete al `configENABLED_NETWORKS` parametro in `vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/aws_iot_network_config.h` (dove *`vendor`* è il nome del fornitore e il *`board`* nome della scheda che stai utilizzando per eseguire le demo).

Ad esempio, se Bluetooth Low Energy e Wi-Fi sono entrambi abilitati, la riga che inizia con `#define configENABLED_NETWORKS` in `aws_iot_network_config.h` contiene quanto segue:

```
#define  configENABLED_NETWORKS  ( AWSIOT_NETWORK_TYPE_BLE | AWSIOT_NETWORK_TYPE_WIFI )
```

Per ottenere un elenco dei tipi di connessione di rete attualmente supportati, vedere le righe che iniziano con `#define AWSIOT_NETWORK_TYPE` in `aws_iot_network.h`.

Applicazione dimostrativa FreerTOS Bluetooth Low Energy Mobile SDK

L'applicazione demo FreeRTOS Bluetooth Low Energy Mobile SDK si trova GitHub su Android SDK per dispositivi Bluetooth FreerTOS sotto e `amazon-freertos-ble-android-sdk/app` su iOS SDK per dispositivi Bluetooth FreerTOS sotto. `amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo` In questo esempio, utilizziamo screenshot della versione iOS dell'applicazione demo per dispositivi mobili.

> **Note**
>
> Se utilizzi un dispositivo iOS, è richiesto Xcode per compilare l'applicazione demo per dispositivi mobili. Se utilizzi un dispositivo Android, puoi utilizzare Android Studio per compilare l'applicazione demo per dispositivi mobili.

Per configurare l'applicazione demo per SDK iOS

Quando definisci le variabili di configurazione, utilizza il formato dei valori segnaposto forniti nei file di configurazione.

1. Verifica che sia installato SDK iOS per dispositivi Bluetooth FreerTOS.
2. Invia il comando seguente da `amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo/`:

```
$ pod install
```

3. Aprire il progetto `amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo/AmazonFreeRTOSDemo.xcworkspace` con Xcode e modificare l'account sviluppatore di firma nell'account dell'utente.
4. Crea una AWS IoT politica nella tua regione (se non l'hai già fatto).

> **Note**
>
> Questa policy è diversa dalla policy IAM creata per l'identità autenticata di Amazon Cognito.

   a. Apri la AWS IoT console.
   b. Nel riquadro di navigazione, selezionare Secure (Sicurezza), scegliere Policies (Policy) e poi Create (Crea). Inserire un nome per identificare la policy. Nella sezione Add statements (Aggiungi istruzioni), scegliere Advanced mode (Modalità avanzata). Copiare e incollare il seguente JSON nella finestra dell'editor policy. Sostituisci *aws-region* e *aws-account* con la tua AWS regione e l'ID dell'account.

```
{
    "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. Scegli Create (Crea) .

5. Apri `amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo/AmazonFreeRTOSDemo/Amazon/AmazonConstants.swift` e ridefinisci le seguenti variabili:

   - `region`: La tua AWS regione.
   - `iotPolicyName`: il nome AWS IoT della tua polizza.
   - `mqttCustomTopic`: l'argomento MQTT in cui effettuare la pubblicazione.

6. Aprire `amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo/AmazonFreeRTOSDemo/Support/awsconfiguration.json`.

   In `CognitoIdentity` ridefinisci le seguenti variabili:

   - `PoolId`: l'ID del tuo pool di identità di Amazon Cognito.
   - `Region`: La tua AWS regione.

   In `CognitoUserPool` ridefinisci le seguenti variabili:

   - `PoolId`: l'ID del tuo pool di utenti di Amazon Cognito.
   - `AppClientId`: l'ID del client dell'app.
   - `AppClientSecret`: il segreto del client dell'app.
   - `Region`: La tua AWS regione.

Per configurare l'applicazione demo per SDK Android

Quando definisci le variabili di configurazione, utilizza il formato dei valori segnaposto forniti nei file di configurazione.

1. Verifica che sia installato Android SDK per dispositivi Bluetooth FreerTOS.
2. Crea una AWS IoT politica nella tua regione (se non l'hai già fatto).

   > **Note**
   >
   > Questa policy è diversa dalla policy IAM creata per l'identità autenticata di Amazon Cognito.

   a. Apri la AWS IoT console.
   b. Nel riquadro di navigazione, selezionare Secure (Sicurezza), scegliere Policies (Policy) e poi Create (Crea). Inserire un nome per identificare la policy. Nella sezione Add statements (Aggiungi istruzioni), scegliere Advanced mode (Modalità avanzata). Copiare e incollare il seguente JSON nella finestra dell'editor policy. Sostituisci *aws-region* e *aws-account* con la tua AWS regione e l'ID dell'account.

```
{
    "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. Scegli Create (Crea) .

3. Apri https://github.com/aws/amazon-freertos-ble-android- sdk/blob/master/app/src/main/java/software/amazon/freertos/demo/DemoConstants .java e ridefinisci le seguenti variabili:

   - `AWS_IOT_POLICY_NAME`: nome della tua AWS IoT politica.
   - `AWS_IOT_REGION`: La tua AWS regione.

4. Apri https://github.com/aws/amazon-freertos-ble-android- sdk/blob/master/app/src/main/res/raw/awsconfiguration .json.

   In `CognitoIdentity` ridefinisci le seguenti variabili:

   - `PoolId`: l'ID del tuo pool di identità di Amazon Cognito.
   - `Region`: La tua AWS regione.

   In `CognitoUserPool` ridefinisci le seguenti variabili:

   - `PoolId`: l'ID del tuo pool di utenti di Amazon Cognito.
   - `AppClientId`: l'ID del client dell'app.
   - `AppClientSecret`: il segreto del client dell'app.
   - `Region`: La tua AWS regione.

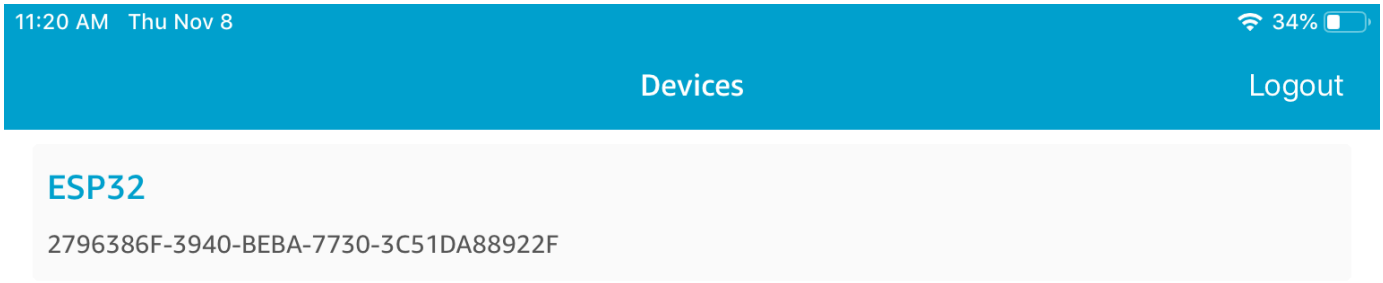
Per individuare e stabilire connessioni sicure con il microcontroller su Bluetooth Low Energy

1. Per associare il microcontrollore e il dispositivo mobile in modo sicuro (fase 6), è necessario un emulatore di terminale seriale con funzionalità di input e output (ad esempio). TeraTerm Configura il terminale per la connessione alla scheda tramite una connessione seriale come indicato in Installazione di un emulatore di terminale.
2. Eseguire il progetto demo Bluetooth Low Energy sul microcontroller.
3. Eseguire l'applicazione demo SDK Bluetooth Low Energy Mobile sul dispositivo mobile.

Per avviare un'applicazione demo di SDK Android dalla riga di comando, eseguire il comando seguente:

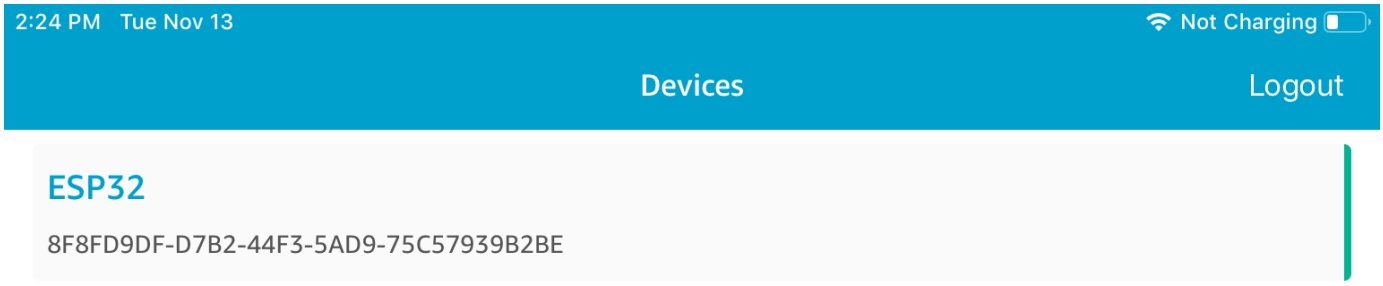
```
$ ./gradlew installDebug
```

4. Verificare che il microcontroller sia visualizzato in Devices (Dispositivi) nell'applicazione demo SDK Bluetooth Low Energy Mobile.

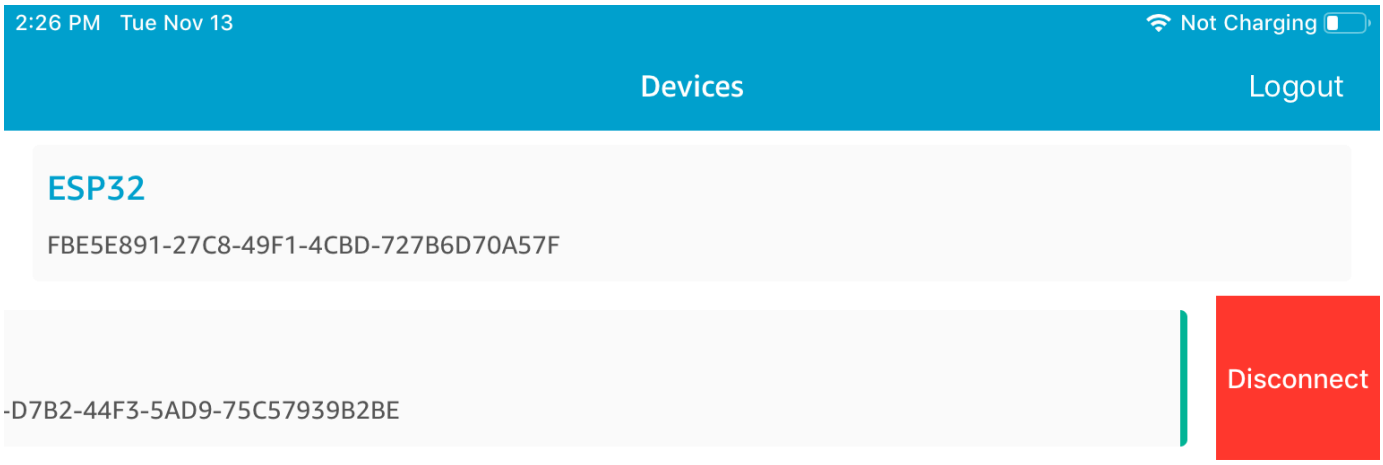


> **Note**
>
> Nell'elenco vengono visualizzati tutti i dispositivi con FreerTOS e il servizio di informazioni sul dispositivo *`freertos`*`/.../device_information` () che si trovano nell'intervallo.

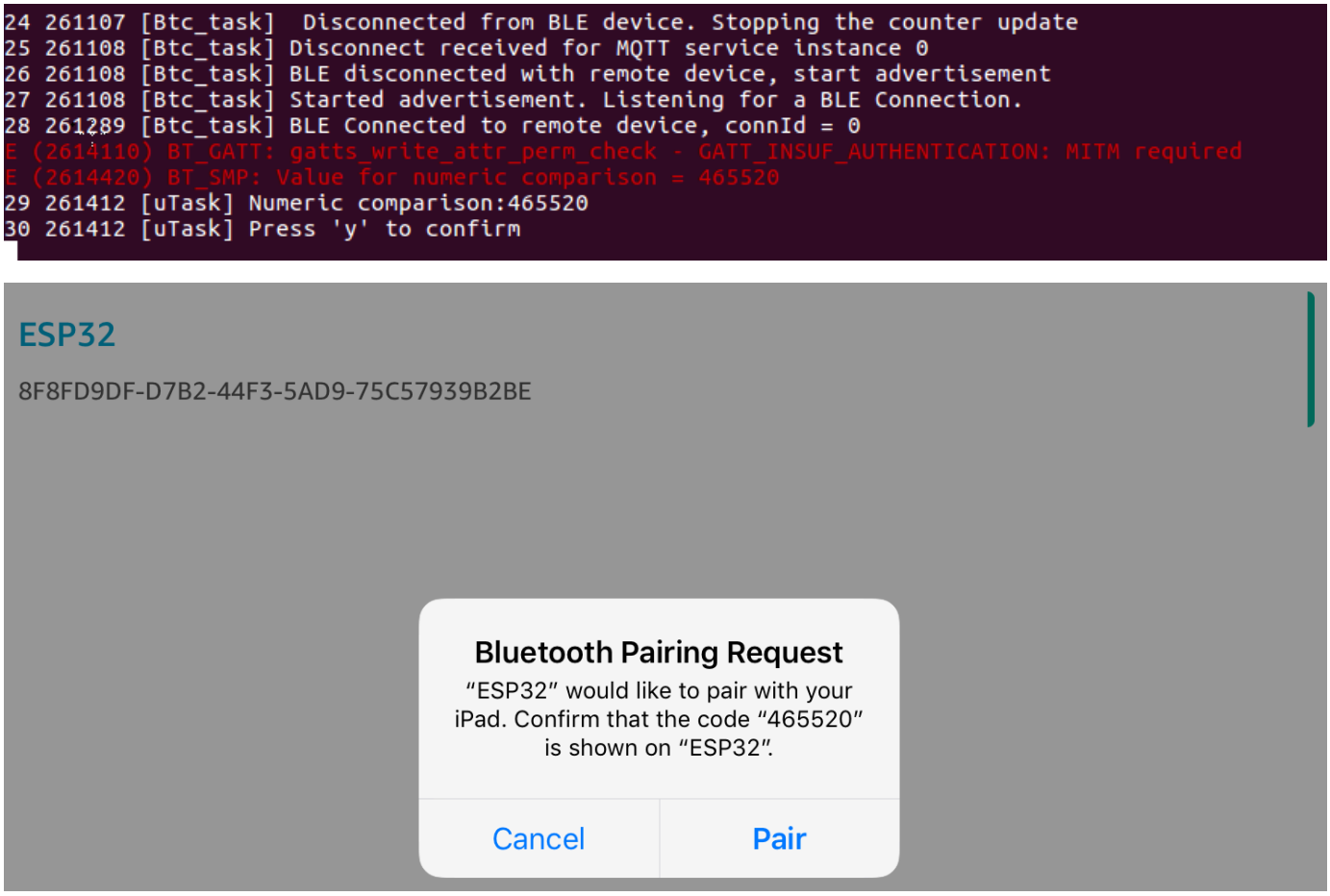
5. Scegliere il microcontroller dall'elenco di dispositivi. L'applicazione stabilisce una connessione con la scheda e una riga verde viene visualizzata accanto al dispositivo connesso.



Per disconnettersi dal microcontroller, trascinare la riga verso sinistra.



6. Se richiesto, abbinare il microcontroller e il dispositivo mobile.



Se il codice per i confronti numerici è identico su entrambi i dispositivi, accoppiare i dispositivi.

> **Note**
>
> L'applicazione demo Bluetooth Low Energy Mobile SDK utilizza Amazon Cognito per l'autenticazione degli utenti. Assicurati di aver configurato un pool di utenti e identità di Amazon Cognito e di aver collegato le policy IAM alle identità autenticate.

MQTT su Bluetooth Low Energy

Nella demo MQTT tramite Bluetooth Low Energy, il microcontrollore pubblica messaggi sul cloud tramite un proxy MQTT. AWS

Per effettuare la sottoscrizione a un argomento MQTT demo

1. Accedi alla console. AWS IoT
2. Nel riquadro di navigazione scegliere Test per aprire il client MQTT.
3. In Argomento sottoscrizione, digitare ***thing-name*/example/topic1**, quindi scegliere Effettua sottoscrizione all'argomento.

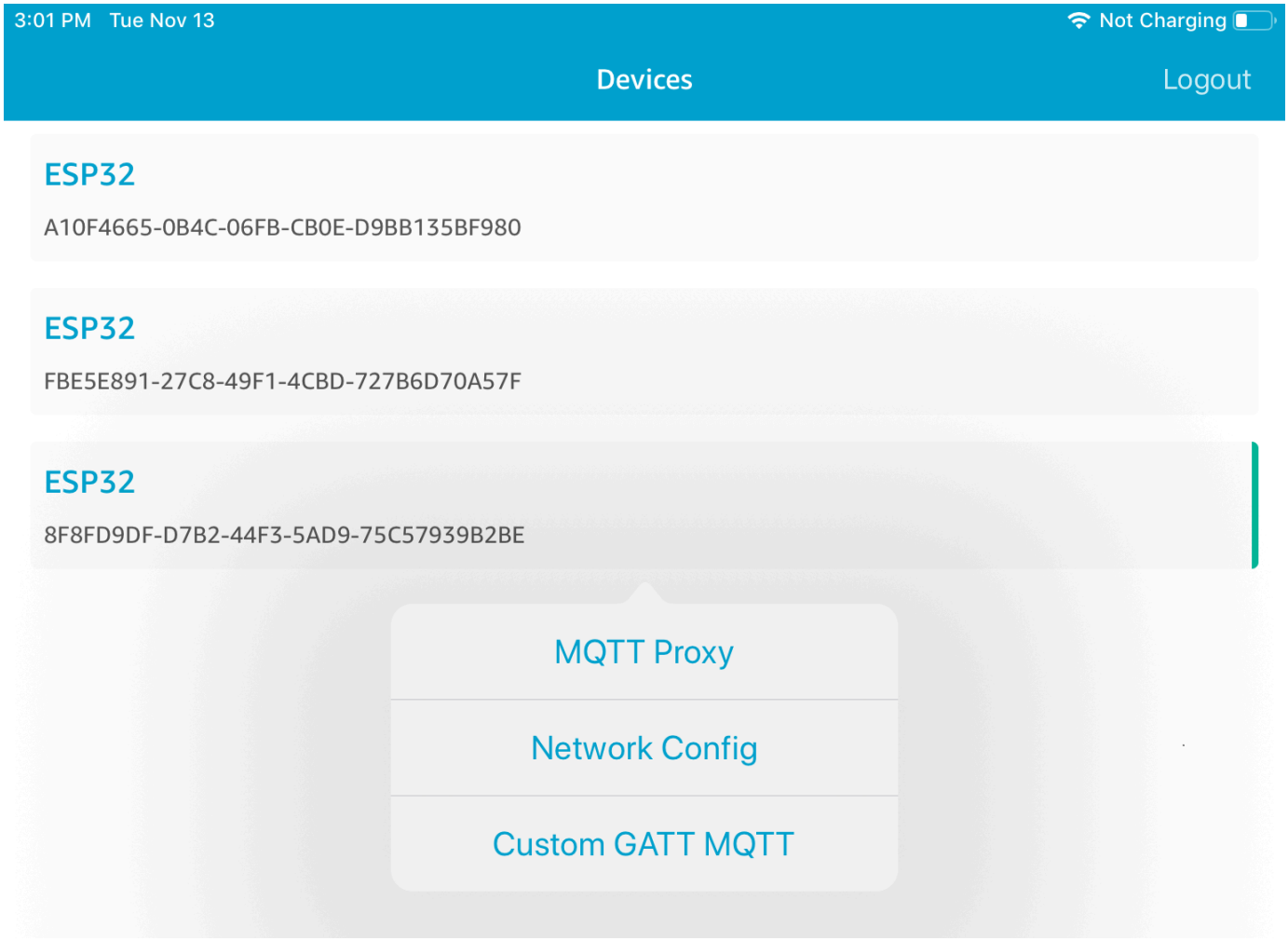
Se utilizzi Bluetooth Low Energy per accoppiare il microcontroller con il dispositivo mobile, i messaggi MQTT vengono instradati al dispositivo mobile mediante l'applicazione demo SDK Bluetooth Low Energy Mobile.

Per abilitare la demo tramite Bluetooth Low Energy

- Aprire `vendors/vendor/boards/board/aws_demos/config_files/aws_demo_config.h` e definire `CONFIG_MQTT_BLE_TRANSPORT_DEMO_ENABLED`.

Per eseguire la demo

1. Compilare ed eseguire il progetto demo sul microcontroller.
2. Assicurati di aver accoppiato la scheda e il dispositivo mobile utilizzando la Applicazione dimostrativa FreerTOS Bluetooth Low Energy Mobile SDK.
3. Dall'elenco Devices (Dispositivi) nell'applicazione demo per dispositivi mobili, scegliere il microcontroller, quindi selezionare MQTT Proxy (Proxy MQTT) per aprire le impostazioni proxy MQTT.



4. Dopo aver abilitato il proxy MQTT, i messaggi MQTT vengono visualizzati sull'argomento *thing-name*`/example/topic1` e i dati vengono stampati sul terminale UART.

Provisioning Wi-Fi

Wi-Fi Provisioning è un servizio FreerTOS Bluetooth Low Energy che consente di inviare in modo sicuro le credenziali di rete Wi-Fi da un dispositivo mobile a un microcontrollore tramite Bluetooth Low Energy. Il codice sorgente per il servizio Provisioning Wi-Fi è disponibile in *freertos*`/.../wifi_provisioning`.

> **Note**
>
> La demo di Wi-Fi Provisioning è attualmente supportata su Espressif - C. ESP32 DevKit

Per abilitare la demo

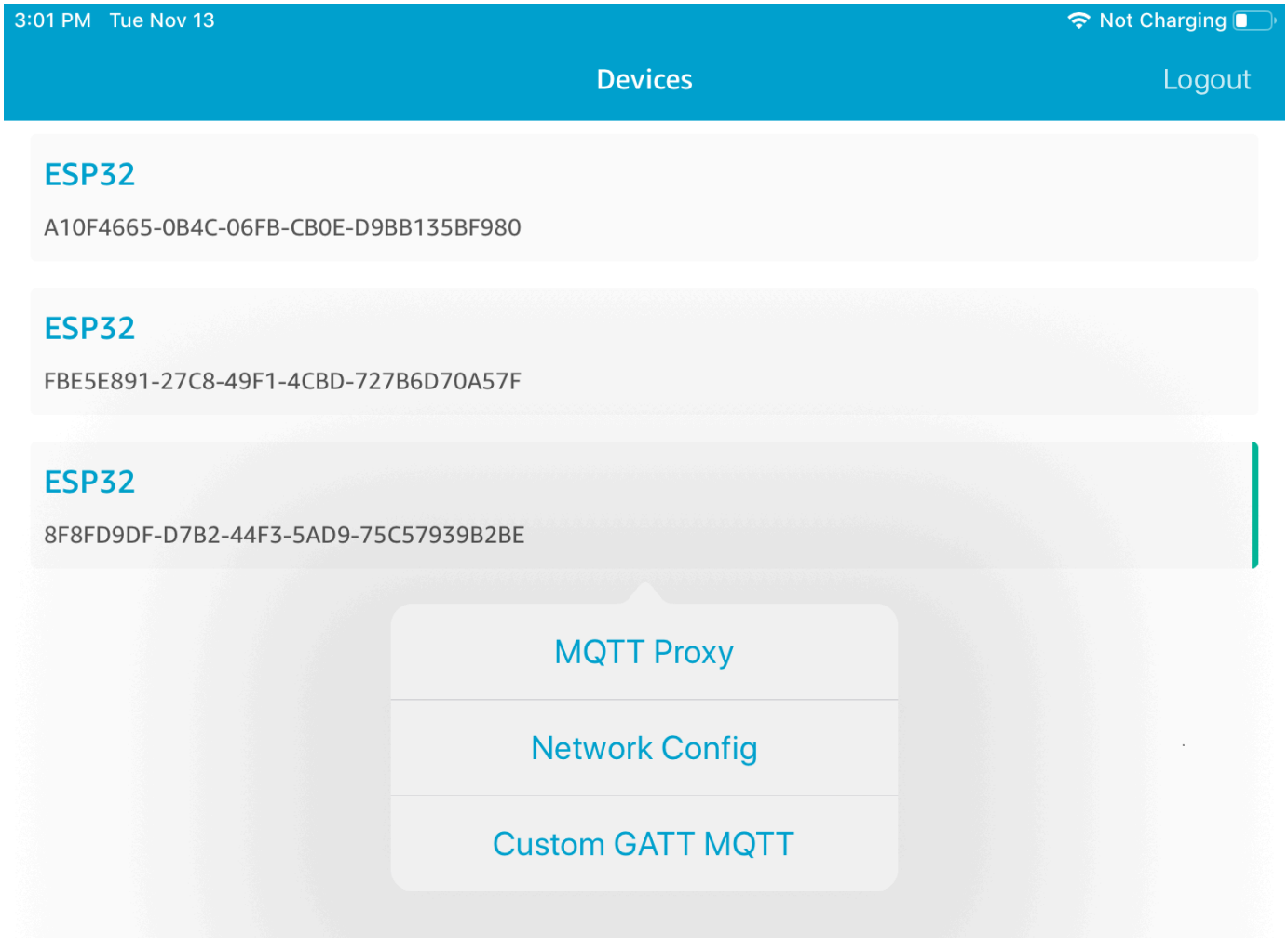
1. Abilitare il servizio di provisioning Wi-Fi. Apri `vendors/vendor/boards/board/aws_demos/config_files/iot_ble_config.h` e imposta su `#define IOT_BLE_ENABLE_WIFI_PROVISIONING 1` (dove *vendor* è il nome del fornitore e il nome della *board* scheda che stai utilizzando per eseguire le demo).

   > **Note**
   >
   > Per impostazione predefinita, il servizio di provisioning Wi-Fi è disabilitato.

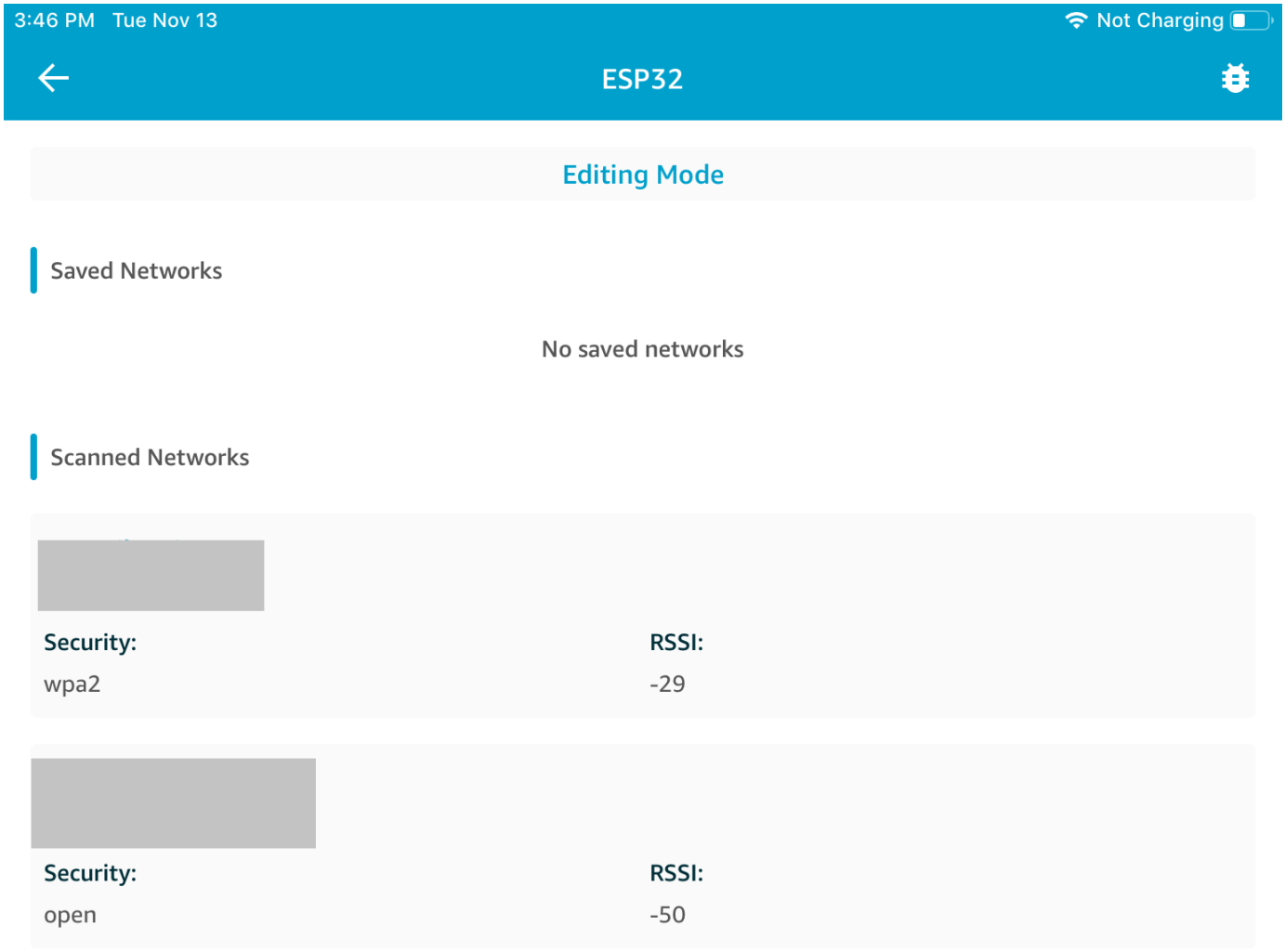
2. Configurare Network Manager per abilitare Bluetooth Low Energy e Wi-Fi.

Per eseguire la demo

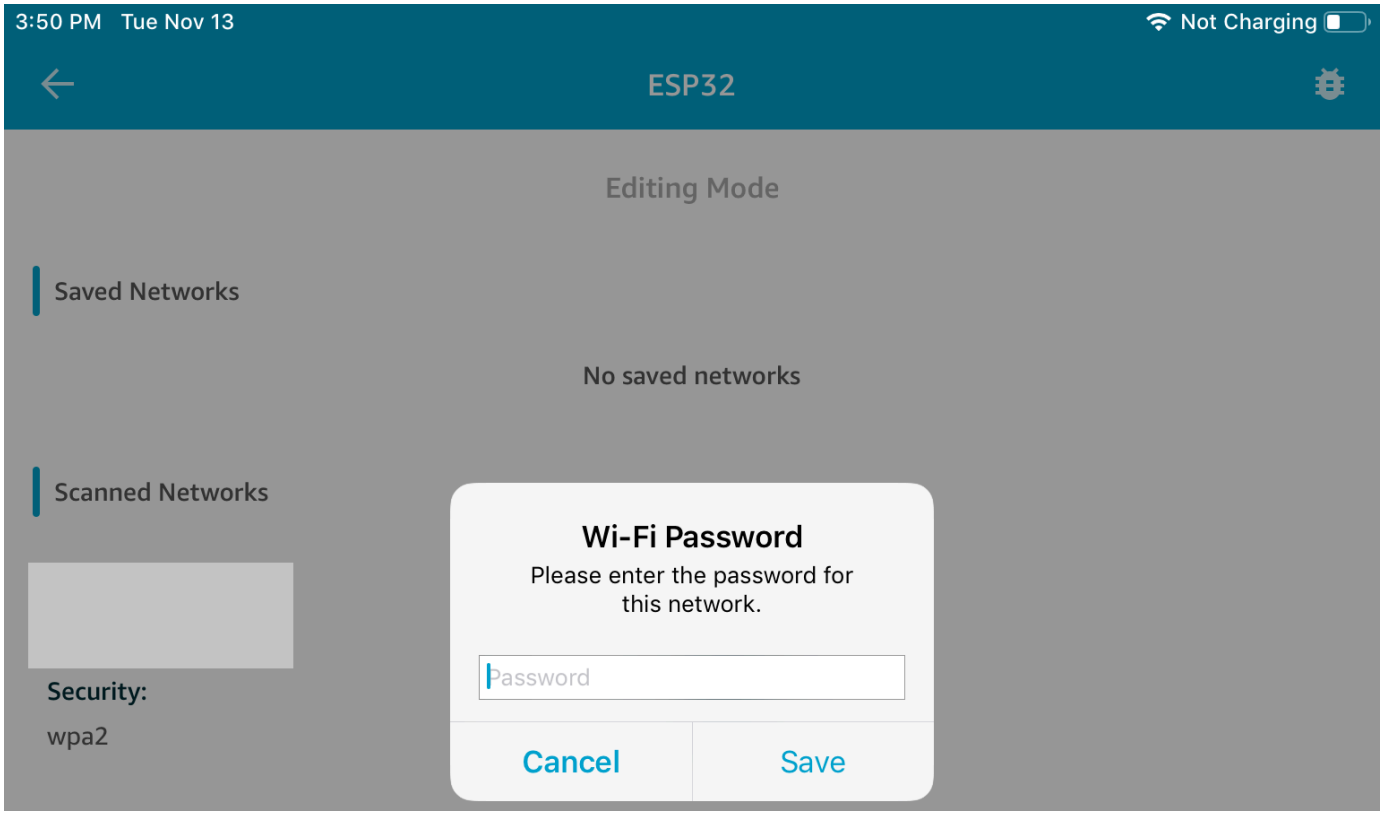
1. Compilare ed eseguire il progetto demo sul microcontroller.
2. Assicurarsi di aver accoppiato il microntroller e il dispositivo mobile utilizzando la Applicazione dimostrativa FreerTOS Bluetooth Low Energy Mobile SDK.
3. Dall'elenco Devices (Dispositivi) nell'app demo per dispositivi mobili, scegliere il microcontroller, quindi selezionare Network Config (Configurazione rete) per aprire le impostazioni di configurazione della rete.



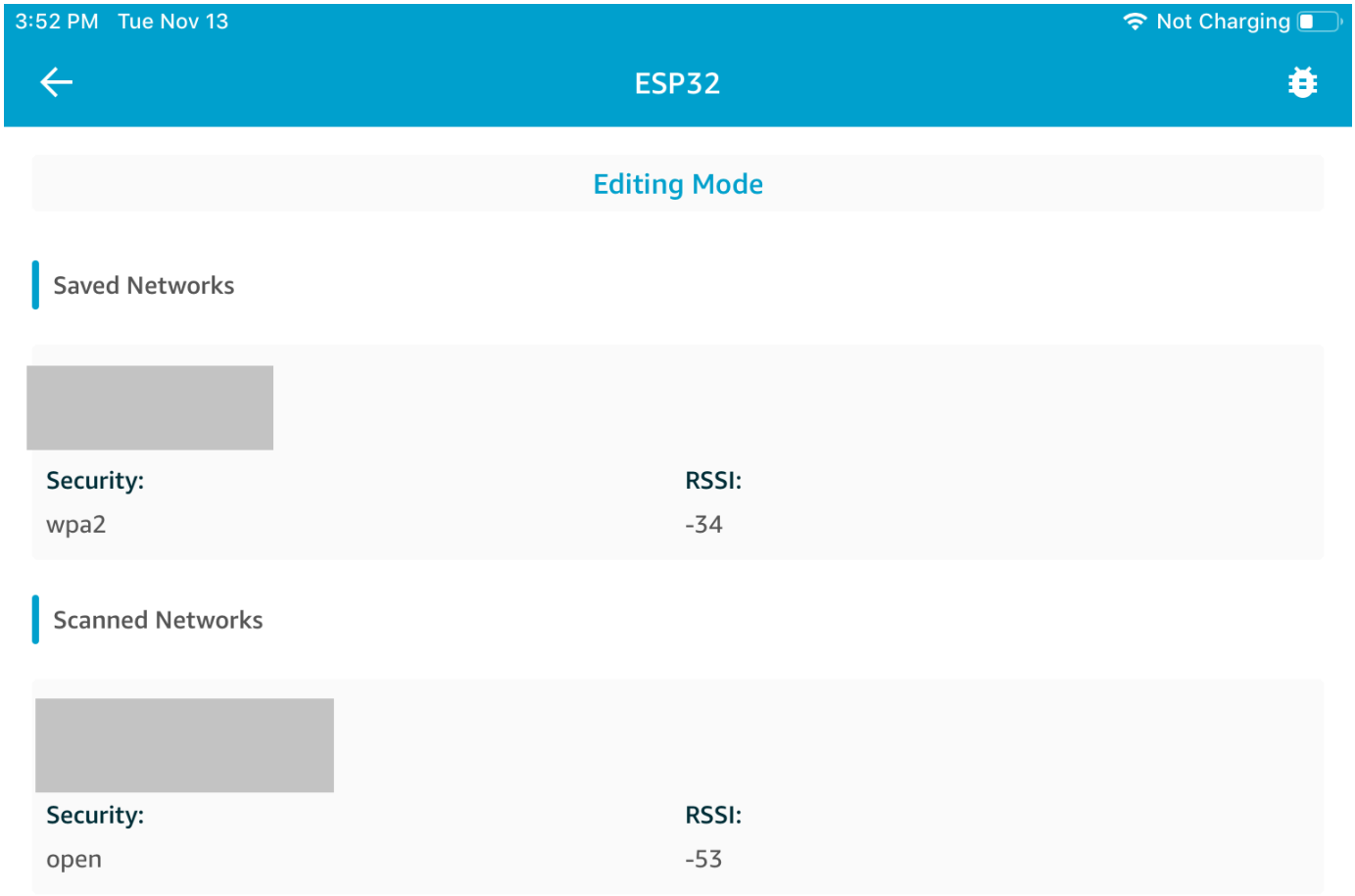
4. Dopo aver scelto Network Config (Configurazione rete) per la scheda, il microcontroller invia un elenco delle reti che si trovano vicino al dispositivo mobile. Le reti Wi-Fi disponibili vengono visualizzate in un elenco in Scanned Networks (Reti scansite).



Dall'elenco Scanned Networks (Reti scansite), scegliere la rete, quindi immettere l'SSID e la password, se richiesti.

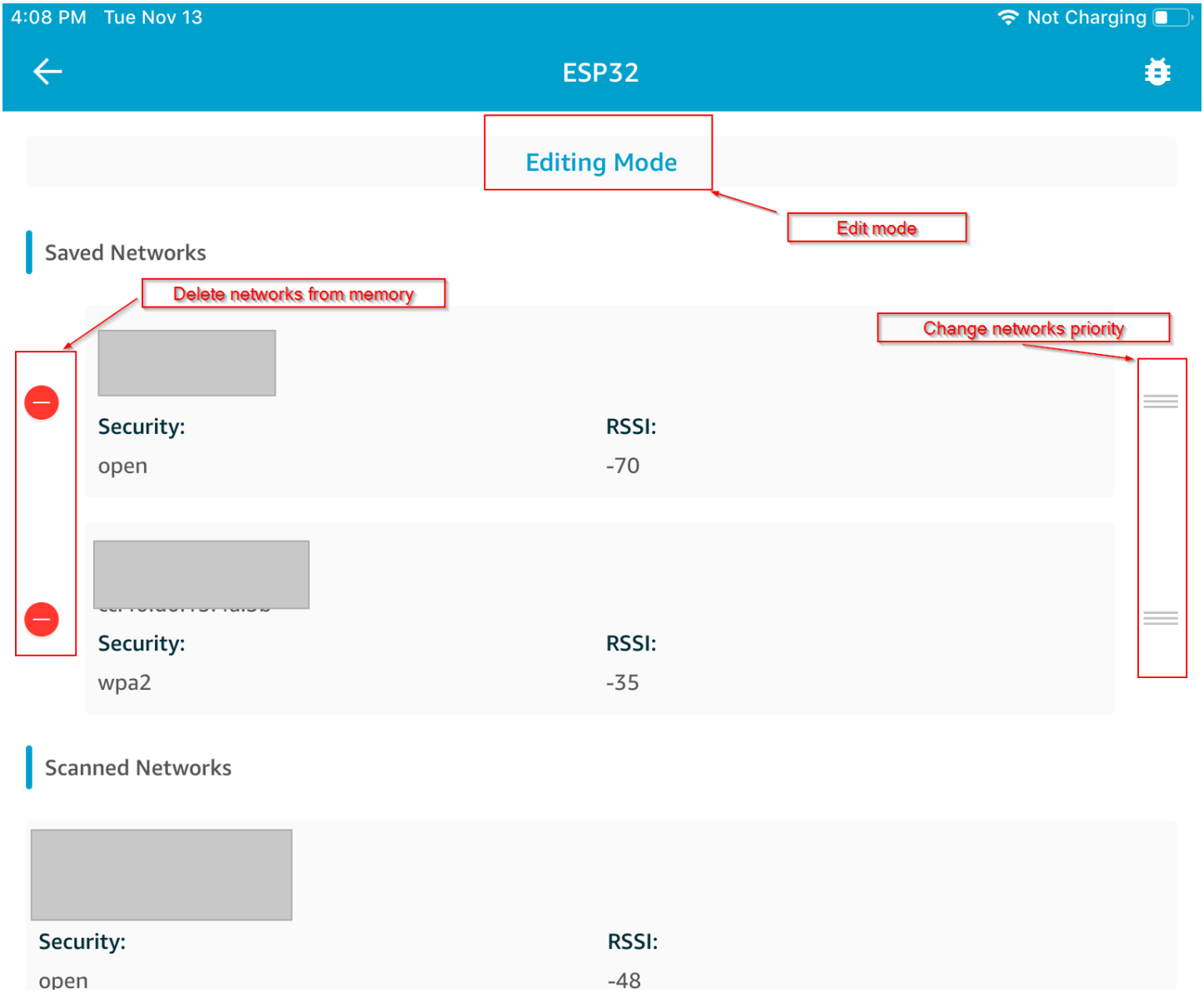


Il microcontroller si collega alla rete e la salva. La rete viene visualizzata in Saved Networks (Reti salvate).



Puoi salvare diverse reti nell'applicazione demo per dispositivi mobili. Quando riavvii l'applicazione e la demo, il microcontroller si connette alla prima rete salvata disponibile, partendo dall'inizio dell'elenco Saved Networks (Reti salvate).

Per modificare l'ordine di priorità di rete o eliminare reti, nella pagina Network Configuration (Configurazione di rete), scegli Editing Mode (Modalità di modifica). Per modificare l'ordine di priorità di rete, scegli la parte destra della rete per la quale desideri riassegnare le priorità e trascina la rete verso l'alto o verso il basso. Per eliminare una rete, scegli il pulsante rosso sul lato sinistro della rete che desideri eliminare.



Server Generic Attributes

In questo esempio, un'applicazione demo Generic Attributes (GATT) Server sul microcontroller invia un semplice valore di un contatore a Applicazione dimostrativa FreeRTOS Bluetooth Low Energy Mobile SDK.

Utilizzando Bluetooth Low Energy Mobile SDKs, puoi creare il tuo client GATT per un dispositivo mobile che si connette al server GATT sul tuo microcontrollore e funziona in parallelo con l'applicazione mobile demo.

Per abilitare la demo

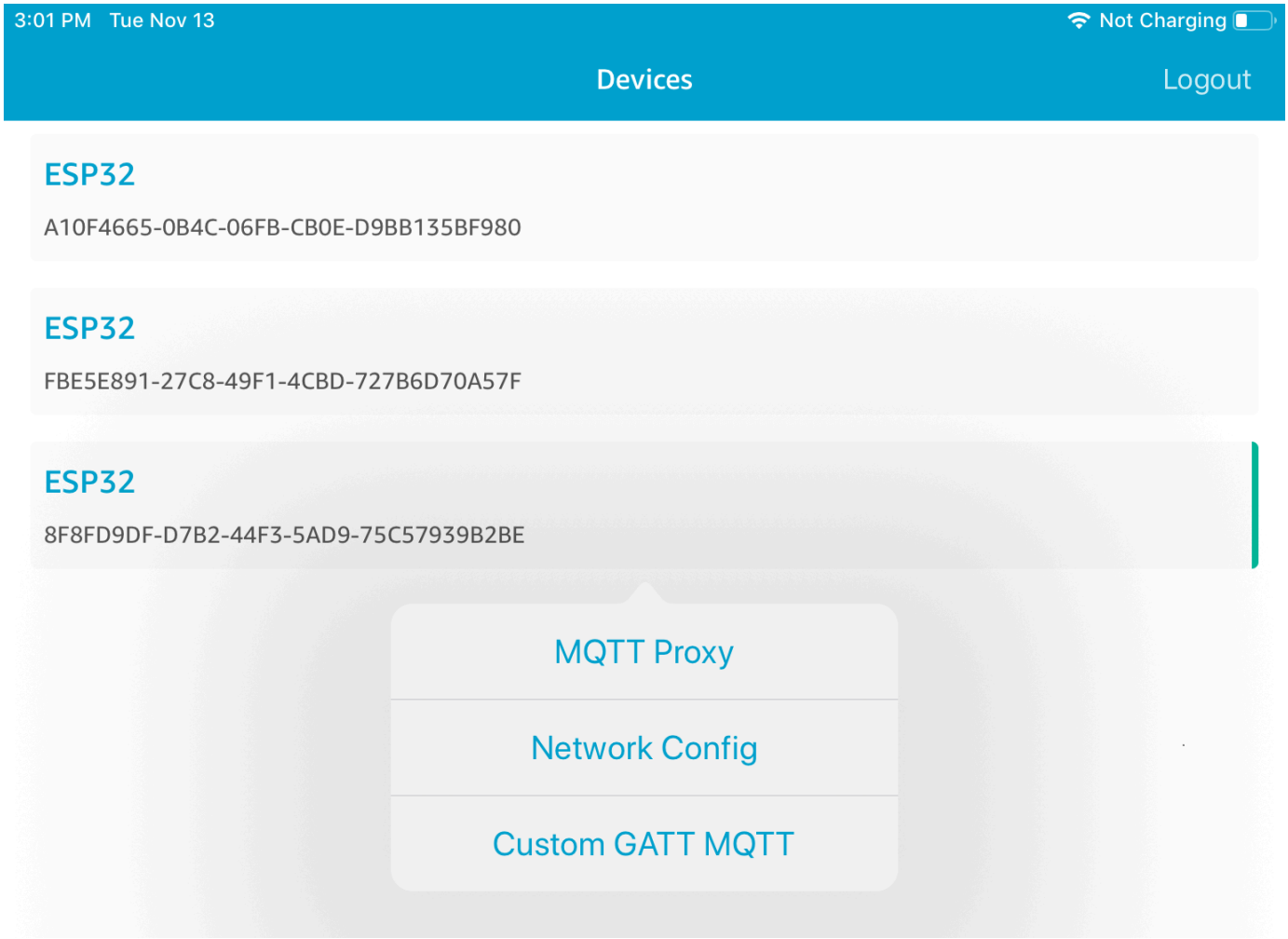
1. Abilitare la demo GATT Bluetooth Low Energy. In `vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/iot_ble_config.h` (dove *`vendor`* è il nome del fornitore e il *`board`* è il nome della scheda che state utilizzando per eseguire le demo), aggiungete `#define IOT_BLE_ADD_CUSTOM_SERVICES ( 1 )` all'elenco delle istruzioni define.

   > **Note**
   >
   > La demo GATT Bluetooth Low Energy è disabilitata per impostazione predefinita.

2. Aprire *`freertos`*`/vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/aws_demo_config.h`, commentare `#define CONFIG_MQTT_DEMO_ENABLED` e definire `CONFIG_BLE_GATT_SERVER_DEMO_ENABLED`.

Per eseguire la demo

1. Compilare ed eseguire il progetto demo sul microcontroller.
2. Assicurati di aver accoppiato la scheda e il dispositivo mobile utilizzando la Applicazione dimostrativa FreerTOS Bluetooth Low Energy Mobile SDK.
3. Dall'elenco Devices (Dispositivi) nell'applicazione, scegliere la scheda, quindi selezionare MQTT Proxy (Proxy MQTT) per aprire le opzioni proxy MQTT.



4. Tornare all'elenco Devices (Dispositivi), scegliere la scheda, quindi selezionare Custom GATT MQTT (MQTT GATT personalizzato) per aprire le opzioni del servizio GATT personalizzato.
5. Scegliere Start Counter (Avvia contatore) per iniziare a pubblicare dati nell'argomento ***your-thing-name*/example/topic** MQTT.

   Dopo aver abilitato il proxy MQTT, i messaggi Hello World e di incremento del contatore vengono visualizzati sull'argomento *your-thing-name*`/example/topic`.

Bootloader dimostrativo per Microchip Curiosity MZEF PIC32

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Questo bootloader demo implementa il controllo della versione firmware, verifica la firma crittografica e il self-test dell'applicazione. Queste funzionalità supportano gli aggiornamenti del firmware over-the-air (OTA) per FreerTOS.

La verifica del firmware include il controllo dell'autenticità e dell'integrità del nuovo firmware ricevuto over-the-air. Il bootloader consente di verificare la firma crittografica dell'applicazione prima dell'avvio. La demo utilizza l'ECDSA (Elliptic-Curve Digital Signature Algorithm) su SHA-256. L'utilità fornita può essere utilizzata per generare un'applicazione firmata che è possibile attivare sul dispositivo.

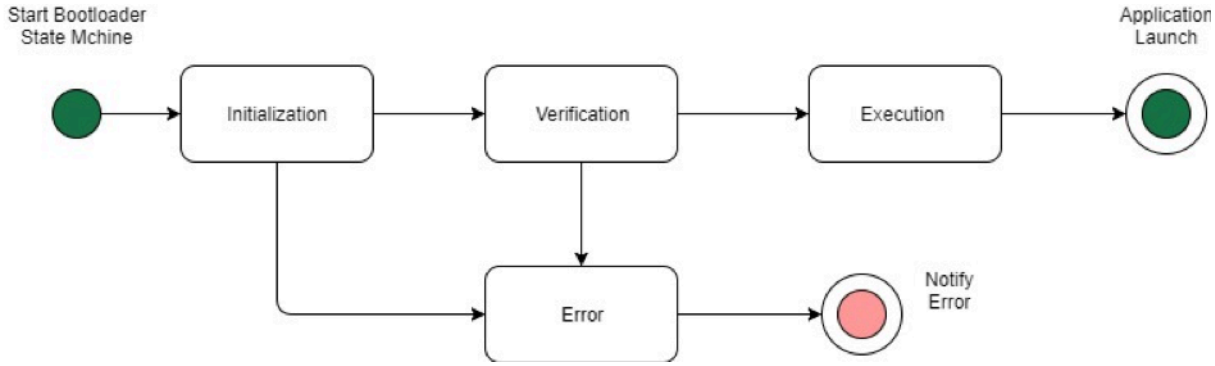
Il bootloader supporta le seguenti caratteristiche necessarie per OTA:

- Mantiene le immagini dell'applicazione sul dispositivo e passa da una all'altra.
- Consente il self-test di un'immagine OTA ricevuta e il rollback in caso di errore.
- Controlla la firma e la versione dell'immagine di aggiornamento OTA.

> **Note**
>
> Per configurare ed eseguire le demo di FreerTOS, segui i passaggi indicati. Guida introduttiva a FreerTOS

Stati del bootloader

Il processo bootloader è illustrato nel seguente stato della macchina.



Nella seguente tabella vengono descritti gli stati del bootloader.

Stato del bootloader	Descrizione
Inizializzazione	Il bootloader è in stato di inizializzazione.
Verifica	Il bootloader verifica le immagini presenti sul dispositivo.
Esegui immagine	Il bootloader lancia l'immagine selezionata.
Esegui predefinita	Il bootloader lancia l'immagine predefinita.
Errore	Il bootloader è in stato di errore.

Nel diagramma precedente, sia `Execute Image` che `Execute Default` vengono visualizzati come stato `Execution`.

Stato di esecuzione del bootloader

Il bootloader è nello `Execution` ed è pronto per avviare l'immagine verificata selezionata. Se l'immagine da avviare si trova nel banco di memoria superiore, i banchi vengono scambiati prima di eseguire l'immagine, poiché l'applicazione è sempre sviluppata per il banco di memoria inferiore.

Stato di esecuzione predefinito del bootloader

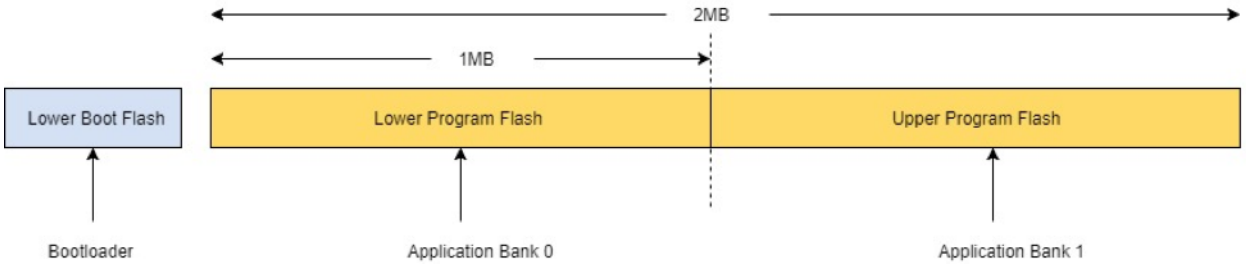
Se l'opzione di configurazione per avviare l'immagine predefinita è abilitata, il bootloader lancia l'applicazione da un indirizzo di esecuzione predefinito. Questa opzione deve essere disabilitata, ma non durante il debug.

Stato di errore del bootloader

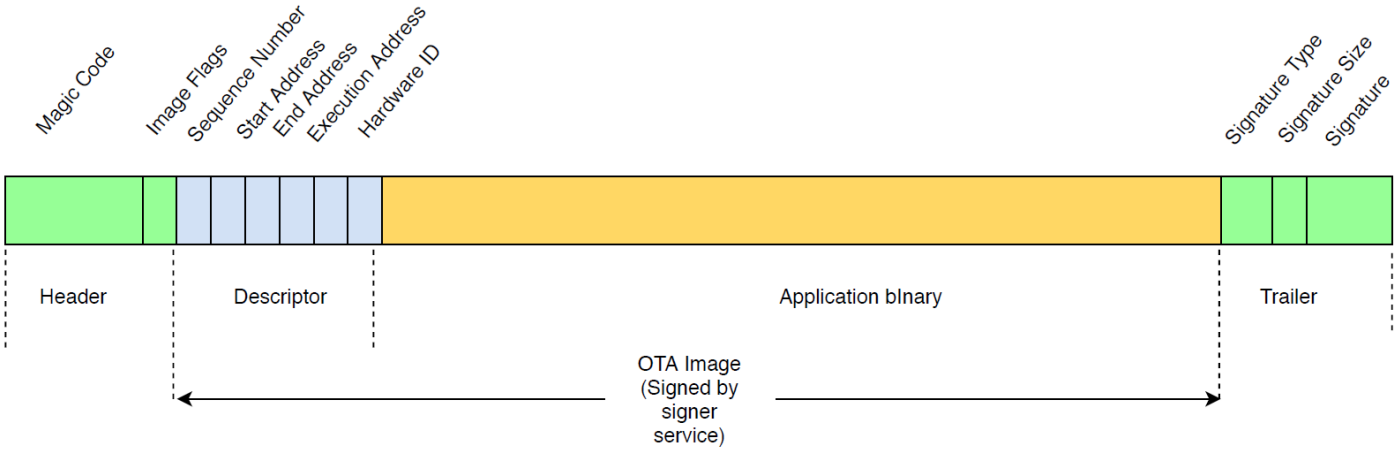
Il bootloader è in stato di errore e sul dispositivo non è presente nessuna immagine valida. Il bootloader deve inviare notifica all'utente. L'implementazione predefinita invia un messaggio di log alla console e il LED sulla scheda lampeggia velocemente in modo continuo.

Dispositivo Flash

La piattaforma Microchip Curiosity PIC32 MZEF contiene un programma flash interno di due megabyte (MB) suddiviso in due banchi. Supporta lo scambio della mappa di memoria tra questi due banchi e gli aggiornamenti in tempo reale. Il bootloader demo è programmato in un'altra regione flash di avvio inferiore separata.



Struttura dell'immagine di un'applicazione



Il diagramma mostra i componenti principali dell'immagine di applicazione archiviata su ogni banco di memoria del dispositivo.

Componente	Dimensione (in byte)
Intestazione immagine	8 byte
Descrittore immagine	24 byte
File binario applicazione	< 1 MB – (324)
Trailer	292 byte

Intestazione immagine

Le immagini dell'applicazione presenti sul dispositivo devono iniziare con un'intestazione che comprende un codice magic e flag dell'immagine.

Campo intestazione	Dimensione (in byte)
Codice magic	7 byte
Flag immagine	1 byte

Codice magic

L'immagine sul dispositivo flash deve iniziare con un codice magic. Il codice magic predefinito è `@AFRTOS`. Il bootloader verifica se il codice magic valido è presente prima di avviare l'immagine. Questa è la prima fase della verifica.

Flag immagine

I flag dell'immagine vengono utilizzati per memorizzare lo stato delle immagini dell'applicazione. I flag sono utilizzati nel processo OTA. I flag dell'immagine di entrambi i banchi di memoria determinano lo stato del dispositivo. Se l'esecuzione di immagine viene contrassegnata come commit in sospeso, significa che il dispositivo è in fase di self-test OTA. Anche se le immagini sui dispositivi contrassegnate come valide, sono sottoposte alle stesse fasi di verifica a ogni avvio. Se un'immagine viene contrassegnato come nuova, il bootloader la contrassegna come commit in sospeso e la avvia per il self-test dopo la verifica. Il bootloader, inoltre, inizializza e avvia il timer di watchdog, in modo

che se il self-test della nuova immagine OTA ha esito negativo, il dispositivo si riavvia e bootloader rifiuta l'immagine cancellandola, quindi esegue l'immagine valida precedente.

Il dispositivo può avere una sola immagine valida. L'altra immagine può essere una nuova immagine OTA o un commit in sospeso (self-test). Dopo aver eseguito correttamente l'aggiornamento OTA, l'immagine precedente viene cancellata dal dispositivo.

Stato	Valore	Descrizione
Nuova immagine	0xFF	Applicazione immagine è nuova e non è stata mai eseguita.
Commit in sospeso	0xFE	L'immagine dell'applicazione viene contrassegnata per l'esecuzione del test.
Valida	0xFC	L'immagine dell'applicazione viene contrassegnata come valida e sottoposta a commit.
Non valido	0xF8	L'immagine dell'applicazione viene contrassegnata come non valida.

Descrittore immagine

L'immagine dell'applicazione sul dispositivo flash deve contenere il descrittore immagine appena dopo l'intestazione immagine. Il descrittore immagine viene generato da una utilità post-build che utilizza i file di configurazione (`ota-descriptor.config`) per generare il descrittore appropriato e lo antepone al file binario dell'applicazione. L'output di questa fase post-build è l'immagine binaria da utilizzare per l'OTA.

Campo descrittore	Dimensione (in byte)
Numero sequenza	4 byte
Indirizzo iniziale	4 byte

Campo descrittore	Dimensione (in byte)
Indirizzo finale	4 byte
Indirizzo di esecuzione	4 byte
ID hardware	4 byte
Riservata	4 byte

Numero sequenza

Il numero di sequenza deve essere incrementato prima di creare una nuova immagine OTA. Consulta il file `ota-descriptor.config`. Il bootloader utilizza questo numero per determinare l'immagine di avvio. I valori validi sono compresi tra 1 e 4294967295.

Indirizzo iniziale

L'indirizzo iniziale dell'immagine dell'applicazione sul dispositivo. Poiché il descrittore dell'immagine viene anteposto al file binario dell'applicazione, questo indirizzo è l'avvio del descrittore dell'immagine.

Indirizzo finale

L'indirizzo finale dell'immagine dell'applicazione sul dispositivo, escluso il trailer dell'immagine.

Indirizzo di esecuzione

L'indirizzo di esecuzione dell'immagine.

ID hardware

Un ID univoco dell'hardware utilizzato dal bootloader per verificare che l'immagine OTA sia creata per la piattaforma corretta.

Riservata

Questo campo è riservato per un utilizzo futuro.

Trailer dell'immagine

Il trailer dell'immagine viene accodato al file binario dell'applicazione. Contiene la stringa di tipo firma, le dimensioni della firma e la firma dell'immagine.

Campo trailer	Dimensione (in byte)
Tipo firma	32 byte
Dimensioni firma	4 byte
Firma	256 byte

Tipo firma

Il tipo di firma è una stringa che rappresenta l'algoritmo crittografico utilizzato e rappresenta un contrassegno per il trailer. Il bootloader supporta l'ECDSA (Elliptic-Curve Digital Signature Algorithm). Il valore predefinito è sig-sha256-ecdsa.

Dimensioni firma

La dimensione della firma crittografica in byte.

Firma

La firma crittografica del file binario dell'applicazione preceduta dal descrittore dell'immagine.

Configurazione del bootloader

Le opzioni di configurazione del bootloader di base sono fornite in *freertos*`/vendors/microchip/boards/curiosity_pic32mzef/bootloader/config_files/aws_boot_config.h`. Alcune opzioni vengono fornite esclusivamente a scopo di debug.

Abilita avvio predefinito

Abilita l'esecuzione dell'applicazione dall'indirizzo predefinito e deve essere abilitato solo per il debug. L'immagine viene eseguita dall'indirizzo predefinito senza alcuna verifica.

Abilita verifica firma crittografica

Abilita verifica firma crittografica. Le immagini con stato non riuscito vengono cancellate dal dispositivo. Questa opzione è disponibile solo per scopi di debug e deve rimanere abilitata in produzione.

Cancella immagine non valida

Abilita la cancellazione dell'intero banco di memoria se la verifica dell'immagine su quel banco ha esito negativo. L'opzione è disponibile solo per scopi di debug e deve rimanere abilitata in produzione.

Abilita verifica ID hardware

Abilita la verifica dell'ID hardware nel descrittore dell'immagine OTA e dell'ID hardware programmati nel bootloader. Si tratta di un'opzione facoltativa che può essere disattivata se la verifica dell'ID hardware non è richiesta.

Abilita verifica indirizzo

Abilita la verifica degli indirizzi di inizio, di fine e di esecuzione nel descrittore dell'immagine OTA. Ti consigliamo di tenere questa opzione abilitata.

Creazione del bootloader

Il bootloader demo è incluso come progetto caricabile nel progetto che si trova nel *freertos*/vendors/microchip/boards/curiosity_pic32mzef/aws_demos/mplab/ repository del aws_demos codice sorgente di FreerTOS. Quando viene creato il progetto aws_demos, crea prima il bootloader, poi l'applicazione. L'output finale è un'immagine esadecimale unificata che include il bootloader e l'applicazione. L'utilità factory_image_generator.py viene fornita per generare un'immagine esadecimale unificata con firma crittografica. Gli script dell'utilità bootloader si trovano in *freertos*/demos/ota/bootloader/utility/.

Fase di pre-compilazione del bootloader

Questa utilità preinstallata esegue uno script dell'utilità chiamato codesigner_cert_utility.py che estrae la chiave pubblica dal certificato di firma del codice e genera un file di intestazione C che contiene la chiave pubblica nel formato codificato ASN.1 (Abstract Syntax Notation One). Questa intestazione è compilata nel progetto bootloader. L'intestazione generata contiene due costanti: una serie di chiavi pubbliche e la lunghezza della chiave. Il progetto bootloader può anche essere creato senza aws_demos e può essere sottoposto a debug come una normale applicazione.

AWS IoT Device Defender demo

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Introduzione

Questa demo mostra come utilizzare la libreria AWS IoT Device Shadow a cui connettersi AWS IoT Device Defender. La demo utilizza la libreria CoreMQTT per stabilire una connessione MQTT con TLS (autenticazione reciproca) al broker AWS IoT MQTT e la libreria CoreJSON per convalidare e analizzare le risposte ricevute. AWS IoT Device Defender La demo mostra come creare un report in formato JSON utilizzando le metriche raccolte dal dispositivo e come inviare il rapporto a. AWS IoT Device Defender La demo mostra anche come registrare una funzione di callback con la libreria CoreMQTT per gestire le risposte AWS IoT Device Defender inviate per confermare se un report è stato accettato o rifiutato.

> **Note**
>
> Per configurare ed eseguire le demo di FreerTOS, segui i passaggi indicati. Guida introduttiva a FreerTOS

Funzionalità

Questa demo crea una singola attività applicativa che dimostra come raccogliere metriche, creare un report Device Defender in formato JSON e inviarlo AWS IoT Device Defender tramite una connessione MQTT sicura a MQTT Broker. AWS IoT

Il modo in cui raccogliamo le metriche dipende dallo stack TCP/IP in uso. Per FreeRTOS+TCP e configurazioni LWIP supportate, forniamo implementazioni di raccolta di metriche che raccolgono metriche reali dal dispositivo e le inviano nel rapporto. AWS IoT Device Defender Puoi trovare le implementazioni per FreeRTOS+TCP e LWip su. GitHub

Per le schede che utilizzano qualsiasi altro stack TCP/IP, vengono fornite le definizioni stub delle funzioni di raccolta delle metriche che restituiscono zeri per tutte le metriche. Per inviare metriche reali alle schede utilizzando questa implementazione stub, implementa le funzioni nel tuo stack di rete. *freertos*`/demos/device_defender_for_aws/metrics_collector/stub/metrics_collector.c` Il file è disponibile anche sul sito Web. GitHub

Infatti ESP32, la configurazione LWip predefinita non utilizza il core locking e quindi la demo utilizzerà metriche stubbed. Se desideri utilizzare l'implementazione della raccolta di metriche LWip di riferimento, definisci le seguenti macro in: `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
```

Di seguito è riportato un esempio di output quando si esegue la demo.

```
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 la scheda non utilizza FreeRTOS+TCP o una configurazione LWIP supportata, l'output sarà simile al seguente.

```
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-------
```

Il codice sorgente della demo si trova nella directory di download o sul sito web. *freertos*/demos/device_defender_for_aws/ GitHub

Iscrizione agli argomenti AWS IoT Device Defender

La funzione subscribeToDefenderTopics sottoscrive gli argomenti MQTT sui quali verranno ricevute le risposte ai report pubblicati da Device Defender. Utilizza la macro `DEFENDER_API_JSON_ACCEPTED` per costruire la stringa dell'argomento su cui vengono ricevute le risposte ai report di Device Defender accettati. Utilizza la macro `DEFENDER_API_JSON_REJECTED` per costruire la stringa dell'argomento in base alla quale verranno ricevute le risposte ai report di Device Defender rifiutati.

Raccolta delle metriche dei dispositivi

La collectDeviceMetricsfunzione raccoglie le metriche di rete utilizzando le funzioni definite in. `metrics_collector.h` Le metriche raccolte sono il numero di byte e pacchetti inviati e ricevuti, le porte TCP aperte, le porte UDP aperte e le connessioni TCP stabilite.

Generazione del report AWS IoT Device Defender

La funzione generateDeviceMetricsReport genera un rapporto Device Defender utilizzando la funzione definita in`report_builder.h`. Questa funzione prende le metriche di rete e un buffer, crea un documento JSON nel formato previsto da AWS IoT Device Defender e lo scrive nel buffer fornito. Il formato del documento JSON previsto da AWS IoT Device Defender è specificato nelle metriche lato dispositivo nella Guida per gli sviluppatori.AWS IoT

Pubblicazione del rapporto AWS IoT Device Defender

Il AWS IoT Device Defender rapporto è pubblicato sull'argomento MQTT per la pubblicazione di report JSON AWS IoT Device Defender . Il report viene creato utilizzando la macro`DEFENDER_API_JSON_PUBLISH`, come illustrato in questo frammento di codice sul sito Web. GitHub

Callback per la gestione delle risposte

La funzione publishCallback gestisce i messaggi di pubblicazione MQTT in entrata. Utilizza l'`Defender_MatchTopic`API della AWS IoT Device Defender libreria per verificare se il messaggio MQTT in arrivo proviene dal servizio. AWS IoT Device Defender Se il messaggio proviene da AWS IoT Device Defender, analizza la risposta JSON ricevuta ed estrae l'ID del report nella risposta. Viene quindi verificato che l'ID del rapporto sia uguale a quello inviato nel rapporto.

AWS IoT Greengrass applicazione demo discovery

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Prima di eseguire la demo di AWS IoT Greengrass Discovery per FreerTOS, devi AWS configurare AWS IoT Greengrass e. AWS IoT Per la configurazione AWS, segui le istruzioni riportate all'indirizzo. Configurazione dell'account e delle autorizzazioni AWS Per configurare AWS IoT Greengrass, è necessario creare un gruppo Greengrass e quindi aggiungere un core Greengrass. Per ulteriori informazioni sulla configurazione AWS IoT Greengrass, consulta Guida introduttiva. AWS IoT Greengrass

Dopo aver configurato AWS e AWS IoT Greengrass, devi configurare alcune autorizzazioni aggiuntive per AWS IoT Greengrass.

Per configurare AWS IoT Greengrass le autorizzazioni

1. Accedi alla console IAM.
2. Dal pannello di navigazione, scegli Ruoli, quindi trova e scegli Greengrass_ ServiceRole.
3. Scegli Allega policy, seleziona AmazonS3 FullAccess e AWSIoTFullAccess, quindi scegli Allega policy.
4. Passare alla console AWS IoT.
5. Nel riquadro di navigazione, selezionare Greengrass (Greengrass), poi Groups (Gruppi), quindi selezionare il gruppo Greengrass creato in precedenza.
6. Selezionare Impostazioni, quindi Add role (Aggiungi ruolo).
7. Scegli Greengrass_ ServiceRole, quindi scegli Salva.

Connect la tua scheda AWS IoT e configura la tua demo di FreerTOS.

1. Registrazione della scheda MCU con AWS IoT

   Dopo avere registrato la scheda, è necessario creare e collegare una policy Greengrass al certificato del dispositivo.

Per creare una nuova politica AWS IoT Greengrass

1. Passare alla console AWS IoT.
2. Nel riquadro di navigazione, selezionare Secure (Sicurezza), scegliere Policies (Policy) e poi Create (Crea).
3. Inserire un nome per identificare la policy.
4. Nella sezione Add statements (Aggiungi istruzioni), scegliere Advanced mode (Modalità avanzata). Copiare e incollare il seguente JSON nella finestra dell'editor policy:

   ```
   {
         "Effect": "Allow",
         "Action": [
           "greengrass:*"
         ],
         "Resource": "*"
   }
   ```

   Questa politica concede AWS IoT Greengrass le autorizzazioni a tutte le risorse.
5. Scegli Create (Crea) .

Per allegare la AWS IoT Greengrass politica al certificato del tuo dispositivo

1. Passare alla console AWS IoT.
2. Nel riquadro di navigazione, selezionare Manage (Gestisci), poi Things (Oggetti), quindi selezionare l'oggetto creato in precedenza.
3. Selezionare Security (Sicurezza), quindi selezionare il certificato collegato al dispositivo.
4. Selezionare Policies (Policy), poi Actions (Operazioni), quindi selezionare Attach Policy (Collega policy).
5. Individuare e selezionare la policy Greengrass creata in precedenza, quindi selezionare Attach (Collega).

2. Scaricare FreeRTOS

> **Note**
>
> Se stai scaricando FreerTOS dalla console FreerTOS, scegli Connetti a - invece di Connetti a AWS IoT Greengrass -. ***Platform*** AWS IoT***Platform***

3. Configurazione delle demo di FreerTOS.

   Aprire *freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h, commentare `#define CONFIG_MQTT_DEMO_ENABLED` e definire `CONFIG_GREENGRASS_DISCOVERY_DEMO_ENABLED`.

Dopo aver installato e scaricato AWS IoT e AWS IoT Greengrass configurato FreerTOS, puoi creare, flashare ed eseguire la demo di Greengrass sul tuo dispositivo. Per configurare l'ambiente di sviluppo hardware e software della scheda, segui le istruzioni indicate in Guide alle operazioni di base specifiche per la scheda.

La demo di Greengrass pubblica una serie di messaggi al core di Greengrass e al client MQTT. AWS IoT Per visualizzare i messaggi nel client AWS IoT MQTT, apri la AWS IoT console, scegli Test, quindi aggiungi un abbonamento a. `freertos/demos/ggd`

Nel client MQTT, consulta le seguenti stringhe:

```
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
```

Utilizzo di un' EC2 istanza Amazon

Se stai lavorando con un' EC2 istanza Amazon

1. Trova il DNS pubblico (IPv4) associato alla tua EC2 istanza Amazon: vai alla EC2 console Amazon e, nel pannello di navigazione a sinistra, scegli Istanze. Scegli la tua EC2 istanza Amazon, quindi scegli il pannello Descrizione. Cerca la voce relativa al DNS pubblico (IPv4) e prendine nota.
2. Trova la voce relativa ai gruppi di sicurezza e scegli il gruppo di sicurezza collegato alla tua EC2 istanza Amazon.

3. Scegli la scheda Inbound rules (Regole in entrata) quindi scegli Edit inbound rules (Modifica regole in entrata) e aggiungi le seguenti regole.

   Regole in entrata

Tipo	Protocollo	Intervallo porte	Origine	Descrizione (facoltativa)
HTTP	TCP	80	0.0.0.0/0	-
HTTP	TCP	80	::/0	-
SSH	TCP	22	0.0.0.0/0	-
TCP personalizzato	TCP	8883	0.0.0.0/0	Comunicazioni MQTT
TCP personalizzato	TCP	8883	::/0	Comunicazioni MQTT
HTTPS	TCP	443	0.0.0.0/0	-
HTTPS	TCP	443	::0/0	-
Tutto ICMP - IPv4	ICMP	Tutti	0.0.0.0/0	-
Tutti gli ICMP - IPv4	ICMP	Tutti	::0/0	-

4. Nella AWS IoT console scegli Greengrass, quindi Gruppi e scegli il gruppo Greengrass che hai creato in precedenza. Seleziona Impostazioni. Modifica Local connection detection (Rileva connessione locale) in Manually manage connection information (Gestisci manualmente le informazioni di connessione).
5. Nel riquadro di navigazione, scegli Cores (Nuclei) quindi seleziona il nucleo del gruppo.
6. Scegli Connectivity (Connettività) e assicurati di avere un solo endpoint principale (elimina tutto il resto) e che non sia un indirizzo IP (perché è soggetto a modifiche). L'opzione migliore è usare il DNS pubblico (IPv4) che hai annotato nel primo passaggio.
7. Aggiungi l’oggetto IoT di FreeRTOS che hai creato al gruppo GG.

a. Scegli la freccia indietro per tornare alla pagina del AWS IoT Greengrass gruppo. Nel riquadro di navigazione, scegli Devices (Dispositivi) quindi scegli Add Device (Aggiungi dispositivo).

b. Scegli Select an IoT Thing (Seleziona un oggetto IoT). Scegli il tuo dispositivo, quindi scegli Finish (Fine).

8. Aggiungi gli abbonamenti necessari: nella pagina Greengrass Group, scegli Abbonamenti, quindi scegli Aggiungi abbonamento e inserisci le informazioni come mostrato qui.

   Sottoscrizioni

Origine	Target	Argomento
TIGG1	IoT Cloud	freertos/demos/ggd

   Dove «Source» è il nome dato all' AWS IoT elemento creato nella AWS IoT console al momento della registrazione della scheda, "TIGG1" nell'esempio riportato qui.

9. Avvia una distribuzione del tuo AWS IoT Greengrass gruppo e assicurati che la distribuzione abbia esito positivo. Ora dovresti essere in grado di eseguire correttamente la demo di AWS IoT Greengrass discovery.

Demo CoreHTTP

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Queste demo possono aiutarti a imparare a usare la libreria CoreHTTP.

Argomenti

- Demo sull'autenticazione reciproca CoreHTTP
- Demo di caricamento di base di CoreHTTP su Amazon S3
- Scarica la demo di CoreHTTP basic S3

- Demo multithread di base di CoreHTTP

Demo sull'autenticazione reciproca CoreHTTP

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Introduzione

Il progetto dimostrativo CoreHTTP (Mutual Authentication) mostra come stabilire una connessione a un server HTTP utilizzando TLS con autenticazione reciproca tra il client e il server. Questa demo utilizza un'implementazione dell'interfaccia di trasporto basata su MBEDTLS per stabilire una connessione TLS autenticata da server e client e dimostra un flusso di lavoro di risposta alle richieste in HTTP.

> **Note**
>
> Per configurare ed eseguire le demo di FreerTOS, segui i passaggi indicati. Guida introduttiva a FreerTOS

Funzionalità

Questa demo crea una singola attività applicativa con esempi che mostrano come completare quanto segue:

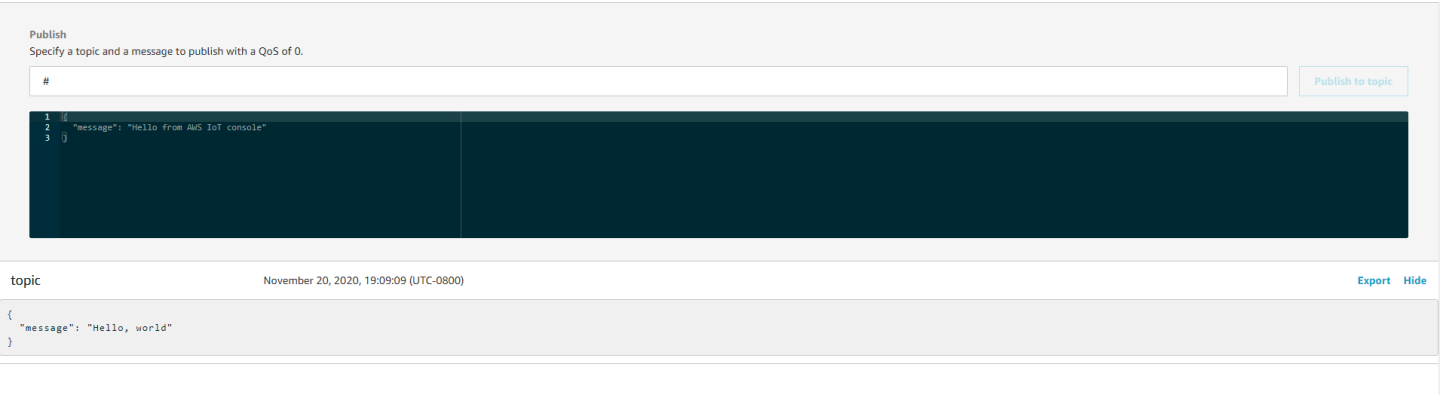
- Connect al server HTTP sull' AWS IoT endpoint.
- Invia una richiesta POST.
- Ricevi la risposta.
- Disconnettiti dal server.

Dopo aver completato questi passaggi, la demo genera un output simile alla schermata seguente.

```
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------
```

La AWS IoT console genera un output simile alla schermata seguente.



Organizzazione del codice sorgente

Il file sorgente della demo è denominato `http_demo_mutual_auth.c` e può essere trovato nella *`freertos`*`/demos/coreHTTP/` directory e sul GitHubsito Web.

Connessione al server AWS IoT HTTP

La connectToServerWithBackoffRetriesfunzione tenta di stabilire una connessione TLS con autenticazione reciproca al server HTTP. AWS IoT Se la connessione fallisce, riprova dopo un timeout. Il valore di timeout aumenta esponenzialmente fino a raggiungere il numero massimo di tentativi o il valore di timeout massimo. La `RetryUtils_BackoffAndSleep` funzione fornisce valori di timeout che aumentano in modo esponenziale e restituisce `RetryUtilsRetriesExhausted` quando è stato raggiunto il numero massimo di tentativi. La `connectToServerWithBackoffRetries` funzione restituisce uno stato di errore se la connessione TLS al broker non può essere stabilita dopo il numero di tentativi configurato.

Invio di una richiesta HTTP e ricezione della risposta

La funzione prvSendHttpRequest dimostra come inviare una richiesta POST al server AWS IoT HTTP. Per ulteriori informazioni su come effettuare una richiesta all'API REST in AWS IoT, vedete

Device communication protocol - HTTPS. La risposta viene ricevuta con la stessa chiamata all'API CoreHTTP,`HTTPClient_Send`.

Demo di caricamento di base di CoreHTTP su Amazon S3

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Introduzione

Questo esempio dimostra come inviare una richiesta PUT al server HTTP Amazon Simple Storage Service (Amazon S3) e caricare un file di piccole dimensioni. Esegue anche una richiesta GET per verificare la dimensione del file dopo il caricamento. Questo esempio utilizza un'interfaccia di trasporto di rete che utilizza mbedTLS per stabilire una connessione reciprocamente autenticata tra un client di dispositivo IoT che esegue CoreHTTP e il server HTTP Amazon S3.

> **Note**
>
> Per configurare ed eseguire le demo di FreerTOS, segui i passaggi indicati. Guida introduttiva a FreerTOS

Filettatura singola contro filettatura multipla

Esistono due modelli di utilizzo CoreHTTP, a thread singolo e multithread (multitasking). Sebbene la demo in questa sezione esegua la libreria HTTP in un thread, in realtà dimostra come utilizzare CoreHTTP in un ambiente a thread singolo. Solo un'attività in questa demo utilizza l'API HTTP. Sebbene le applicazioni a thread singolo debbano chiamare ripetutamente la libreria HTTP, le applicazioni multithread possono invece inviare richieste HTTP in background all'interno di un'attività agente (o daemon).

Organizzazione del codice sorgente

Il file sorgente della demo è denominato `http_demo_s3_upload.c` e può essere trovato nella *`freertos`*`/demos/coreHTTP/` directory e sul GitHubsito Web.

Configurazione della connessione al server HTTP Amazon S3

Questa demo utilizza un URL prefirmato per connettersi al server HTTP Amazon S3 e autorizzare l'accesso all'oggetto da scaricare. La connessione TLS del server HTTP Amazon S3 utilizza solo l'autenticazione del server. A livello di applicazione, l'accesso all'oggetto viene autenticato con i parametri della query URL prefirmata. Segui i passaggi seguenti per configurare la connessione a. AWS

1. Configura un AWS account:

   a. Se non l'hai già fatto, crea un AWS account.

   b. Gli account e le autorizzazioni vengono impostati utilizzando AWS Identity and Access Management (IAM). Utilizzi IAM per gestire le autorizzazioni per ogni utente del tuo account. Per impostazione predefinita, un utente non dispone delle autorizzazioni fino a quando non le concede il proprietario principale.

      i. Per aggiungere un utente IAM al tuo AWS account, consulta la IAM User Guide.

      ii. Concedi l'autorizzazione al tuo AWS account per accedere a FreerTOS AWS IoT e aggiungendo questa politica:

      - Amazon S3 FullAccess

2. Crea un bucket in Amazon S3 seguendo la procedura descritta in Come si crea un bucket S3? nella Guida per l'utente di Amazon Simple Storage Service.
3. Carica un file su Amazon S3 seguendo la procedura descritta in Come si caricano file e cartelle in un bucket S3? .
4. Genera un URL prefirmato utilizzando lo script che si trova nel `FreeRTOS-Plus/Demo/coreHTTP_Windows_Simulator/Common/presigned_url_generator/presigned_urls_gen.py` file.

   Per istruzioni sull'uso, consulta il `FreeRTOS-Plus/Demo/coreHTTP_Windows_Simulator/Common/presigned_url_generato/README.md` file.

Funzionalità

La demo si connette innanzitutto al server HTTP Amazon S3 con l'autenticazione del server TLS. Quindi, crea una richiesta HTTP per caricare i dati specificati in. `democonfigDEMO_HTTP_UPLOAD_DATA` Dopo aver caricato il file, verifica che il file sia stato

caricato correttamente richiedendo la dimensione del file. Il codice sorgente della demo è disponibile sul sito Web. GitHub

Connessione al server HTTP Amazon S3

La connectToServerWithBackoffRetriesfunzione tenta di stabilire una connessione TCP al server HTTP. Se la connessione fallisce, riprova dopo un timeout. Il valore di timeout aumenterà esponenzialmente fino al raggiungimento del numero massimo di tentativi o al raggiungimento del valore di timeout massimo. La `connectToServerWithBackoffRetries` funzione restituisce uno stato di errore se non è possibile stabilire la connessione TCP al server dopo il numero di tentativi configurato.

La `prvConnectToServer` funzione dimostra come stabilire una connessione al server HTTP Amazon S3 utilizzando solo l'autenticazione del server. Utilizza l'interfaccia di trasporto basata su MBEDTLS implementata nel file. `FreeRTOS-Plus/Source/Application-Protocols/network_transport/freertos_plus_tcp/using_mbedtls/using_mbedtls.c` La definizione di `prvConnectToServer` può essere trovata sul sito web. GitHub

Caricare dati

La `prvUploadS3ObjectFile` funzione dimostra come creare una richiesta PUT e specificare il file da caricare. Il bucket Amazon S3 in cui viene caricato il file e il nome del file da caricare sono specificati nell'URL prefirmato. Per risparmiare memoria, viene utilizzato lo stesso buffer sia per le intestazioni della richiesta che per ricevere la risposta. La risposta viene ricevuta in modo sincrono utilizzando la `HTTPClient_Send` funzione API. È previsto un codice di stato della `200 OK` risposta dal server HTTP Amazon S3. Qualsiasi altro codice di stato è un errore.

Il codice sorgente di `prvUploadS3ObjectFile()` è disponibile sul GitHubsito Web.

Verifica del caricamento

La `prvVerifyS3ObjectFileSize` funzione chiama `prvGetS3ObjectFileSize` per recuperare la dimensione dell'oggetto nel bucket S3. Il server HTTP Amazon S3 attualmente non supporta le richieste HEAD che utilizzano un URL prefirmato, quindi è richiesto il byte 0. La dimensione del file è contenuta nel campo di intestazione della risposta. `Content-Range` È prevista una `206 Partial Content` risposta dal server. Qualsiasi altro codice di stato della risposta è un errore.

Il codice sorgente di `prvGetS3ObjectFileSize()` è disponibile sul GitHubsito Web.

Scarica la demo di CoreHTTP basic S3

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Introduzione

Questa demo mostra come utilizzare le richieste di intervallo per scaricare file dal server HTTP Amazon S3. Le richieste di intervallo sono supportate nativamente nell'API CoreHTTP quando vengono utilizzate `HTTPClient_AddRangeHeader` per creare la richiesta HTTP. Per un ambiente con microcontrollore, le richieste di intervallo sono altamente consigliate. Scaricando un file di grandi dimensioni in intervalli separati, anziché in una singola richiesta, ogni sezione del file può essere elaborata senza bloccare il socket di rete. Le richieste di intervallo riducono il rischio di perdita di pacchetti, il che richiede ritrasmissioni sulla connessione TCP, e quindi migliorano il consumo energetico del dispositivo.

Questo esempio utilizza un'interfaccia di trasporto di rete che utilizza mbedTLS per stabilire una connessione reciprocamente autenticata tra un client di dispositivo IoT che esegue CoreHTTP e il server HTTP Amazon S3.

> **Note**
>
> Per configurare ed eseguire le demo di FreerTOS, segui i passaggi indicati. Guida introduttiva a FreerTOS

Filettatura singola contro filettatura multipla

Esistono due modelli di utilizzo CoreHTTP, a thread singolo e multithread (multitasking). Sebbene la demo in questa sezione esegua la libreria HTTP in un thread, in realtà dimostra come utilizzare CoreHTTP in un ambiente a thread singolo (solo un'attività utilizza l'API HTTP nella demo). Sebbene le applicazioni a thread singolo debbano chiamare ripetutamente la libreria HTTP, le applicazioni multithread possono invece inviare richieste HTTP in background all'interno di un'attività agente (o daemon).

Organizzazione del codice sorgente

Il progetto dimostrativo è denominato `http_demo_s3_download.c` e può essere trovato nella *`freertos`*`/demos/coreHTTP/` directory e sul GitHubsito Web.

Configurazione della connessione al server HTTP Amazon S3

Questa demo utilizza un URL prefirmato per connettersi al server HTTP Amazon S3 e autorizzare l'accesso all'oggetto da scaricare. La connessione TLS del server HTTP Amazon S3 utilizza solo l'autenticazione del server. A livello di applicazione, l'accesso all'oggetto viene autenticato con i parametri della query URL prefirmata. Segui i passaggi seguenti per configurare la connessione a. AWS

1. Configura un AWS account:
   a. Se non l'hai già fatto, crea e attiva un AWS account.
   b. Gli account e le autorizzazioni vengono impostati utilizzando AWS Identity and Access Management (IAM). IAM ti consente di gestire le autorizzazioni per ogni utente del tuo account. Per impostazione predefinita, un utente non dispone delle autorizzazioni fino a quando non le concede il proprietario principale.
      i. Per aggiungere un utente IAM al tuo AWS account, consulta la IAM User Guide.
      ii. Concedi l'autorizzazione al tuo AWS account per accedere a FreerTOS aggiungendo queste AWS IoT politiche:
         - Amazon S3 FullAccess
2. Crea un bucket in S3 seguendo la procedura descritta in Come si crea un bucket S3? nella Guida per l'utente della console di Amazon Simple Storage Service.
3. Carica un file su S3 seguendo la procedura descritta in Come faccio a caricare file e cartelle in un bucket S3? .
4. Genera un URL prefirmato utilizzando lo script che si trova in`FreeRTOS-Plus/Demo/coreHTTP_Windows_Simulator/Common/presigned_url_generator/presigned_urls_gen.py`. Per istruzioni d'uso, vedere`FreeRTOS-Plus/Demo/coreHTTP_Windows_Simulator/Common/presigned_url_generato/README.md`.

Funzionalità

La demo recupera innanzitutto la dimensione del file. Quindi richiede ogni intervallo di byte in sequenza, in un ciclo, con dimensioni dell'intervallo di. `democonfigRANGE_REQUEST_LENGTH`

Il codice sorgente per la demo è disponibile sul sito Web. GitHub

Connessione al server HTTP Amazon S3

La funzione `connectToServerWithBackoffRetries()` tenta di stabilire una connessione TCP al server HTTP. Se la connessione fallisce, riprova dopo un timeout. Il valore di timeout aumenterà esponenzialmente fino al raggiungimento del numero massimo di tentativi o al raggiungimento del valore di timeout massimo. `connectToServerWithBackoffRetries()`restituisce uno stato di errore se la connessione TCP al server non può essere stabilita dopo il numero di tentativi configurato.

La funzione `prvConnectToServer()` dimostra come stabilire una connessione al server HTTP Amazon S3 utilizzando solo l'autenticazione del server. Utilizza l'interfaccia di trasporto basata su MbeDTLS implementata nel file FreerTOS- _mbedtls.c. Plus/Source/Application-Protocols/network_transport/freertos_plus_tcp/using_mbedtls/using

Il codice `prvConnectToServer()` sorgente GitHubdi è disponibile su.

Creazione di una richiesta di intervallo

La funzione API `HTTPClient_AddRangeHeader()` supporta la serializzazione di un intervallo di byte nelle intestazioni della richiesta HTTP per formare una richiesta di intervallo. Le richieste di intervallo vengono utilizzate in questa demo per recuperare la dimensione del file e per richiedere ogni sezione del file.

La funzione `prvGetS3ObjectFileSize()` recupera la dimensione del file nel bucket S3. L'`Connection: keep-alive`intestazione viene aggiunta in questa prima richiesta ad Amazon S3 per mantenere aperta la connessione dopo l'invio della risposta. Il server HTTP S3 attualmente non supporta le richieste HEAD che utilizzano un URL prefirmato, quindi è richiesto il byte 0. La dimensione del file è contenuta nel campo di intestazione della risposta. `Content-Range` È prevista una `206 Partial Content` risposta dal server; qualsiasi altro codice di stato della risposta ricevuto è un errore.

Il codice sorgente di `prvGetS3ObjectFileSize()` è disponibile su. GitHub

Dopo aver recuperato la dimensione del file, questa demo crea una nuova richiesta di intervallo per ogni intervallo di byte del file da scaricare. Viene utilizzato `HTTPClient_AddRangeHeader()` per ogni sezione del file.

Invio di richieste di intervallo e ricezione di risposte

La funzione `prvDownloadS3ObjectFile()` invia le richieste di intervallo in un ciclo fino al download dell'intero file. La funzione API `HTTPClient_Send()` invia una richiesta e riceve la risposta in modo sincrono. Quando la funzione ritorna, la risposta viene ricevuta in un`xResponse`. Viene quindi verificato che il codice di stato sia valido `206 Partial Content` e il numero di byte scaricati finora viene incrementato del valore dell'intestazione. `Content-Length`

Il codice sorgente di è disponibile su. `prvDownloadS3ObjectFile()` GitHub

Demo multithread di base di CoreHTTP

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Introduzione

Questa demo utilizza le code thread-safe di FreeRTOS per trattenere richieste e risposte in attesa di essere elaborate. In questa demo, ci sono tre attività di cui prendere nota.

- L'attività principale attende che le richieste compaiano nella coda delle richieste. Invierà tali richieste tramite la rete, quindi inserirà la risposta nella coda di risposta.
- Un'attività di richiesta crea oggetti di richiesta della libreria HTTP da inviare al server e li inserisce nella coda delle richieste. Ogni oggetto di richiesta specifica un intervallo di byte del file S3 che l'applicazione ha configurato per il download.
- Un'attività di risposta attende che le risposte compaiano nella coda di risposta. Registra ogni risposta che riceve.

Questa demo multithread di base è configurata per utilizzare una connessione TLS con la sola autenticazione del server, richiesta dal server HTTP Amazon S3. L'autenticazione a livello di applicazione viene eseguita utilizzando i parametri Signature Version 4 nella query URL prefirmata.

Organizzazione del codice sorgente

Il progetto dimostrativo è denominato `http_demo_s3_download_multithreaded.c` e può essere trovato nella *`freertos`*`/demos/coreHTTP/` directory e nel GitHubsito Web.

Creazione del progetto dimostrativo

Il progetto demo utilizza l'edizione comunitaria gratuita di Visual Studio. Per creare la demo:

1. Apri il file della soluzione di `mqtt_multitask_demo.sln` Visual Studio dall'interno dell'IDE di Visual Studio.
2. Seleziona Build Solution dal menu Build dell'IDE.

> **Note**
>
> Se utilizzi Microsoft Visual Studio 2017 o versioni precedenti, devi selezionare un Platform Toolset compatibile con la tua versione: Progetto -> RTOSDemos Proprietà -> Platform Toolset.

Configurazione del progetto demo

La demo utilizza lo stack TCP/IP FreeRTOS+TCP, quindi segui le istruzioni fornite per il progetto TCP/IP starter per:

1. Installa i componenti necessari (come Win). PCap
2. È possibile impostare un indirizzo IP statico o dinamico, un indirizzo gateway e una maschera di rete.
3. Opzionalmente, imposta un indirizzo MAC.
4. Seleziona un'interfaccia di rete Ethernet sul tuo computer host.
5. È importante verificare la connessione di rete prima di provare a eseguire la demo HTTP.

Configurazione della connessione al server HTTP Amazon S3

Segui le istruzioni riportate Configurazione della connessione al server HTTP Amazon S3 nella demo di base per il download di CoreHTTP.

Funzionalità

La demo crea tre attività in totale:

- Una che invia richieste e riceve risposte attraverso la rete.
- Uno che crea richieste da inviare.
- Uno che elabora le risposte ricevute.

In questa demo, l'attività principale:

1. Crea le code di richiesta e risposta.
2. Crea la connessione al server.
3. Crea le attività di richiesta e risposta.
4. Attende che la coda delle richieste invii le richieste sulla rete.
5. Inserisce le risposte ricevute in rete nella coda di risposta.

L'attività di richiesta:

1. Crea ciascuna delle richieste di intervallo.

L'attività di risposta:

1. Elabora ciascuna delle risposte ricevute.

Typedefs

La demo definisce le seguenti strutture per supportare il multithreading.

Richiedi articoli

Le seguenti strutture definiscono un elemento di richiesta da inserire nella coda delle richieste. L'elemento della richiesta viene copiato nella coda dopo che l'attività di richiesta ha creato una richiesta HTTP.

```
/**
 * @brief Data type for the request queue.
 *
 * Contains the request header struct and its corresponding buffer, to be
 * populated and enqueued by the request task, and read by the main task. The
 * buffer is included to avoid pointer inaccuracy during queue copy operations.
 */
typedef struct RequestItem
{
    HTTPRequestHeaders_t xRequestHeaders;
    uint8_t ucHeaderBuffer[ democonfigUSER_BUFFER_LENGTH ];
} RequestItem_t;
```

Elemento di risposta

Le seguenti strutture definiscono un elemento di risposta da inserire nella coda di risposta. L'elemento di risposta viene copiato nella coda dopo che l'attività HTTP principale ha ricevuto una risposta sulla rete.

```
/**
 * @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;
```

Attività principale di invio HTTP

L'attività principale dell'applicazione:

1. Analizza l'URL predefinito per l'indirizzo host per stabilire una connessione con il server HTTP Amazon S3.
2. Analizza l'URL predefinito per il percorso degli oggetti nel bucket S3.

3. Si connette al server HTTP Amazon S3 tramite TLS con autenticazione del server.
4. Crea le code di richiesta e risposta.
5. Crea le attività di richiesta e risposta.

La funzione `prvHTTPDemoTask()` esegue questa configurazione e fornisce lo stato della demo. Il codice sorgente di questa funzione è disponibile su Github.

Nella funzione`prvDownloadLoop()`, l'attività principale blocca e attende le richieste dalla coda delle richieste. Quando riceve una richiesta, la invia utilizzando la funzione API. `HTTPClient_Send()` Se la funzione API ha avuto successo, inserisce la risposta nella coda di risposta.

Il codice sorgente di `prvDownloadLoop()` è disponibile su Github.

Attività di richiesta HTTP

L'attività di richiesta è specificata nella funzione`prvRequestTask`. Il codice sorgente di questa funzione è disponibile su Github.

L'attività di richiesta recupera la dimensione del file nel bucket Amazon S3. Questa operazione viene eseguita nella funzione. `prvGetS3ObjectFileSize` L'intestazione «Connection: keep-alive» viene aggiunta a questa richiesta ad Amazon S3 per mantenere aperta la connessione dopo l'invio della risposta. Il server HTTP Amazon S3 attualmente non supporta le richieste HEAD che utilizzano un URL predefinito, quindi è richiesto il byte 0. La dimensione del file è contenuta nel campo di intestazione della risposta. `Content-Range` È prevista una `206 Partial Content` risposta dal server; qualsiasi altro codice di stato della risposta ricevuto è un errore.

Il codice sorgente di `prvGetS3ObjectFileSize` è disponibile su Github.

Dopo aver recuperato la dimensione del file, l'attività di richiesta continua a richiedere ogni intervallo del file. Ogni richiesta di intervallo viene inserita nella coda delle richieste per l'invio dell'attività principale. Gli intervalli di file sono configurati dall'utente demo nella macro`democonfigRANGE_REQUEST_LENGTH`. Le richieste di intervallo sono supportate nativamente nell'API della libreria client HTTP utilizzando la funzione`HTTPClient_AddRangeHeader`. La funzione `prvRequestS3ObjectRange` dimostra come usare. `HTTPClient_AddRangeHeader()`

Il codice sorgente della funzione è `prvRequestS3ObjectRange` disponibile su Github.

Attività di risposta HTTP

Le attività di risposta attendono nella coda di risposta le risposte ricevute sulla rete. L'attività principale popola la coda di risposta quando riceve correttamente una risposta HTTP. Questa attività elabora le risposte registrando il codice di stato, le intestazioni e il corpo. Un'applicazione reale può elaborare la risposta, ad esempio, scrivendo il corpo della risposta nella memoria flash. Se il codice di stato della risposta non lo è`206 partial content`, l'attività notifica all'attività principale che la demo dovrebbe fallire. L'attività di risposta è specificata in funzione. `prvResponseTask` Il codice sorgente di questa funzione è disponibile su Github.

AWS IoT Demo della libreria Jobs

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Introduzione

La demo della libreria AWS IoT Jobs mostra come connettersi al servizio AWS IoT Jobs tramite una connessione MQTT, recuperare un lavoro ed elaborarlo su un dispositivo. AWS IoT Il progetto demo di AWS IoT Jobs utilizza la porta FreerTOS per Windows, quindi può essere creato e valutato con la versione Visual Studio Community su Windows. Non è necessario alcun hardware per microcontrollori. La demo stabilisce una connessione sicura al broker AWS IoT MQTT utilizzando TLS allo stesso modo della demo di autenticazione reciproca MQTT.

> **Note**
>
> Per configurare ed eseguire le demo di FreerTOS, segui i passaggi indicati. Guida introduttiva a FreerTOS

Organizzazione del codice sorgente

Il codice demo si trova nel `jobs_demo.c` file e può essere trovato sul GitHubsito Web o nella *`freertos`*`/demos/jobs_for_aws/` directory.

Configura la connessione del broker AWS IoT MQTT

In questa demo, si utilizza una connessione MQTT al broker AWS IoT MQTT. Questa connessione è configurata nello stesso modo della demo di autenticazione reciproca MQTT.

Funzionalità

La demo mostra il flusso di lavoro utilizzato per ricevere lavori da un dispositivo AWS IoT ed elaborarli su un dispositivo. La demo è interattiva e richiede la creazione di lavori utilizzando la AWS IoT console o AWS Command Line Interface (AWS CLI). Per ulteriori informazioni sulla creazione di un lavoro, vedere create-job nel AWS CLI Command Reference. La demo richiede che nel documento di lavoro sia impostata una `action` chiave `print` per stampare un messaggio sulla console.

Vedi il seguente formato per questo documento di lavoro.

```
{
    "action": "print",
    "message": "ADD_MESSAGE_HERE"
}
```

È possibile utilizzare il AWS CLI per creare un lavoro come nel seguente comando di esempio.

```
aws iot create-job \
    --job-id t12 \
    --targets arn:aws:iot:region:123456789012:thing/device1 \
    --document '{"action":"print","message":"hello world!"}'
```

La demo utilizza anche un documento di lavoro con la `action` chiave impostata `publish` per ripubblicare il messaggio su un argomento. Vedi il seguente formato per questo documento di lavoro.

```
{
    "action": "publish",
    "message": "ADD_MESSAGE_HERE",
    "topic": "topic/name/here"
}
```

La demo continua finché non riceve un documento di lavoro con la `action` chiave impostata `exit` per uscire dalla demo. Il formato del documento di lavoro è il seguente.

```
{
    "action: "exit"
}
```

Punto di accesso alla demo di Jobs

Il codice sorgente per la funzione del punto di ingresso della demo di Jobs è disponibile su GitHub. Questa funzione esegue le seguenti operazioni:

1. Stabilire una connessione MQTT utilizzando le funzioni di supporto in. `mqtt_demo_helpers.c`
2. Abbonatevi all'argomento MQTT relativo all'`NextJobExecutionChanged`API, utilizzando le funzioni di supporto in. `mqtt_demo_helpers.c` La stringa dell'argomento è stata assemblata in precedenza, utilizzando le macro definite dalla libreria Jobs. AWS IoT
3. Pubblica nell'argomento MQTT relativo all'`StartNextPendingJobExecution`API, utilizzando le funzioni di supporto in. `mqtt_demo_helpers.c` La stringa dell'argomento è stata assemblata in precedenza, utilizzando le macro definite dalla libreria Jobs. AWS IoT
4. Chiama ripetutamente `MQTT_ProcessLoop` per ricevere messaggi in arrivo che vengono consegnati `prvEventCallback` per l'elaborazione.
5. Dopo che la demo ha ricevuto l'azione di uscita, annullate l'iscrizione all'argomento MQTT e disconnettetevi utilizzando le funzioni di supporto presenti nel file. `mqtt_demo_helpers.c`

Richiamata per i messaggi MQTT ricevuti

La prvEventCallbackfunzione chiama `Jobs_MatchTopic` dalla libreria AWS IoT Jobs per classificare il messaggio MQTT in arrivo. Se il tipo di messaggio corrisponde a un nuovo lavoro, `prvNextJobHandler()` viene chiamato.

La funzione prvNextJobHandler e le funzioni che chiama analizzano il documento di lavoro dal messaggio in formato JSON ed eseguono l'azione specificata dal processo. Di particolare interesse è la funzione. `prvSendUpdateForJob`

Invia un aggiornamento per un processo in esecuzione

La function prvSendUpdateForJob() chiama `Jobs_Update()` dalla libreria Jobs per compilare la stringa dell'argomento utilizzata nell'operazione di pubblicazione MQTT che segue immediatamente.

Demo CoreMQTT

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Queste demo possono aiutarti a imparare a usare la libreria CoreMQTT.

Argomenti

- Demo di autenticazione reciproca CoreMQTT
- Demo sulla condivisione della connessione CoreMQTT

Demo di autenticazione reciproca CoreMQTT

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Introduzione

Il progetto dimostrativo di autenticazione reciproca CoreMQTT mostra come stabilire una connessione a un broker MQTT utilizzando TLS con autenticazione reciproca tra il client e il server. Questa demo utilizza un'implementazione dell'interfaccia di trasporto basata su MBEDTLS per stabilire una connessione TLS autenticata dal server e dal client e dimostra il flusso di lavoro di sottoscrizione-pubblicazione di MQTT a livello QoS 1. Si iscrive a un filtro per argomenti, quindi pubblica gli argomenti che corrispondono al filtro e attende la ricezione di tali messaggi dal server a livello QoS 1. Questo ciclo di pubblicazione sul broker e ricezione dello stesso messaggio dal broker si ripete all'infinito. I messaggi in questa demo vengono inviati con QoS 1, che garantisce almeno una consegna secondo le specifiche MQTT.

> **Note**
>
> Per configurare ed eseguire le demo di FreerTOS, segui i passaggi indicati. Guida introduttiva a FreerTOS

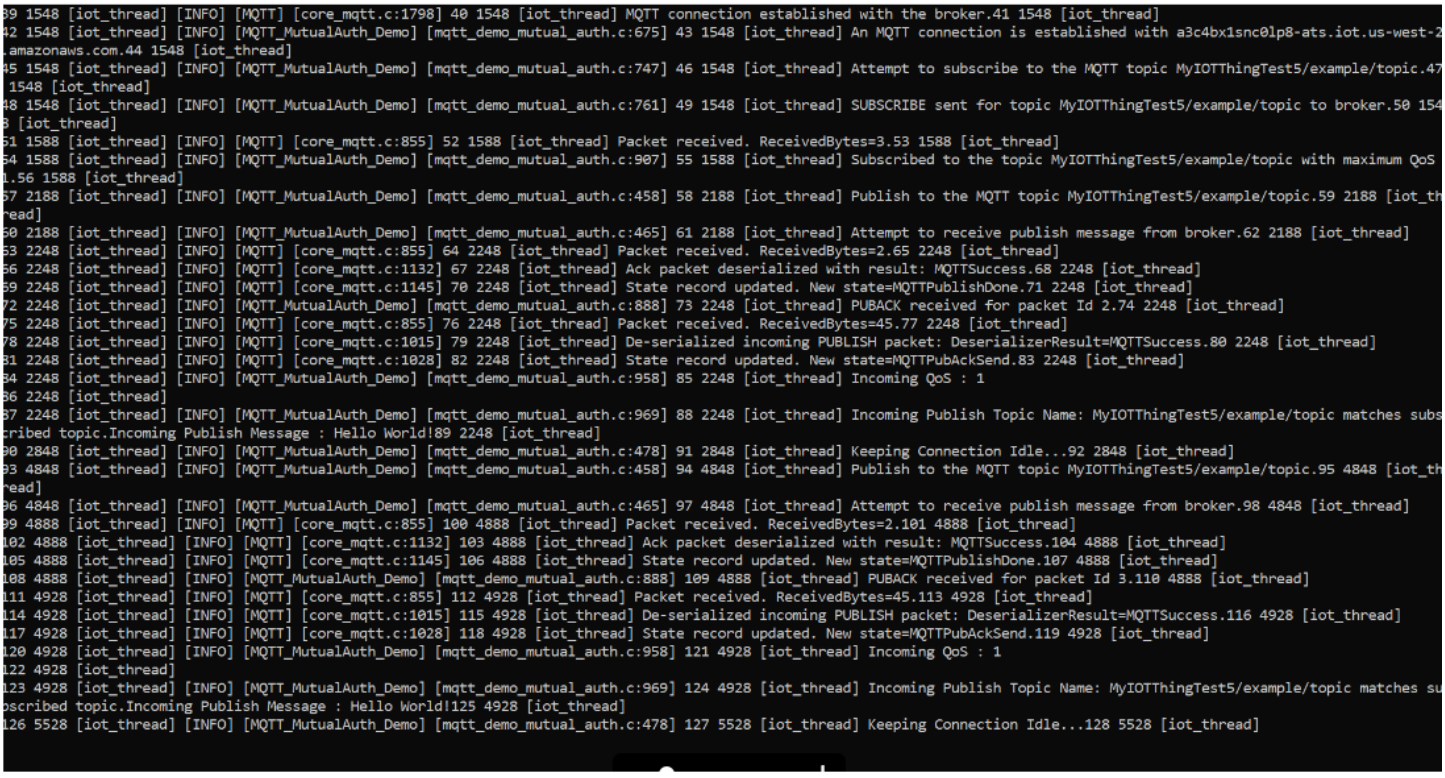
Codice sorgente

Il file sorgente della demo è denominato `mqtt_demo_mutual_auth.c` e può essere trovato nella *`freertos`*`/demos/coreMQTT/` directory e nel sito Web. GitHub

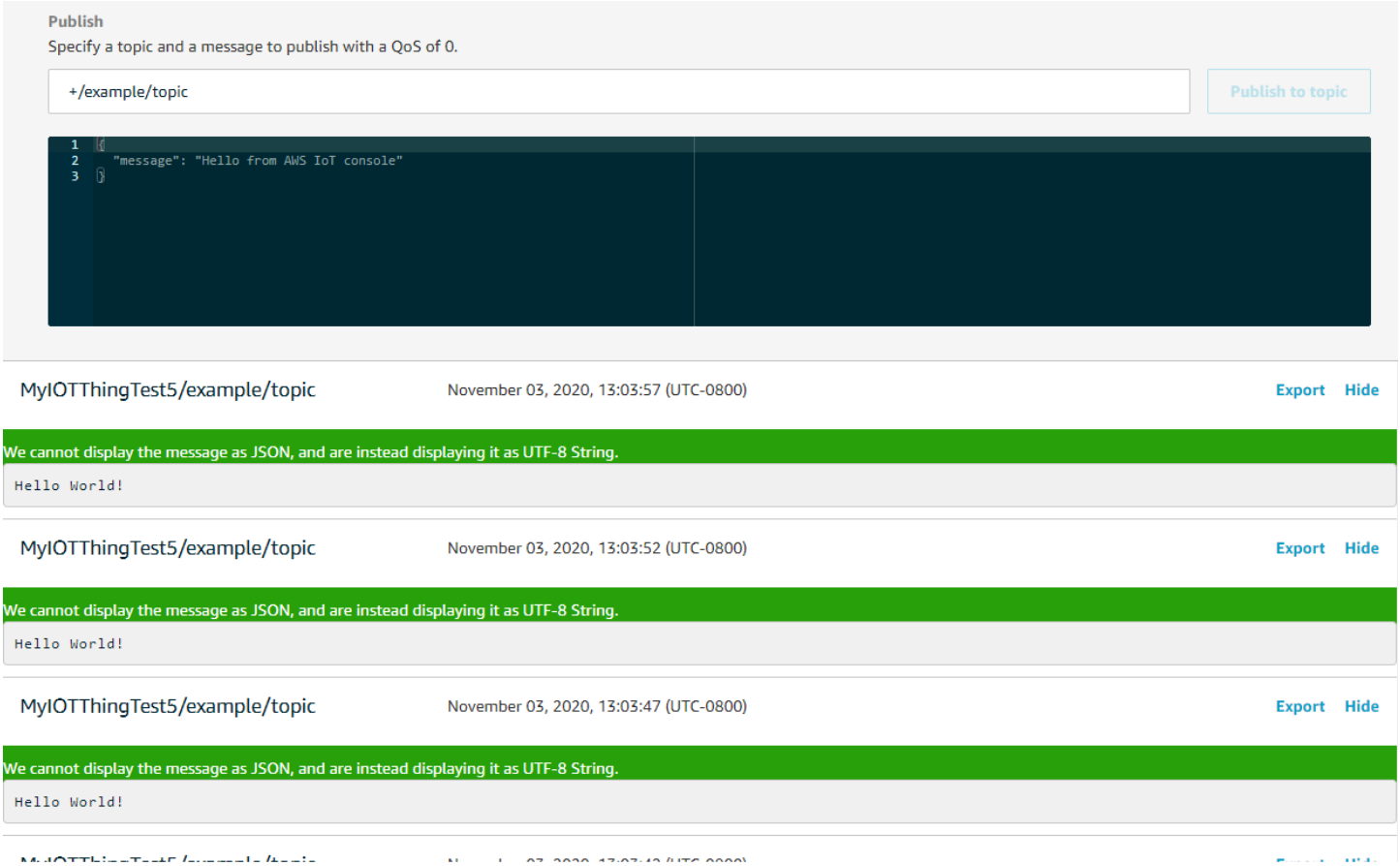
Funzionalità

La demo crea una singola attività applicativa che ripercorre una serie di esempi che dimostrano come connettersi al broker, iscriversi a un argomento sul broker, pubblicare un argomento sul broker e infine disconnettersi dal broker. L'applicazione demo sottoscrive e pubblica lo stesso argomento. Ogni volta che la demo pubblica un messaggio al broker MQTT, il broker invia lo stesso messaggio all'applicazione demo.

Un completamento con successo della demo genererà un output simile all'immagine seguente.



La AWS IoT console genererà un output simile all'immagine seguente.



Riprova la logica con backoff e jitter esponenziali

La funzione prvBackoffForRetry mostra come le operazioni di rete non riuscite con il server, ad esempio connessioni TLS o richieste di sottoscrizione MQTT, possono essere ritentate con backoff e jitter esponenziali. La funzione calcola il periodo di backoff per il tentativo successivo ed esegue il ritardo del backoff se i tentativi non sono stati esauriti. Poiché il calcolo del periodo di backoff richiede la generazione di un numero casuale, la funzione utilizza il modulo per generare il numero casuale. PKCS11 L'uso del PKCS11 modulo consente l'accesso a un True Random Number Generator (TRNG) se la piattaforma del fornitore lo supporta. Si consiglia di assegnare al generatore di numeri casuali una fonte di entropia specifica del dispositivo in modo da ridurre la probabilità di collisioni tra i dispositivi durante i tentativi di connessione.

Connessione al broker MQTT

La prvConnectToServerWithBackoffRetriesfunzione tenta di stabilire una connessione TLS con autenticazione reciproca al broker MQTT. Se la connessione fallisce, riprova dopo un periodo di backoff. Il periodo di backoff aumenterà esponenzialmente fino al raggiungimento del numero massimo di tentativi o al raggiungimento del periodo massimo di backoff. La

`BackoffAlgorithm_GetNextBackoff` funzione fornisce un valore di backoff che aumenta esponenzialmente e ritorna `RetryUtilsRetriesExhausted` quando è stato raggiunto il numero massimo di tentativi. La `prvConnectToServerWithBackoffRetries` funzione restituisce uno stato di errore se la connessione TLS al broker non può essere stabilita dopo il numero di tentativi configurato.

La MQTTConnection WithBroker funzione PrvCreate dimostra come stabilire una connessione MQTT a un broker MQTT con una sessione pulita. Utilizza l'interfaccia di trasporto TLS, implementata nel file. `FreeRTOS-Plus/Source/Application-Protocols/platform/freertos/transport/src/tls_freertos.c` Tieni presente che stiamo impostando i secondi di mantenimento in vita per il broker. `xConnectInfo`

La funzione successiva mostra come l'interfaccia di trasporto TLS e la funzione time sono impostate in un contesto MQTT utilizzando la funzione. `MQTT_Init` Mostra anche come viene impostata una funzione di callback di eventi pointer ()`prvEventCallback`. Questo callback viene utilizzato per segnalare i messaggi in arrivo.

Iscrizione a un argomento MQTT

La MQTTSubscribe WithBackoffRetries funzione prv dimostra come sottoscrivere un filtro per argomenti sul broker MQTT. L'esempio dimostra come sottoscrivere un filtro di argomento, ma è possibile passare un elenco di filtri di argomento nella stessa chiamata API di sottoscrizione per sottoscrivere più di un filtro di argomento. Inoltre, nel caso in cui il broker MQTT rifiuti la richiesta di abbonamento, l'abbonamento riproverà, con un backoff esponenziale, per. `RETRY_MAX_ATTEMPTS`

Pubblicazione in un argomento

La MQTTPublish ToTopic funzione prv dimostra come pubblicare su un argomento sul broker MQTT.

Ricezione di messaggi in arrivo

L'applicazione registra una funzione di callback degli eventi prima di connettersi al broker, come descritto in precedenza. La `prvMQTTDemoTask` funzione chiama la `MQTT_ProcessLoop` funzione per ricevere i messaggi in arrivo. Quando viene ricevuto un messaggio MQTT in arrivo, richiama la funzione di callback degli eventi registrata dall'applicazione. La prvEventCallbackfunzione è un esempio di tale funzione di callback degli eventi. `prvEventCallback`esamina il tipo di pacchetto in entrata e chiama il gestore appropriato. Nell'esempio seguente, la funzione richiede la gestione dei messaggi `prvMQTTProcessIncomingPublish()` di pubblicazione in entrata o la gestione dei riconoscimenti (`prvMQTTProcessResponse()`ACK).

Elaborazione dei pacchetti di pubblicazione MQTT in entrata

La MQTTProcess IncomingPublish funzione prv dimostra come elaborare un pacchetto di pubblicazione dal broker MQTT.

Annullamento dell'iscrizione a un argomento

L'ultimo passaggio del flusso di lavoro consiste nell'annullare l'iscrizione all'argomento in modo che il broker non invii alcun messaggio pubblicato da. `mqttexampleTOPIC` Ecco la definizione della funzione MQTTUnsubscribeFromTopicprv.

Modifica della CA principale utilizzata nella demo

Per impostazione predefinita, le demo FreerTOS utilizzano il certificato Amazon Root CA 1 (chiave RSA 2048 bit) per l'autenticazione con il server. AWS IoT Core È possibile utilizzare altri certificati CA per l'autenticazione del server, incluso il certificato Amazon Root CA 3 (chiave ECC a 256 bit). Per modificare la CA principale per la demo di autenticazione reciproca CoreMQTT:

1. In un editor di testo, aprire il file *`freertos`*`/vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/mqtt_demo_mutual_auth_config.h`.
2. Nel file, individuate la riga seguente.

   ```
   * #define democonfigROOT_CA_PEM    "...insert here..."
   ```

   Decommenta questa riga e, se necessario, spostala oltre la fine `*/` del blocco dei commenti.
3. Copia il certificato CA che desideri utilizzare e incollalo nel `"...insert here..."` testo. Il risultato sarà simile al seguente esempio:

   ```
   #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. (Facoltativo) È possibile modificare la CA principale per altre demo. Ripetere i passaggi da 1 a 3 per ogni *freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/*demo-name*_config.h file.

Demo sulla condivisione della connessione CoreMQTT

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Introduzione

Il progetto dimostrativo di condivisione della connessione CoreMQTT mostra come utilizzare un'applicazione multithread per stabilire una connessione al broker AWS MQTT utilizzando TLS con autenticazione reciproca tra il client e il server. Questa demo utilizza un'implementazione dell'interfaccia di trasporto basata su MBEDTLS per stabilire una connessione TLS autenticata dal server e dal client e dimostra il flusso di lavoro di sottoscrizione-pubblicazione di MQTT a livello QoS 1. La demo sottoscrive un filtro per argomenti, pubblica gli argomenti che corrispondono al filtro e quindi attende di ricevere quei messaggi dal server a livello QoS 1. Questo ciclo di pubblicazione sul broker e ricezione dello stesso messaggio dal broker si ripete all'infinito. I messaggi in questa demo vengono inviati con QoS 1, che garantisce almeno una consegna secondo le specifiche MQTT.

> **Note**
>
> Per configurare ed eseguire le demo di FreerTOS, segui i passaggi indicati. Guida introduttiva a FreerTOS

Questa demo utilizza una coda thread-safe per contenere i comandi per interagire con l'API MQTT. Ci sono quattro attività da prendere in considerazione in questa demo.

- Un'attività di comando (principale) prende i comandi dalla coda dei comandi e li elabora. Le altre attività inseriscono i comandi in questa coda per l'elaborazione. Questa operazione entra in un ciclo durante il quale elabora i comandi. Se viene ricevuto un comando di terminazione, questa operazione verrà interrotta dal ciclo.

- Un'attività di pubblicazione sincrona crea una serie di operazioni di pubblicazione e le inserisce nella coda dei comandi. Queste operazioni vengono quindi eseguite dal comando task. Questa attività utilizza la pubblicazione sincrona, il che significa che aspetterà il completamento di ogni operazione di pubblicazione prima di pianificare quella successiva.
- Un'attività di pubblicazione asincrona crea una serie di operazioni di pubblicazione e le inserisce nella coda dei comandi. Queste operazioni vengono quindi eseguite dal comando task. La differenza tra questa attività e quella precedente è che non aspetterà il completamento di un'operazione di pubblicazione prima di pianificare quella successiva. Controlla lo stato di ogni operazione di pubblicazione dopo che tutte le relative operazioni di pubblicazione sono state aggiunte alla coda. Nota che la distinzione tra pubblicazione sincrona e asincrona riguarda solo il comportamento di queste attività. Non ci sono differenze negli effettivi comandi di pubblicazione.
- Un'attività di sottoscrizione crea una sottoscrizione MQTT a un filtro di argomenti che corrisponde agli argomenti di tutti i messaggi pubblicati dalle due attività di pubblicazione. Questa attività entra in un ciclo continuo e attende di ricevere i messaggi pubblicati dalle altre attività.

Le attività possono avere code per contenere i messaggi ricevuti. L'operazione di comando inserirà i messaggi in arrivo nella coda di qualsiasi attività sottoscritta all'argomento in arrivo.

Questa demo utilizza una connessione TLS con autenticazione reciproca a cui connettersi. AWS Se la rete si disconnette inaspettatamente durante la demo, il client tenta di riconnettersi utilizzando la logica di backoff esponenziale. Se il client si riconnette con successo, ma il broker non riesce a riprendere la sessione precedente, il client riscriverà gli stessi argomenti della sessione precedente.

Thread singolo vs multithread

Esistono due modelli di utilizzo CoreMQTT, a thread singolo e a thread multiplo (multitasking). Questa demo mostra come creare uno schema di multithreading personalizzato. C'è anche un altro esempio multithread che esegue il protocollo MQTT in background all'interno di un'attività agente (o demone). Per ulteriori informazioni, vedere MWTT Agent and Demos using CoreMQTT. Se si esegue il protocollo MQTT in un'attività di agente, non è necessario gestire in modo esplicito alcuno stato MQTT o chiamare la funzione. `MQTT_ProcessLoop` Inoltre, se si utilizza un'attività agente, più attività dell'applicazione possono condividere una singola connessione MQTT senza la necessità di primitive di sincronizzazione, come i mutex.

Codice sorgente

Il file sorgente della demo è denominato `mqtt_demo_connection_sharing.c` e può essere trovato nella directory e nel sito Web. *`freertos`*`/demos/coreMQTT/` GitHub

Funzionalità

Questa demo crea quattro attività in totale: tre che richiedono chiamate API MQTT e una che elabora tali richieste, che è l'attività principale. In questa demo, l'attività principale entra in un ciclo che crea le tre sottoattività, richiama il ciclo di elaborazione e successivamente si pulisce. L'attività principale crea una singola connessione MQTT al broker condivisa tra le sottoattività. Due delle sottoattività pubblicano messaggi sul broker e la terza riceve i messaggi, utilizzando un abbonamento MQTT a un filtro per argomenti che corrisponde a tutti gli argomenti dei messaggi pubblicati.

Typedef

La demo definisce le seguenti strutture, enumerazioni e puntatori di funzioni.

Comandi

Anziché effettuare direttamente le chiamate all'API MQTT, le attività utilizzano `Command_t` strutture per creare comandi che indicano all'attività principale di richiamare l'operazione API appropriata. I comandi hanno i seguenti tipi:

- `PROCESSLOOP`
- `PUBLISH`
- `SUBSCRIBE`
- `UNSUBSCRIBE`
- `PING`
- `DISCONNECT`
- `RECONNECT`
- `TERMINATE`

Il `TERMINATE` comando non ha un'operazione API MQTT corrispondente. Viene utilizzato nella demo per indicare all'attività principale di interrompere l'elaborazione dei comandi e iniziare le operazioni di pulizia. Perché alcune informazioni aggiuntive, ad esempio informazioni di pubblicazione o sottoscrizione, sono necessarie per alcuni comandi MQTT come`MQTT_Publish`, e `MQTT_Subscribe`MQTT_Unsubscribe, utilizziamo il campo. `CommandContext_t` Questo campo è obbligatorio per questi tre comandi ed è facoltativo per gli altri.

Poiché questo contesto è necessario per questi comandi, non modificate questo valore una volta che il comando è stato messo in coda, fino al completamento del comando. Al termine di un comando, è possibile richiamare un callback opzionale. In questa demo, utilizziamo un callback che crea una notifica di attività per informare l'attività chiamante che il comando è stato

completato. Per le operazioni MQTT che richiedono riconoscimenti (sottoscrizione, annullamento dell'iscrizione e pubblicazione con QoS superiore a 0), il comando viene considerato completato una volta ricevuta la conferma. In caso contrario, il comando viene completato una volta restituita la chiamata API MQTT corrispondente.

Nel `mqtt_demo_connection_sharing.c` file sono disponibili le seguenti definizioni:

- Struttura Command_t
- CommandContextstruttura _t
- CommandTypeenumerazione _t
- CommandCallbackpuntatore alla funzione _t

Riconoscimenti

Poiché alcune operazioni MQTT richiedono un riconoscimento, utilizzano un array di AckInfo_t che contiene l'identificatore di pacchetto del riconoscimento previsto e il comando originale che lo prevede, in modo da poterne richiamare il callback di completamento.

Sottoscrizioni

Questa demo può tenere traccia delle sottoscrizioni per ogni attività. A tal fine, ogni attività che richiede un abbonamento deve fornire una coda di messaggi in cui riceverà i messaggi pubblicati (SubscriptionElement_t). Più attività possono sottoscrivere lo stesso filtro per argomenti, poiché è previsto che utilizzino code di risposta separate.

Messaggi pubblicati ricevuti

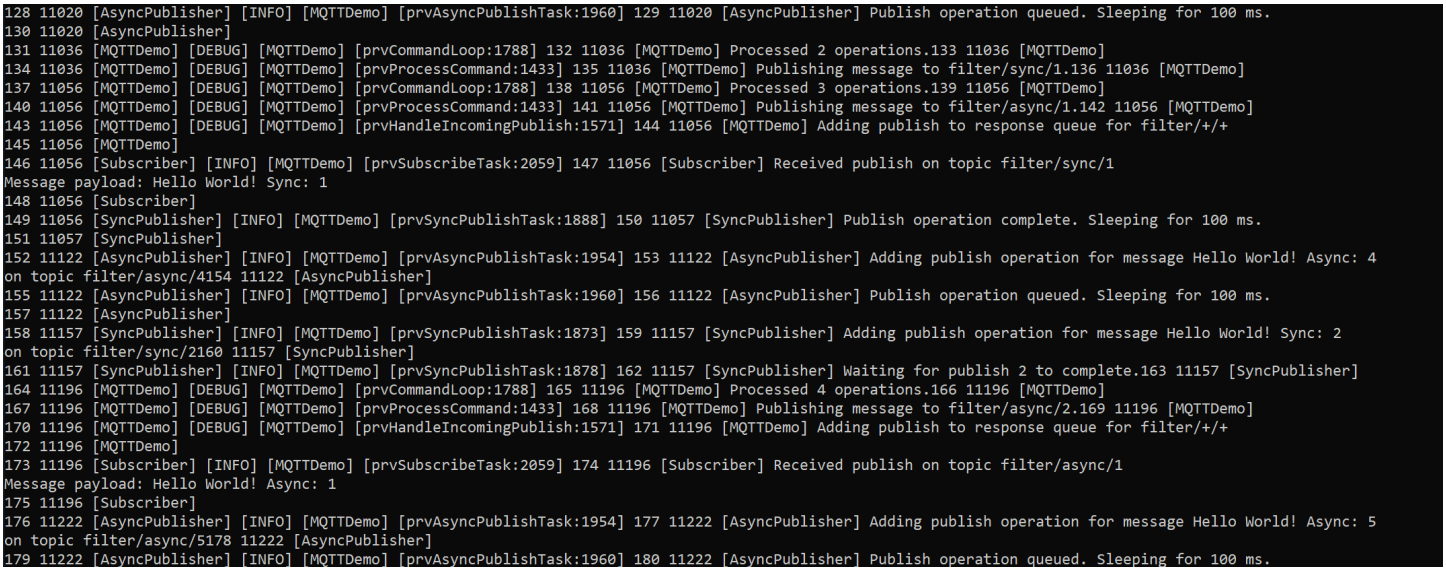
Poiché le attività vengono eseguite in parallelo all'attività principale, sarebbe difficile e dispendioso in termini di tempo per l'attività principale dover attendere che ogni attività sottoscritta legga un messaggio pubblicato ricevuto. Pertanto, ogni messaggio ricevuto viene copiato nella coda di risposta di qualsiasi attività sottoscritta all'argomento del messaggio pubblicato (_t). PublishElement Poiché i pacchetti di pubblicazione ricevuti dal client MQTT contengono puntatori al buffer di rete del client, il payload e il nome dell'argomento del messaggio in arrivo vengono copiati in buffer separati prima di essere inseriti in una coda di risposta. In questo modo, l'operazione sottoscritta può ancora leggere le informazioni ricevute dopo che il client MQTT ha cancellato il buffer di rete.

Attività principale

L'attività principale dell'applicazione stabilisce una sessione MQTT persistente, crea tre sottoattività ed esegue il ciclo di elaborazione fino alla ricezione di un comando di terminazione.

RunCoreMQTTConnectionSharingDemo Viene utilizzata una sessione persistente, quindi se la rete si disconnette inaspettatamente, la demo si riconnetterà al broker in background, senza perdere gli abbonamenti o i messaggi pubblicati in arrivo dal broker. Per creare una nuova sessione persistente per ogni esecuzione, la demo si connette al broker con il flag impostato, quindi si disconnette e si riconnette con il `clean session` flag unset. Una volta terminato, il ciclo di elaborazione si disconnette dal broker e riparte dal punto in cui aveva effettuato la riconnessione di rete.

Un completamento con successo della demo genererà un output simile all'immagine seguente.



Ciclo di comando

Il ciclo di comandi prvCommandLoop, attende che i comandi vengano inseriti nella coda dei comandi, quindi chiama l'API MQTT appropriata. Anche tutti i comandi tranne `DISCONNECT` e `TERMINATE` risultano `MQTT_ProcessLoop` richiamati. Questa demo imposta un callback di riattivazione del socket per aggiungere un `PROCESSLOOP` comando alla coda quando i dati sono disponibili sul socket. Tuttavia, a quel punto potrebbero esserci molti comandi in coda. Per assicurarci di non trascurare i dati in arrivo mentre vengono elaborati altri comandi, `MQTT_ProcessLoop` viene richiesta una singola iterazione dopo ogni comando.

Comandi di elaborazione

Vedi la prvProcessCommandfunzione.

Attività sincrona di pubblicazione

L'attività sincrona di pubblicazione, prvSyncPublishTask, crea `PUBLISH` le operazioni in modo sincrono e attende il completamento di ciascuna operazione prima di pianificare quella successiva.

Questa demo utilizza QoS 1 per pubblicare messaggi, il che significa che queste operazioni non vengono considerate completate fino alla ricezione del pacchetto di conferma della pubblicazione.

Attività di pubblicazione asincrona

L'attività di pubblicazione asincrona, prvAsyncPublishTask, non attende il completamento di una pubblicazione prima di inserire quella successiva nella coda. Ciò dimostra che non è sempre necessario che un'attività attenda il completamento di un'operazione MQTT prima di poter essere ripresa. Poiché ogni comando di pubblicazione richiede una propria struttura di contesto, questa attività non può riutilizzare una singola struttura di contesto come fa l'attività sincrona di Publisher, poiché un comando precedente potrebbe ancora averne bisogno. Pertanto, alloca memoria per ogni struttura di contesto e quindi attende di liberare tutta la memoria allocata dopo che tutti i messaggi da pubblicare sono stati inseriti nella coda.

Attività per abbonati

L'attività di sottoscrizione prvSubscribeTask, sottoscrive un filtro per argomenti che corrisponde a tutti gli argomenti dei messaggi pubblicati dalle attività sincrone e asincrone. Attende quindi di ricevere nuovamente tutti i messaggi pubblicati prima di annullare l'iscrizione. Questa attività è anche responsabile della creazione dell'`TERMINATE`operazione che segnala all'attività principale di terminare il ciclo di comandi.

Over-the-air aggiorna l'applicazione demo

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

FreerTOS include un'applicazione demo che dimostra la funzionalità della over-the-air libreria (OTA). L'applicazione demo OTA si trova nel file *`freertos`*`/demos/ota/aws_iot_ota_update_demo.c`.

L'applicazione demo OTA esegue le seguenti operazioni:

1. Inizializza lo stack della rete FreeRTOS e il pool di buffer MQTT.
2. Crea un'attività con cui esercitare la libreria OTA. `vRunOTAUpdateDemo()`

3. Crea un client MQTT utilizzando `_establishMqttConnection()`.
4. Si connette al broker AWS IoT MQTT utilizzando `IotMqtt_Connect()` e registra un callback di disconnessione MQTT:. `prvNetworkDisconnectCallback`
5. Chiama `OTA_AgentInit()` per creare l'attività OTA e registra un callback da utilizzare quando l'attività OTA è completa.
6. Riutilizza la connessione MQTT con `xOTAConnectionCtx.pvControlClient = _mqttConnection;`
7. Se MQTT si disconnette, l'applicazione sospende l'agente OTA, tenta di riconnettersi utilizzando un ritardo esponenziale con jitter, quindi riprende l'agente OTA.

Prima di poter utilizzare gli aggiornamenti OTA, soddisfa tutti i prerequisiti descritti in Aggiornamenti FreerTOS Over-the-Air

Dopo aver completato la configurazione per gli aggiornamenti OTA, scarica, crea, esegui il flash ed esegui la demo OTA di FreerTOS su una piattaforma che supporta la funzionalità OTA. Sono disponibili istruzioni dimostrative specifiche per i seguenti dispositivi certificati FreeRTOS:

- Texas Instruments 0SF-LAUNCHXL CC322
- Microchip Curiosity MZEF PIC32
- Espressif ESP32
- Scarica, crea, esegui il flashing ed esegui la demo OTA di FreerTOS su Renesas N RX65

Dopo aver creato, flashato ed eseguito l'applicazione demo OTA sul dispositivo, puoi utilizzare la AWS IoT console o creare un processo di aggiornamento OTA. AWS CLI Dopo aver creato un lavoro di aggiornamento OTA, collega un emulatore di terminale per visualizzare l'avanzamento dell'aggiornamento OTA. Annota tutti gli errori generati durante il processo.

Un lavoro di aggiornamento OTA con esito positivo visualizza un output come il seguente. Alcune righe in questo esempio sono state rimosse dall'elenco per ragioni di spazio.

```
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 configurazioni demo

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Le configurazioni demo OTA sono opzioni di configurazione specifiche per la demo fornite in. `aws_iot_ota_update_demo.c` Queste configurazioni sono diverse dalle configurazioni della libreria OTA fornite nel file di configurazione della libreria OTA.

OTA_DEMO_KEEP_ALIVE_SECONDS

Per il client MQTT, questa configurazione è l'intervallo di tempo massimo che può trascorrere tra la fine della trasmissione di un pacchetto di controllo e l'inizio dell'invio del successivo. In assenza di un pacchetto di controllo, viene inviato un PINGREQ. Il broker deve disconnettere un client che non invia un messaggio o un pacchetto PINGREQ entro una volta e mezza di questo intervallo di mantenimento in vita. Questa configurazione deve essere regolata in base ai requisiti dell'applicazione.

OTA_DEMO_CONN_RETRY_BASE_INTERVAL_SECONDS

L'intervallo di base, in secondi, prima di riprovare la connessione di rete. La demo OTA proverà a riconnettersi dopo questo intervallo di tempo di base. L'intervallo viene raddoppiato dopo ogni tentativo fallito. All'intervallo viene inoltre aggiunto un ritardo casuale, fino a un massimo di questo ritardo di base.

OTA_DEMO_CONN_RETRY_MAX_INTERVAL_SECONDS

Intervallo massimo, in secondi, prima di riprovare la connessione di rete. Il ritardo di riconnessione viene raddoppiato per ogni tentativo fallito, ma può arrivare solo fino a questo valore massimo, sommato a un jitter dello stesso intervallo.

Scarica, crea, esegui il flashing ed esegui la demo OTA di FreerTOS su Texas Instruments 0SF-LAUNCHXL CC322

Important

Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Per scaricare FreerTOS e il codice demo OTA

- Puoi scaricare il codice sorgente dal GitHub sito https://github.com/FreeRTOS/FreerTOS.

Per creare l'applicazione demo

1. Segui le istruzioni Guida introduttiva a FreerTOS per importare il `aws_demos` progetto in Code Composer Studio, configurare l' AWS IoT endpoint, l'SSID e la password Wi-Fi e una chiave privata e un certificato per la scheda.
2. Aprire *`freertos`*`/vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/aws_demo_config.h`, commentare `#define CONFIG_MQTT_DEMO_ENABLED` e definire `CONFIG_OTA_UPDATE_DEMO_ENABLED`.
3. Creare la soluzione e accertarsi che venga compilata senza errori.
4. Avviare un emulatore di terminale e utilizzare le seguenti impostazioni per effettuare la connessione alla scheda:
   - Velocità in baud: 115200
   - Bit di dati: 8
   - Parità: nessuna
   - Bit di stop: 1
5. Eseguire il progetto sulla scheda per accertarsi che riesca a connettersi alla rete WiFi e al broker di messaggi MQTT AWS IoT .

Durante l'esecuzione, l'emulatore di terminale dovrebbe visualizzare testo simile al seguente:

```
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
```

Scarica, crea, esegui il flashing ed esegui la demo OTA di FreerTOS su Microchip Curiosity MZEF PIC32

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Per scaricare il codice demo di FreerTOS OTA

- Puoi scaricare il codice sorgente dal GitHub sito https://github.com/FreeRTOS/FreerTOS.

Per creare l'applicazione demo dell'aggiornamento OTA

1. Segui le istruzioni Guida introduttiva a FreerTOS per importare il `aws_demos` progetto nell'IDE MPLAB X, configurare l' AWS IoT endpoint, l'SSID e la password Wi-Fi e una chiave privata e un certificato per la tua scheda.
2. Aprire `demos/include/aws_ota_codesigner_certificate.h`.
3. Incollare il contenuto del certificato di firma del codice nella variabile `static const char signingcredentialSIGNING_CERTIFICATE_PEM`. Seguendo lo stesso formato di `aws_clientcredential_keys.h`, ogni riga deve terminare con il carattere di nuova riga ('\ n') ed essere racchiusa tra virgolette.

   Ad esempio, il certificato dovrebbe essere simile a quanto segue:

```
"-----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. Installare Python 3 o versione successiva.
5. Installare pyOpenSSL eseguendo `pip install pyopenssl`.
6. Copiare il certificato di firma del codice in formato .pem nel percorso `demos/ota/bootloader/utility/codesigner_cert_utility/`. Rinominare il file del certificato `aws_ota_codesigner_certificate.pem`.
7. Aprire *`freertos`*`/vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/aws_demo_config.h`, commentare `#define CONFIG_MQTT_DEMO_ENABLED` e definire `CONFIG_OTA_UPDATE_DEMO_ENABLED`.
8. Creare la soluzione e accertarsi che venga compilata senza errori.

9. Avviare un emulatore di terminale e utilizzare le seguenti impostazioni per effettuare la connessione alla scheda:
   - Velocità in baud: 115200
   - Bit di dati: 8
   - Parità: nessuna
   - Bit di stop: 1
10. Scollega il debugger dalla scheda ed esegui il progetto sulla scheda per confermare che è in grado di connettersi al Wi-Fi e al broker di messaggi MQTT. AWS IoT

Quando si esegue il progetto, il modulo MPLAB X IDE dovrebbe aprire una finestra di output. Assicurati che la scheda sia selezionata. ICD4 Vedrai il seguente output.

```
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
```

L'emulatore di terminale dovrebbe visualizzare testo simile al seguente:

```
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
```

Questo output mostra che il Microchip Curiosity PIC32 MZEF può connettersi AWS IoT e sottoscrivere gli argomenti MQTT richiesti per gli aggiornamenti OTA. Vengono sicuramente visualizzati messaggi `Missing job parameter` perché non ci sono processi di aggiornamento OTA in sospeso.

Scarica, crea, esegui il flashing ed esegui la demo OTA di FreerTOS su Espressif ESP32

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

1. Scarica il codice sorgente di FreerTOS da. GitHub Consultare il file README.md per le istruzioni. Creare un progetto nell'IDE che includa tutti i codici sorgente e tutte le librerie richiesti.
2. Seguire le istruzioni nella pagina delle nozioni di base su Espressif per impostare la toolchain basata su GCC richiesta.
3. Aprire *`freertos`*`/vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/aws_demo_config.h`, commentare `#define CONFIG_MQTT_DEMO_ENABLED` e definire `CONFIG_OTA_UPDATE_DEMO_ENABLED`.
4. Creare il progetto dimostrativo eseguendo `make` nella directory `vendors/espressif/boards/esp32/aws_demos`. È possibile memorizzare nella flash il programma dimostrativo e verificare l'output tramite l'esecuzione di `make flash monitor`, come descritto nella pagina delle nozioni di base su Espressif.
5. Prima di eseguire la demo dell'aggiornamento OTA:

- Aprire *freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h, commentare #define CONFIG_MQTT_DEMO_ENABLED e definire CONFIG_OTA_UPDATE_DEMO_ENABLED.
- Accertarsi che il certificato di firma del codice SHA-256/ECDSA venga copiato in demos/include/aws_ota_codesigner_certificate.h.

Scarica, crea, esegui il flashing ed esegui la demo OTA di FreerTOS su Renesas N RX65

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Questo capitolo mostra come scaricare, creare, eseguire il flashing ed eseguire le applicazioni demo FreerTOS OTA su Renesas N. RX65

Argomenti

- Configura il tuo ambiente operativo
- Configura le tue risorse AWS
- Importa, configura il file di intestazione e crea aws_demos e boot_loader

Configura il tuo ambiente operativo

Le procedure descritte in questa sezione utilizzano i seguenti ambienti:

- IDE: ^{2} studio 7.8.0, e ^{2} studio 2020-07
- Catene di strumenti: CCRX Compiler v3.0.1
- Dispositivi di destinazione: RSKRX65 N-2 MB
- Debugger$^{: \text{emulatore E 2, E } 2 \text{ Lite}}$
- Software: Renesas Flash Programmatore, Renesas Secure Flash Programmer.exe, Tera Term

Per configurare l'hardware

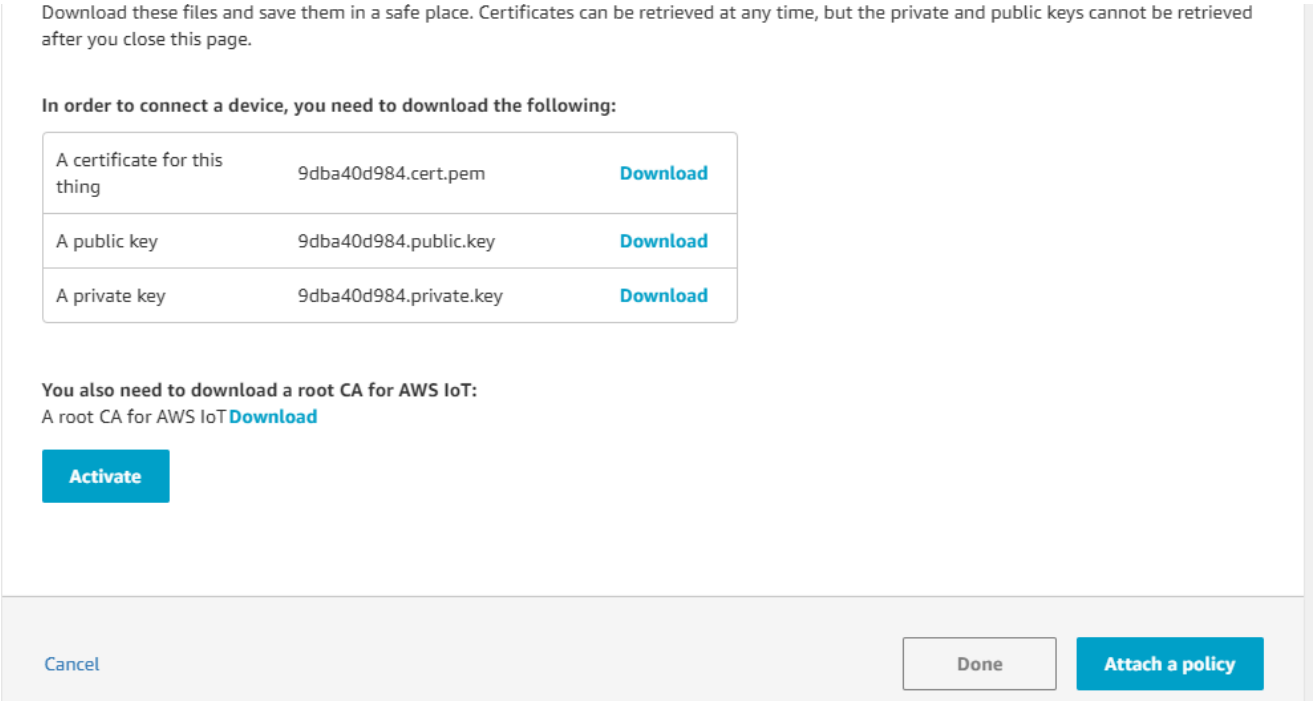
1. Connect l'emulatore E ^{2} Lite e la porta seriale USB alla scheda RX65 N e al PC.
2. Connect la fonte di alimentazione al RX65 N.

Configura le tue risorse AWS

1. Per eseguire le demo di FreerTOS, è necessario disporre di AWS un account con AWS Identity and Access Management un utente (IAM) autorizzato ad accedere ai servizi. AWS IoT Se non l'hai già fatto, segui i passaggi indicati. Configurazione dell'account e delle autorizzazioni AWS
2. Per configurare gli aggiornamenti OTA, segui i passaggi indicatiPrerequisiti per l'aggiornamento OTA. In particolare, segui i passaggi indicati inPrerequisiti per gli aggiornamenti OTA mediante MQTT.
3. Apri la AWS IoT console.
4. Nel riquadro di navigazione a sinistra, scegli Gestisci, quindi scegli Cose per creare un oggetto.

   Una cosa è una rappresentazione di un dispositivo o di un'entità logica in AWS IoT. Può trattarsi di un dispositivo fisico o un sensore, ad esempio una lampadina o un interruttore su un muro. Può anche essere un'entità logica come un'istanza di un'applicazione o un'entità fisica a cui non si connette AWS IoT, ma è correlata a dispositivi che lo fanno (ad esempio, un'auto dotata di sensori del motore o un pannello di controllo). AWS IoT fornisce un registro degli oggetti che ti aiuta a gestire le tue cose.

   a. Scegli Crea, quindi scegli Crea una singola cosa.
   b. Inserisci un nome per il tuo oggetto, quindi scegli Avanti.
   c. Scegli Crea certificato.
   d. Scarica i tre file creati, quindi scegli Attiva.
   e. Scegliere Attach a policy (Collega policy).



f. Seleziona la politica in cui hai creatoPolicy dei dispositivi.

Ogni dispositivo che riceve un aggiornamento OTA tramite MQTT deve essere registrato come oggetto in AWS IoT e deve avere una politica allegata come quella elencata. Ulteriori informazioni sugli elementi degli oggetti `"Action"` e `"Resource"` sono disponibili in Operazioni di policy AWS IoT Core e Risorse per operazioni AWS IoT Core.

Note

- Le `iot:Connect` autorizzazioni consentono al dispositivo di connettersi tramite MQTT AWS IoT .
- Le autorizzazioni `iot:Publish` e `iot:Subscribe` sugli argomenti dei processi AWS IoT (`.../jobs/*`) consentono al dispositivo connesso di ricevere notifiche e documenti relativi ai processi e di pubblicare lo stato di completamento dell'esecuzione di un processo.
- Le `iot:Publish` autorizzazioni `iot:Subscribe` e sugli argomenti degli stream AWS IoT OTA (`.../streams/*`) consentono al dispositivo connesso di recuperare i dati di aggiornamento OTA da. AWS IoT Queste autorizzazioni sono necessarie per eseguire gli aggiornamenti del firmware tramite MQTT.
- Le `iot:Receive` autorizzazioni consentono di AWS IoT Core pubblicare messaggi su tali argomenti sul dispositivo connesso. Questa autorizzazione viene controllata ad ogni

recapito di un messaggio MQTT. È possibile utilizzare questa autorizzazione per revocare l'accesso ai client attualmente sottoscritti a un argomento.

5. Per creare un profilo di firma del codice e registrare un certificato di firma del codice su. AWS

   a. Per creare le chiavi e la certificazione, vedere la sezione 7.3 «Generazione di coppie di SHA256 chiavi ECDSA con OpenSSL» nella politica di progettazione degli aggiornamenti del firmware MCU di Renesas.

   b. Apri la AWS IoT console. Nel riquadro di navigazione a sinistra, seleziona Gestisci, quindi Jobs. Seleziona Crea un lavoro, quindi Crea processo di aggiornamento OTA.

   c. In Seleziona i dispositivi da aggiornare scegli Seleziona, quindi scegli l'elemento che hai creato in precedenza. Seleziona Avanti.

   d. Nella pagina Crea un processo di aggiornamento OTA FreerTOS, procedi come segue:

      i. Per Seleziona il protocollo per il trasferimento delle immagini del firmware, scegli MQTT.

      ii. Per Seleziona e firma l'immagine del firmware, scegli Firma una nuova immagine del firmware per me.

      iii. Per il profilo Code Signing, scegli Crea.

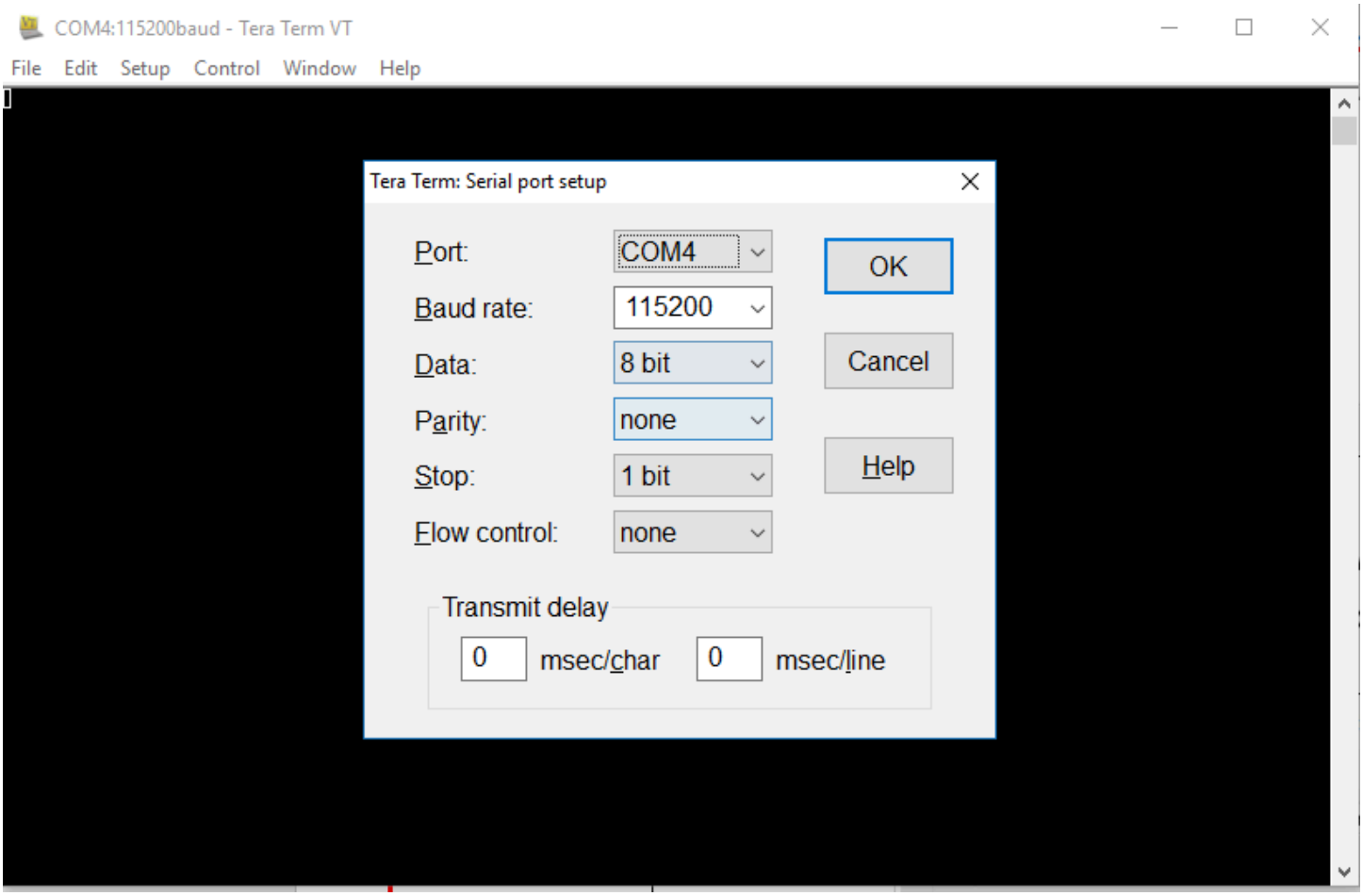
      iv. Nella finestra Crea un profilo di firma del codice, inserisci un nome di profilo. Per la piattaforma hardware del dispositivo, seleziona Windows Simulator. Per il certificato di firma del codice, scegli Importa.

      v. Sfoglia per selezionare il certificato (`secp256r1.crt`), la chiave privata del certificato (`secp256r1.key`) e la catena di certificati (`ca.crt`).

      vi. Inserisci un nome di percorso del certificato di firma del codice sul dispositivo. Quindi, scegli Crea.

6. Per concedere l'accesso alla firma del codice per AWS IoT, segui i passaggi indicati inConcedi l'accesso alla firma del codice per AWS IoT.

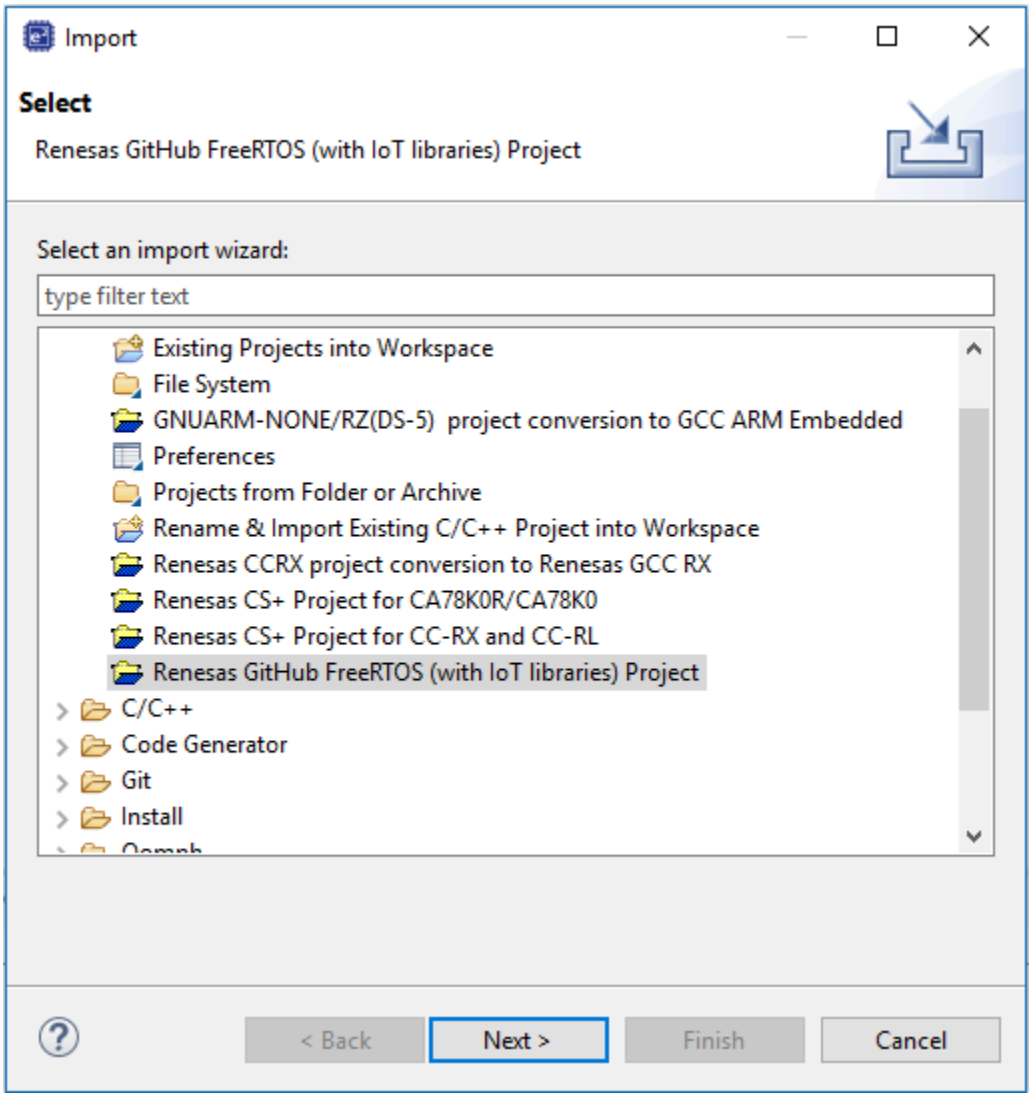
Se non hai Tera Term installato sul tuo PC, puoi scaricarlo da https://ttssh2.osdn.jp/index.html.en e configurarlo come mostrato qui. Assicurati di collegare la porta seriale USB dal dispositivo al PC.



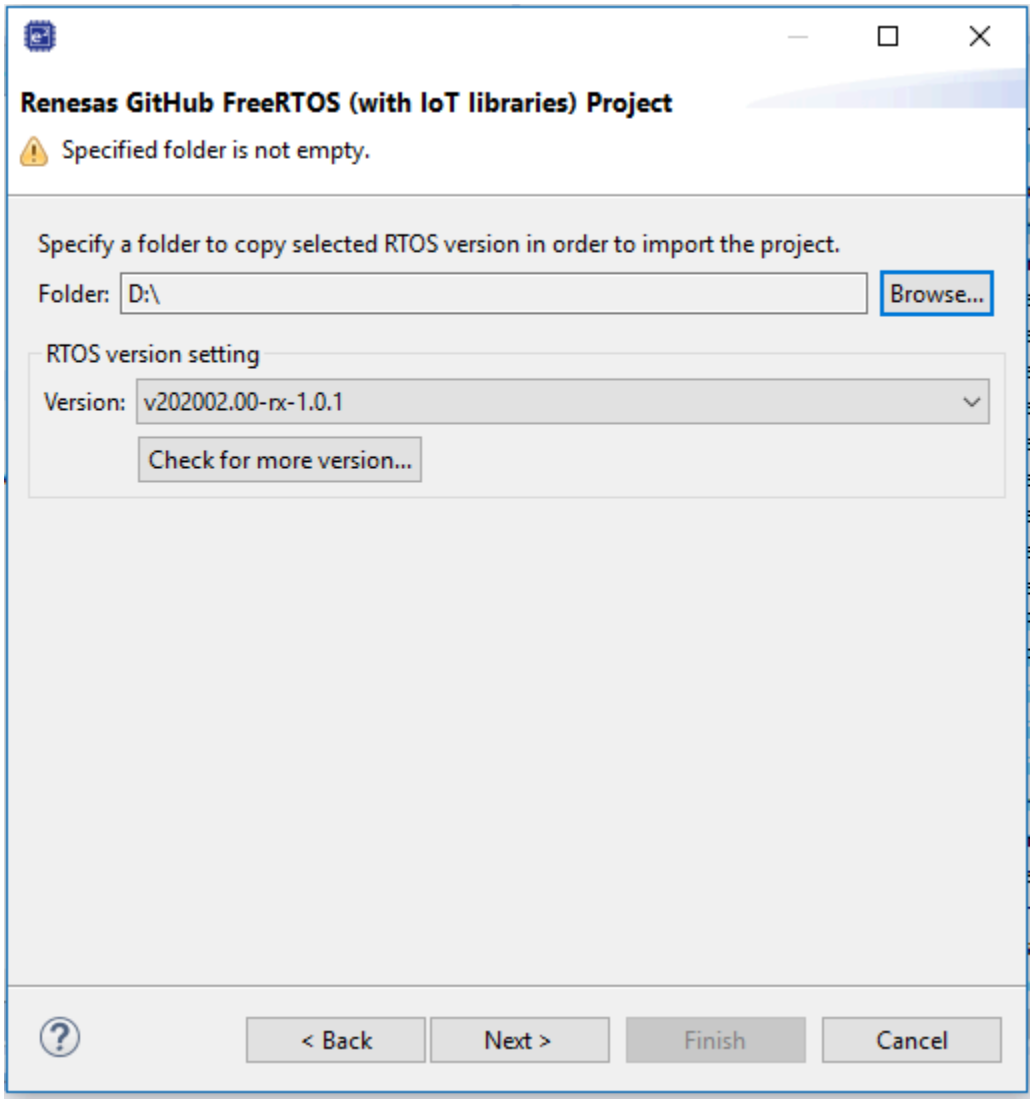
Importa, configura il file di intestazione e crea aws_demos e boot_loader

Per iniziare, selezioni l'ultima versione del pacchetto FreerTOS, che verrà scaricata GitHub e importata automaticamente nel progetto. In questo modo puoi concentrarti sulla configurazione di FreerTOS e sulla scrittura del codice dell'applicazione.

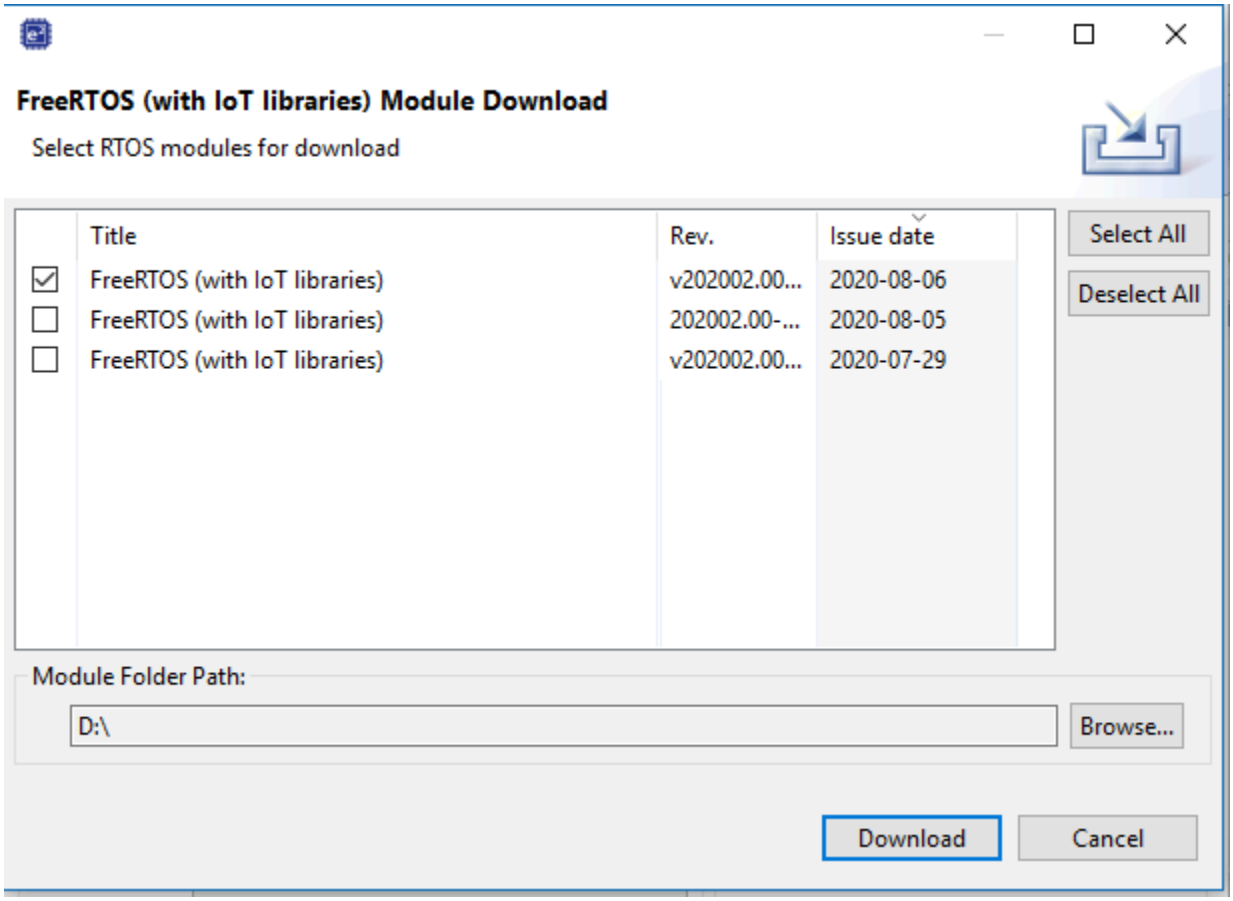
1. Avvia e 2 studio.
2. Scegli File, quindi scegli Importa...
3. Seleziona il progetto Renesas GitHub FreerTOS (con librerie IoT).



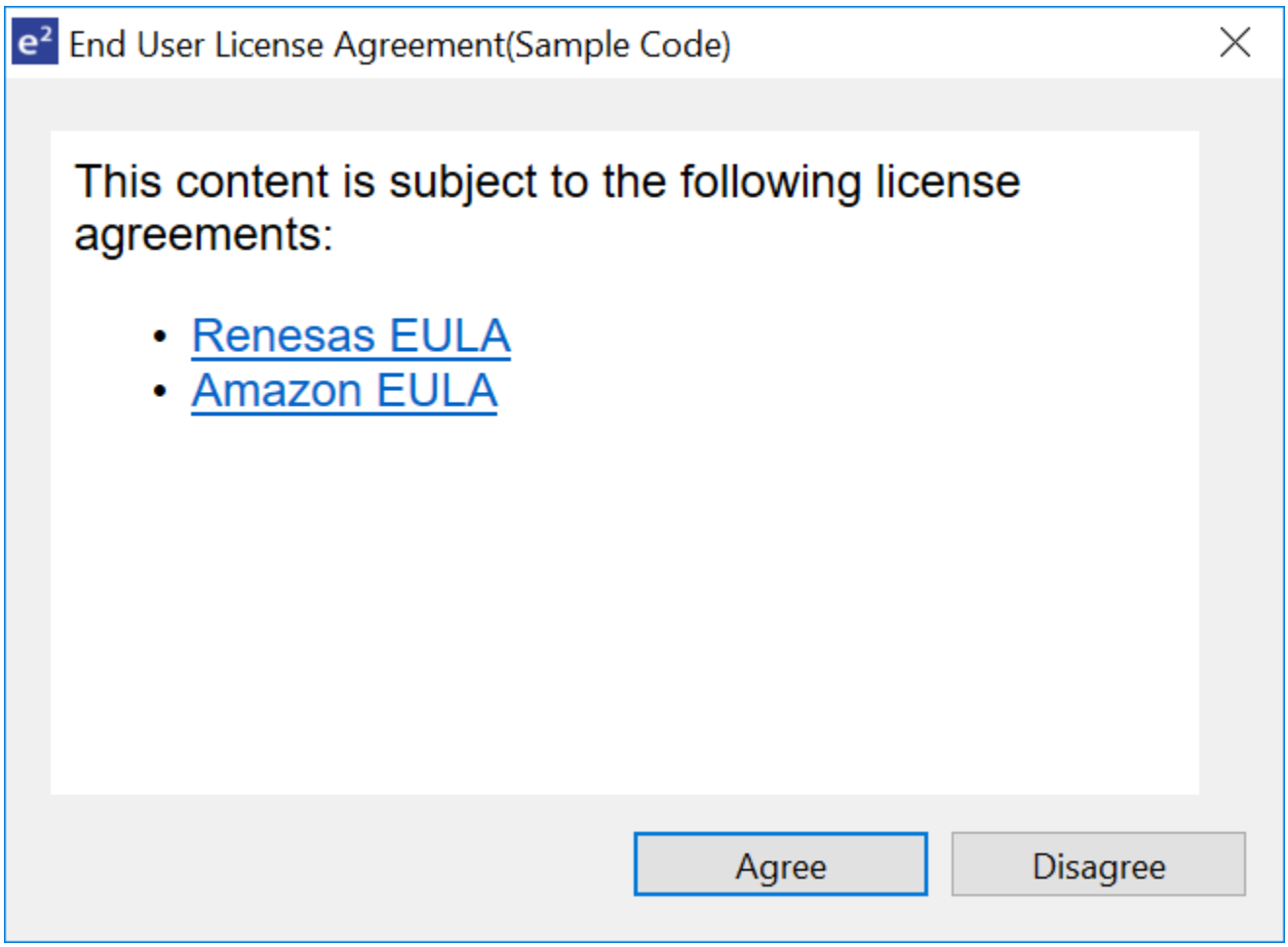
4. Scegli Controlla altre versioni... per mostrare la finestra di dialogo per il download.



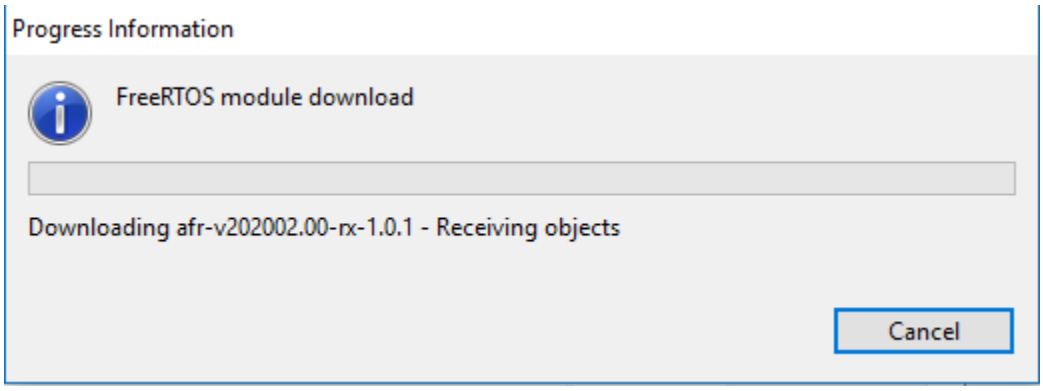
5. Seleziona il pacchetto più recente.



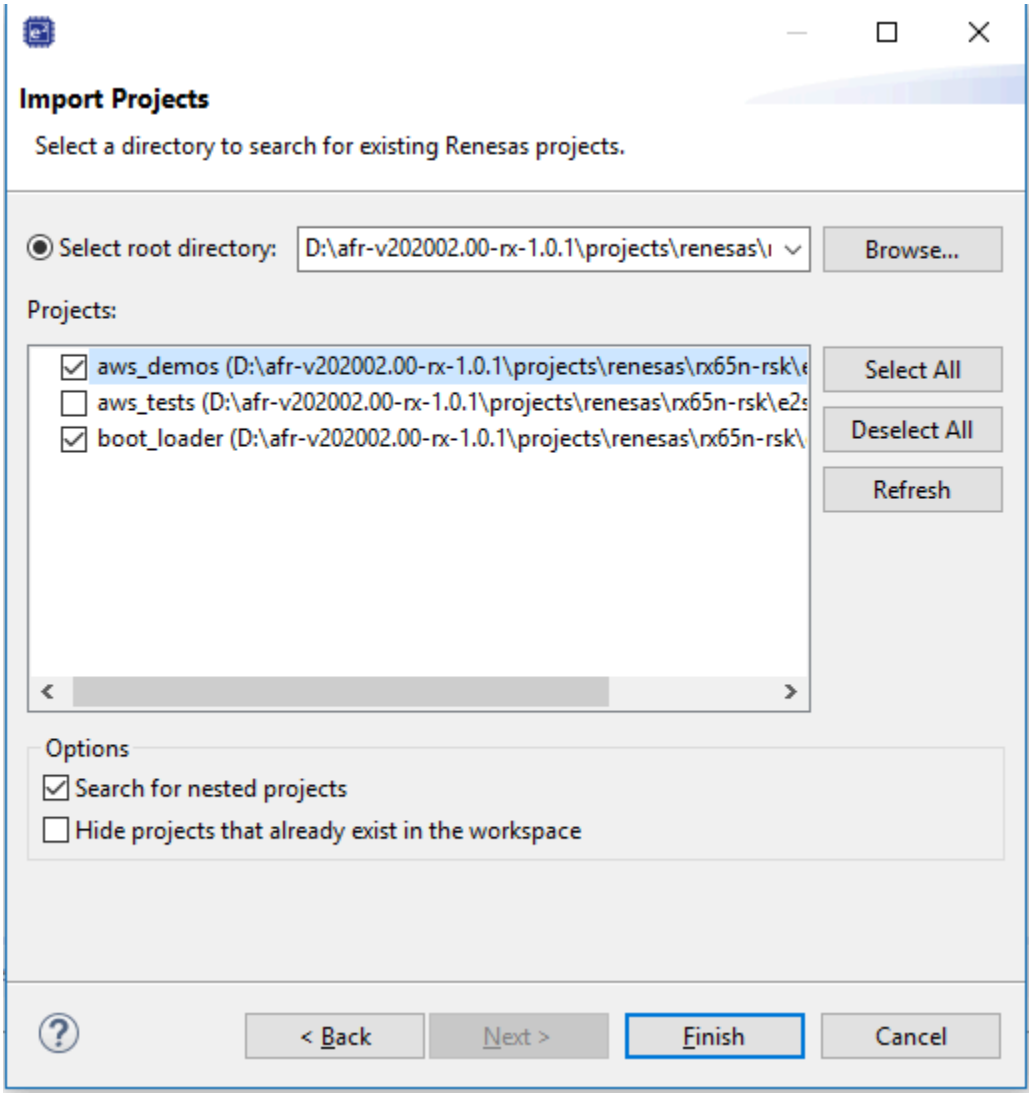
6. Scegli Accetto per accettare il contratto di licenza con l'utente finale.



7. Attendi il completamento del download.



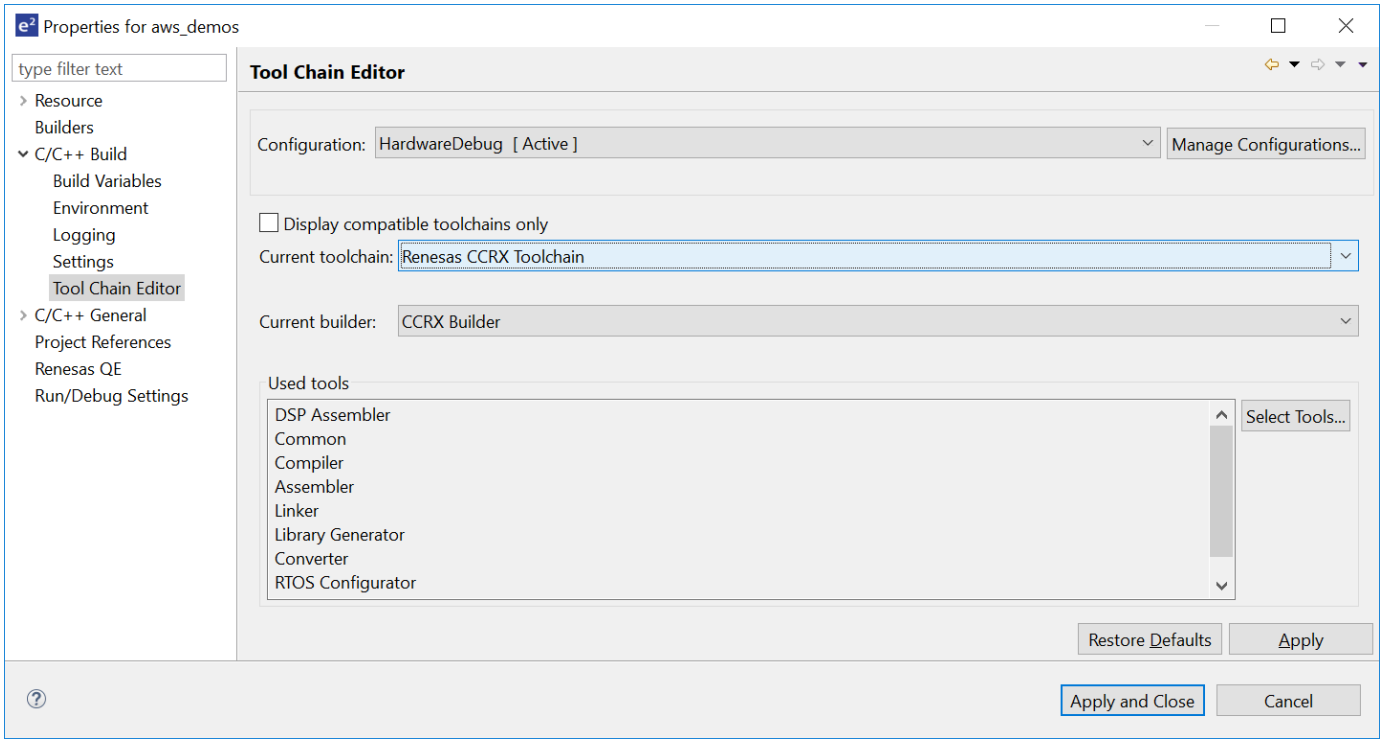
8. Seleziona i progetti aws_demos e boot_loader, quindi scegli Fine per importarli.



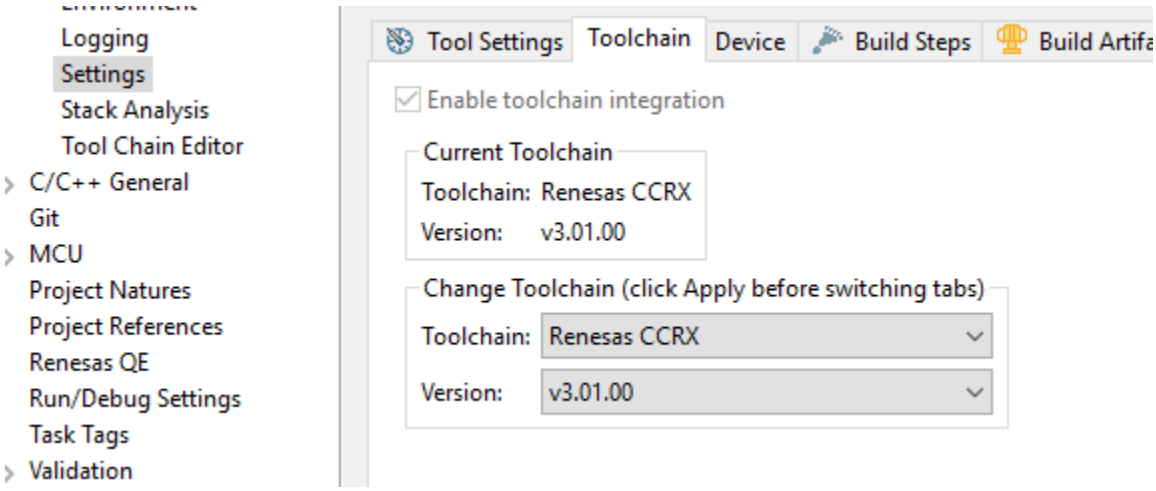
9. Per entrambi i progetti, apri le proprietà del progetto. Nel riquadro di navigazione, scegliete Tool Chain Editor.

    a. Scegliete la toolchain corrente.

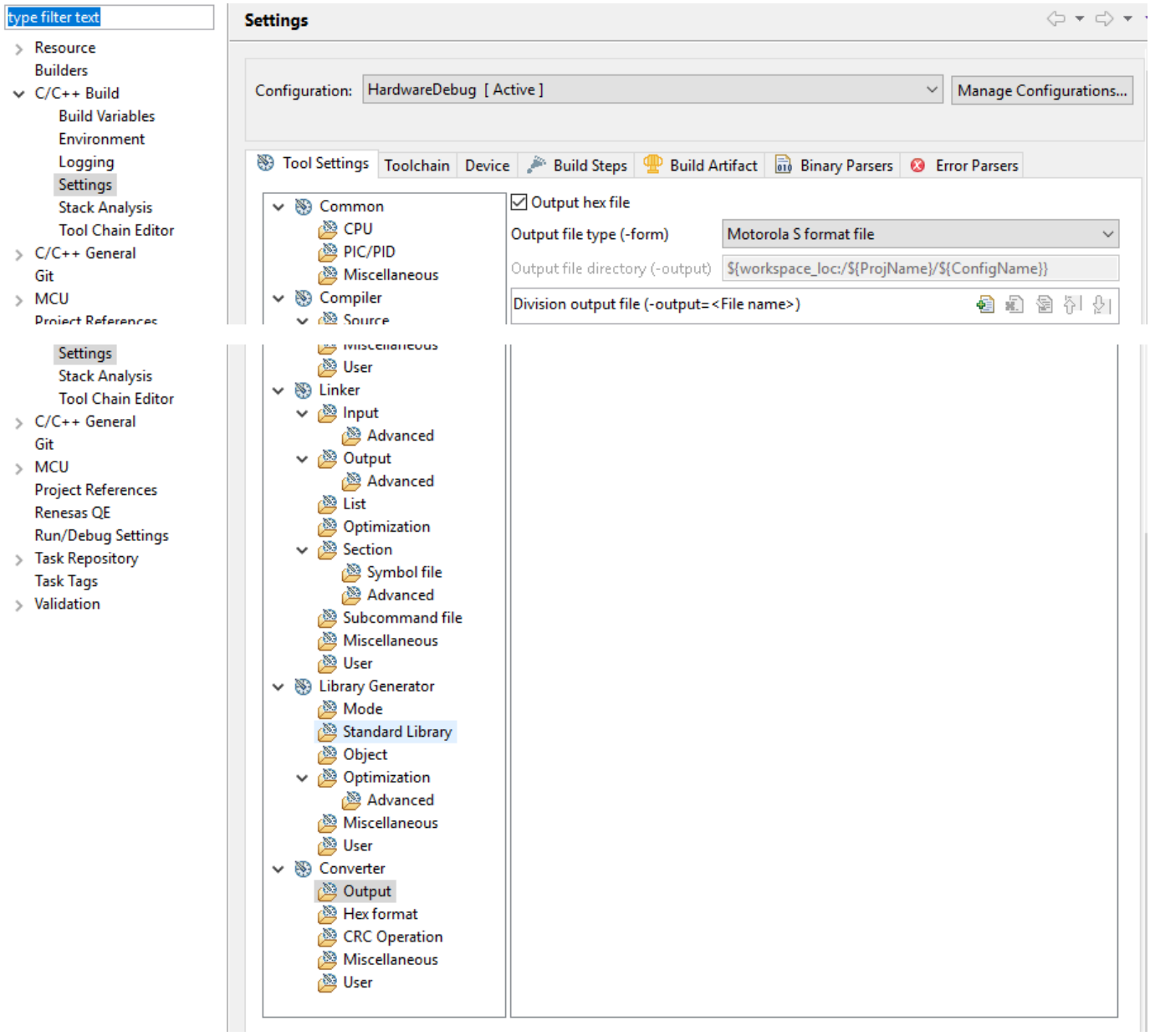
    b. Scegli il generatore Current.



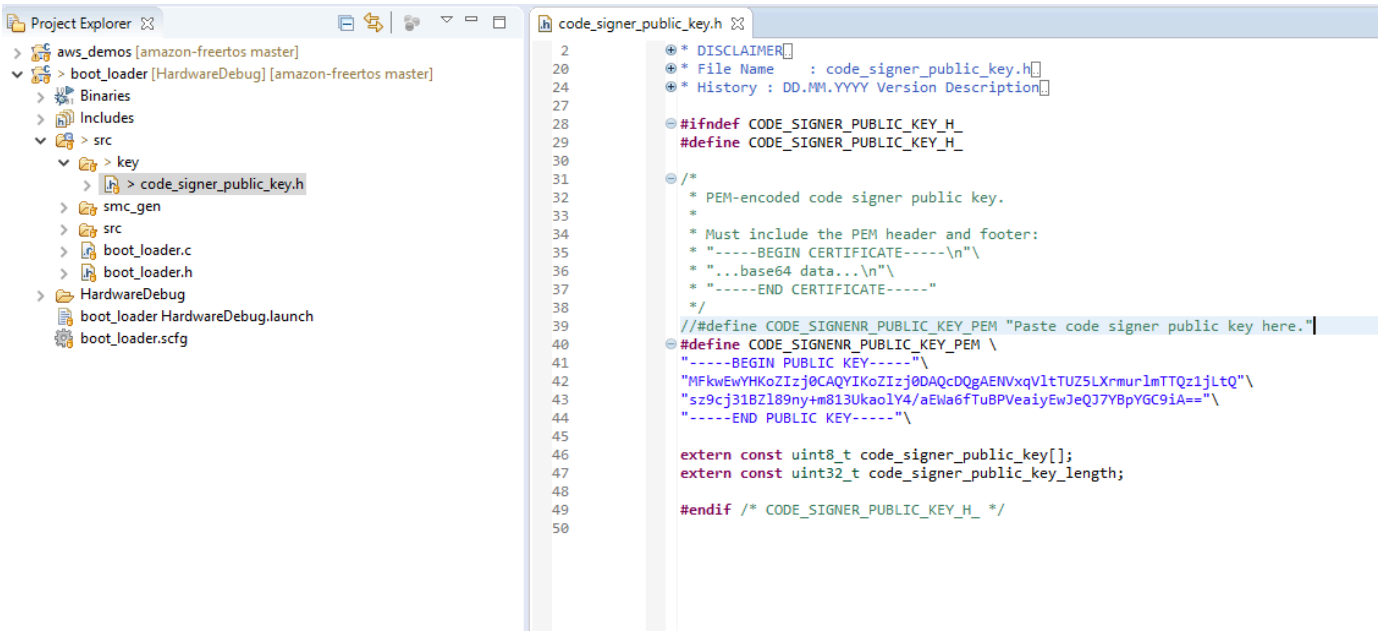
10. Nel pannello di navigazione scegli Impostazioni. Scegli la scheda Toolchain, quindi scegli la versione della toolchain.



Scegliete la scheda Impostazioni dello strumento, espandete Convertitore e scegliete Output. Nella finestra principale, assicurati che il file esadecimale di output sia selezionato, quindi scegli il tipo di file di output.



11. Nel progetto bootloader, apri `projects\renesas\rx65n-rsk\e2studio\boot_loader\src\key\code_signer_public_key.h` e inserisci la chiave pubblica. Per informazioni su come creare una chiave pubblica, consulta Come implementare FreerTOS OTA utilizzando Amazon Web Services RX65 su N e la sezione 7.3 «Generazione di coppie di chiavi ECDSA con OpenSSL» nella politica di progettazione dell'SHA256 aggiornamento del firmware MCU di Renesas.



Quindi `boot_loader.mot` crea il progetto da creare.

12. Apri il `aws_demos` progetto.

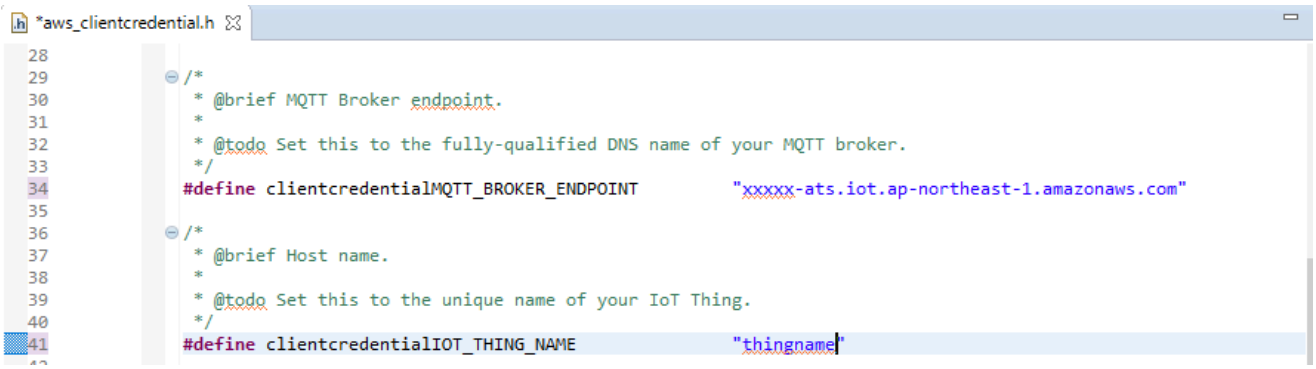
    a. Apri la AWS IoT console.

    b. Nel riquadro di navigazione a sinistra scegliere Impostazioni. Prendi nota del tuo endpoint personalizzato.

    c. Scegli Gestisci, quindi scegli Cose. Prendi nota del nome dell' AWS IoT oggetto della tua bacheca.

    d. Nel `aws_demos` progetto, aprite `demos/include/aws_clientcredential.h` e specificate i seguenti valori.

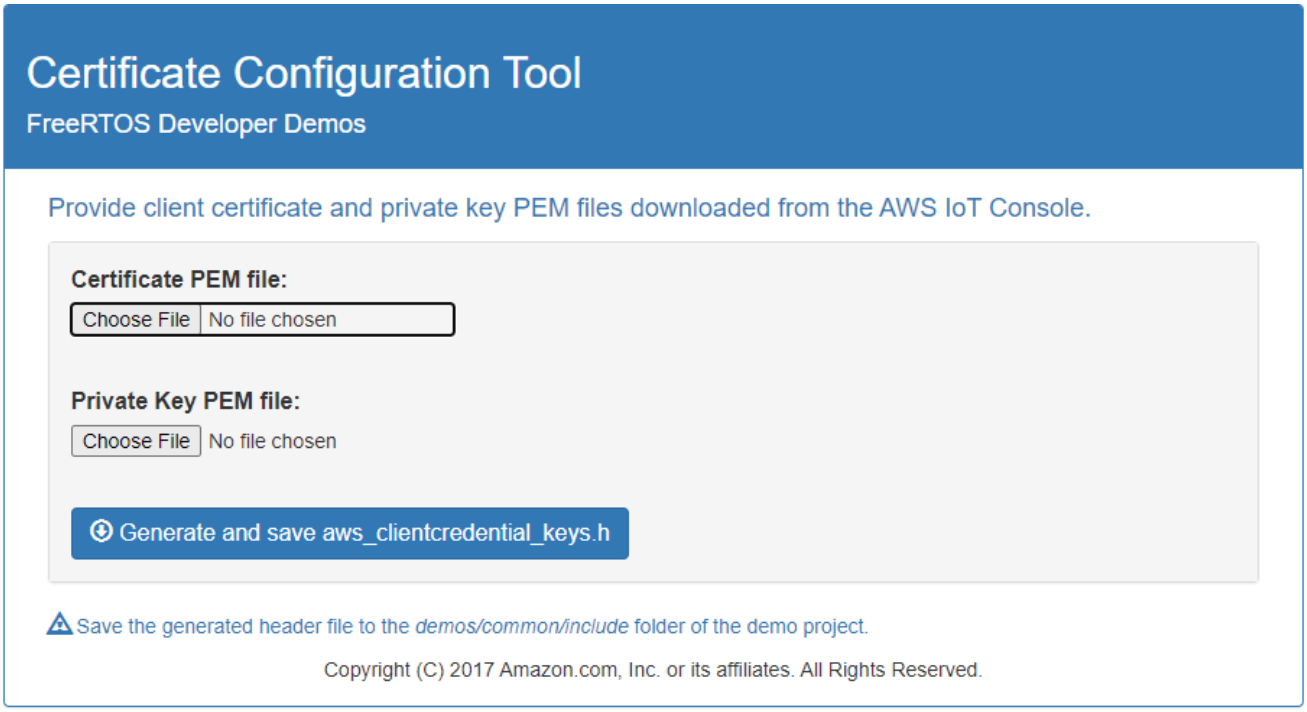
```
#define clientcredentialMQTT_BROKER_ENDPOINT[] = "Your AWS IoT endpoint";
#define clientcredentialIOT_THING_NAME "The AWS IoT thing name of your board"
```



e. Apri il file `tools/certificate_configuration/CertificateConfigurator.html`.

f. Importate il file PEM del certificato e il file PEM a chiave privata che avete scaricato in precedenza.

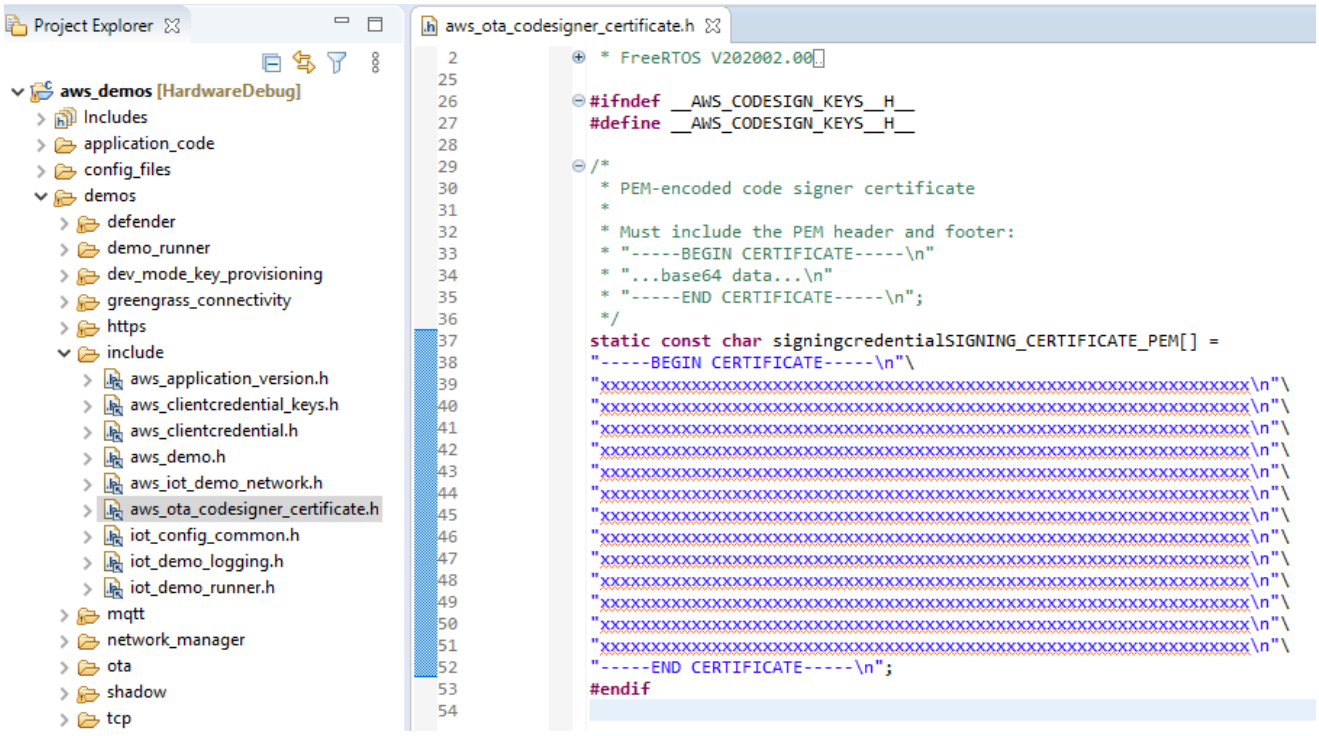
g. Scegli Genera e salva aws_clientcredential_keys.h e sostituisci questo file nella directory. `demos/include/`



h. `demos/include/aws_ota_codesigner_certificate.h`Apri il file e specifica questi valori.

```
#define signingcredentialSIGNING_CERTIFICATE_PEM [] = "your-certificate-key";
```

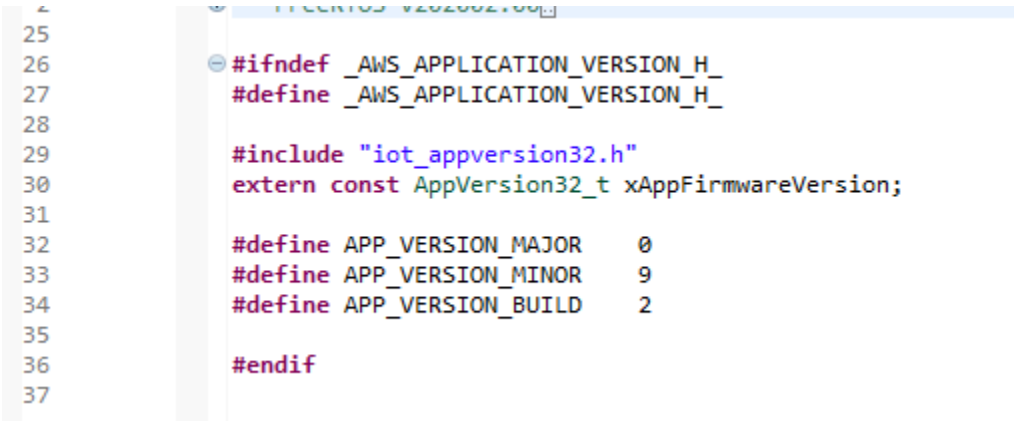
*`your-certificate-key`*Dov'è il valore del file`secp256r1.crt`. Ricordati di aggiungere «\» dopo ogni riga della certificazione. Per ulteriori informazioni sulla creazione del `secp256r1.crt` file, consulta Come implementare FreerTOS OTA utilizzando Amazon Web Services RX65 su N e la sezione 7.3 «Generazione di coppie di chiavi ECDSA SHA256 con OpenSSL» nella politica di progettazione dell'aggiornamento del firmware MCU di Renesas.



13. Attività A: installare la versione iniziale del firmware

    a. Apri il `vendors/renesas/boards/board/aws_demos/config_files/aws_demo_config.h` file, commenta e definisci`CONFIG_OTA_UPDATE_DEMO_ENABLED`. `#define CONFIG_MQTT_DEMO_ENABLED`

    b. Aprire il `demos/include/ aws_application_version.h` file e impostare la versione iniziale del firmware su`0.9.2`.



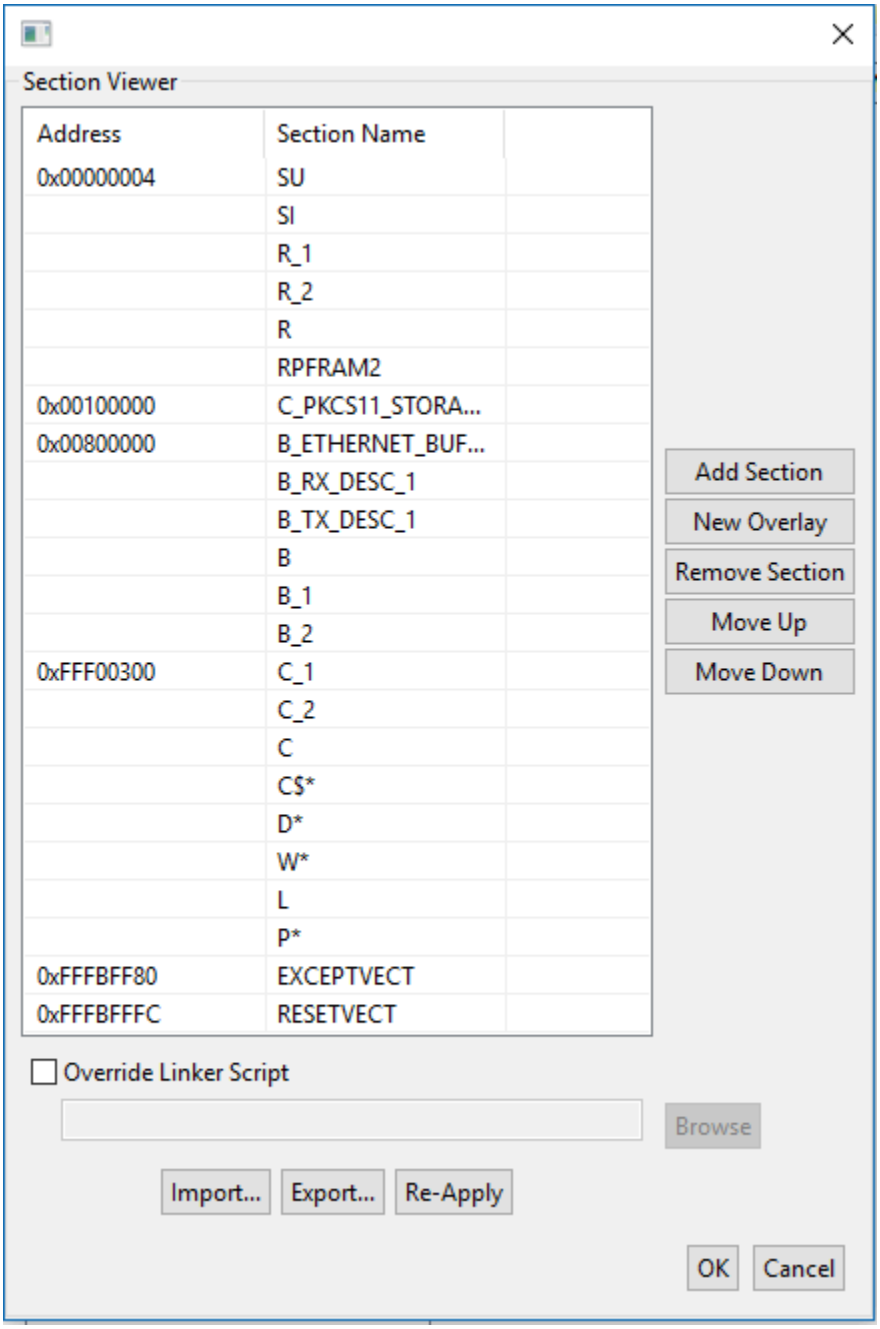
```
#ifndef _AWS_APPLICATION_VERSION_H_
#define _AWS_APPLICATION_VERSION_H_

#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. Modificate le seguenti impostazioni nel Section Viewer.



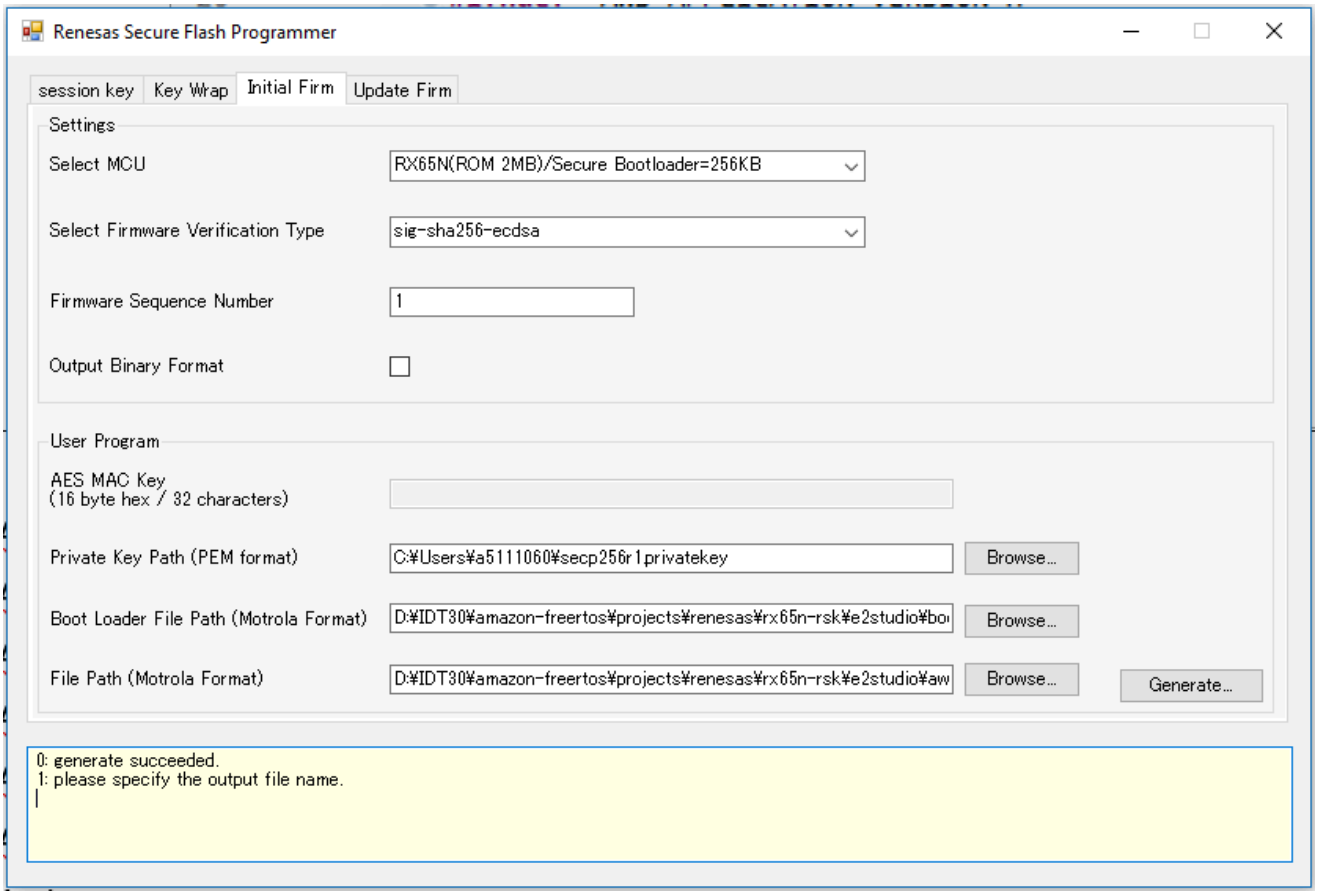
d. Scegliete Build per creare il `aws_demos.mot` file.

14. Crea il file `userprog.mot` con Renesas Secure Flash Programmer. `userprog.mot`è una combinazione di e. `aws_demos.mot` `boot_loader.mot` È possibile eseguire il flashing di questo file su RX65 N-RSK per installare il firmware iniziale.

    a. Scarica https://github.com/renesas/Amazon-FreeRTOS-Tools e apri. `Renesas Secure Flash Programmer.exe`

    b. Scegli la scheda Initial Firm e imposta i seguenti parametri:

- Private Key Path: la posizione di`secp256r1.privatekey`.
- Percorso del file del loader di avvio: la posizione di `boot_loader.mot` (`projects\renesas\rx65n-rsk\e2studio\boot_loader\HardwareDebug`).
- Percorso del file: la posizione di `aws_demos.mot` (`projects\renesas\rx65n-rsk\e2studio\aws_demos\HardwareDebug`).



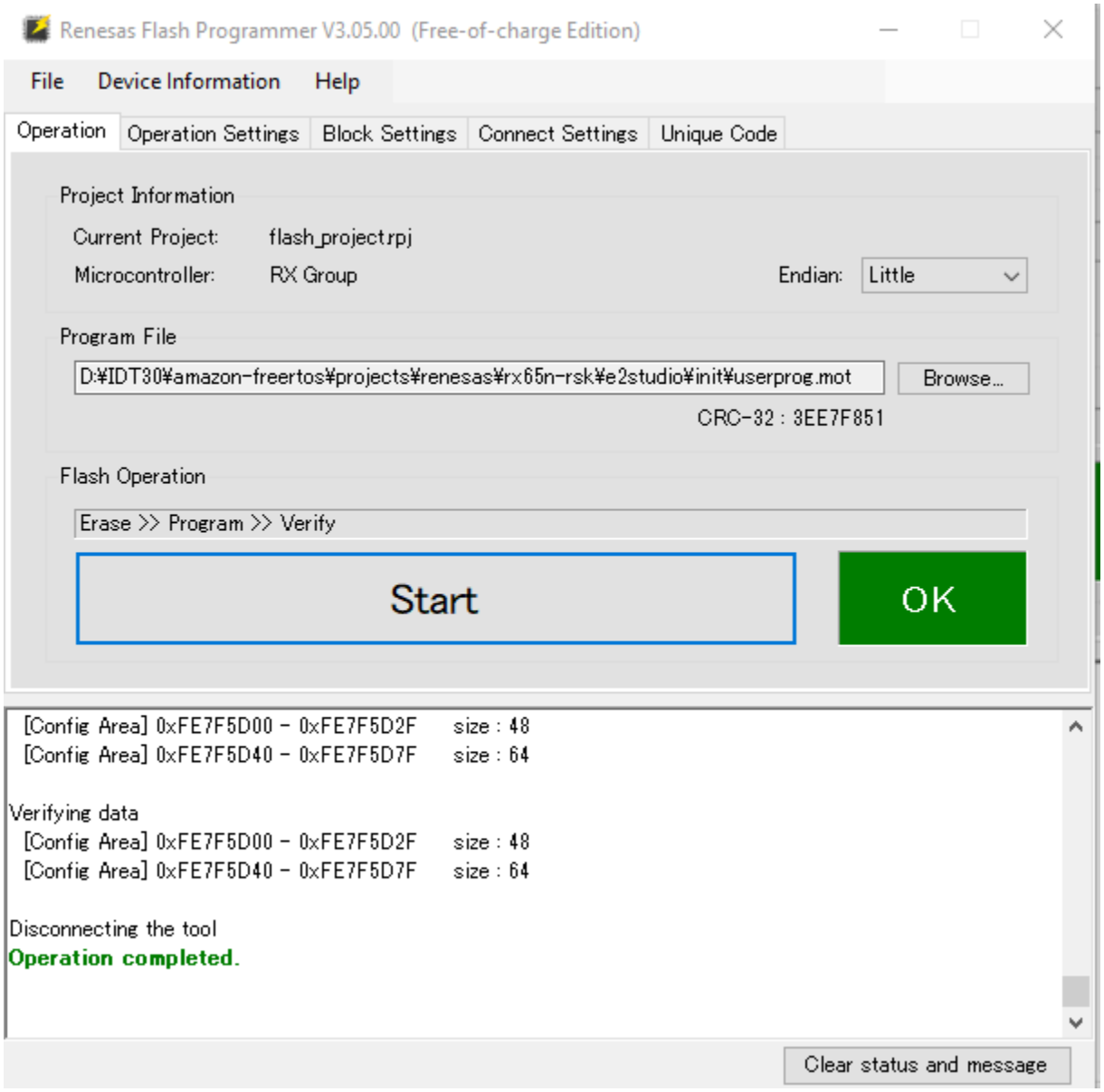
c. Create una directory `init_firmware` denominata `userprog.mot` Generate e salvatela `init_firmware` nella directory. Verificate che la generazione sia riuscita.

15. Esegui il flashing del firmware iniziale sull' RX65N-RSK.

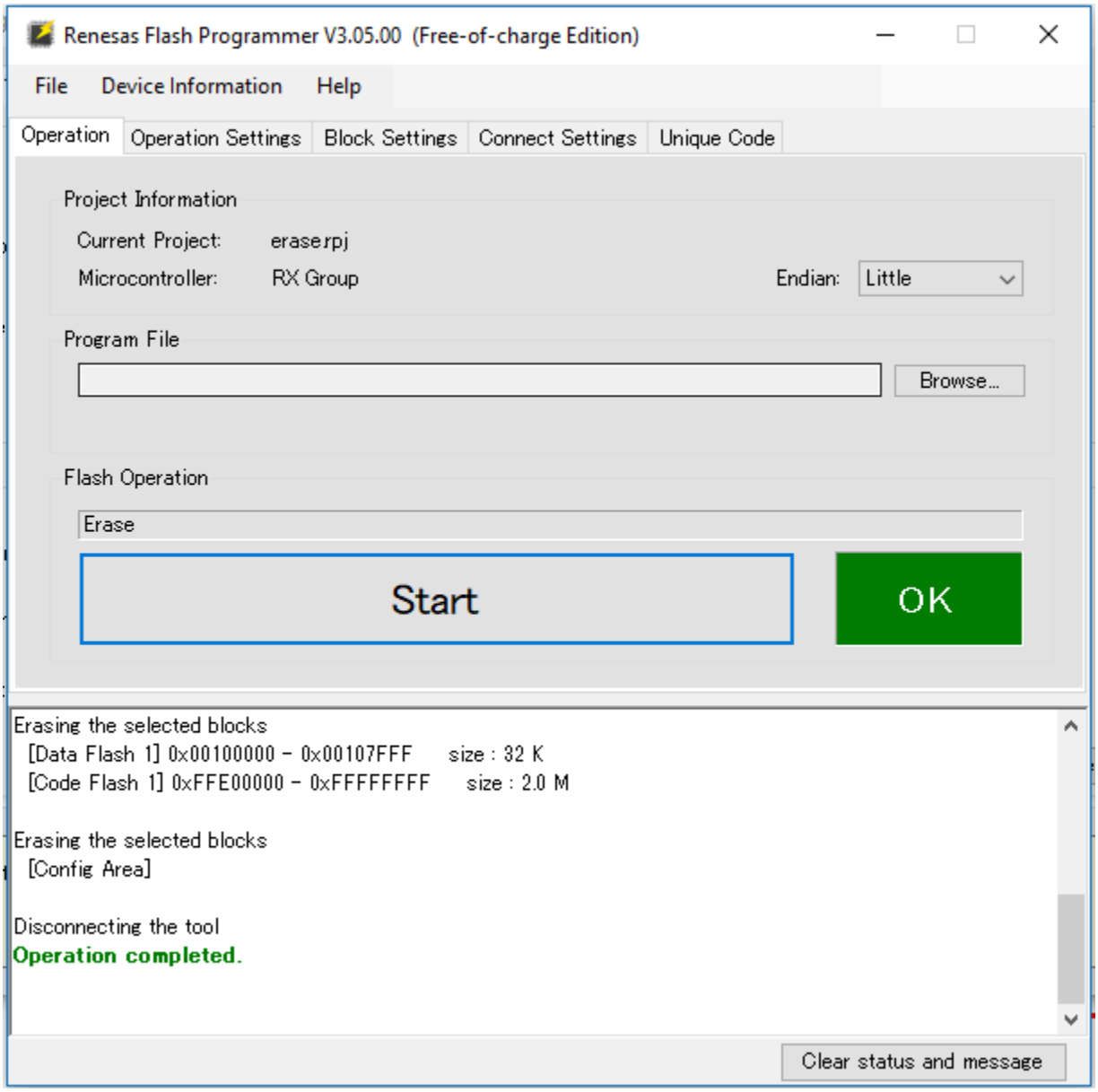
a. Scarica l'ultima versione di Renesas Flash Programmer (Programming GUI) da - .html. https://www.renesas.com/tw/ en/products/software-tools/tools/programmer/renesas flash-programmer-programming-gui

b. Apri il `vendors\renesas\rx_mcu_boards\boards\rx65n-rsk\aws_demos\flash_project\erase_from_bank\ erase.rpj` file per cancellare i dati sulla banca.

c. Scegli Inizia per cancellare la banca.



d. Per lampeggiare`userprog.mot`, scegli Sfoglia... e accedi alla `init_firmware` directory, seleziona il `userprog.mot` file e scegli Avvia.



16. La versione 0.9.2 (versione iniziale) del firmware è stata installata sul tuo RX65 N-RSK. La scheda RX65 N-RSK è ora in attesa degli aggiornamenti OTA. Se hai aperto Tera Term sul tuo PC, all'esecuzione del firmware iniziale viene visualizzato qualcosa di simile a quanto segue.

```
------------------------------------------------
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. Attività B: aggiorna la versione del firmware

    a. Aprire il `demos/include/aws_application_version.h` file e incrementare il valore del `APP_VERSION_BUILD` token a`0.9.3`.

b. Ricreare il progetto.

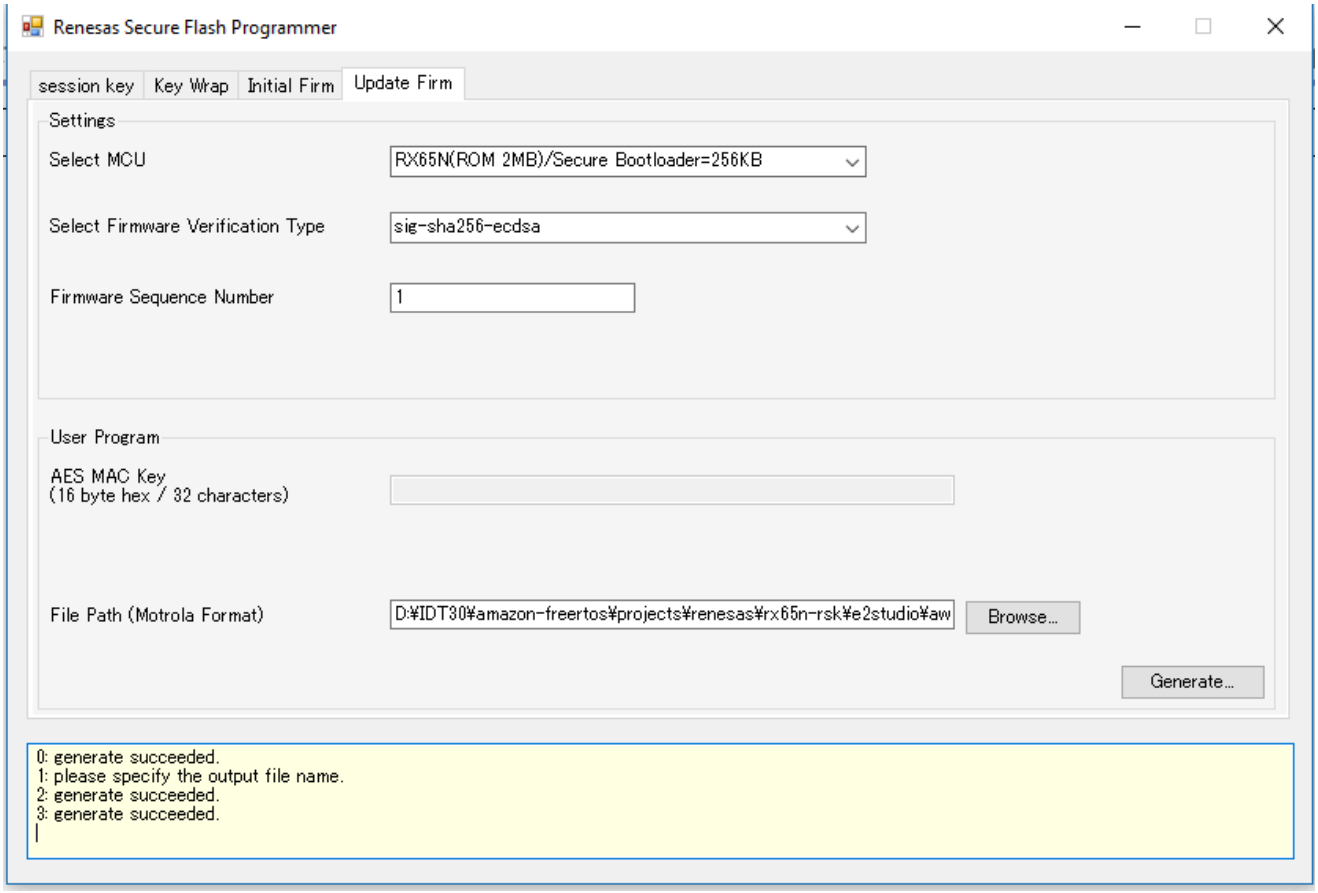
18. Crea il `userprog.rsu` file con Renesas Secure Flash Programmer per aggiornare la versione del firmware.

    a. Apri il file `Amazon-FreeRTOS-Tools\Renesas Secure Flash Programmer.exe`.

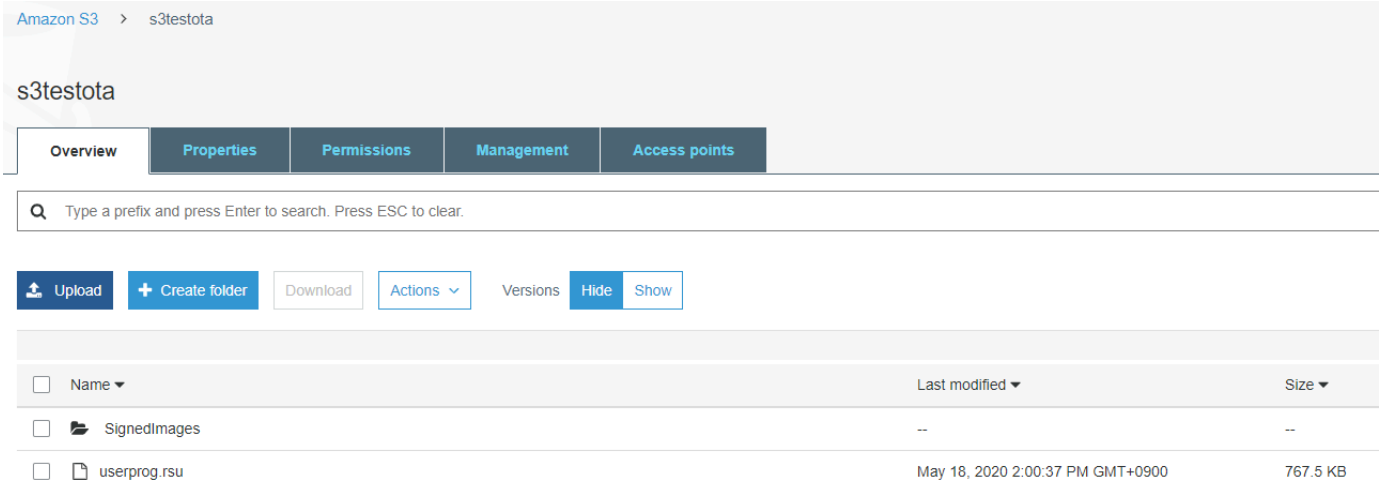
    b. Scegli la scheda Update Firm e imposta i seguenti parametri:

    - Percorso del file: la posizione del `aws_demos.mot` file (`projects\renesas\rx65n-rsk\e2studio\aws_demos\HardwareDebug`).

    c. Crea una directory denominata `update _firmware`. Genera `userprog.rsu` e salvalo nella `update_firmware` directory. Verifica che la generazione sia riuscita.



19. Carica l'aggiornamento del firmware in un bucket Amazon S3 come descritto in. `userproj.rsu` [Crea un bucket Amazon S3 per archiviare l'aggiornamento]



20. Crea un processo per aggiornare il firmware sull'N-RSK. RX65

    AWS IoT Jobs è un servizio che notifica a uno o più dispositivi connessi un Job in sospeso. Un job può essere utilizzato per gestire una flotta di dispositivi, aggiornare il firmware e i certificati di sicurezza sui dispositivi o eseguire attività amministrative come il riavvio dei dispositivi e l'esecuzione di diagnosi.

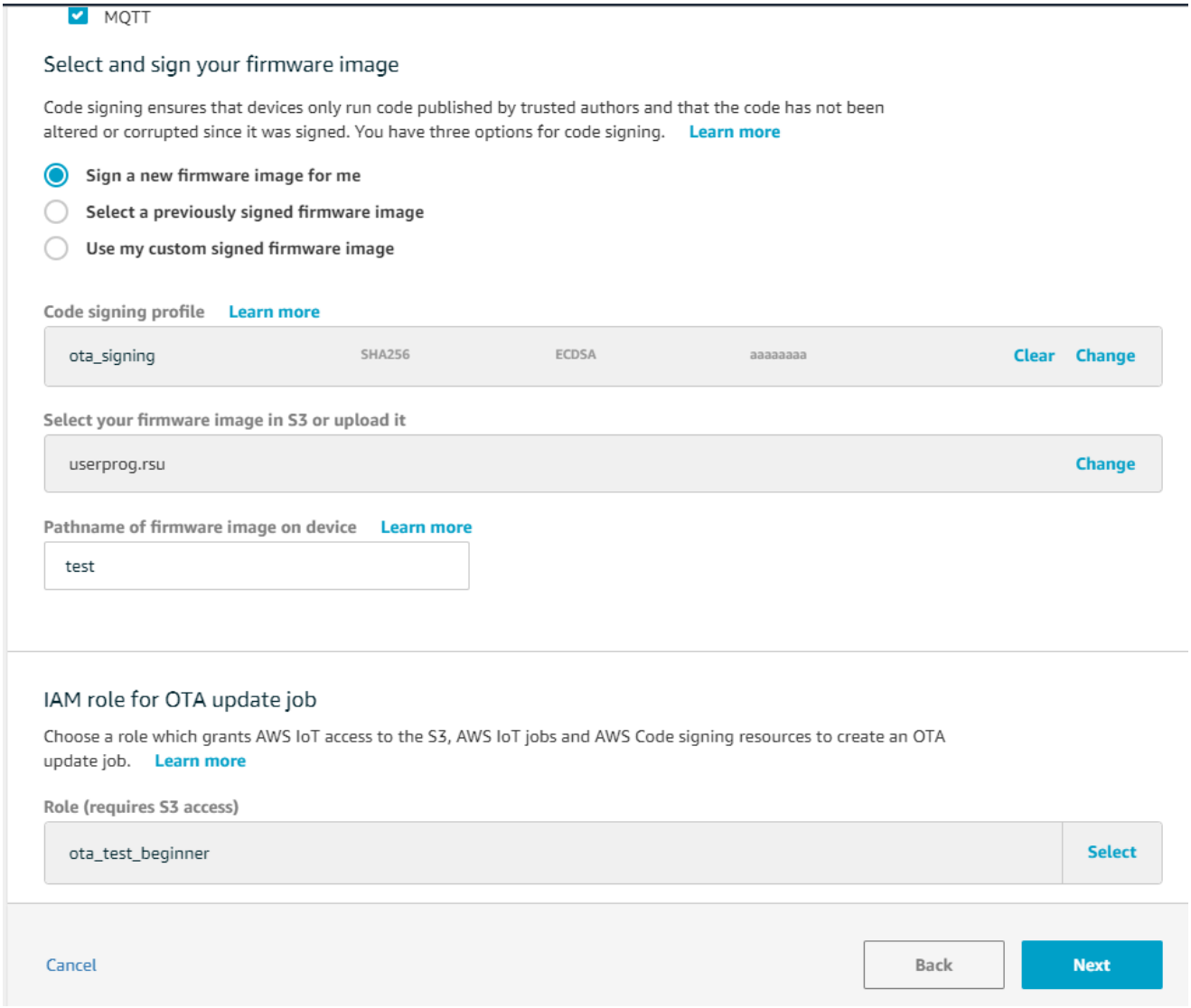
    a. Accedere alla console AWS IoT. Nel riquadro di navigazione, scegli Gestisci e scegli Lavori.

    b. Scegli Crea un lavoro, quindi scegli Crea lavoro OTA Update. Seleziona una cosa, quindi scegli Avanti.

    c. Crea un processo di aggiornamento OTA FreerTOS come segue:

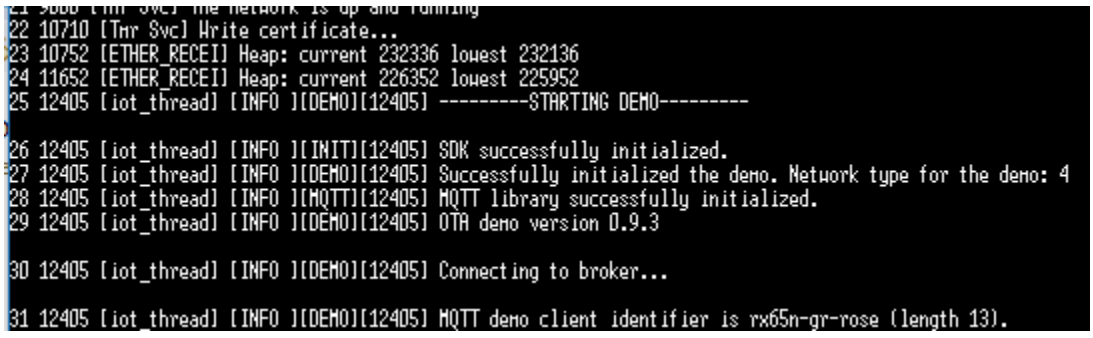
    - Scegli MQTT.
    - Seleziona il profilo di firma del codice che hai creato nella sezione precedente.
    - Seleziona l'immagine del firmware che hai caricato in un bucket Amazon S3.
    - Per il nome del percorso dell'immagine del firmware sul dispositivo, inserisci. **`test`**
    - Scegli il ruolo IAM che hai creato nella sezione precedente.

    d. Scegli Next (Successivo).

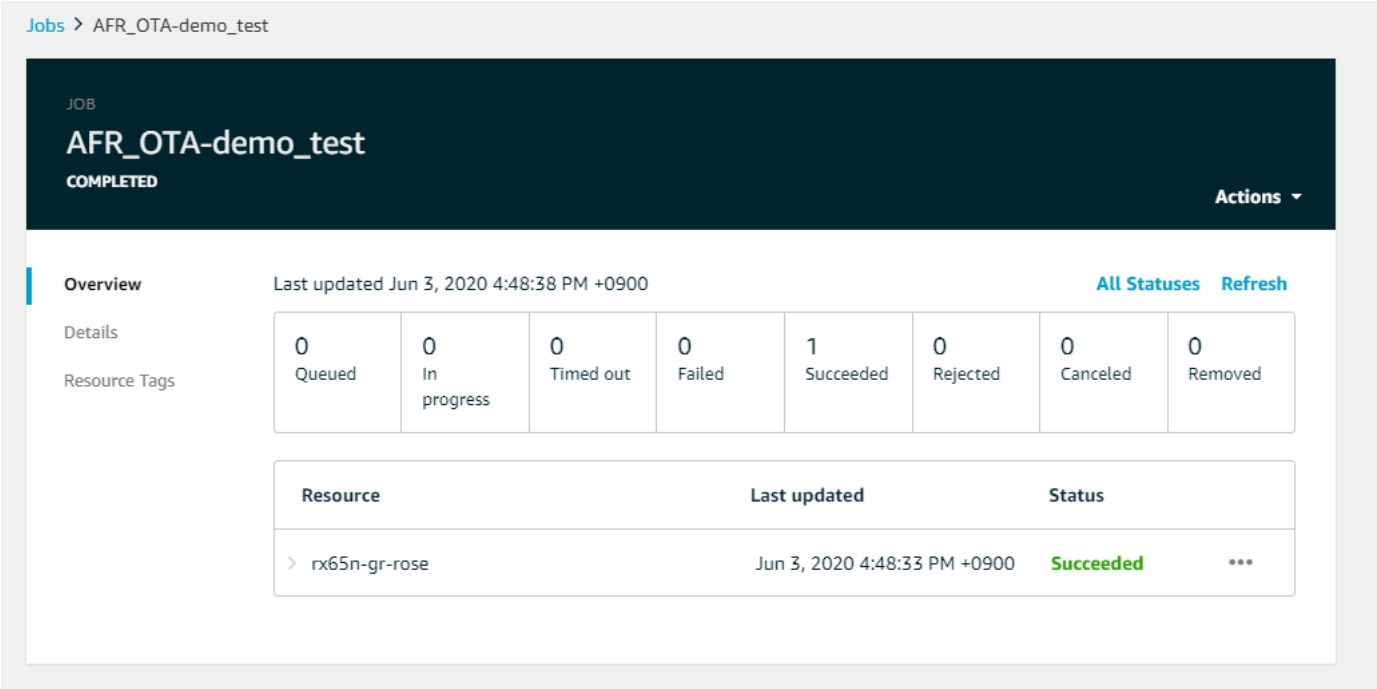


e. Inserisci un ID e poi scegli Crea.

21. Riapri Tera Term per verificare che il firmware sia stato aggiornato correttamente alla versione demo OTA 0.9.3.



22. Sulla AWS IoT console, verifica che lo stato del processo sia Riuscito.



AWS IoT Applicazione demo Device Shadow

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Introduzione

Questa demo mostra come utilizzare la libreria AWS IoT Device Shadow per connettersi al servizio AWS Device Shadow. Utilizza la libreria CoreMQTT per stabilire una connessione MQTT con TLS (Mutual Authentication) al broker AWS IoT MQTT e il parser della libreria CoreJSON per analizzare i documenti shadow ricevuti dal servizio Shadow. AWS La demo mostra le operazioni shadow di base, come aggiornare un documento shadow e come eliminare un documento shadow. La demo mostra anche come registrare una funzione di callback con la libreria CoreMQTT per gestire messaggi come l'shadow `/update` e i `/update/delta` messaggi inviati dal servizio Device Shadow AWS IoT .

Questa demo è intesa come esercizio di apprendimento solo perché la richiesta di aggiornamento del documento ombra (stato) e la risposta all'aggiornamento vengono eseguite dalla stessa applicazione.

In uno scenario di produzione realistico, un'applicazione esterna richiederebbe un aggiornamento dello stato del dispositivo in remoto, anche se il dispositivo non è attualmente connesso. Il dispositivo riconoscerà la richiesta di aggiornamento quando sarà connesso.

> **Note**
>
> Per configurare ed eseguire le demo di FreerTOS, segui i passaggi indicati. Guida introduttiva a FreerTOS

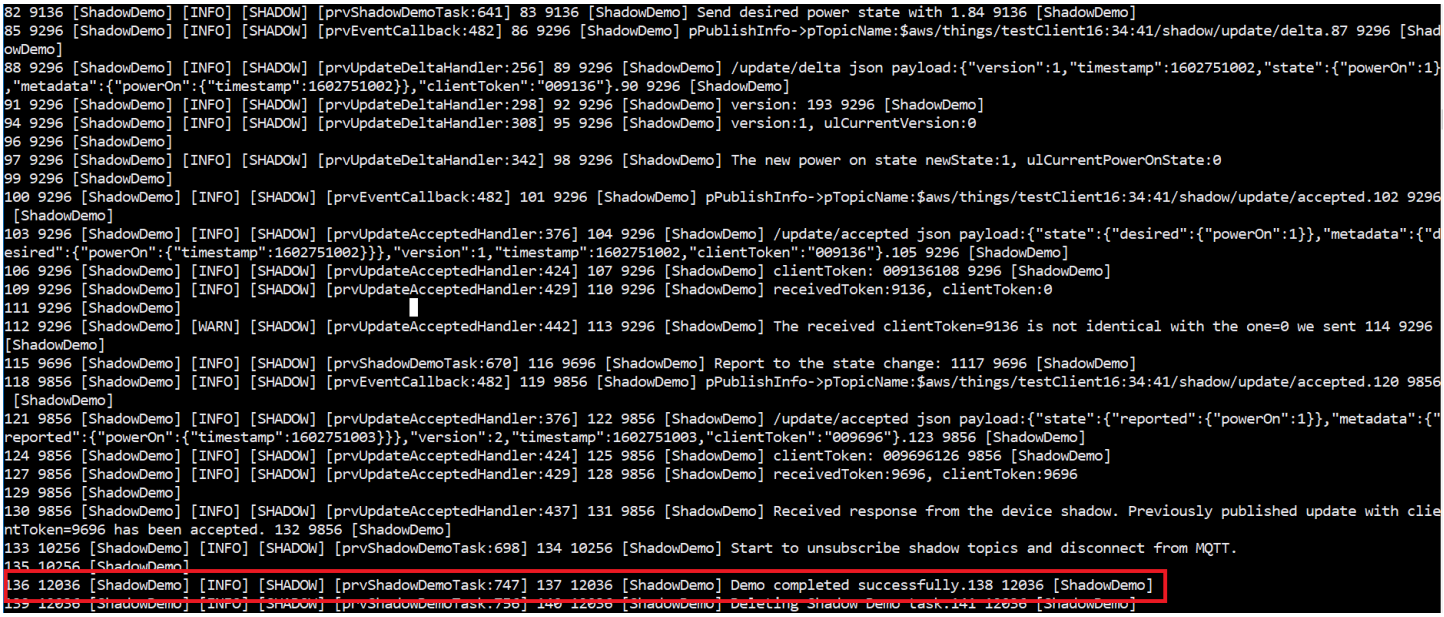
Funzionalità

La demo crea una singola attività applicativa che ripercorre una serie di esempi che mostrano shadow `/update` e `/update/delta` callback per simulare la commutazione dello stato di un dispositivo remoto. Invia un aggiornamento shadow con il nuovo `desired` stato e attende che il dispositivo cambi il suo `reported` stato in risposta al nuovo stato. `desired` Inoltre, viene utilizzato un `/update` callback shadow per stampare i cambiamenti degli stati ombra. Questa demo utilizza anche una connessione MQTT sicura al broker AWS IoT MQTT e presuppone che esista uno `power0n` stato nell'ombra del dispositivo.

La demo esegue le seguenti operazioni:

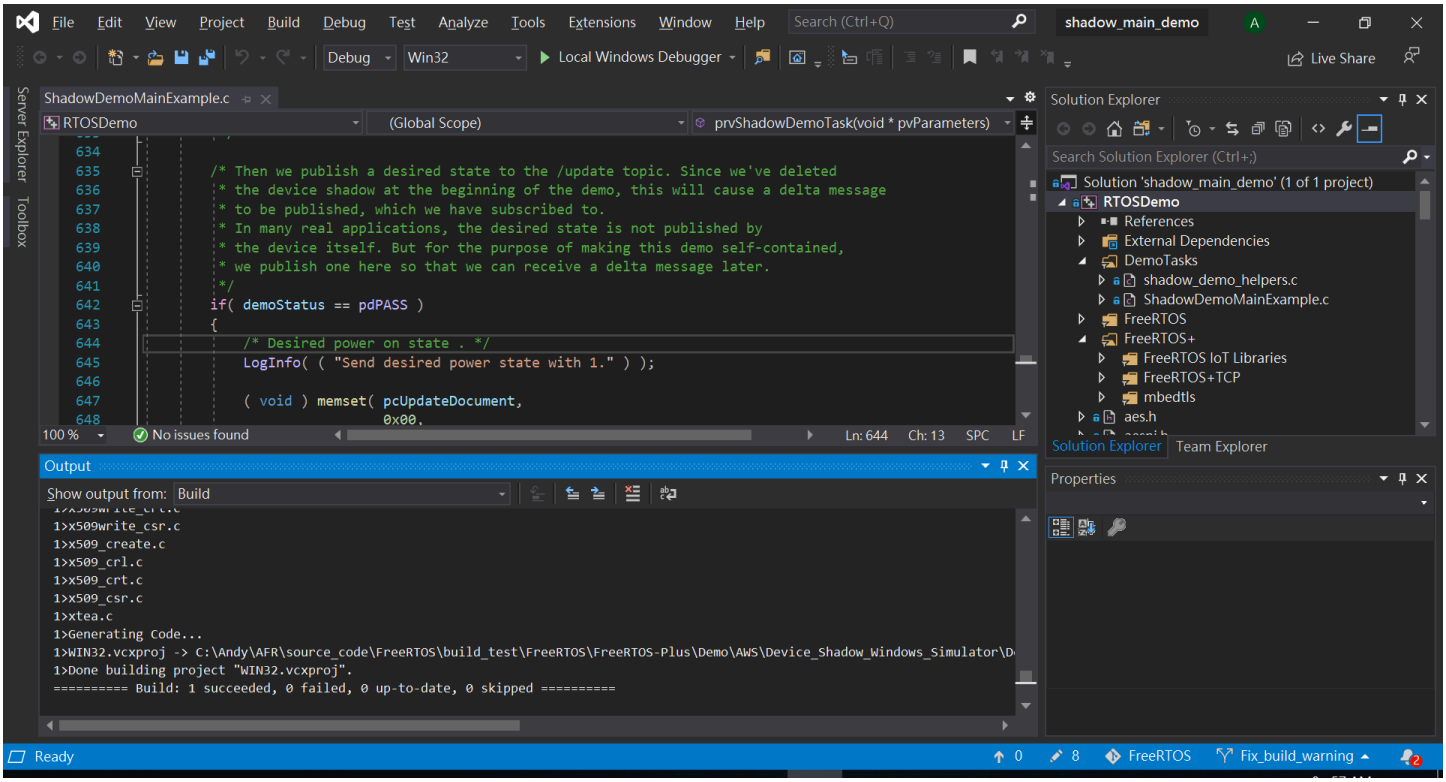
1. Stabilire una connessione MQTT utilizzando le funzioni di supporto in. `shadow_demo_helpers.c`
2. Assemblate stringhe di argomenti MQTT per le operazioni di shadow del dispositivo, utilizzando le macro definite dalla libreria Device Shadow AWS IoT .
3. Pubblicate sull'argomento MQTT utilizzato per eliminare l'ombra di un dispositivo per eliminare qualsiasi ombra del dispositivo esistente.
4. Abbonatevi agli argomenti MQTT per `/update/accepted` e come `/update/rejected` utilizzare `/update/delta` le funzioni di supporto in. `shadow_demo_helpers.c`
5. Pubblica lo stato desiderato di `power0n` utilizzo delle funzioni di supporto in. `shadow_demo_helpers.c` Ciò provocherà l'invio di un `/update/delta` messaggio al dispositivo.
6. Gestisci i messaggi MQTT in `prvEventCallback` arrivo e determina se il messaggio è correlato all'ombra del dispositivo utilizzando una funzione definita dalla libreria AWS IoT Device Shadow ()`Shadow_MatchTopic`. Se il messaggio è un `/update/delta` messaggio shadow del dispositivo, la funzione demo principale pubblicherà un secondo messaggio a cui aggiornare lo

stato segnalato. `powerOn` Se viene ricevuto un `/update/accepted` messaggio, verificate che sia lo `clientToken` stesso pubblicato in precedenza nel messaggio di aggiornamento. Questo segnerà la fine della demo.



La demo può essere trovata nel file *freertos*`/demos/device_shadow_for_aws/shadow_demo_main.c` o su GitHub.

La schermata seguente mostra l'output previsto quando la demo ha esito positivo.



Connect al broker AWS IoT MQTT

Per connetterci al broker AWS IoT MQTT, utilizziamo lo stesso metodo `MQTT_Connect()` di. [Demo di autenticazione reciproca CoreMQTT]

Eliminare il documento ombra

Per eliminare il documento shadow, chiamate `xPublishToTopic` con un messaggio vuoto, utilizzando le macro definite dalla libreria AWS IoT Device Shadow. Viene utilizzato `MQTT_Publish` per pubblicare sull'`/delete`argomento. La seguente sezione di codice mostra come questa operazione viene eseguita nella funzione`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 );
```

Iscriviti agli argomenti ombra

Iscriviti agli argomenti Device Shadow per ricevere notifiche dal AWS IoT broker sulle modifiche dello shadow. Gli argomenti Device Shadow sono assemblati da macro definite nella libreria Device Shadow. La seguente sezione di codice mostra come questa operazione viene eseguita nella `prvShadowDemoTask` funzione.

```
/* 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 ) );
}
```

Invia aggiornamenti Shadow

Per inviare un aggiornamento shadow, la demo chiama `xPublishToTopic` con un messaggio in formato JSON, utilizzando macro definite dalla libreria Device Shadow. Viene utilizzato `MQTT_Publish` per pubblicare sull'`/delete`argomento. La seguente sezione di codice mostra come questa operazione viene eseguita nella `prvShadowDemoTask` funzione.

```
#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 ) );
```

Gestisci i messaggi shadow delta e i messaggi shadow update

La funzione di callback utente, che è stata registrata nella libreria client CoreMQTT utilizzando la `MQTT_Init` funzione, ci avviserà di un evento relativo a un pacchetto in arrivo. Vedi la funzione di callback attiva. prvEventCallback GitHub

La funzione di callback conferma che il pacchetto in entrata è di tipo `MQTT_PACKET_TYPE_PUBLISH` e utilizza l'API Device Shadow Library `Shadow_MatchTopic` per confermare che il messaggio in arrivo è un messaggio shadow.

Se il messaggio in arrivo è un messaggio ombra di tipo`ShadowMessageTypeUpdateDelta`, chiamiamo prvUpdateDeltaHandler per gestire questo messaggio. Il gestore `prvUpdateDeltaHandler` utilizza la libreria CoreJSON per analizzare il messaggio per ottenere il valore delta per `powerOn` lo stato e lo confronta con lo stato corrente del dispositivo gestito localmente. Se questi sono diversi, lo stato del dispositivo locale viene aggiornato in modo da riflettere il nuovo valore `powerOn` dello stato del documento ombra.

Se il messaggio in arrivo è un messaggio shadow con tipo`ShadowMessageTypeUpdateAccepted`, chiamiamo prvUpdateAcceptedHandler per gestire questo messaggio. Il gestore `prvUpdateAcceptedHandler` analizza il messaggio utilizzando la libreria CoreJSON per ottenere il contenuto del messaggio. `clientToken` Questa funzione di gestione verifica che il token client del messaggio JSON corrisponda al token client utilizzato dall'applicazione. Se non corrisponde, la funzione registra un messaggio di avviso.

Demo Client Echo Secure Sockets

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

L'esempio seguente utilizza un'attività RTOS singola. Il codice sorgente per questo esempio è disponibile all'indirizzo `demos/tcp/aws_tcp_echo_client_single_task.c`.

Prima di iniziare, verifica di aver scaricato FreeRTOS sul tuo microcontrollore e di aver creato ed eseguito i progetti demo di FreerTOS. Puoi clonare o scaricare FreerTOS da. GitHub Consultare il file README.md per le istruzioni.

Per eseguire la demo

> **Note**
>
> Per configurare ed eseguire le demo di FreerTOS, segui i passaggi indicati. Guida introduttiva a FreerTOS
> Le demo del server e del client TCP non sono attualmente supportate sui kit di sviluppo Cypress CYW9439 07 AEVAL1 F e 07 F. CYW9549 AEVAL1

1. Segui le istruzioni in Configurazione del TLS Echo Server nella FreerTOS Porting Guide.

   Il server echo TLS deve essere in esecuzione e in ascolto sulla porta 9000.

   Durante la configurazione, occorre aver generato quattro file:

   - `client.pem` (certificato del client)
   - `client.key` (chiave privata del client)
   - `server.pem` (certificato del server)
   - `server.key` (chiave privata del server)

2. Utilizzare lo strumento `tools/certificate_configuration/CertificateConfigurator.html` per copiare il certificato del client (`client.pem`) e la chiave privata del client (`client.key`) in `aws_clientcredential_keys.h`.
3. Apri il file `FreeRTOSConfig.h`.
4. Impostare le variabili `configECHO_SERVER_ADDR0`, `configECHO_SERVER_ADDR1`, `configECHO_SERVER_ADDR2` e `configECHO_SERVER_ADDR3` sui quattro interi che costituiscono l'indirizzo IP in cui il TLS Echo Server è in esecuzione.
5. Impostare la variabile `configTCP_ECHO_CLIENT_PORT` su `9000`, la porta su cui il TLS Echo Server è in ascolto.
6. Impostare la variabile `configTCP_ECHO_TASKS_SINGLE_TASK_TLS_ENABLED` su `1`.
7. Utilizzare lo strumento `tools/certificate_configuration/PEMfileToCString.html` per copiare il certificato del server (`server.pem`) in `cTlsECHO_SERVER_CERTIFICATE_PEM` nel file `aws_tcp_echo_client_single_task.c`.
8. Aprire *`freertos`*`/vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/aws_demo_config.h`, commentare `#define CONFIG_MQTT_DEMO_ENABLED` e definire `CONFIG_TCP_ECHO_CLIENT_DEMO_ENABLED`.

Il microcontroller e il server echo TLS devono trovarsi sulla stessa rete. All'avvio della demo (`main.c`), viene visualizzato un messaggio di log contenente `Received correct string from echo server`.

Utilizzo AWS IoT Device Tester per FreerTOS

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Puoi usare AWS IoT Device Tester (IDT) for FreerTOS per verificare che il sistema operativo FreeRTOS funzioni localmente sul tuo dispositivo e possa comunicare con il Cloud. AWS IoT In particolare, verifica che le interfacce del livello di porting per le librerie FreerTOS siano implementate correttamente. Esegue anche test con. end-to-end AWS IoT Core Ad esempio, verifica che la scheda sia in grado di inviare e ricevere messaggi MQTT ed elaborarli correttamente. I test eseguiti da IDT per FreerTOS sono definiti nel repository FreerTOS. GitHub

I test vengono eseguiti come applicazioni incorporate di cui viene eseguito il flashing nella scheda. Le immagini binarie dell'applicazione includono FreeRTOS, le interfacce FreeRTOS con porting del fornitore di semiconduttori e i driver dei dispositivi di scheda. Lo scopo dei test è verificare che le interfacce FreerTOS portate funzionino correttamente sui driver del dispositivo.

IDT for FreerTOS genera report di test che puoi inviare AWS IoT per aggiungere il tuo hardware al AWS Partner Device Catalog. Per ulteriori informazioni, consulta AWS Device Qualification Program.

IDT per FreerTOS funziona su un computer host (Windows, macOS o Linux) collegato alla scheda da testare. IDT esegue i casi di test e aggrega i risultati. Inoltre offre un'interfaccia a riga di comando per gestire l'esecuzione dei test.

Oltre ai dispositivi di test, IDT for FreeRTOS crea risorse (ad esempio AWS IoT oggetti, gruppi FreeRTOS, funzioni Lambda e così via) per facilitare il processo di qualificazione.

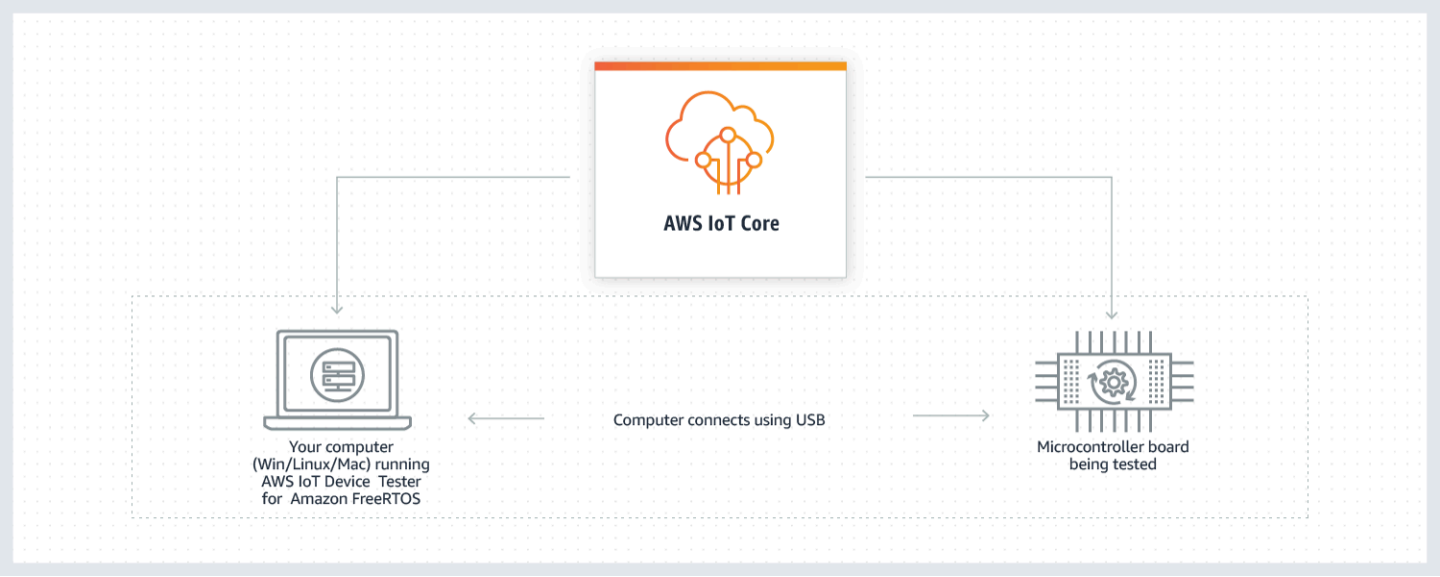
Per creare queste risorse, IDT for FreerTOS utilizza AWS le credenziali configurate in per effettuare chiamate API per `config.json` tuo conto. Il provisioning di queste risorse viene effettuato varie volte nel corso di un test.

Quando esegui IDT for FreerTOS sul tuo computer host, esegue i seguenti passaggi:

1. Carica e convalida la configurazione del dispositivo e delle credenziali.

2. Esegue i test selezionati con le risorse locali e cloud richieste.
3. Esegue la pulizia di risorse locali e cloud.
4. Genera i report di test che indicano se la scheda ha superato i test richiesti per la qualifica.

Il seguente diagramma illustra la configurazione dell'infrastruttura di test.



Per eseguire IDT for FreerTOS, puoi utilizzare le risorse di test. Esistono due tipi di risorse:

- Una suite di test è l'insieme di gruppi di test utilizzati per verificare che un dispositivo funzioni con versioni particolari di FreerTOS.
- Un gruppo di test è il set di singoli test relativi a una particolare funzionalità, come la messaggistica BLE e MQTT.

Per ulteriori informazioni, consulta AWS IoT Device Tester per le versioni della suite di test FreerTOS.

Versioni supportate di AWS IoT Device Tester per FreerTOS

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Questo argomento elenca le versioni supportate di IDT per FreerTOS. Come best practice, ti consigliamo di utilizzare l'ultima versione di IDT per FreeRTOS che supporti la versione di destinazione di FreerTOS. Ogni versione di IDT per FreerTOS ha una o più versioni corrispondenti di FreerTOS. Le nuove versioni di FreerTOS potrebbero richiedere il download di una nuova versione di IDT per FreerTOS.

Scaricando il software, accetti il Contratto di licenza IDT for FreerTOS.

Versione più recente di AWS IoT Device Tester for FreerTOS

Usa i seguenti link per scaricare l'ultima versione di IDT per FreerTOS.

IDT v4.0.3 e versione della suite di test 1.5.0 per FreeRTOS 202012.00 (utilizza le librerie FreeRTOS 202012.00 LTS)

- IDT v4.0.3 con suite di test FRQ_1.5.0 per Linux
- IDT v4.0.3 con suite di test FRQ_1.5.0 per macOS
- IDT v4.0.3 con suite di test FRQ_1.5.0 per Windows

> **Note**
>
> Non è consigliabile che più utenti eseguano IDT da un percorso condiviso, ad esempio una directory NFS o una cartella condivisa di rete Windows. Questa pratica potrebbe causare arresti anomali o il danneggiamento dei dati. Si consiglia di estrarre il pacchetto IDT in un'unità locale ed eseguire il file binario IDT sulla workstation locale.

Note di rilascio

- Support per la qualificazione di dispositivi con credenziali bloccate su un Hardware Security Module.
- Miglioramenti e correzioni di bug minori.

Versioni della suite di test

- FRQ_1.5.0
  - Rilasciato il 2021.04.16.

Versioni IDT precedenti per FreerTOS

Sono supportate anche le seguenti versioni precedenti di IDT per FreerTOS.

IDT v4.0.1 e versione della suite di test 1.4.1 per FreeRTOS 202012.00 (utilizza le librerie FreeRTOS 202012.00 LTS)

- IDT v4.0.1 con suite di test FRQ_1.4.1 per Linux
- IDT v4.0.1 con suite di test FRQ_1.4.1 per macOS
- IDT v4.0.1 con suite di test FRQ_1.4.1 per Windows

Note di rilascio

- Supporta FreeRTOS 202012.00 che utilizza le librerie FreerTOS LTS. Per ulteriori informazioni su ciò che è incluso nella versione FreerTOS 202012.00, consulta il file ChangeLog.md in. GitHub
- Presenta casi di test OTA (O) E2E () aggiuntivi. ver-the-air end-to-end
- Supporta la qualificazione delle schede di sviluppo che eseguono FreerTOS 202012.00 che utilizzano le librerie FreeRTOS LTS.
- Aggiunge il supporto per la qualificazione delle schede di sviluppo FreerTOS utilizzando la connettività cellulare.
- Risolve un bug nella configurazione del server echo.
- Ti consente di sviluppare ed eseguire le tue suite di test personalizzate utilizzando AWS IoT Device Tester FreerTOS. Per ulteriori informazioni, consulta Usa IDT per sviluppare ed eseguire le tue suite di test.
- Fornisce applicazioni IDT con firma di codice, quindi non devi concedere autorizzazioni quando lo esegui su Windows o macOS.
- Ha perfezionato la logica di analisi dei risultati del test BLE.

Versioni della suite di test

- FRQ_1.4.1
  - Rilasciato il 2021.01.19.

IDT v3.4.0 e versione 1.3.0 della suite di test per FreerTOS 202011.01

- IDT v3.4.0 con suite di test FRQ_1.3.0 per Linux
- IDT v3.4.0 con suite di test FRQ_1.3.0 per macOS
- IDT v3.4.0 con suite di test FRQ_1.3.0 per Windows

Note di rilascio

- Supporta FreerTOS 202011.01. Per ulteriori informazioni su ciò che è incluso nella versione FreerTOS 202011.01, consulta il file ChangeLog.md in. GitHub
- Risolto un bug per cui 'RSA' non era un'opzione di configurazione valida. PKCS11
- Risolto un bug per cui i bucket Amazon S3 non venivano puliti correttamente dopo i test OTA.
- Aggiornamenti per supportare i nuovi casi di test all'interno del gruppo di test FullMQTT.

Versioni della suite di test

- FRQ_1.3.0
  - Rilasciato il 2020.11.05.

IDT v3.3.0 e Test Suite versione 1.2.0 per FreerTOS 202007.00

- IDT v3.3.0 con suite di test FRQ_1.2.0 per Linux
- IDT v3.3.0 con suite di test FRQ_1.2.0 per macOS
- IDT v3.3.0 con suite di test FRQ_1.2.0 per Windows

Note di rilascio

- Supporta FreerTOS 202007.00. Per ulteriori informazioni su ciò che è incluso nella versione FreerTOS 202007.00, consulta il file ChangeLog.md in. GitHub
- Nuovi test end-to-end per convalidare la funzionalità di sospensione e ripresa degli aggiornamenti Over The Air (OTA).
- È stato risolto un bug che impediva agli utenti della regione eu-central-1 di superare la convalida della configurazione per i test OTA.
- Parametro aggiunto al comando`--update-idt`. `run-suite` È possibile utilizzare questa opzione per impostare la risposta per il prompt di aggiornamento IDT.

- `--update-managed-policy`Parametro aggiunto al comando. `run-suite` È possibile utilizzare questa opzione per impostare la risposta per il prompt di aggiornamento delle politiche gestite.
- Miglioramenti interni e correzioni di bug, tra cui:
  - Per gli aggiornamenti automatici della suite di test, miglioramenti all'aggiornamento del file di configurazione.

Versioni della suite di test

- FRQ_1.2.0
  - Rilasciato il 2020.09.17.
  - Miglioramenti interni e correzioni di bug, tra cui:
    - Per OTA, il test Disconnect And Resume ora utilizza correttamente il dataplane MQTT se MQTT è configurato.
    - Per BLE, ora puoi cambiare la password dell'immagine Raspberry Pi.

Per ulteriori informazioni, consulta Politica di supporto AWS IoT Device Tester per FreerTOS.

Versioni IDT non supportate per FreerTOS

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Questa sezione elenca le versioni non supportate di IDT per FreerTOS. Le versioni non supportate non ricevono correzioni di bug o aggiornamenti. Per ulteriori informazioni, consulta Politica di supporto AWS IoT Device Tester per FreerTOS.

Le seguenti versioni di IDT-FreeRTOS non sono più supportate.

IDT v3.0.2 per FreerTOS 202002.00

Note di rilascio

- Supporta FreerTOS 202002.00. Per ulteriori informazioni su ciò che è incluso nella versione FreerTOS 202002.00, consulta il file ChangeLog.md in. GitHub
- Aggiunge l'aggiornamento automatico delle suite di test in IDT. IDT può ora scaricare le ultime suite di test disponibili per la tua versione di FreerTOS. Con questa funzione puoi:
  - Scaricare le suite di test più recenti utilizzando il comando `upgrade-test-suite`.
  - Scaricare le suite di test più recenti impostando un flag all'avvio di IDT.

    Usa l'`-u` *`flag`*opzione dove *`flag`* può essere 'y' per scaricare sempre o 'n' per usare la versione esistente.

    Quando sono disponibili più versioni della suite di test, viene utilizzata la versione più recente a meno che non viene specificato un ID suite di test all'avvio di IDT.
  - Usa la nuova `list-supported-versions` opzione per elencare le versioni di FreerTOS e della suite di test supportate dalla versione installata di IDT.
  - Elencare i casi di test in un gruppo ed eseguire i singoli test.

  Le versioni della suite di test sono definite utilizzando il formato `major.minor.patch` a partire da 1.0.0.
- Aggiunge il `list-supported-products` comando: elenca le versioni di FreerTOS e della suite di test supportate dalla versione installata di IDT.
- Aggiunge `list-test-cases` il comando: elenca i casi di test disponibili in un gruppo di test.
- Aggiunge l'`test-id`opzione per il `run-suite` comando: utilizzate questa opzione per eseguire singoli casi di test in un gruppo di test.

Versioni della suite di test

- FRQ_1.0.1

IDT v1.7.1 per FreerTOS 202002.00

Note di rilascio

- Supporta FreerTOS 202002.00. Per ulteriori informazioni su ciò che è incluso nella versione FreerTOS 202002.00, consulta il file ChangeLog.md in. GitHub

- Supporta il metodo di firma del codice personalizzato per i casi di end-to-end test over-the-air (OTA) in modo da poter utilizzare i propri comandi e script di firma del codice per firmare i payload OTA.
- Aggiunge un controllo preliminare per le porte seriali prima dell'inizio dei test. I test avranno esito negativo rapidamente con una migliore messaggistica degli errori se la porta seriale non è configurata correttamente nel file `device.json`.
- È stata aggiunta una politica AWS gestita `AWSIoTDeviceTesterForFreeRTOSFullAccess` con le autorizzazioni necessarie per l'esecuzione. AWS IoT Device Tester Se le nuove versioni richiedono autorizzazioni aggiuntive, le aggiungiamo a questa policy gestita in modo da non dover aggiornare manualmente le autorizzazioni IAM.
- Il file denominato `AFQ_Report.xml` nella directory dei risultati è ora `FRQ_Report.xml`.

IDT v1.6.2 per FreerTOS 201912.00

Note di rilascio

- Supporta FreerTOS 201912.00.
- Supporta test opzionali per OTA su HTTPS per qualificare le tue schede di sviluppo FreerTOS.
- Supporta l'endpoint AWS IoT ATS durante i test.
- Supporta la funzionalità di informare gli utenti sull'ultima versione IDT prima dell'inizio della suite di test.

IDT v1.5.2 per FreerTOS 201910.00

Note di rilascio

- Supporta la qualificazione dei dispositivi FreerTOS con elemento sicuro (chiave integrata).
- Supporta porte server echo configurabili per gruppi di test Secure Sockets e Wi-Fi.
- Supporta il flag moltiplicatore di timeout per aumentare i timeout, utile quando si risolvono gli errori correlati al timeout.
- Aggiunta correzione di bug per l'analisi dei log.
- Supporta l'endpoint iot ats nei test.

IDT v1.4.1 per FreerTOS 201908.00

Note di rilascio

- È stato aggiunto il supporto per gli aggiornamenti di nuove librerie e test case. PKCS11
- Introdotti codici di errore fruibili. Per ulteriori informazioni, consulta Codici di errore IDT
- È stata aggiornata la policy IAM utilizzata per eseguire IDT.

IDT v1.3.2 per FreerTOS 201906.00 Major

Note di rilascio

- Aggiunto il supporto per il test Bluetooth Low Energy (BLE).
- Miglioramento dell'esperienza utente per i comandi dell'interfaccia a riga di comando (CLI) IDT.
- È stata aggiornata la policy IAM utilizzata per eseguire IDT.

IDT-FreeRTOS v1.2

- Versioni FreerTOS:
  - FreerTOS v1.4.9
  - FreerTOS v1.4.8
- Note di rilascio:
  - Aggiunto il supporto per FreerTOS v1.4.8 e v1.4.9.
  - Aggiunto il supporto per testare i dispositivi FreerTOS con il sistema di build CMAKE.

IDT-FreeRTOS v1.1

- Versioni FreerTOS:
  - FreerTOS v1.4.7

IDT-FreeRTOS v1.0

- Versioni FreerTOS:
  - FreerTOS v1.4.6
  - FreerTOS v1.4.5
  - FreerTOS v1.4.4

- FreerTOS v1.4.3
- FreerTOS v1.4.2

Prerequisiti

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Questa sezione descrive i prerequisiti per testare i microcontrollori con. AWS IoT Device Tester

Scarica FreerTOS

Puoi scaricare una versione di FreerTOS GitHubda con il seguente comando:

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

dove <FREERTOS_RELEASE_VERSION>è una versione di FreeRTOS (ad esempio, 202007.00) corrispondente a una versione IDT elencata in. Versioni supportate di AWS IoT Device Tester per FreerTOS Questo ti assicura di avere il codice sorgente completo, compresi i sottomoduli, e di utilizzare la versione corretta di IDT per la tua versione di FreerTOS e viceversa.

In Windows esiste un limite di lunghezza del percorso di 260 caratteri. La struttura dei percorsi di FreerTOS è profonda a molti livelli, quindi se usi Windows, mantieni i percorsi dei file al di sotto del limite di 260 caratteri. Ad esempio, clona FreerTOS su invece `C:\FreeRTOS` di. `C:\Users\username\programs\projects\myproj\FreeRTOS\`

Qualifica LTS (qualificazione per FreerTOS che utilizza librerie LTS)

- Affinché il tuo microcontrollore sia designato per supportare le versioni di FreeRTOS basate sul supporto a lungo termine (LTS) nel AWS Partner Device Catalog, devi fornire un file manifest.

Per ulteriori informazioni, consulta la Checklist per la qualificazione di FreerTOS nella FreerTOS Qualification Guide.

- Per verificare che il microcontrollore supporti le versioni basate su LTS di FreeRTOS e qualificarlo per l'invio al AWS Partner Device Catalog, è necessario utilizzare (AWS IoT Device Tester IDT) con la suite di test FreerTOS Qualification (FRQ) v1.4.x.
- Il supporto per le versioni basate su LTS di FreerTOS è limitato alla versione 202012.xx di FreerTOS.

Scarica IDT per FreerTOS

Ogni versione di FreerTOS ha una versione corrispondente di IDT per FreerTOS per eseguire test di qualificazione. Scarica la versione appropriata di IDT per FreerTOS da. Versioni supportate di AWS IoT Device Tester per FreerTOS

Estrai IDT for FreerTOS in una posizione del file system in cui disponi dei permessi di lettura e scrittura. Poiché Microsoft Windows ha un limite di caratteri per la lunghezza del percorso, estrai IDT per FreerTOS in una directory principale come o. `C:\` `D:\`

> **Note**
>
> Non è consigliabile che più utenti eseguano IDT da un percorso condiviso, ad esempio una directory NFS o una cartella condivisa di rete Windows. Questo comportamento potrebbe causare arresti anomali o il danneggiamento dei dati. Si consiglia di estrarre il pacchetto IDT in un'unità locale.

Registrati per un Account AWS

Se non ne hai uno Account AWS, completa i seguenti passaggi per crearne uno.

Per iscriverti a un Account AWS

1. Apri la https://portal.aws.amazon.com/billing/registrazione.
2. Segui le istruzioni online.

   Parte della procedura di registrazione prevede la ricezione di una telefonata o di un messaggio di testo e l'immissione di un codice di verifica sulla tastiera del telefono.

Quando ti iscrivi a un Account AWS, Utente root dell'account AWSviene creato un. L'utente root dispone dell'accesso a tutte le risorse e tutti i Servizi AWS nell'account. Come best practice di sicurezza, assegna l'accesso amministrativo a un utente e utilizza solo l'utente root per eseguire attività che richiedono l'accesso di un utente root.

AWS ti invia un'email di conferma dopo il completamento della procedura di registrazione. In qualsiasi momento, puoi visualizzare l'attività corrente del tuo account e gestirlo accedendo a https://aws.amazon.com/e scegliendo Il mio account.

Crea un utente con accesso amministrativo

Dopo esserti registrato Account AWS, proteggi Utente root dell'account AWS AWS IAM Identity Center, abilita e crea un utente amministrativo in modo da non utilizzare l'utente root per le attività quotidiane.

Proteggi i tuoi Utente root dell'account AWS

1. Accedi AWS Management Consolecome proprietario dell'account scegliendo Utente root e inserendo il tuo indirizzo Account AWS email. Nella pagina successiva, inserisci la password.

   Per informazioni sull'accesso utilizzando un utente root, consulta la pagina Signing in as the root user della Guida per l'utente di Accedi ad AWS .
2. Abilita l'autenticazione a più fattori (MFA) per l'utente root.

   Per istruzioni, consulta Abilitare un dispositivo MFA virtuale per l'utente Account AWS root (console) nella Guida per l'utente IAM.

Crea un utente con accesso amministrativo

1. Abilita Centro identità IAM.

   Per istruzioni, consulta Abilitazione di AWS IAM Identity Center nella Guida per l'utente di AWS IAM Identity Center .
2. In IAM Identity Center, assegna l'accesso amministrativo a un utente.

   Per un tutorial sull'utilizzo di IAM Identity Center directory come fonte di identità, consulta Configurare l'accesso utente con l'impostazione predefinita IAM Identity Center directory nella Guida per l'AWS IAM Identity Center utente.

Accesso come utente amministratore

- Per accedere con l'utente IAM Identity Center, utilizza l'URL di accesso che è stato inviato al tuo indirizzo e-mail quando hai creato l'utente IAM Identity Center.

  Per informazioni sull'accesso utilizzando un utente IAM Identity Center, consulta AWS Accedere al portale di accesso nella Guida per l'Accedi ad AWS utente.

Assegna l'accesso a ulteriori utenti

1. In IAM Identity Center, crea un set di autorizzazioni conforme alla best practice dell'applicazione di autorizzazioni con il privilegio minimo.

   Segui le istruzioni riportate nella pagina Creazione di un set di autorizzazioni nella Guida per l'utente di AWS IAM Identity Center .
2. Assegna al gruppo prima gli utenti e poi l'accesso con autenticazione unica (Single Sign-On).

   Per istruzioni, consulta Aggiungere gruppi nella Guida per l'utente di AWS IAM Identity Center .

AWS IoT Device Tester politica gestita

La policy gestita `AWSIoTDeviceTesterForFreeRTOSFullAccess` contiene le seguenti autorizzazioni per consentire l'esecuzione e la raccolta di parametri di Device Tester:

- `iot-device-tester:SupportedVersion`

  Concede il permesso di ottenere l'elenco delle versioni di FreerTOS e delle versioni della suite di test supportate da IDT, in modo che siano disponibili da. AWS CLI
- `iot-device-tester:LatestIdt`

  Concede l'autorizzazione per ottenere la AWS IoT Device Tester versione più recente disponibile per il download.
- `iot-device-tester:CheckVersion`

  Concede l'autorizzazione a verificare la compatibilità di una combinazione di prodotto, suite di test e AWS IoT Device Tester versioni.
- `iot-device-tester:DownloadTestSuite`

Concede l'autorizzazione a AWS IoT Device Tester scaricare suite di test.

- `iot-device-tester:SendMetrics`

  Concede l'autorizzazione a pubblicare i dati delle metriche AWS IoT Device Tester di utilizzo.

(Facoltativo) Installa AWS Command Line Interface

Potresti preferire utilizzare il AWS CLI per eseguire alcune operazioni. Se non lo hai AWS CLI installato, segui le istruzioni in Installare il AWS CLI.

Configuralo AWS CLI per la AWS regione che desideri utilizzare eseguendolo aws configure da una riga di comando. Per informazioni sulle AWS regioni che supportano IDT for FreerTOS,AWS consulta Regioni ed endpoint. Per ulteriori informazioni, aws configure vedere Configurazione rapida con. aws configure

Preparazione dei test per la prima verifica della scheda del microcontrollore

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Puoi usare IDT for FreeRTOS per eseguire il test durante il porting delle interfacce FreeRTOS. Dopo aver effettuato il porting delle interfacce FreerTOS per i driver di dispositivo della scheda, si AWS IoT Device Tester eseguono i test di qualificazione sulla scheda del microcontrollore.

Aggiunta di livelli di porting delle librerie

Per portare FreerTOS sul tuo dispositivo, segui le istruzioni nella FreerTOS Porting Guide.

AWS Configura le tue credenziali

Devi configurare le tue AWS credenziali AWS IoT Device Tester per comunicare con il AWS Cloud. Per ulteriori informazioni, consulta Configurare AWS le credenziali e la regione per lo sviluppo.

AWS Le credenziali valide devono essere specificate nel *devicetester_extract_location*/devicetester_afreertos_*[win|mac|linux]*/configs/config.json file di configurazione.

Crea un pool di dispositivi in IDT per FreerTOS

I dispositivi da testare sono organizzati in pool di dispositivi. Ogni pool di dispositivi è composto da uno o più dispositivi identici. Puoi configurare IDT per FreerTOS per testare un singolo dispositivo in un pool o più dispositivi in un pool. Per accelerare il processo di qualificazione, IDT for FreerTOS può testare dispositivi con le stesse specifiche in parallelo. Lo strumento utilizza un metodo Round Robin per eseguire un gruppo di test differente su ciascun dispositivo di un pool.

È possibile aggiungere uno o più dispositivi a un pool di dispositivi modificando la sezione `devices` del modello `device.json` nella cartella `configs`.

> **Note**
>
> La specifica tecnica e il codice SKU dei dispositivi in uno stesso pool devono essere identici.

Per abilitare le build parallele del codice sorgente per diversi gruppi di test, IDT for FreeRTOS copia il codice sorgente in una cartella dei risultati all'interno della cartella estratta IDT for FreerTOS. Il percorso del codice sorgente nel comando build o flash deve essere referenziato utilizzando la variabile or. `testdata.sourcePath` `sdkPath` IDT for FreerTOS sostituisce questa variabile con un percorso temporaneo del codice sorgente copiato. Per ulteriori informazioni, consultare IDT per variabili FreerTOS.

Di seguito è riportato un esempio di file `device.json` utilizzato per creare un pool di dispositivi con più dispositivi.

```
[
    {
        "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"
            }
          ]
        }
      ]
    }
]
```

I seguenti attributi vengono utilizzati nel file `device.json`:

`id`

Un ID alfanumerico definito dall'utente che identifica in modo univoco un pool di dispositivi. I dispositivi appartenenti a un pool devono essere dello stesso tipo. Durante l'esecuzione di una suite di test, i dispositivi del pool vengono utilizzati per parallelizzare il carico di lavoro.

`sku`

Un valore alfanumerico che identifica in modo univoco la scheda da testare. Il codice SKU viene utilizzato per tenere traccia delle schede qualificate.

> **Note**
>
> Se desideri inserire la tua scheda nel AWS Partner Device Catalog, lo SKU che specifichi qui deve corrispondere allo SKU utilizzato nel processo di pubblicazione.

`features`

Un array che contiene le funzionalità supportate dal dispositivo. AWS IoT Device Tester utilizza queste informazioni per selezionare i test di qualificazione da eseguire.

I valori supportati sono:

`TCP/IP`

Indica se la scheda supporta uno TCP/IP stack e se è supportata su chip (MCU) o trasferita su un altro modulo. TCP/IP è necessario per la qualificazione.

`WIFI`

Indica se la scheda dispone di funzionalità Wi-Fi. Deve essere impostato su `No` se `Cellular` è impostato su. `Yes`

`Cellular`

Indica se la scheda è compatibile con la rete cellulare. Deve essere impostato su `No` se `WIFI` è impostato su`Yes`. Quando questa funzionalità è impostata su`Yes`, il FullSecureSockets test verrà eseguito utilizzando EC2 istanze AWS t2.micro e ciò potrebbe comportare costi aggiuntivi per l'account. Per ulteriori informazioni, consulta i EC2 prezzi di Amazon.

`TLS`

Indica se la scheda supporta TLS. TLS è obbligatorio per la qualifica.

`PKCS11`

Indica l'algoritmo di crittografia a chiave pubblica supportato dalla scheda. PKCS11 è necessario per la qualificazione. I valori supportati sono `ECC`, `RSA`, `Both` e `No`. `Both` indica che la scheda supporta sia gli algoritmi `ECC` e `RSA`.

`KeyProvisioning`

Indica il metodo di scrittura di un certificato client X.509 attendibile sulla scheda. I valori validi sono `Import`, `Onboard` e `No`. Per la qualifica è necessario il provisioning delle chiavi.

- Utilizzare `Import` se la scheda consente l'importazione di chiavi private. IDT creerà una chiave privata e la inserirà nel codice sorgente di FreerTOS.
- Utilizzare `Onboard` se la scheda supporta la generazione di chiavi private integrate (ad esempio, se il dispositivo ha un elemento protetto o se si preferisce generare la coppia di chiavi del dispositivo e il certificato). Assicurarsi di aggiungere un elemento `secureElementConfig` in ciascuna delle sezioni del dispositivo e inserire il percorso assoluto del file della chiave pubblica nel campo `publicKeyAsciiHexFilePath`.
- Utilizzare `No` se la scheda non supporta il provisioning delle chiavi.

`OTA`

Indica se la scheda supporta la funzionalità di over-the-air aggiornamento (OTA). L'attributo `OtaDataPlaneProtocol` indica quale protocollo di piano dei dati OTA supporta il dispositivo.

L'attributo viene ignorato se la caratteristica OTA non è supportata dal dispositivo. Quando è selezionato "Both", il tempo di esecuzione del test OTA viene prolungato a causa dell'esecuzione di MQTT, HTTP e test misti.

`BLE`

Indica se la scheda supporta Bluetooth Low Energy (BLE).

`devices.id`

Un identificativo univoco definito dall'utente del dispositivo sottoposto a test.

`devices.connectivity.protocol`

Il protocollo di comunicazione utilizzato per comunicare con questo dispositivo. Valore supportato: `uart`.

`devices.connectivity.serialPort`

La porta seriale del computer host utilizzato per connettersi ai dispositivi da testare.

`devices.secureElementConfig.PublicKeyAsciiHexFilePath`

Il percorso assoluto del file che contiene la chiave pubblica in byte esadecimali estratta dalla chiave privata integrata.

Formato di esempio:

```
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 la tua chiave pubblica è in formato.der, puoi codificare direttamente in esadecimale la chiave pubblica per generare il file esadecimale.

Comando di esempio per la chiave pubblica.der per generare un file esadecimale:

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

Se la chiave pubblica è in formato.pem, puoi estrarre la parte codificata in base64, decodificarla in formato binario e quindi codificarla in esadecimale per generare il file esadecimale.

Ad esempio, utilizzate questi comandi per generare un file esadecimale per una chiave pubblica.pem:

1. Estrai la parte codificata in base64 della chiave (elimina l'intestazione e il piè di pagina) e memorizzala in un file, ad esempio assegnagli un nome`base64key`, esegui questo comando per convertirla in formato.der:

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

2. Esegui il `xxd` comando per convertirlo in formato esadecimale.

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

`devices.secureElementConfig.SecureElementSerialNumber`

(Facoltativo) Il numero di serie dell'elemento sicuro. Fornisci questo campo quando il numero di serie viene stampato insieme alla chiave pubblica del dispositivo quando esegui il progetto FreerTOS demo/test .

`devices.secureElementConfig.preProvisioned`

(Facoltativo) Imposta su «Sì» se il dispositivo dispone di un elemento sicuro prestabilito con credenziali bloccate, che non può importare, creare o distruggere oggetti. Questa configurazione `features` ha effetto solo se è `KeyProvisioning` impostata su «Onboard», insieme a «ECC». `PKCS11`

`identifiers`

(Facoltativo) Un array di coppie nome/valore arbitrarie. Puoi utilizzare questi valori nei comandi di compilazione e flashing descritti nella sezione successiva.

Configurazione delle impostazioni di compilazione, flashing e test

Affinché IDT for FreerTOS possa creare ed eseguire test flash sulla scheda automaticamente, è necessario configurare IDT per eseguire i comandi build e flash per il proprio hardware. Le impostazioni dei comandi di compilazione e flashing sono configurate nel file di modello `userdata.json` che si trova nella cartella `config`.

Configurazione delle impostazioni per il testing dei dispositivi

Le impostazioni di compilazione, flashing e test vengono eseguite nel file `configs/userdata.json`. Supportiamo la configurazione di Echo Server caricando i certificati e le chiavi del client e del server in. `customPath` Per ulteriori informazioni, consulta Configurazione di un server echo nella FreerTOS Porting Guide. Il seguente esempio JSON mostra come configurare IDT per FreerTOS per testare più dispositivi:

```
{
    "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"
            }
        ]
    }
}
```

Il seguente elenca gli attributi utilizzati in `userdata.json`:

`sourcePath`

Il percorso alla radice del codice sorgente di FreerTOS portato. Per i test paralleli con un SDK, è `sourcePath` possibile impostarlo utilizzando il `{{userData.sdkConfiguration.path}}` segnaposto. Per esempio:

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

`vendorPath`

Il percorso verso il codice FreerTOS specifico del fornitore. Per i test seriali, `vendorPath` può essere impostato come un percorso assoluto. Per esempio:

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

Per test paralleli, `vendorPath` può essere impostato utilizzando il segnaposto `{{testData.sourcePath}}`. Per esempio:

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

La `vendorPath` variabile è necessaria solo quando viene eseguita senza un SDK, altrimenti può essere rimossa.

> **Note**
>
> Quando si eseguono test in parallelo senza un SDK, il `{{testData.sourcePath}}` segnaposto deve essere utilizzato nei campi`vendorPath`,,`buildTool`. `flashTool` Quando si esegue il test con un singolo dispositivo, i percorsi assoluti devono essere utilizzati nei campi `vendorPath`, `buildTool`, `flashTool`. Quando si esegue con un SDK, il `{{sdkPath}}` segnaposto deve essere utilizzato nei comandi, e. `sourcePath` `buildTool` `flashTool`

`sdkConfiguration`

Se stai qualificando FreerTOS con modifiche alla struttura di file e cartelle oltre a quelle richieste per il porting, dovrai configurare le informazioni del tuo SDK in questo blocco. Se non ti qualifichi con un FreerTOS portato all'interno di un SDK, allora dovresti omettere completamente questo blocco.

`sdkConfiguration.name`

Il nome dell'SDK che stai usando con FreerTOS. Se non stai utilizzando un SDK, l'intero `sdkConfiguration` blocco dovrebbe essere omesso.

`sdkConfiguration.version`

La versione dell'SDK che stai usando con FreerTOS. Se non stai utilizzando un SDK, l'intero `sdkConfiguration` blocco dovrebbe essere omesso.

`sdkConfiguration.path`

Il percorso assoluto della directory SDK che contiene il codice FreerTOS. Se non stai usando un SDK, allora l'intero `sdkConfiguration` blocco dovrebbe essere omesso.

`buildTool`

Il percorso completo allo script di build (.bat o.sh) contenente i comandi per creare il codice sorgente. Tutti i riferimenti al percorso del codice sorgente nel comando build devono essere sostituiti dalla AWS IoT Device Tester variabile `{{testdata.sourcePath}}` e i riferimenti al percorso SDK devono essere sostituiti da. `{{sdkPath}}`

`buildImageInfo`

`testsImageName`

Il nome del file prodotto dal comando build durante la creazione dei test dalla *`freertos-source`*`/tests` cartella.

`demosImageName`

Il nome del file prodotto dal comando build durante la creazione dei test dalla *`freertos-source`*`/demos` cartella.

`flashTool`

Il percorso completo allo script di flash (.sh o.bat) contenente i comandi flash per il dispositivo. Tutti i riferimenti al percorso del codice sorgente nel comando flash devono essere sostituiti dalla variabile IDT per FreerTOS `{{testdata.sourcePath}}` e tutti i riferimenti al percorso SDK devono essere sostituiti dalla variabile IDT per FreeRTOS. `{{sdkPath}}`

`clientWifiConfig`

Configurazione del Wi-Fi client. I test della libreria Wi-Fi richiedono la connessione di una scheda MCU a due punti di accesso. I due punti di accesso possono essere uguali. Questo attributo configura le impostazioni di rete Wi-Fi per il primo punto di accesso. Alcuni dei casi di test del Wi-Fi prevedono che il punto di accesso disponga di una certa sicurezza e che non sia aperto. Assicurati che entrambi i punti di accesso si trovino sulla stessa sottorete del computer host su cui è in esecuzione IDT.

`wifi_ssid`

SSID del Wi-Fi.

`wifi_password`

Password del Wi-Fi.

`wifiSecurityType`

Tipo di sicurezza Wi-Fi utilizzato. Uno dei valori:

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

- `eWiFiSecurityWPA3`

> **Note**
>
> Se la scheda non supporta il Wi-Fi, devi comunque includere la sezione `clientWifiConfig` nel file `device.json`, ma puoi omettere i valori per questi attributi.

`testWifiConfig`

Effettua il test della configurazione Wi-Fi. I test della libreria Wi-Fi richiedono la connessione di una scheda MCU a due punti di accesso. I due punti di accesso possono essere uguali. Questo attributo configura l'impostazione del Wi-Fi per il secondo punto di accesso. Alcuni dei casi di test del Wi-Fi prevedono che il punto di accesso disponga di una certa sicurezza e che non sia aperto. Assicurati che entrambi i punti di accesso si trovino sulla stessa sottorete del computer host su cui è in esecuzione IDT.

`wifiSSID`

SSID del Wi-Fi.

`wifiPassword`

Password del Wi-Fi.

`wifiSecurityType`

Tipo di sicurezza Wi-Fi utilizzato. Uno dei valori:

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

> **Note**
>
> Se la scheda non supporta il Wi-Fi, devi comunque includere la sezione `testWifiConfig` nel file `device.json`, ma puoi omettere i valori per questi attributi.

`echoServerCertificateConfiguration`

Il segnaposto configurabile per la generazione di certificati del server echo per i test dei socket sicuri. Questo campo è obbligatorio.

`certificateGenerationMethod`

Speciifica se il certificato del server viene generato automaticamente o fornito manualmente.

`customPath`

Sono «Personalizzati» `certificatePath` e `privateKeyPath` sono obbligatori. `certificateGenerationMethod`

`certificatePath`

Speciifica il percorso del file per il certificato del server.

`privateKeyPath`

Speciifica il percorso del file per la chiave privata.

`eccCurveFormat`

Specifica il formato della curva supportato dalla scheda. Richiesto quando `PKCS11` è impostato su «ecc» in`device.json`. I valori validi sono «P224", «P256", «P384" o «P521".

`echoServerConfiguration`

Le porte configurabili del server echo per i test dei socket sicuri. WiFi Questo campo è facoltativo.

`securePortForSecureSocket`

La porta che viene utilizzata per configurare un server echo con TLS per il test di Secure Sockets. Il valore predefinito è 33333. Verificare che la porta configurata non sia bloccata da un firewall o dalla rete aziendale.

`insecurePortForSecureSocket`

La porta che viene utilizzata per configurare un server echo senza TLS per il test di Secure Sockets. Il valore predefinito utilizzato nel test è 33334. Verificare che la porta configurata non sia bloccata da un firewall o dalla rete aziendale.

`insecurePortForWiFi`

La porta utilizzata per configurare il server echo senza TLS per il test. WiFi Il valore predefinito utilizzato nel test è 33335. Verificare che la porta configurata non sia bloccata da un firewall o dalla rete aziendale.

`otaConfiguration`

La configurazione OTA. [Facoltativo].

`otaFirmwareFilePath`

Il percorso completo dell'immagine OTA creata dopo il build. Ad esempio, `{{testData.sourcePath}}/`*`relative-path/to/ota/image/from/source/root`*.

`deviceFirmwareFileName`

Il percorso completo del file sul dispositivo MCU in cui si trova il firmware OTA. Anche se alcuni dispositivi non utilizzano questo campo, è comunque necessario fornire un valore.

`otaDemoConfigFilePath`

Il percorso completo di `aws_demo_config.h`, disponibile in *`afr-source`*`/vendors/vendor/boards/board/aws_demos/config_files/`. Questi file sono inclusi nel modello di codice di porting fornito da FreerTOS.

`codeSigningConfiguration`

La configurazione della firma del codice.

`signingMethod`

Il metodo di firma del codice. I valori possibili sono `AWS` o `Custom`.

> **Note**
>
> Per le regioni di Pechino e Ningxia, usa. `Custom` AWSla firma del codice non è supportata in queste regioni.

`signerHashingAlgorithm`

L'algoritmo hash supportato sul dispositivo. I valori possibili sono `SHA1` o `SHA256`.

`signerSigningAlgorithm`

L'algoritmo di firma supportato sul dispositivo. I valori possibili sono `RSA` o `ECDSA`.

`signerCertificate`

Il certificato attendibile utilizzato per OTA.

Per il metodo di firma del AWS codice, usa Amazon Resource Name (ARN) per il certificato affidabile caricato su. AWS Certificate Manager

Per il metodo di firma del codice personalizzato, utilizza il percorso assoluto del file del certificato firmatario.

Per ulteriori informazioni sulla creazione di un certificato attendibile, consulta Creazione di un certificato di firma del codice.

`signerCertificateFileName`

Il percorso del certificato di firma del codice sul dispositivo.

`compileSignerCertificate`

Imposta questa `true` opzione se il certificato di verifica della firma del firmatario del codice non è fornito o aggiornato, quindi deve essere compilato nel progetto. AWS IoT Device Tester recupera il certificato affidabile e lo compila in. `aws_codesigner_certifiate.h`

`untrustedSignerCertificateArn`

L'ARN per il certificato di firma del codice caricato su ACM.

`signerPlatform`

L'algoritmo di firma e hashing utilizzato da AWS Code Signer durante la creazione del processo di aggiornamento OTA. Al momento, i valori possibili per questo campo sono `AmazonFreeRTOS-TI-CC3220SF` e `AmazonFreeRTOS-Default`.

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

Se ti occorre `SHA256` | `RSA` o `SHA1` | `ECDSA` per la configurazione, contattaci per ulteriore supporto.

Configura `signCommand` se hai scelto `Custom` per `signingMethod`.

`signCommand`

Il comando utilizzato per eseguire la firma del codice personalizzato. Puoi trovare il modello nella directory `/configs/script_templates`.

Sono necessari due segnaposto `{{inputImageFilePath}}` e `{{outputSignatureFilePath}}` nel comando. `{{inputImageFilePath}}` è il percorso

del file dell'immagine creata da IDT da firmare. `{{outputSignatureFilePath}}` è il percorso del file della firma che verrà generato dallo script.

`otaDemoConfigFilePath`

Il percorso completo di `aws_demo_config.h`, disponibile all'interno di *`afr-source`*`/vendors/vendor/boards/board/aws_demos/config_files/`. Questi file sono inclusi nel modello di codice di porting fornito da FreerTOS.

`cmakeConfiguration`

CMake configurazione [Opzionale]

> **Note**
>
> Per eseguire casi di CMake test, è necessario fornire il nome della scheda, il nome del fornitore e l'opzione `frToolchainPath` o`compilerName`. `cmakeToolchainPathSe` disponi di un percorso personalizzato per accedere alla CMake toolchain, puoi anche fornire il.

`boardName`

Nome della scheda sottoposta a test. Il nome della scheda deve essere uguale al nome della cartella in *`path/to/afr/source/code`*`/vendors/`*`vendor`*`/boards/`*`board`*.

`vendorName`

Nome del fornitore della scheda sottoposta a test. Il nome del fornitore deve essere uguale al nome della cartella in *`path/to/afr/source/code`*`/vendors/`*`vendor`*.

`compilerName`

Nome del compilatore.

`frToolchainPath`

Percorso completo del toolchain del compilatore.

`cmakeToolchainPath`

Il percorso completo verso la toolchain. CMake Questo campo è facoltativo

`freertosFileConfiguration`

La configurazione dei file FreerTOS che IDT modifica prima di eseguire i test.

`required`

Questa sezione specifica i test obbligatori di cui hai spostato i file di configurazione PKCS11, ad esempio TLS e così via.

`configName`

Il nome del test che viene configurato.

`filePath`

Il percorso assoluto dei file di configurazione all'interno del *`freertos`* repository. Utilizzate la `{{testData.sourcePath}}` variabile per definire il percorso.

`optional`

Questa sezione specifica i test opzionali di cui sono stati spostati i file di configurazione, ad esempio OTA e così via. WiFi

`configName`

Il nome del test che viene configurato.

`filePath`

Il percorso assoluto dei file di configurazione all'interno del *`freertos`* repository. Utilizzate la `{{testData.sourcePath}}` variabile per definire il percorso.

> **Note**
>
> Per eseguire i CMake test case, è necessario fornire il nome della scheda, il nome del fornitore e l'opzione `afrToolchainPath` `ocompilerName`. Puoi anche fornire `cmakeToolchainPath` se disponi di un percorso personalizzato per la CMake toolchain.

IDT per variabili FreerTOS

I comandi per creare il codice e eseguire il flashing del dispositivo potrebbero richiedere connettività o altre informazioni sui dispositivi per funzionare correttamente. AWS IoT Device Tester consente di fare riferimento alle informazioni sul dispositivo in flash e di creare comandi utilizzando JsonPath. Utilizzando JsonPath espressioni semplici, è possibile recuperare le informazioni richieste specificate nel `device.json` file.

Variabili di percorso

IDT per FreerTOS definisce le seguenti variabili di percorso che possono essere utilizzate nelle righe di comando e nei file di configurazione:

`{{testData.sourcePath}}`

Si estende al percorso del codice sorgente. Se utilizzi questa variabile, deve essere utilizzata sia nei comandi flash che di compilazione.

`{{sdkPath}}`

Si espande fino al valore del tuo `userData.sdkConfiguration.path` quando viene utilizzato nei comandi build e flash.

`{{device.connectivity.serialPort}}`

Si estende alla porta seriale.

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

Estende al numero seriale del dispositivo.

`{{enableTests}}`

Valore intero che indica se la compilazione è per test (valore 1) o demo (valore 0).

`{{buildImageName}}`

Il nome del file dell'immagine creata dal comando di compilazione.

`{{otaCodeSignerPemFile}}`

File PEM per il firmatario del codice OTA.

Esecuzione dei test Bluetooth Low Energy

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Questa sezione descrive come configurare ed eseguire i test Bluetooth utilizzando AWS IoT Device Tester FreerTOS. I test Bluetooth non sono richiesti per la qualifica core. Se non vuoi testare il tuo dispositivo con il supporto Bluetooth di FreerTOS puoi saltare questa configurazione, assicurati di lasciare la funzionalità BLE in device.json impostata su. No

Prerequisiti

- Segui le istruzioni in Preparazione dei test per la prima verifica della scheda del microcontrollore.
- Un Raspberry Pi 4B o 3B+. Obbligatorio per eseguire l'applicazione companion Raspberry Pi BLE.
- Una scheda microSD e una scheda SD per il software Raspberry Pi.

Configurazione di Raspberry Pi

Per testare le funzionalità BLE del dispositivo in prova (DUT), è necessario disporre di un Raspberry Pi Model 4B o 3B+.

Per configurare il Raspberry Pi per eseguire i test BLE

1. Scarica una delle immagini Yocto personalizzate che contengono il software necessario per eseguire i test.

   - Immagine per Raspberry Pi 4B
   - Immagine per Raspberry Pi 3B+

   > **Note**
   >
   > L'immagine Yocto deve essere usata solo per i test con AWS IoT Device Tester FreerTOS e non per altri scopi.

2. Memorizzare sulla memoria flash l'immagine Yocto sulla scheda SD per Raspberry Pi.

   - Utilizzando uno strumento di scrittura per scheda SD come Etcher, memorizzare sulla memoria flash il file *image-name*`.rpi-sd.img` scaricato sulla scheda SD. Poiché l'immagine del sistema operativo è di grandi dimensioni, l'operazione potrebbe richiedere alcuni minuti. Espelli la scheda SD dal computer e inserisci la scheda microSD nel Raspberry Pi.

3. Configurare il Raspberry Pi.

   a. Per il primo avvio, ti consigliamo di connettere il Raspberry Pi a un monitor, una tastiera e un mouse.

   b. Collegare il Raspberry Pi a una fonte di alimentazione micro USB.

   c. Accedere utilizzando le credenziali predefinite. Per ID utente, immettere **`root`**. Per la password, immettere **`idtafr`**.

   d. Utilizzando una connessione Wi-Fi o Ethernet, connettere il Raspberry Pi alla rete.

      i. Per collegare il Raspberry Pi tramite Wi-Fi, aprire `/etc/wpa_supplicant.conf` sul Raspberry Pi e aggiungere le credenziali Wi-Fi alla configurazione di `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. Eseguire `ifup wlan0` per avviare la connessione Wi-Fi. Potrebbe essere necessario qualche minuto per la connessione alla rete Wi-Fi.

   e. Per una connessione Ethernet, eseguire `ifconfig eth0`. Per una connessione Wi-Fi, eseguire `ifconfig wlan0`. Annotare l'indirizzo IP che viene visualizzato come `inet addr` nell'output del comando. L'indirizzo IP sarà necessario più avanti in questa procedura.

   f. (Facoltativo) I test eseguono i comandi sul Raspberry Pi tramite SSH utilizzando le credenziali predefinite per l'immagine yocto. Per una maggiore sicurezza, si consiglia di impostare l'autenticazione della chiave pubblica per SSH e disabilitare l'SSH basato su password.

      i. Creare un chiave SSH utilizzando il comando `ssh-keygen` OpenSSL. Se hai già una coppia di chiavi SSK sul tuo computer host, è consigliabile crearne una nuova AWS IoT Device Tester per consentire a FreerTOS di accedere al tuo Raspberry Pi.

> **Note**
>
> Windows non viene fornito con un client SSH preinstallato. Per informazioni su come installare un client SSH su Windows, consultare l'argomento che descrive come scaricare il software SSH.

ii. Il comando `ssh-keygen` richiede di specificare un nome e un percorso di archiviazione della coppia di chiavi. Per impostazione predefinita, i file della coppia di chiavi sono `id_rsa` (chiave privata) e `id_rsa.pub` (chiave pubblica). In macOS e Linux, il percorso predefinito di questi file è `~/.ssh/`. In Windows, la posizione predefinita è `C:\Users\`*`user-name`*.

iii. Quando viene richiesta una frase chiave, è sufficiente premere INVIO per continuare.

iv. Per aggiungere la tua chiave SSH al tuo Raspberry Pi in modo che AWS IoT Device Tester FreeRTOS possa accedere al dispositivo, usa il `ssh-copy-id` comando dal tuo computer host. Questo comando aggiunge la chiave pubblica al file `~/.ssh/authorized_keys` sul Raspberry Pi.

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

v. Quando viene richiesta una password, immettere **`idtafr`**. Questa è la password predefinita per l'immagine yocto.

> **Note**
>
> Il comando `ssh-copy-id` presume che la chiave pubblica sia denominata `id_rsa.pub`. Su macOS e Linux, la posizione predefinita è `~/.ssh/`. In Windows, la posizione predefinita è `C:\Users\`*`user-name`*`\.ssh`. Se hai attribuito un nome diverso alla chiave pubblica o la hai archiviata in una posizione diversa, devi specificare il percorso completo della chiave pubblica SSH con l'opzione `-i` di `ssh-copy-id`, ad esempio `ssh-copy-id -i ~/my/path/myKey.pub`. Per ulteriori informazioni sulla creazione di chiavi SSH e sulla copia di chiavi pubbliche, consulta SSH-COPY-ID.

vi. Per verificare che l'autenticazione della chiave pubblica funzioni, eseguire `ssh -i /`*`my/path/myKey`* `root@`*`raspberry-pi-device-ip`*.

Se non viene richiesta una password, l'autenticazione della chiave pubblica è attiva.

vii. Verificare che sia possibile accedere al Raspberry Pi utilizzando una chiave pubblica e quindi disabilitare SSH basato su password.

A. Sul Raspberry Pi, modificare il file `/etc/ssh/sshd_config`.

B. Impostare l'attributo `PasswordAuthentication` su no.

C. Salvare e chiudere il file `sshd_config`.

D. Ricaricare il server SSH eseguendo `/etc/init.d/sshd reload`.

g. Creare un file `resource.json`.

i. Nella directory in cui hai estratto AWS IoT Device Tester, crea un file denominato. `resource.json`

ii. Aggiungi le seguenti informazioni sul tuo Raspberry Pi al file, sostituendole *`rasp-pi-ip-address`* con l'indirizzo IP del tuo 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 non hai scelto di utilizzare l'autenticazione a chiave pubblica per SSH, aggiungi quanto segue alla `connectivity` sezione del file. `resource.json`

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

```
        "credentials": {
            "user": "root",
            "password": "idtafr"
        }
    }
}
```

iv. (Facoltativo) Se è stato scelto di utilizzare l'autenticazione della chiave pubblica per SSH, aggiungere quanto segue alla sezione `connectivity` del file `resource.json`.

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

Configurazione del dispositivo FreerTOS

Nel file `device.json`, imposta la caratteristica `BLE` su `Yes`. Se si inizia da un file `device.json` prima che i test Bluetooth fossero disponibili, è necessario aggiungere la caratteristica per BLE all'array `features`:

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

Esecuzione dei test BLE

Dopo aver abilitato la caratteristica BLE in `device.json`, i test BLE vengono eseguiti quando esegui `devicetester_[linux | mac | win_x86-64] run-suite` senza specificare un ID gruppo.

Se vuoi eseguire i test BLE separatamente, puoi specificare l'ID gruppo per BLE: `devicetester_[linux | mac | win_x86-64] run-suite --userdata path-to-userdata/userdata.json --group-id FullBLE`.

Per le prestazioni più affidabili, posiziona il Raspberry Pi vicino al dispositivo sottoposto a test.

Risoluzione dei problemi relativi ai test BLE

Verifica di aver seguito i passaggi descritti in Preparazione dei test per la prima verifica della scheda del microcontrollore. Se i test diversi da BLE non riescono, il problema probabilmente non è dovuto alla configurazione Bluetooth.

Esecuzione della suite di qualificazione FreerTOS

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Si utilizza l'eseguibile AWS IoT Device Tester for FreeRTOS per interagire con IDT for FreerTOS. Gli esempi della riga di comando seguenti illustrano come eseguire i test di qualifica per un pool di dispositivi (serie di dispositivi identici).

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
```

Esegue una suite di test in un determinato pool di dispositivi. Il file `userdata.json` si deve trovare nella directory *devicetester_extract_location*`/devicetester_afreertos_`*[win|mac|linux]*`/configs/`.

> **Note**
>
> Se stai usando IDT for FreerTOS su Windows, usa le barre (/) per specificare il percorso del file. `userdata.json`

Utilizza il comando seguente per eseguire un gruppo di test specifico:

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

I parametri `suite-id` e `pool-id` sono facoltativi se esegui una singola suite di test su un singolo pool di dispositivi (ovvero nel tuo file `device.json` c'è un solo pool di dispositivi definito).

Utilizza il comando seguente per eseguire un caso di test specifico in un gruppo di test:

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

Puoi utilizzare il comando `list-test-cases` per elencare i casi di test in un gruppo di test.

IDT per le opzioni della riga di comando di FreerTOS

group-id

(Facoltativo) I gruppi di test da eseguire, come elenco con valori separati da virgole. Se non specificato, IDT esegue tutti i gruppi di test nella suite di test.

pool-id

(Facoltativo) Il pool di dispositivi da testare. Questo è necessario se si definiscono più pool di dispositivi in `device.json`. Se hai un solo pool di dispositivi, questa opzione può essere omessa.

suite-id

(Facoltativo) Versione della suite di test da eseguire. Se non specificato, IDT utilizza la versione più recente nella directory dei test del sistema.

> **Note**
>
> A partire da IDT v3.0.0, IDT controlla online le suite di test più recenti. Per ulteriori informazioni, consulta Versioni della suite di test.

test-id

(Facoltativo) I test da eseguire, come elenco con valori separati da virgole. Se specificato, `group-id` deve includere un singolo gruppo.

Example

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

update-idt

(Facoltativo) Se questo parametro non è impostato ed è disponibile una versione IDT più recente, ti verrà richiesto di aggiornare IDT. Se questo parametro è impostato su`Y`, IDT interromperà l'esecuzione del test se rileva che è disponibile una versione più recente. Se questo parametro è impostato su`N`, IDT continuerà l'esecuzione del test.

update-managed-policy

(Facoltativo) Se questo parametro non viene utilizzato e IDT rileva che la tua politica gestita non lo è up-to-date, ti verrà richiesto di aggiornare la politica gestita. Se questo parametro è impostato su`Y`, IDT interromperà l'esecuzione del test se rileva che la politica gestita non lo è. up-to-date Se questo parametro è impostato su`N`, IDT continuerà l'esecuzione del test.

upgrade-test-suite

(Facoltativo) Se non utilizzato ed è disponibile una versione più recente della suite di test, viene richiesto di scaricarla. Per nascondere il prompt, specifica y per scaricare sempre la suite di test più recente o n per utilizzare la suite di test specificata o l'ultima versione nel sistema.

Example

Per scaricare e utilizzare sempre la suite di test più recente, utilizza il seguente comando.

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

Per utilizzare la suite di test più recente nel sistema, utilizza il seguente comando.

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

h

Utilizza l'opzione di aiuto per ulteriori informazioni sulle opzioni `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
```

Il file `userdata.json` deve trovarsi nella directory *devicetester_extract_location*`/devicetester_afreertos_`*[win|mac|linux]*`/configs/`.

> **Note**
>
> Se stai usando IDT for FreerTOS su Windows, usa le barre (/) per specificare il percorso del file. `userdata.json`

Utilizza il comando seguente per eseguire un gruppo di test specifico.

```
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` sono facoltativi se esegui una singola suite di test su un singolo pool di dispositivi (ovvero nel tuo file `device.json` c'è un solo pool di dispositivi definito).

IDT per le opzioni della riga di comando di FreerTOS

group-id

(Facoltativo) Specifica il gruppo di test.

pool-id

Specifica il pool di dispositivi da testare. Se hai un solo pool di dispositivi, questa opzione può essere omessa.

suite-id

(Facoltativo) Specifica la suite di test da eseguire.

Comandi IDT per FreerTOS

Il comando IDT for FreerTOS supporta le seguenti operazioni:

IDT v3.0.0 and later

`help`

Elenca le informazioni sul comando specificato.

`list-groups`

Elenca i gruppi in una determinata suite.

`list-suites`

Elenca le suite disponibili.

`list-supported-products`

Elenca i prodotti e le versioni della suite di test supportati.

`list-supported-versions`

Elenca le versioni di FreerTOS e della suite di test supportate dalla versione IDT corrente.

`list-test-cases`

Elenca i casi di test in un gruppo specificato.

`run-suite`

Esegue una suite di test in un determinato pool di dispositivi.

Utilizza l'opzione `--suite-id` per specificare una versione della suite di test oppure omettila per utilizzare la versione più recente nel sistema.

Utilizza `--test-id` per eseguire un singolo caso di test.

Example

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

Per l'elenco completo delle opzioni, consulta Esecuzione della suite di qualificazione FreerTOS.

> **Note**
>
> A partire da IDT v3.0.0, IDT controlla online le suite di test più recenti. Per ulteriori informazioni, consulta Versioni della suite di test.

IDT v1.7.0 and earlier

`help`

Elenca le informazioni sul comando specificato.

`list-groups`

Elenca i gruppi in una determinata suite.

`list-suites`

Elenca le suite disponibili.

`run-suite`

Esegue una suite di test in un determinato pool di dispositivi.

Test di riqualifica

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Man mano che vengono rilasciate nuove versioni di IDT per i test di qualificazione FreerTOS o quando aggiorni i pacchetti o i driver di dispositivo specifici della scheda, puoi utilizzare IDT per FreerTOS per testare le tue schede microcontrollori. Per le qualifiche successive, assicurati di disporre delle versioni più recenti di FreeRTOS e IDT for FreerTOS ed esegui nuovamente i test di qualificazione.

AWS IoT Device Tester per le versioni della suite di test FreerTOS

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

IDT for FreerTOS organizza i test in suite di test e gruppi di test.

- Una suite di test è l'insieme di gruppi di test utilizzati per verificare che un dispositivo funzioni con versioni particolari di FreerTOS.
- Un gruppo di test è l'insieme di singoli test relativi a una particolare funzionalità, come PKCS, MQTT o OTA.

A partire da IDT v3.0.0, le versioni della suite di test sono definite utilizzando il formato `major.minor.patch` a partire da 1.0.0. Quando scarichi IDT, il pacchetto include la versione più recente della suite di test.

Quando avvii IDT nell'interfaccia a riga di comando, IDT verifica se è disponibile una versione più recente della suite di test. In tal caso, viene richiesto di eseguire l'aggiornamento alla nuova versione. Puoi scegliere di aggiornare o continuare con i test correnti.

> **Note**
>
> IDT supporta le tre versioni più recenti della suite di test per la qualifica. Per ulteriori informazioni, consulta Politica di supporto AWS IoT Device Tester per FreerTOS.

Puoi scaricare le suite di test utilizzando il comando `upgrade-test-suite`. In alternativa, è possibile utilizzare il parametro `-upgrade-test-suite` *`flag`* opzionale all'avvio di IDT, dove *`flag`* può essere 'y' per scaricare sempre la versione più recente o 'n' per utilizzare la versione esistente.

Puoi anche eseguire il `list-supported-versions` comando per elencare le versioni di FreerTOS e della suite di test supportate dalla versione corrente di IDT.

Nuovi test potrebbero introdurre nuove impostazioni di configurazione IDT. Se le impostazioni sono facoltative, IDT ti invia una notifica e continua a eseguire i test. Se le impostazioni sono necessarie, IDT invia una notifica all'utente e interrompe l'esecuzione. Una volta configurate le impostazioni, puoi continuare a eseguire i test.

Informazioni su risultati e log

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Questa sezione descrive come visualizzare e interpretare i log e i report dei risultati di IDT.

Visualizzazione dei risultati

Durante l'esecuzione, IDT scrive gli errori nella console, i file di log e i report di test. Al termine della suite di test di qualifica, IDT scrive un riepilogo dell'esecuzione dei test nella console e genera due report di test. Questi report sono disponibili in *`devicetester-extract-location`*`/results/`*`execution-id`*`/`. Entrambi i report acquisiscono i risultati dall'esecuzione della suite di test di qualifica.

`awsiotdevicetester_report.xml`È il rapporto del test di qualificazione che invii AWS per inserire il tuo dispositivo nel AWS Partner Device Catalog. Il report contiene i seguenti elementi:

- La versione IDT per FreerTOS.
- La versione di FreerTOS che è stata testata.
- Le funzionalità di FreerTOS supportate dal dispositivo in base ai test superati.
- Il codice SKU e il nome del dispositivo specificato nel file `device.json`.
- Le caratteristiche del dispositivo specificato nel file `device.json`.
- Il riepilogo aggregato dei risultati dei casi di test.
- Un'analisi dei risultati dei test case in base alle librerie testate in base alle funzionalità del dispositivo (ad esempio FullWiFi, FullMQTT e così via).
- Se questa qualifica di FreerTOS è per la versione 202012.00 che utilizza le librerie LTS.

`FRQ_Report.xml`Si tratta di un rapporto in formato XML standard. JUnit Puoi integrarlo in CI/CD piattaforme come Jenkins, Bamboo e così via. Il report contiene i seguenti elementi:

- Un riepilogo aggregato dei risultati dei casi di test.

- Un'analisi dei risultati dei casi di test suddivisi per le librerie sottoposte a test in base alle caratteristiche del dispositivo.

Interpretazione dei risultati di IDT per FreerTOS

La sezione dei report in `awsiotdevicetester_report.xml` o `FRQ_Report.xml` elenca i risultati dei test eseguiti.

Il primo tag XML `<testsuites>` contiene il riepilogo complessivo dell'esecuzione dei test. Per esempio:

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

Attributi utilizzati nel tag **`<testsuites>`**

`name`

Il nome della suite di test.

`time`

Il tempo, espresso in secondi, impiegato per eseguire la suite di qualifica.

`tests`

Il numero dei casi di test eseguiti.

`failures`

Il numero dei casi di test eseguiti ma non superati.

`errors`

Il numero di casi di test che IDT for FreerTOS non è riuscito a eseguire.

`disabled`

Questo attributo non è utilizzato e si può ignorare.

Se non ci sono guasti o errori nel test case, il tuo dispositivo soddisfa i requisiti tecnici per eseguire FreerTOS e può interagire con i servizi. AWS IoT Se scegli di inserire il tuo dispositivo nel AWS Partner Device Catalog, puoi utilizzare questo rapporto come prova di idoneità.

In caso di esiti negativi o errori nei casi di test, puoi identificare il caso di test non riuscito esaminando i tag XML `<testsuites>`. I tag XML `<testsuite>` all'interno del tag `<testsuites>` mostrano il riepilogo dei risultati dei casi di test per un gruppo di test.

```
<testsuite name="FullMQTT" package="" tests="16" failures="0" time="76"
disabled="0" errors="0" skipped="0">
```

Il formato è simile al tag `<testsuites>`, ma con un attributo denominato `skipped` che non viene utilizzato e si può ignorare. All'interno di ogni tag XML `<testsuite>` ci sono tag `<testcase>` per ciascuno dei casi di test eseguiti per un gruppo di test. Per esempio:

```
<testcase classname="mcu.Full_MQTT" name="AFQP_MQTT_Connect_HappyCase"
attempts="1"></testcase>
```

Attributi utilizzati nel tag **`<awsproduct>`**

`name`

Il nome del prodotto sottoposto a test.

`version`

La versione del prodotto sottoposto a test.

`sdk`

Se hai eseguito IDT con un SDK, questo blocco contiene il nome e la versione del tuo SDK. Se non hai eseguito IDT con un SDK, questo blocco contiene:

```
<sdk>
    <name>N/A</vame>
    <version>N/A</version>
</sdk>
```

`features`

Le caratteristiche convalidate. Le caratteristiche contrassegnate come `required` sono necessarie per inviare la scheda per la qualifica. Il seguente frammento mostra come questo appare nel file. `awsiotdevicetester_report.xml`

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

Caratteristiche contrassegnate come `optional` non sono necessarie per la qualifica. I seguenti snippet mostrano caratteristiche facoltative.

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

Se non si verificano errori o errori nei test per le funzionalità richieste, il dispositivo soddisfa i requisiti tecnici per eseguire FreerTOS e può interagire con i servizi. AWS IoT Se desideri inserire il tuo dispositivo nel AWS Partner Device Catalog, puoi utilizzare questo rapporto come prova di idoneità.

In caso di esiti negativi o errori nei test, puoi identificare il test non riuscito esaminando i tag XML `<testsuites>`. I tag XML `<testsuite>` all'interno del tag `<testsuites>` mostrano il riepilogo dei risultati dei test per un gruppo di test. Per esempio:

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

Il formato è simile al `<testsuites>` tag, ma ha un `skipped` attributo che non viene utilizzato e può essere ignorato. All'interno di ogni tag XML `<testsuite>` ci sono tag `<testcase>` per ciascuno dei test eseguiti per un gruppo di test. Per esempio:

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

`lts`

Vero se sei idoneo per una versione di FreeRTOS che utilizza librerie LTS, false in caso contrario.

Attributi utilizzati nel tag **`<testcase>`**

`name`

Il nome del caso di test.

`attempts`

Il numero di volte in cui IDT per FreerTOS ha eseguito il test case.

Quando un test non riesce o si verifica un errore, i tag `<failure>` o `<error>` vengono aggiunti al tag `<testcase>` con informazioni per la risoluzione dei problemi. Per esempio:

```
<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>
```

Per ulteriori informazioni, consulta Risoluzione dei problemi.

Visualizzazione dei registri di

Puoi trovare i log che IDT for FreerTOS genera dall'esecuzione del test in. *devicetester-extract-location*`/results/`*execution-id*`/logs` Vengono generate due serie di log:

`test_manager.log`

Contiene i log generati da IDT per FreerTOS (ad esempio, la configurazione relativa ai log e la generazione di report).

test_group_id`__`*test_case_id*`.log` (ad esempio, `FullMQTT__Full_MQTT.log`)

Il file di log per un caso di test, incluso l'output dal dispositivo sottoposto a test. Il file di log viene denominato in base al gruppo di test e al caso di test che è stato eseguito.

Usa IDT per sviluppare ed eseguire le tue suite di test

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

A partire da IDT v4.0.0, IDT per FreerTOS combina una configurazione di configurazione standardizzata e un formato di risultati con un ambiente di suite di test che consente di sviluppare suite di test personalizzate per i dispositivi e il software del dispositivo. Potete aggiungere test personalizzati per la vostra convalida interna o fornirli ai clienti per la verifica dei dispositivi.

Utilizza IDT per sviluppare ed eseguire suite di test personalizzate, come segue:

Per sviluppare suite di test personalizzate

- Crea suite di test con logica di test personalizzata per il dispositivo che desideri testare.
- Fornisci a IDT le tue suite di test personalizzate per i test runner. Includi informazioni sulle configurazioni di impostazioni specifiche per le tue suite di test.

Per eseguire suite di test personalizzate

- Configura il dispositivo che desideri testare.
- Implementa le configurazioni delle impostazioni come richiesto dalle suite di test che desideri utilizzare.
- Usa IDT per eseguire le tue suite di test personalizzate.
- Visualizza i risultati dei test e i registri di esecuzione per i test eseguiti da IDT.

Scarica l'ultima versione di AWS IoT Device Tester per FreerTOS

Scaricate l'ultima versione di IDT ed estraete il software in una posizione del file system in cui disponete delle autorizzazioni di lettura e scrittura.

> **Note**
>
> IDT non supporta l'esecuzione da parte di più utenti da un percorso condiviso, ad esempio una directory NFS o una cartella condivisa di rete Windows. Si consiglia di estrarre il pacchetto IDT in un'unità locale ed eseguire il file binario IDT sulla workstation locale.
> In Windows esiste un limite di lunghezza del percorso di 260 caratteri. Se stai usando Windows, estrai IDT in una directory root come `C:\` o `D:\` per mantenere i percorsi entro il limite di 260 caratteri.

Flusso di lavoro per la creazione della suite

Le suite di test sono composte da tre tipi di file:

- File di configurazione JSON che forniscono a IDT informazioni su come eseguire la suite di test.
- Esegui il test dei file eseguibili utilizzati da IDT per eseguire i test case.
- File aggiuntivi necessari per eseguire i test.

Completa i seguenti passaggi di base per creare test IDT personalizzati:

1. Crea file di configurazione JSON per la tua suite di test.
2. Crea eseguibili di test case che contengano la logica di test per la tua suite di test.
3. Verifica e documenta le informazioni di configurazione richieste ai test runner per eseguire la suite di test.
4. Verifica che IDT sia in grado di eseguire la tua suite di test e produrre i risultati dei test come previsto.

Per creare rapidamente una suite personalizzata di esempio ed eseguirla, segui le istruzioni riportate inTutorial: crea ed esegui la suite di test IDT di esempio.

Per iniziare a creare una suite di test personalizzata in Python, vedi. Tutorial: Sviluppa una semplice suite di test IDT

Tutorial: crea ed esegui la suite di test IDT di esempio

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Il AWS IoT Device Tester download include il codice sorgente per una suite di test di esempio. Puoi completare questo tutorial per creare ed eseguire la suite di test di esempio per capire come utilizzare FreerTOS AWS IoT Device Tester per eseguire suite di test personalizzate. Sebbene questo tutorial utilizzi SSH, è utile imparare a usarlo AWS IoT Device Tester con i dispositivi FreerTOS.

In questo tutorial, completerai i seguenti passaggi:

1. Crea la suite di test di esempio
2. Usa IDT per eseguire la suite di test di esempio

Prerequisiti

Per completare questo tutorial, è necessario quanto segue:

- Requisiti del computer host
  - Versione più recente di AWS IoT Device Tester
  - Python 3.7 o successivo

    Per verificare la versione di Python installata sul tuo computer, esegui il seguente comando:

    ```
    python3 --version
    ```

    In Windows, se l'utilizzo di questo comando restituisce un errore, usalo `python --version` invece. Se il numero di versione restituito è 3.7 o superiore, esegui il comando seguente in un terminale Powershell da impostare `python3` come alias per il comando. `python`

    ```
    Set-Alias -Name "python3" -Value "python"
    ```

    Se non viene restituita alcuna informazione sulla versione o se il numero di versione è inferiore a 3.7, segui le istruzioni in Downloading Python per installare Python 3.7+. Per ulteriori informazioni, consulta la documentazione di Python.
  - urllib3

    Per verificare che `urllib3` sia installato correttamente, esegui il seguente comando:

    ```
    python3 -c 'import urllib3'
    ```

    Se non `urllib3` è installato, esegui il seguente comando per installarlo:

    ```
    python3 -m pip install urllib3
    ```
- Requisiti per il dispositivo
  - Un dispositivo con un sistema operativo Linux e una connessione di rete alla stessa rete del computer host.

    Ti consigliamo di utilizzare un Raspberry Pi con sistema operativo Raspberry Pi. Assicurati di aver configurato SSH sul tuo Raspberry Pi per connetterti in remoto ad esso.

Configura le informazioni sul dispositivo per IDT

Configura le informazioni sul dispositivo per consentire a IDT di eseguire il test. È necessario aggiornare il `device.json` modello che si trova nella *<device-tester-extract-location>*`/configs` cartella con le seguenti informazioni.

```
[
  {
    "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>"
            }
          }
        }
      }
    ]
  }
]
```

Nell'`devices`oggetto, fornite le seguenti informazioni:

`id`

Un identificatore univoco definito dall'utente per il dispositivo.

`connectivity.ip`

L'indirizzo IP del dispositivo.

`connectivity.port`

Facoltativo. Il numero di porta da utilizzare per le connessioni SSH al dispositivo.

`connectivity.auth`

Informazioni di autenticazione per la connessione.

Questa proprietà si applica solo se `connectivity.protocol` è impostata su `ssh`.

`connectivity.auth.method`

Il metodo di autorizzazione utilizzato per accedere a un dispositivo con un determinato protocollo di connettività.

I valori supportati sono:

- `pki`
- `password`

`connectivity.auth.credentials`

Le credenziali utilizzate per l'autenticazione.

`connectivity.auth.credentials.user`

Il nome utente utilizzato per accedere al dispositivo.

`connectivity.auth.credentials.privKeyPath`

Il percorso completo della chiave privata utilizzata per accedere al dispositivo.

Questo valore si applica solo se `connectivity.auth.method` è impostato su `pki`.

`devices.connectivity.auth.credentials.password`

La password utilizzata per accedere al dispositivo.

Questo valore si applica solo se `connectivity.auth.method` è impostato su `password`.

> **Note**
>
> Specificare `privKeyPath` solo se `method` è impostato su `pki`.
> Specificare `password` solo se `method` è impostato su `password`.

Crea la suite di test di esempio

La *<device-tester-extract-location>*/samples/python cartella contiene file di configurazione di esempio, codice sorgente e IDT Client SDK che puoi combinare in una suite di test utilizzando gli script di build forniti. Il seguente albero di directory mostra la posizione di questi file di esempio:

```
<device-tester-extract-location>
### ...
### tests
### samples
#   ### ...
#   ### python
#       ### configuration
#       ### src
#       ### build-scripts
#           ### build.sh
#           ### build.ps1
### sdks
    ### ...
    ### python
        ### idt_client
```

Per creare la suite di test, esegui i seguenti comandi sul tuo computer 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
```

In questo modo viene creata la suite di test di esempio nella IDTSampleSuitePython_1.0.0 cartella all'interno della *<device-tester-extract-location>*/tests cartella. Esamina i file nella IDTSampleSuitePython_1.0.0 cartella per capire come è strutturata la suite di test di esempio e guarda vari esempi di eseguibili per test case e file JSON di configurazione dei test.

Passaggio successivo: usa IDT per eseguire la suite di test di esempio che hai creato.

Usa IDT per eseguire la suite di test di esempio

Per eseguire la suite di test di esempio, esegui i seguenti comandi sul tuo computer host:

```
cd <device-tester-extract-location>/bin
./devicetester_[linux | mac | win_x86-64] run-suite --suite-id IDTSampleSuitePython
```

IDT esegue la suite di test di esempio e trasmette i risultati alla console. Al termine dell'esecuzione del test, vengono visualizzate le seguenti informazioni:

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

Risoluzione dei problemi

Utilizza le seguenti informazioni per risolvere eventuali problemi relativi al completamento del tutorial.

Il test case non viene eseguito correttamente

Se il test non viene eseguito correttamente, IDT trasmette i log degli errori alla console per aiutarti a risolvere i problemi relativi all'esecuzione del test. Assicurati di soddisfare tutti i prerequisiti per questo tutorial.

Impossibile connettersi al dispositivo in prova

Verificare quanto segue:

- Il `device.json` file contiene l'indirizzo IP, la porta e le informazioni di autenticazione corretti.
- Puoi connetterti al tuo dispositivo tramite SSH dal tuo computer host.

Tutorial: Sviluppa una semplice suite di test IDT

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Una suite di test combina quanto segue:

- Eseguibili di test che contengono la logica di test
- File di configurazione JSON che descrivono la suite di test

Questo tutorial mostra come usare IDT per FreerTOS per sviluppare una suite di test Python che contenga un singolo test case. Sebbene questo tutorial utilizzi SSH, è utile imparare a usarlo AWS IoT Device Tester con i dispositivi FreerTOS.

In questo tutorial, completerai i seguenti passaggi:

1. Crea una directory per la suite di test
2. Crea file di configurazione JSON
3. Crea l'eseguibile del test case
4. Esegui la suite di test

Prerequisiti

Per completare questo tutorial, è necessario quanto segue:

- Requisiti del computer host
  - Versione più recente di AWS IoT Device Tester
  - Python 3.7 o successivo

Per verificare la versione di Python installata sul tuo computer, esegui il seguente comando:

```
python3 --version
```

In Windows, se l'utilizzo di questo comando restituisce un errore, usalo `python --version` invece. Se il numero di versione restituito è 3.7 o superiore, esegui il comando seguente in un terminale Powershell da impostare `python3` come alias per il comando. `python`

```
Set-Alias -Name "python3" -Value "python"
```

Se non viene restituita alcuna informazione sulla versione o se il numero di versione è inferiore a 3.7, segui le istruzioni in Downloading Python per installare Python 3.7+. Per ulteriori informazioni, consulta la documentazione di Python.

- urllib3

  Per verificare che `urllib3` sia installato correttamente, esegui il seguente comando:

  ```
  python3 -c 'import urllib3'
  ```

  Se non `urllib3` è installato, esegui il seguente comando per installarlo:

  ```
  python3 -m pip install urllib3
  ```

- Requisiti per il dispositivo
  - Un dispositivo con un sistema operativo Linux e una connessione di rete alla stessa rete del computer host.

    Ti consigliamo di utilizzare un Raspberry Pi con sistema operativo Raspberry Pi. Assicurati di aver configurato SSH sul tuo Raspberry Pi per connetterti in remoto ad esso.

Crea una directory per la suite di test

IDT separa logicamente i casi di test in gruppi di test all'interno di ciascuna suite di test. Ogni test case deve essere all'interno di un gruppo di test. Per questo tutorial, crea una cartella chiamata `MyTestSuite_1.0.0` e crea il seguente albero di directory all'interno di questa cartella:

```
MyTestSuite_1.0.0
```

```
### suite
    ### myTestGroup
        ### myTestCase
```

Crea file di configurazione JSON

La tua suite di test deve contenere i seguenti file di configurazione JSON richiesti:

File JSON richiesti

`suite.json`

Contiene informazioni sulla suite di test. Consultare Configura suite.json.

`group.json`

Contiene informazioni su un gruppo di test. È necessario creare un `group.json` file per ogni gruppo di test nella suite di test. Consultare Configura group.json.

`test.json`

Contiene informazioni su un test case. È necessario creare un `test.json` file per ogni test case nella suite di test. Consultare Configura test.json.

1. Nella `MyTestSuite_1.0.0/suite` cartella, create un `suite.json` file con la seguente struttura:

```
{
    "id": "MyTestSuite_1.0.0",
    "title": "My Test Suite",
    "details": "This is my test suite.",
    "userDataRequired": false
}
```

2. Nella `MyTestSuite_1.0.0/myTestGroup` cartella, create un `group.json` file con la seguente struttura:

```
{
    "id": "MyTestGroup",
    "title": "My Test Group",
    "details": "This is my test group.",
    "optional": false
```

```
}
```

3. Nella `MyTestSuite_1.0.0/myTestGroup/myTestCase` cartella, create un `test.json` file con la seguente struttura:

```
{
    "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"
            ]
        }
    }
}
```

L'albero delle cartelle della `MyTestSuite_1.0.0` cartella dovrebbe ora avere il seguente aspetto:

```
MyTestSuite_1.0.0
### suite
    ### suite.json
    ### myTestGroup
        ### group.json
        ### myTestCase
            ### test.json
```

Scarica l'SDK del client IDT

Utilizzi l'SDK del client IDT per consentire a IDT di interagire con il dispositivo sottoposto a test e di riportare i risultati del test. Per questo tutorial, utilizzerai la versione Python dell'SDK.

Dalla *<device-tester-extract-location>*`/sdks/python/` cartella, copia la `idt_client` cartella nella tua `MyTestSuite_1.0.0/suite/myTestGroup/myTestCase` cartella.

Per verificare che l'SDK sia stato copiato correttamente, esegui il comando seguente.

```
cd MyTestSuite_1.0.0/suite/myTestGroup/myTestCase
python3 -c 'import idt_client'
```

Crea l'eseguibile del test case

Gli eseguibili del test case contengono la logica di test che si desidera eseguire. Una suite di test può contenere più eseguibili di test case. Per questo tutorial, creerai un solo eseguibile di test case.

1. Crea il file della suite di test.

   Nella `MyTestSuite_1.0.0/suite/myTestGroup/myTestCase` cartella, crea un `myTestCase.py` file con il seguente contenuto:

   ```
   from idt_client import *

   def main():
       # Use the client SDK to communicate with IDT
       client = Client()

   if __name__ == "__main__":
       main()
   ```

2. Utilizza le funzioni dell'SDK del client per aggiungere la seguente logica di test al tuo `myTestCase.py` file:

   a. Esegui un comando SSH sul dispositivo sottoposto a test.

      ```
      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. Invia il risultato del test a IDT.

```
from idt_client import *

def main():
    # Use the client SDK to communicate with IDT
    client = Client()

    # Create an execute on device request
    exec_req = ExecuteOnDeviceRequest(ExecuteOnDeviceCommand("echo 'hello
 world'"))

    # Run the command
    exec_resp = client.execute_on_device(exec_req)

    # Print the standard output
    print(exec_resp.stdout)

    # Create a send result request
    sr_req = SendResultRequest(TestResult(passed=True))

    # Send the result
    client.send_result(sr_req)

if __name__ == "__main__":
    main()
```

Configura le informazioni sul dispositivo per IDT

Configura le informazioni sul dispositivo per consentire a IDT di eseguire il test. È necessario aggiornare il `device.json` modello che si trova nella *<device-tester-extract-location>*`/configs` cartella con le seguenti informazioni.

```
[
  {
    "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>"
            }
          }
        }
      }
    ]
  }
]
```

Nell'`devices`oggetto, fornite le seguenti informazioni:

`id`

Un identificatore univoco definito dall'utente per il dispositivo.

`connectivity.ip`

L'indirizzo IP del dispositivo.

`connectivity.port`

Facoltativo. Il numero di porta da utilizzare per le connessioni SSH al dispositivo.

`connectivity.auth`

Informazioni di autenticazione per la connessione.

Questa proprietà si applica solo se `connectivity.protocol` è impostata su `ssh`.

`connectivity.auth.method`

Il metodo di autorizzazione utilizzato per accedere a un dispositivo con un determinato protocollo di connettività.

I valori supportati sono:

- `pki`
- `password`

`connectivity.auth.credentials`

Le credenziali utilizzate per l'autenticazione.

`connectivity.auth.credentials.user`

Il nome utente utilizzato per accedere al dispositivo.

`connectivity.auth.credentials.privKeyPath`

Il percorso completo della chiave privata utilizzata per accedere al dispositivo.

Questo valore si applica solo se `connectivity.auth.method` è impostato su `pki`.

`devices.connectivity.auth.credentials.password`

La password utilizzata per accedere al dispositivo.

Questo valore si applica solo se `connectivity.auth.method` è impostato su `password`.

> **Note**
>
> Specificare `privKeyPath` solo se `method` è impostato su `pki`.
> Specificare `password` solo se `method` è impostato su `password`.

Esegui la suite di test

Dopo aver creato la suite di test, devi assicurarti che funzioni come previsto. Completa i seguenti passaggi per eseguire la suite di test con il pool di dispositivi esistente a tale scopo.

1. Copia la tua `MyTestSuite_1.0.0` cartella in*<device-tester-extract-location>*/`tests`.
2. Esegui i comandi seguenti:

```
cd <device-tester-extract-location>/bin
./devicetester_[linux | mac | win_x86-64] run-suite --suite-id MyTestSuite
```

IDT esegue la tua suite di test e trasmette i risultati alla console. Al termine dell'esecuzione del test, vengono visualizzate le seguenti informazioni:

```
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
```

Risoluzione dei problemi

Utilizza le seguenti informazioni per risolvere eventuali problemi relativi al completamento del tutorial.

Il test case non viene eseguito correttamente

Se il test non viene eseguito correttamente, IDT trasmette i log degli errori alla console per aiutarti a risolvere i problemi relativi all'esecuzione del test. Prima di controllare i log degli errori, verifica quanto segue:

- L'SDK del client IDT si trova nella cartella corretta, come descritto in questo passaggio.
- Soddisfi tutti i prerequisiti per questo tutorial.

Impossibile connettersi al dispositivo in prova

Verificare quanto segue:

- Il `device.json` file contiene l'indirizzo IP, la porta e le informazioni di autenticazione corretti.
- Puoi connetterti al tuo dispositivo tramite SSH dal tuo computer host.

Crea file di configurazione della suite di test IDT

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Questa sezione descrive i formati in cui crei i file di configurazione JSON da includere quando scrivi una suite di test personalizzata.

File JSON richiesti

`suite.json`

Contiene informazioni sulla suite di test. Consultare Configura suite.json.

`group.json`

Contiene informazioni su un gruppo di test. È necessario creare un `group.json` file per ogni gruppo di test nella suite di test. Consultare Configura group.json.

`test.json`

Contiene informazioni su un test case. È necessario creare un `test.json` file per ogni test case nella suite di test. Consultare Configura test.json.

File JSON opzionali

`state_machine.json`

Definisce come vengono eseguiti i test quando IDT esegue la suite di test. Consultare Configura state_machine.json.

`userdata_schema.json`

Definisce lo schema per il `userdata.json`file che i test runner possono includere nella configurazione delle impostazioni. Il `userdata.json` file viene utilizzato per tutte le informazioni di configurazione aggiuntive necessarie per eseguire il test ma che non sono presenti nel `device.json` file. Consultare Configura userdata_schema.json.

I file di configurazione JSON vengono inseriti nel file *<custom-test-suite-folder>* come mostrato qui.

```
<custom-test-suite-folder>
### suite
    ### suite.json
    ### state_machine.json
    ### userdata_schema.json
    ### <test-group-folder>
        ### group.json
        ### <test-case-folder>
            ### test.json
```

Configura suite.json

Il `suite.json` file imposta le variabili di ambiente e determina se i dati utente sono necessari per eseguire la suite di test. Utilizzate il seguente modello per configurare il *<custom-test-suite-folder>*`/suite/suite.json` file:

```
{
    "id": "<suite-name>_<suite-version>",
    "title": "<suite-title>",
    "details": "<suite-details>",
    "userDataRequired": true | false,
    "environmentVariables": [
        {
            "key": "<name>",
            "value": "<value>",
        },
        ...
        {
            "key": "<name>",
            "value": "<value>",
        }
    ]
}
```

Tutti i campi che includono valori sono obbligatori, come descritto di seguito:

`id`

Un ID univoco definito dall'utente per la suite di test. Il valore di `id` deve corrispondere al nome della cartella della suite di test in cui si trova il `suite.json` file. Il nome e la versione della suite devono inoltre soddisfare i seguenti requisiti:

- *<suite-name>*non può contenere caratteri di sottolineatura.
- *<suite-version>*è indicato come*x.x.x*, dove x è un numero.

L'ID viene visualizzato nei rapporti di test generati da IDT.

`title`

Un nome definito dall'utente per il prodotto o la funzionalità testata da questa suite di test. Il nome viene visualizzato nella CLI IDT per i test runner.

`details`

Una breve descrizione dello scopo della suite di test.

`userDataRequired`

Definisce se i test runner devono includere informazioni personalizzate in un `userdata.json` file. Se imposti questo valore su`true`, devi anche includere il `userdata_schema.json`file nella cartella della suite di test.

`environmentVariables`

Facoltativo. Una serie di variabili di ambiente da impostare per questa suite di test.

`environmentVariables.key`

Il nome della variabile di ambiente.

`environmentVariables.value`

Il valore della variabile di ambiente.

Configura group.json

Il `group.json` file definisce se un gruppo di test è obbligatorio o facoltativo. Utilizzate il seguente modello per configurare il *`<custom-test-suite-folder>`*`/suite/`*`<test-group>`*`/` `group.json` file:

```
{
    "id": "<group-id>",
    "title": "<group-title>",
    "details": "<group-details>",
    "optional": true | false,
}
```

Tutti i campi che includono valori sono obbligatori, come descritto di seguito:

`id`

Un ID univoco definito dall'utente per il gruppo di test. Il valore di `id` deve corrispondere al nome della cartella del gruppo di test in cui si trova il `group.json` file. L'ID viene utilizzato nei report di test generati da IDT.

`title`

Un nome descrittivo per il gruppo di test. Il nome viene visualizzato nella CLI IDT per i test runner.

`details`

Una breve descrizione dello scopo del gruppo di test.

`optional`

Facoltativo. Imposta `true` per visualizzare questo gruppo di test come gruppo opzionale dopo che IDT ha terminato l'esecuzione dei test richiesti. Il valore predefinito è `false`.

Configura test.json

Il `test.json` file determina gli eseguibili del test case e le variabili di ambiente utilizzate da un test case. Per ulteriori informazioni sulla creazione di eseguibili per i test case, vedere. Crea eseguibili per i test case IDT

Utilizzate il seguente modello per configurare il *<custom-test-suite-folder>*/suite/*<test-group>*/*<test-case>*/test.json file:

```
{
    "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>",
        }
    ]
}
```

Tutti i campi che includono valori sono obbligatori, come descritto di seguito:

`id`

Un ID univoco definito dall'utente per il test case. Il valore di `id` deve corrispondere al nome della cartella del test case in cui si trova il `test.json` file. L'ID viene utilizzato nei report di test generati da IDT.

`title`

Un nome descrittivo per il test case. Il nome viene visualizzato nella CLI IDT per i test runner.

`details`

Una breve descrizione dello scopo del test case.

`requireDUT`

Facoltativo. Imposta `true` se è necessario un dispositivo per eseguire questo test, altrimenti imposta su`false`. Il valore predefinito è `true`. I test runner configureranno i dispositivi che useranno per eseguire il test nel proprio `device.json` file.

`requiredResources`

Facoltativo. Un array che fornisce informazioni sui dispositivi di risorse necessari per eseguire questo test.

`requiredResources.name`

Il nome univoco da assegnare al dispositivo di risorse durante l'esecuzione di questo test.

`requiredResources.features`

Una serie di funzionalità del dispositivo di risorse definite dall'utente.

`requiredResources.features.name`

Il nome della funzionalità. La funzionalità del dispositivo per cui si desidera utilizzare questo dispositivo. Questo nome viene confrontato con il nome della funzionalità fornito dal test runner nel `resource.json` file.

`requiredResources.features.version`

Facoltativo. La versione della funzionalità. Questo valore viene confrontato con la versione della funzionalità fornita dal test runner nel `resource.json` file. Se non viene fornita una versione, la funzionalità non viene verificata. Se non è richiesto un numero di versione per la funzionalità, lascia vuoto questo campo.

`requiredResources.features.jobSlots`

Facoltativo. Il numero di test simultanei che questa funzionalità può supportare. Il valore predefinito è `1`. Se desideri che IDT utilizzi dispositivi distinti per le singole funzionalità, ti consigliamo di impostare questo valore su. 1

`execution.timeout`

La quantità di tempo (in millisecondi) che IDT attende prima che il test finisca. Per ulteriori informazioni sull'impostazione di questo valore, vedere. Crea eseguibili per i test case IDT

`execution.os`

Gli eseguibili del test case da eseguire in base al sistema operativo del computer host che esegue IDT. I valori supportati sono `linux`, `mac` e `win`.

`execution.os.cmd`

Il percorso dell'eseguibile del test case che si desidera eseguire per il sistema operativo specificato. Questa posizione deve trovarsi nel percorso di sistema.

`execution.os.args`

Facoltativo. Gli argomenti da fornire per eseguire l'eseguibile del test case.

`environmentVariables`

Facoltativo. Una serie di variabili di ambiente impostate per questo test case.

`environmentVariables.key`

Il nome della variabile di ambiente.

`environmentVariables.value`

Il valore della variabile di ambiente.

> **Note**
>
> Se specificate la stessa variabile di ambiente nel `test.json` file e nel `suite.json` file, il valore nel `test.json` file ha la precedenza.

Configura state_machine.json

Una macchina a stati è un costrutto che controlla il flusso di esecuzione della suite di test. Determina lo stato iniziale di una suite di test, gestisce le transizioni di stato in base a regole definite dall'utente e continua la transizione attraverso tali stati fino a raggiungere lo stato finale.

Se la vostra suite di test non include una macchina a stati definita dall'utente, IDT genererà una macchina a stati per voi. La macchina a stati predefinita svolge le seguenti funzioni:

- Fornisce ai test runner la possibilità di selezionare ed eseguire gruppi di test specifici, anziché l'intera suite di test.
- Se non sono selezionati gruppi di test specifici, esegue ogni gruppo di test nella suite di test in ordine casuale.
- Genera report e stampa un riepilogo della console che mostra i risultati dei test per ogni gruppo di test e test case.

Per ulteriori informazioni sul funzionamento della macchina a stati IDT, vedereConfigura la macchina a stati IDT.

Configura userdata_schema.json

Il `userdata_schema.json` file determina lo schema in cui i test runner forniscono i dati degli utenti. I dati utente sono necessari se la suite di test richiede informazioni che non sono presenti nel `device.json` file. Ad esempio, i test potrebbero richiedere credenziali di rete Wi-Fi, porte aperte specifiche o certificati che un utente deve fornire. Queste informazioni possono essere fornite a IDT come parametro di input chiamato`userdata`, il cui valore è un `userdata.json` file, che gli utenti creano nella propria *`<device-tester-extract-location>`*`/config` cartella. Il formato del `userdata.json` file si basa sul `userdata_schema.json` file incluso nella suite di test.

Per indicare che i test runner devono fornire un `userdata.json` file:

1. Nel `suite.json` file, imposta su`userDataRequired`. `true`
2. Nel tuo*`<custom-test-suite-folder>`*, crea un `userdata_schema.json` file.
3. Modifica il `userdata_schema.json` file per creare uno schema JSON IETF Draft v4 valido.

Quando IDT esegue la suite di test, legge automaticamente lo schema e lo usa per convalidare il `userdata.json` file fornito dal test runner. Se valido, il contenuto del `userdata.json` file è disponibile sia nel contesto IDT che nel contesto della macchina a stati.

Configura la macchina a stati IDT

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Una macchina a stati è un costrutto che controlla il flusso di esecuzione della suite di test. Determina lo stato iniziale di una suite di test, gestisce le transizioni di stato in base a regole definite dall'utente e continua la transizione attraverso tali stati fino a raggiungere lo stato finale.

Se la vostra suite di test non include una macchina a stati definita dall'utente, IDT genererà una macchina a stati per voi. La macchina a stati predefinita svolge le seguenti funzioni:

- Fornisce ai test runner la possibilità di selezionare ed eseguire gruppi di test specifici, anziché l'intera suite di test.

- Se non sono selezionati gruppi di test specifici, esegue ogni gruppo di test nella suite di test in ordine casuale.
- Genera report e stampa un riepilogo della console che mostra i risultati dei test per ogni gruppo di test e test case.

La macchina a stati per una suite di test IDT deve soddisfare i seguenti criteri:

- Ogni stato corrisponde a un'azione che IDT deve intraprendere, ad esempio eseguire un gruppo di test o produrre un file di report.
- La transizione a uno stato esegue l'azione associata allo stato.
- Ogni stato definisce la regola di transizione per lo stato successivo.
- Lo stato finale deve essere `Succeed` o `Fail`.

Formato macchina a stati

È possibile utilizzare il seguente modello per configurare il proprio *<custom-test-suite-folder>*`/suite/state_machine.json` file:

```
{
  "Comment": "<description>",
  "StartAt": "<state-name>",
  "States": {
    "<state-name>": {
      "Type": "<state-type>",
      // Additional state configuration
    }

    // Required states
    "Succeed": {
      "Type": "Succeed"
    },
    "Fail": {
      "Type": "Fail"
    }
  }
}
```

Tutti i campi che includono valori sono obbligatori, come descritto di seguito:

`Comment`

Una descrizione della macchina a stati.

`StartAt`

Il nome dello stato in cui IDT inizia a eseguire la suite di test. Il valore di `StartAt` deve essere impostato su uno degli stati elencati nell'`States`oggetto.

`States`

Un oggetto che associa i nomi di stato definiti dall'utente a stati IDT validi. Ogni stato. *`state-name`*l'oggetto contiene la definizione di uno stato valido mappato su. *`state-name`*

L'`States`oggetto deve includere gli `Fail` stati `Succeed` e. Per informazioni sugli stati validi, vedereStati e definizioni di stato validi.

Stati e definizioni di stato validi

Questa sezione descrive le definizioni di stato di tutti gli stati validi che possono essere utilizzati nella macchina a stati IDT. Alcuni dei seguenti stati supportano configurazioni a livello di test case. Tuttavia, si consiglia di configurare le regole di transizione degli stati a livello di gruppo di test anziché a livello di test case, a meno che non sia assolutamente necessario.

Definizioni di stato

- RunTask
- Choice
- Parallel
- AddProductFeatures
- Report
- LogMessage
- SelectGroup
- Fail
- Succeed

RunTask

Lo `RunTask` stato esegue casi di test da un gruppo di test definito nella suite di test.

```
{
    "Type": "RunTask",
    "Next": "<state-name>",
    "TestGroup": "<group-id>",
    "TestCases": [
        "<test-id>"
    ],
    "ResultVar": "<result-name>"
}
```

Tutti i campi che includono valori sono obbligatori, come descritto di seguito:

`Next`

Il nome dello stato a cui passare dopo l'esecuzione delle azioni nello stato corrente.

`TestGroup`

Facoltativo. L'ID del gruppo di test da eseguire. Se questo valore non è specificato, IDT esegue il gruppo di test selezionato dal test runner.

`TestCases`

Facoltativo. Una serie di test case del gruppo IDs specificato in. `TestGroup` In base ai valori di `TestGroup` and`TestCases`, IDT determina il comportamento di esecuzione del test nel modo seguente:

- Quando entrambi `TestGroup` `TestCases` sono specificati, IDT esegue i casi di test specificati dal gruppo di test.
- Quando `TestCases` sono specificati ma non `TestGroup` sono specificati, IDT esegue i casi di test specificati.
- Quando `TestGroup` è specificato, ma non `TestCases` è specificato, IDT esegue tutti i casi di test all'interno del gruppo di test specificato.
- Quando nessuno dei due `TestGroup` `TestCases` è specificato, IDT esegue tutti i casi di test dal gruppo di test selezionato dal test runner dalla CLI IDT. Per abilitare la selezione di gruppo per i test runner, è necessario includere entrambi `RunTask` gli stati nel file. `Choice` `statemachine.json` Per un esempio di come funziona, vedi Example state machine: Esegui gruppi di test selezionati dall'utente.

  Per ulteriori informazioni sull'abilitazione dei comandi CLI IDT per i test runner, vedere. the section called “Abilita i comandi CLI IDT”

`ResultVar`

Il nome della variabile di contesto da impostare con i risultati dell'esecuzione del test. Non specificate questo valore se non avete specificato un valore per`TestGroup`. IDT imposta il valore della variabile definita in `true` o in `false` base `ResultVar` a quanto segue:

- Se il nome della variabile è nel formato*`text`*`_`*`text`*`_passed`, il valore viene impostato in base al fatto che tutti i test del primo gruppo di test siano stati superati o ignorati.
- In tutti gli altri casi, il valore è impostato in base al fatto che tutti i test di tutti i gruppi di test siano stati superati o saltati.

In genere, si utilizza `RunTask` lo stato per specificare un ID del gruppo di test senza specificare il singolo test case IDs, in modo che IDT esegua tutti i casi di test nel gruppo di test specificato. Tutti i casi di test eseguiti da questo stato vengono eseguiti in parallelo, in ordine casuale. Tuttavia, se tutti i casi di test richiedono l'esecuzione di un dispositivo ed è disponibile un solo dispositivo, i test case verranno invece eseguiti in sequenza.

Gestione errori

Se uno dei gruppi di test o dei test case IDs specificati non è valido, questo stato genera l'errore di `RunTaskError` esecuzione. Se lo stato rileva un errore di esecuzione, imposta anche la `hasExecutionError` variabile nel contesto della macchina a stati su. `true`

Choice

Lo `Choice` stato consente di impostare dinamicamente lo stato successivo a cui passare in base a condizioni definite dall'utente.

```
{
    "Type": "Choice",
    "Default": "<state-name>",
    "FallthroughOnError": true | false,
    "Choices": [
        {
            "Expression": "<expression>",
            "Next": "<state-name>"
        }
    ]
}
```

Tutti i campi che includono valori sono obbligatori, come descritto di seguito:

`Default`

Lo stato predefinito a cui passare se nessuna delle espressioni definite in `Choices` può essere valutata. `true`

`FallthroughOnError`

Facoltativo. Speciifica il comportamento quando lo stato rileva un errore nella valutazione delle espressioni. Imposta su `true` se desideri saltare un'espressione se la valutazione genera un errore. Se nessuna espressione corrisponde, la macchina a stati passa allo `Default` stato. Se il `FallthroughOnError` valore non è specificato, il valore predefinito è. `false`

`Choices`

Un array di espressioni e stati per determinare a quale stato passare dopo aver eseguito le azioni nello stato corrente.

`Choices.Expression`

Una stringa di espressione che restituisce un valore booleano. Se l'espressione restituisce a`true`, la macchina a stati passa allo stato definito in. `Choices.Next` Le stringhe di espressione recuperano i valori dal contesto della macchina a stati e quindi eseguono operazioni su di essi per ottenere un valore booleano. Per informazioni sull'accesso al contesto della macchina a stati, vedere. Contesto della macchina a stati

`Choices.Next`

Il nome dello stato a cui passare se l'espressione definita in `Choices.Expression` restituisce. `true`

Gestione errori

Lo `Choice` stato può richiedere la gestione degli errori nei seguenti casi:

- Alcune variabili nelle espressioni di scelta non esistono nel contesto della macchina a stati.
- Il risultato di un'espressione non è un valore booleano.
- Il risultato di una ricerca JSON non è una stringa, un numero o un valore booleano.

Non è possibile utilizzare un `Catch` blocco per gestire gli errori in questo stato. Se si desidera interrompere l'esecuzione della macchina a stati quando rileva un errore, è necessario

impostare su`FallthroughOnError`. `false` Tuttavia, ti consigliamo di impostare `etrue`, `FallthroughOnError` a seconda del caso d'uso, di eseguire una delle seguenti operazioni:

- Se in alcuni casi si prevede che una variabile a cui stai accedendo non esista, utilizza il valore `Default` e i `Choices` blocchi aggiuntivi per specificare lo stato successivo.
- Se una variabile a cui stai accedendo deve sempre esistere, imposta lo `Default` stato su`Fail`.

Parallel

Lo `Parallel` stato consente di definire ed eseguire nuove macchine a stati in parallelo tra loro.

```
{
    "Type": "Parallel",
    "Next": "<state-name>",
    "Branches": [
        <state-machine-definition>
    ]
}
```

Tutti i campi che includono valori sono obbligatori, come descritto di seguito:

`Next`

Il nome dello stato a cui passare dopo l'esecuzione delle azioni nello stato corrente.

`Branches`

Una serie di definizioni di macchine a stati da eseguire. Ogni definizione di macchina a stati deve contenere `StartAt` `Succeed` i propri `Fail` stati. Le definizioni delle macchine a stati in questo array non possono fare riferimento a stati al di fuori della propria definizione.

> **Note**
>
> Poiché ogni macchina a stati della filiale condivide lo stesso contesto della macchina a stati, l'impostazione delle variabili in un ramo e la successiva lettura di tali variabili da un altro ramo potrebbe causare un comportamento imprevisto.

Lo `Parallel` stato passa allo stato successivo solo dopo aver eseguito tutte le macchine a stati della filiale. Ogni stato che richiede un dispositivo aspetterà di funzionare finché il dispositivo non

sarà disponibile. Se sono disponibili più dispositivi, questo stato esegue casi di test da più gruppi in parallelo. Se non sono disponibili dispositivi sufficienti, i test case verranno eseguiti in sequenza. Poiché i casi di test vengono eseguiti in ordine casuale quando vengono eseguiti in parallelo, è possibile utilizzare dispositivi diversi per eseguire i test dello stesso gruppo di test.

Gestione errori

Assicurati che sia la macchina a stati della filiale che la macchina a stati principale passino allo `Fail` stato per gestire gli errori di esecuzione.

Poiché le macchine a stato della filiale non trasmettono errori di esecuzione alla macchina a stato principale, non è possibile utilizzare un `Catch` blocco per gestire gli errori di esecuzione nelle macchine a stato filiale. Utilizzate invece il `hasExecutionErrors` valore nel contesto della macchina a stati condivisa. Per un esempio di come funziona, vediEsempio di macchina a stati: esecuzione di due gruppi di test in parallelo.

AddProductFeatures

Lo `AddProductFeatures` stato consente di aggiungere funzionalità del prodotto al `awsiotdevicetester_report.xml` file generato da IDT.

Una funzionalità del prodotto è costituita da informazioni definite dall'utente su criteri specifici che un dispositivo potrebbe soddisfare. Ad esempio, la funzionalità del `MQTT` prodotto può indicare che il dispositivo pubblica correttamente i messaggi MQTT. Nel rapporto, le caratteristiche del prodotto sono impostate come o come valore personalizzato `supportednot-supported`, in base al superamento dei test specificati.

> **Note**
>
> Lo `AddProductFeatures` stato non genera report da solo. Questo stato deve passare allo `Reportstato` per generare report.

```
{
    "Type": "Parallel",
    "Next": "<state-name>",
    "Features": [
        {
            "Feature": "<feature-name>",
```

```
            "Groups": [
                "<group-id>"
            ],
            "OneOfGroups": [
                "<group-id>"
            ],
            "TestCases": [
                "<test-id>"
            ],
            "IsRequired": true | false,
            "ExecutionMethods": [
                "<execution-method>"
            ]
        }
    ]
}
```

Tutti i campi che includono valori sono obbligatori, come descritto di seguito:

`Next`

Il nome dello stato a cui passare dopo l'esecuzione delle azioni nello stato corrente.

`Features`

Una serie di caratteristiche del prodotto da mostrare nel `awsiotdevicetester_report.xml` file.

`Feature`

Il nome della funzionalità

`FeatureValue`

Facoltativo. Il valore personalizzato da utilizzare nel rapporto anzichésupported. Se questo valore non è specificato, in base ai risultati del test, il valore della funzionalità viene impostato su `supported` o`not-supported`.

Se utilizzate un valore personalizzato per`FeatureValue`, potete testare la stessa funzionalità con condizioni diverse e IDT concatena i valori delle caratteristiche per le condizioni supportate. Ad esempio, il seguente estratto mostra la funzionalità con due valori di `MyFeature` funzionalità separati:

```
...
```

```
{
    "Feature": "MyFeature",
    "FeatureValue": "first-feature-supported",
    "Groups": ["first-feature-group"]
},
{
    "Feature": "MyFeature",
    "FeatureValue": "second-feature-supported",
    "Groups": ["second-feature-group"]
},
...
```

Se entrambi i gruppi di test superano il test, il valore della funzionalità viene impostato su. `first-feature-supported, second-feature-supported`

`Groups`

Facoltativo. Una serie di gruppi di test IDs. Tutti i test all'interno di ogni gruppo di test specificato devono essere superati affinché la funzionalità sia supportata.

`OneOfGroups`

Facoltativo. Una serie di gruppi di test IDs. Tutti i test all'interno di almeno uno dei gruppi di test specificati devono essere superati affinché la funzionalità sia supportata.

`TestCases`

Facoltativo. Una serie di test case IDs. Se si specifica questo valore, si applica quanto segue:

- Tutti i casi di test specificati devono essere superati affinché la funzionalità sia supportata.
- `Groups`deve contenere un solo ID del gruppo di test.
- `OneOfGroups`non deve essere specificato.

`IsRequired`

Facoltativo. Imposta `false` su per contrassegnare questa funzionalità come opzionale nel rapporto. Il valore predefinito è `true`.

`ExecutionMethods`

Facoltativo. Una serie di metodi di esecuzione che corrispondono al `protocol` valore specificato nel `device.json` file. Se viene specificato questo valore, i test runner devono specificare un `protocol` valore che corrisponda a uno dei valori di questo array per includere la funzionalità nel report. Se questo valore non è specificato, la funzionalità verrà sempre inclusa nel rapporto.

Per utilizzare lo `AddProductFeatures` stato, è necessario impostare il valore di `ResultVar` in the `RunTask` state su uno dei seguenti valori:

- Se hai specificato un singolo test case IDs, imposta `ResultVar` su*`group-id_test-id`*`_passed`.
- Se non hai specificato un singolo test case IDs, imposta `ResultVar` su*`group-id`*`_passed`.

Lo `AddProductFeatures` stato verifica i risultati dei test nel modo seguente:

- Se non è stato specificato alcun test case IDs, il risultato per ogni gruppo di test viene determinato dal valore della *`group-id`*`_passed` variabile nel contesto della macchina a stati.
- Se hai specificato il test case IDs, il risultato di ciascuno dei test viene determinato dal valore della *`group-id_test-id`*`_passed` variabile nel contesto della macchina a stati.

Gestione errori

Se un ID di gruppo fornito in questo stato non è un ID di gruppo valido, questo stato genera l'errore di `AddProductFeaturesError` esecuzione. Se lo stato rileva un errore di esecuzione, imposta anche la `hasExecutionErrors` variabile nel contesto della macchina a stati su. `true`

Report

Lo `Report` stato genera i file *`suite-name`*`_Report.xml` and`awsiotdevicetester_report.xml`. Questo stato trasmette inoltre il report alla console.

```
{
    "Type": "Report",
    "Next": "<state-name>"
}
```

Tutti i campi che includono valori sono obbligatori, come descritto di seguito:

`Next`

Il nome dello stato a cui passare dopo l'esecuzione delle azioni nello stato corrente.

Dovresti sempre passare `Report` allo stato verso la fine del flusso di esecuzione del test in modo che i test runner possano visualizzare i risultati del test. In genere, lo stato successivo a questo è`Succeed`.

Gestione errori

Se questo stato riscontra problemi con la generazione dei report, genera l'errore di `ReportError` esecuzione.

LogMessage

Lo `LogMessage` stato genera il `test_manager.log` file e trasmette il messaggio di registro alla console.

```
{
    "Type": "LogMessage",
    "Next": "<state-name>"
    "Level": "info | warn | error"
    "Message": "<message>"
}
```

Tutti i campi che includono valori sono obbligatori, come descritto di seguito:

`Next`

Il nome dello stato a cui passare dopo l'esecuzione delle azioni nello stato corrente.

`Level`

Il livello di errore al quale creare il messaggio di registro. Se si specifica un livello non valido, questo stato genera un messaggio di errore e lo elimina.

`Message`

Il messaggio da registrare.

SelectGroup

Lo `SelectGroup` stato aggiorna il contesto della macchina a stati per indicare quali gruppi sono selezionati. I valori impostati da questo stato vengono utilizzati da tutti `Choice` gli stati successivi.

```
{
    "Type": "SelectGroup",
    "Next": "<state-name>"
    "TestGroups": [
        <group-id>"
```

```
    ]
}
```

Tutti i campi che includono valori sono obbligatori, come descritto di seguito:

`Next`

Il nome dello stato a cui passare dopo l'esecuzione delle azioni nello stato corrente.

`TestGroups`

Una serie di gruppi di test che verranno contrassegnati come selezionati. Per ogni ID del gruppo di test in questo array, la *`group-id`*`_selected` variabile è impostata su `true` nel contesto. Assicuratevi di fornire un gruppo di test valido IDs perché IDT non verifica l'esistenza dei gruppi specificati.

Fail

Lo `Fail` stato indica che la macchina a stati non è stata eseguita correttamente. Si tratta di uno stato finale per la macchina a stati e ogni definizione di macchina a stati deve includere questo stato.

```
{
    "Type": "Fail"
}
```

Succeed

Lo `Succeed` stato indica che la macchina a stati è stata eseguita correttamente. Questo è lo stato finale della macchina a stati e ogni definizione di macchina a stati deve includere questo stato.

```
{
    "Type": "Succeed"
}
```

Contesto della macchina a stati

Il contesto della macchina a stati è un documento JSON di sola lettura che contiene dati disponibili per la macchina a stati durante l'esecuzione. Il contesto della macchina a stati è accessibile solo dalla macchina a stati e contiene informazioni che determinano il flusso di test. Ad esempio, è possibile

utilizzare le informazioni configurate dai test runner nel `userdata.json` file per determinare se è necessario eseguire un test specifico.

Il contesto della macchina a stati utilizza il seguente 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`

Informazioni sul pool di dispositivi selezionato per l'esecuzione del test. Per un pool di dispositivi selezionato, queste informazioni vengono recuperate dal corrispondente elemento dell'array del pool di dispositivi di primo livello definito nel `device.json` file.

`userData`

Informazioni contenute nel file. `userdata.json`

`config`

Le informazioni bloccano il `config.json` file.

`suiteFailed`

Il valore è impostato su `false` quando viene avviata la macchina a stati. Se un gruppo di test fallisce in uno `RunTask` stato, questo valore viene impostato `true` per la durata residua dell'esecuzione della macchina a stati.

`specificTestGroups`

Se il test runner seleziona gruppi di test specifici da eseguire anziché l'intera suite di test, questa chiave viene creata e contiene l'elenco dei gruppi di test specifici. IDs

`specificTestCases`

Se il test runner seleziona casi di test specifici da eseguire anziché l'intera suite di test, questa chiave viene creata e contiene l'elenco di casi di test specifici. IDs

`hasExecutionErrors`

Non esce all'avvio della macchina a stati. Se uno stato rileva errori di esecuzione, questa variabile viene creata e impostata `true` per la durata residua dell'esecuzione della macchina a stati.

È possibile interrogare il contesto utilizzando JSONPath la notazione. La sintassi per le JSONPath interrogazioni nelle definizioni di stato è. `{{$.query}}` È possibile utilizzare JSONPath le query come stringhe segnaposto in alcuni stati. IDT sostituisce le stringhe segnaposto con il valore della query valutata dal contesto. JSONPath È possibile utilizzare i segnaposto per i seguenti valori:

- Il `TestCases` valore in `RunTask` stati.
- Lo `Choice` stato `Expression` del valore.

Quando accedete ai dati dal contesto della macchina a stati, assicuratevi che siano soddisfatte le seguenti condizioni:

- I percorsi JSON devono iniziare con `$.`
- Ogni valore deve restituire una stringa, un numero o un valore booleano.

Per ulteriori informazioni sull'utilizzo della JSONPath notazione per accedere ai dati dal contesto, vedere. Usa il contesto IDT

Errori di esecuzione

Gli errori di esecuzione sono errori nella definizione della macchina a stati che la macchina a stati incontra durante l'esecuzione di uno stato. IDT registra le informazioni su ogni errore nel `test_manager.log` file e trasmette il messaggio di registro alla console.

È possibile utilizzare i seguenti metodi per gestire gli errori di esecuzione:

- Aggiungere un `Catch`blocco nella definizione dello stato.
- Controlla il valore del `hasExecutionErrors`valore nel contesto della macchina a stati.

Cattura

Per utilizzarlo`Catch`, aggiungi quanto segue alla definizione dello stato:

```
"Catch": [
    {
        "ErrorEquals": [
            "<error-type>"
        ]
        "Next": "<state-name>"
    }
]
```

Tutti i campi che includono valori sono obbligatori, come descritto di seguito:

`Catch.ErrorEquals`

Una serie di tipi di errore da catturare. Se un errore di esecuzione corrisponde a uno dei valori specificati, la macchina a stati passa allo stato specificato in`Catch.Next`. Consultate la definizione di ogni stato per informazioni sul tipo di errore che produce.

`Catch.Next`

Lo stato successivo a cui passare se lo stato corrente rileva un errore di esecuzione che corrisponde a uno dei valori specificati in. `Catch.ErrorEquals`

I blocchi Catch vengono gestiti in sequenza fino a quando non ne corrisponde uno. Se gli errori no corrispondono a quelli elencati nei blocchi Catch, le macchine a stati continuano a essere eseguite. Poiché gli errori di esecuzione sono il risultato di definizioni di stato errate, si consiglia di passare allo stato Fail quando uno stato rileva un errore di esecuzione.

hasExecutionError

Quando alcuni stati riscontrano errori di esecuzione, oltre a emettere l'errore, impostano anche il `hasExecutionError` valore su `true` nel contesto della macchina a stati. È possibile utilizzare questo valore per rilevare quando si verifica un errore e quindi utilizzare uno `Choice` stato per trasferire la macchina a stati allo `Fail` stato.

Questo metodo presenta le seguenti caratteristiche.

- La macchina a stati non si avvia con alcun valore assegnato a `hasExecutionError` e questo valore non è disponibile finché non viene impostato da uno stato particolare. Ciò significa che è necessario impostare in modo esplicito il `FallthroughOnError` to `false` per gli `Choice` stati che accedono a questo valore per evitare che la macchina a stati si fermi se non si verificano errori di esecuzione.
- Una volta impostato su`true`, non `hasExecutionError` viene mai impostato su false o rimosso dal contesto. Ciò significa che questo valore è utile solo la prima volta che viene impostato su e per tutti gli stati successivi non fornisce un valore significativo. `true`
- Il `hasExecutionError` valore è condiviso con tutte le macchine a stati delle filiali presenti `Parallel` nello stato, il che può generare risultati imprevisti a seconda dell'ordine in cui viene effettuato l'accesso.

A causa di queste caratteristiche, non è consigliabile utilizzare questo metodo se è possibile utilizzare invece un blocco Catch.

Esempi di macchine a stati

Questa sezione fornisce alcuni esempi di configurazioni di macchine a stati.

Esempi

- Esempio di macchina a stati: Esegui un singolo gruppo di test
- Esempio di macchina a stati: esegue gruppi di test selezionati dall'utente
- Esempio di macchina a stati: esegui un singolo gruppo di test con le caratteristiche del prodotto
- Esempio di macchina a stati: esecuzione di due gruppi di test in parallelo

Esempio di macchina a stati: Esegui un singolo gruppo di test

Questa macchina a stati:

- Esegue il gruppo di test con id`GroupA`, che deve essere presente nella suite in un `group.json` file.
- Verifica la presenza di errori di esecuzione e le `Fail` eventuali transizioni rilevate.
- Genera un rapporto e passa a `Succeed` se non ci sono errori e `Fail` altro.

```
{
    "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"
        }
    }
}
```

Esempio di macchina a stati: esegue gruppi di test selezionati dall'utente

Questa macchina a stati:

- Verifica se il test runner ha selezionato gruppi di test specifici. La macchina a stati non verifica la presenza di casi di test specifici perché i test runner non possono selezionare casi di test senza selezionare anche un gruppo di test.
- Se sono selezionati gruppi di test:
  - Esegue i casi di test all'interno dei gruppi di test selezionati. A tale scopo, la macchina a stati non specifica esplicitamente alcun gruppo di test o caso di test nello `RunTask` stato.
  - Genera un rapporto dopo aver eseguito tutti i test e le uscite.
- Se i gruppi di test non sono selezionati:
  - Esegue i test nel gruppo di test`GroupA`.
  - Genera report ed uscite.

```
{
    "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"
        }
    }
}
```

Esempio di macchina a stati: esegui un singolo gruppo di test con le caratteristiche del prodotto

Questa macchina a stati:

- Esegue il gruppo di test`GroupA`.
- Verifica la presenza di errori di esecuzione e le `Fail` eventuali transizioni rilevate.
- Aggiunge la `FeatureThatDependsOnGroupA` funzionalità al `awsiotdevicetester_report.xml` file:

  - Se ha `GroupA` esito positivo, la funzionalità è impostata su`supported`.
  - La funzionalità non è contrassegnata come opzionale nel rapporto.
- Genera un rapporto e passa a `Succeed` se non ci sono errori e altro `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"
        }
    }
}
```

Esempio di macchina a stati: esecuzione di due gruppi di test in parallelo

Questa macchina a stati:

- Esegue i gruppi `GroupA` e `GroupB` test in parallelo. Le `ResultVar` variabili memorizzate nel contesto dagli `RunTask` stati nelle macchine a stati della filiale di sono disponibili per lo `AddProductFeatures` stato.
- Verifica la presenza di errori di esecuzione e le `Fail` eventuali transizioni rilevate. Questa macchina a stati non utilizza un `Catch` blocco perché tale metodo non rileva errori di esecuzione nelle macchine a stati delle filiali.
- Aggiunge funzionalità al `awsiotdevicetester_report.xml` file in base ai gruppi che passano
  - Se ha `GroupA` esito positivo, la funzionalità è impostata su`supported`.
  - La funzionalità non è contrassegnata come opzionale nel rapporto.
- Genera un rapporto e passa a `Succeed` se non ci sono errori e altro `Fail`

Se due dispositivi sono configurati nel pool di dispositivi, entrambi `GroupA` `GroupB` possono funzionare contemporaneamente. Tuttavia, se uno `GroupA` o `GroupB` più test sono inclusi, entrambi i dispositivi possono essere assegnati a tali test. Se è configurato un solo dispositivo, i gruppi di test verranno eseguiti in sequenza.

```
{
    "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"
        }
    }
}
```

Crea eseguibili per i test case IDT

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

È possibile creare e inserire gli eseguibili dei test case in una cartella di test suite nei seguenti modi:

- Per le suite di test che utilizzano argomenti o variabili di ambiente dei `test.json` file per determinare quali test eseguire, è possibile creare un singolo test case eseguibile per l'intera suite di test o un eseguibile di test per ogni gruppo di test nella suite di test.
- Per una suite di test in cui desideri eseguire test specifici in base a comandi specifici, crei un eseguibile di test case per ogni test case della suite di test.

In qualità di scrittore di test, puoi determinare quale approccio è appropriato per il tuo caso d'uso e strutturare di conseguenza il tuo eseguibile del test case. Assicurati di fornire il percorso eseguibile corretto del test case in ogni `test.json` file e che l'eseguibile specificato funzioni correttamente.

Quando tutti i dispositivi sono pronti per l'esecuzione di un test case, IDT legge i seguenti file:

- Il test case `test.json` per il test case selezionato determina i processi da avviare e le variabili di ambiente da impostare.
- La suite `suite.json` for the test determina le variabili di ambiente da impostare.

IDT avvia il processo eseguibile di test richiesto in base ai comandi e agli argomenti specificati nel `test.json` file e passa le variabili di ambiente richieste al processo.

Usa l'IDT Client SDK

Il client IDT ti SDKs consente di semplificare il modo in cui scrivi la logica di test nel tuo eseguibile di test con comandi API che puoi utilizzare per interagire con IDT e i tuoi dispositivi sottoposti a test. IDT attualmente fornisce quanto segue: SDKs

- SDK del client IDT per Python
- SDK del client IDT per Go

Questi si SDKs trovano nella cartella. *`<device-tester-extract-location>`*`/sdks` Quando crei un nuovo eseguibile del test case, devi copiare l'SDK che desideri utilizzare nella cartella che contiene l'eseguibile del test case e fare riferimento all'SDK nel codice. Questa sezione fornisce una breve descrizione dei comandi API disponibili che puoi utilizzare negli eseguibili del test case.

In questa sezione

- Interazione con il dispositivo
- Interazione IDT
- Interazione con l'host

Interazione con il dispositivo

I seguenti comandi consentono di comunicare con il dispositivo sottoposto a test senza dover implementare ulteriori funzioni di interazione con il dispositivo e di gestione della connettività.

`ExecuteOnDevice`

Consente alle suite di test di eseguire comandi shell su un dispositivo che supporta connessioni shell SSH o Docker.

`CopyToDevice`

Consente alle suite di test di copiare un file locale dalla macchina host che esegue IDT in una posizione specificata su un dispositivo che supporta connessioni shell SSH o Docker.

`ReadFromDevice`

Consente alle suite di test di leggere dalla porta seriale dei dispositivi che supportano le connessioni UART.

> **Note**
>
> Poiché IDT non gestisce le connessioni dirette ai dispositivi effettuate utilizzando le informazioni di accesso ai dispositivi provenienti dal contesto, consigliamo di utilizzare questi comandi API di interazione con i dispositivi negli eseguibili dei test case. Tuttavia, se questi comandi non soddisfano i requisiti del test case, potete recuperare le informazioni di accesso al dispositivo dal contesto IDT e utilizzarle per stabilire una connessione diretta al dispositivo dalla suite di test.
> Per stabilire una connessione diretta, recuperate le informazioni nei campi `device.connectivity` e nei `resource.devices.connectivity` campi per il dispositivo in esame e per i dispositivi di risorse, rispettivamente. Per ulteriori informazioni sull'utilizzo del contesto IDT, vedere. Usa il contesto IDT

Interazione IDT

I seguenti comandi consentono alle suite di test di comunicare con IDT.

`PollForNotifications`

Consente alle suite di test di verificare la presenza di notifiche da IDT.

`GetContextValue` e `GetContextString`

Consente alle suite di test di recuperare valori dal contesto IDT. Per ulteriori informazioni, consulta Usa il contesto IDT.

`SendResult`

Consente alle suite di test di riportare i risultati dei test case a IDT. Questo comando deve essere chiamato alla fine di ogni test case in una suite di test.

Interazione con l'host

Il comando seguente consente alle suite di test di comunicare con la macchina host.

`PollForNotifications`

Consente alle suite di test di verificare la presenza di notifiche da IDT.

`GetContextValue` e `GetContextString`

Consente alle suite di test di recuperare valori dal contesto IDT. Per ulteriori informazioni, consulta Usa il contesto IDT.

`ExecuteOnHost`

Consente alle suite di test di eseguire comandi sulla macchina locale e consente a IDT di gestire il ciclo di vita eseguibile del test case.

Abilita i comandi CLI IDT

Il `run-suite` comando IDT CLI fornisce diverse opzioni che consentono a test runner di personalizzare l'esecuzione del test. Per consentire ai test runner di utilizzare queste opzioni per eseguire la suite di test personalizzata, è necessario implementare il supporto per la CLI IDT. Se non si implementa il supporto, i test runner saranno comunque in grado di eseguire i test, ma alcune opzioni della CLI non funzioneranno correttamente. Per fornire un'esperienza cliente ideale, si consiglia di implementare il supporto per i seguenti argomenti per il `run-suite` comando nella CLI IDT:

`timeout-multiplier`

Speciifica un valore maggiore di 1,0 che verrà applicato a tutti i timeout durante l'esecuzione dei test.

I test runner possono utilizzare questo argomento per aumentare il timeout per i test case che desiderano eseguire. Quando un test runner specifica questo argomento nel proprio `run-suite` comando, IDT lo utilizza per calcolare il valore della variabile di ambiente IDT_TEST_TIMEOUT

e imposta il campo nel contesto IDT. `config.timeoutMultiplier` Per supportare questo argomento, devi fare quanto segue:

- Invece di utilizzare direttamente il valore di timeout del `test.json` file, leggete la variabile di ambiente IDT_TEST_TIMEOUT per ottenere il valore di timeout calcolato correttamente.
- Recuperate il `config.timeoutMultiplier` valore dal contesto IDT e applicatelo a timeout di esecuzione prolungati.

Per ulteriori informazioni sull'uscita anticipata a causa di eventi di timeout, consulta. Specificate il comportamento di uscita

`stop-on-first-failure`

Speciifica che IDT deve interrompere l'esecuzione di tutti i test in caso di errore.

Quando un test runner specifica questo argomento nel proprio `run-suite` comando, IDT interromperà l'esecuzione dei test non appena riscontra un errore. Tuttavia, se i test case vengono eseguiti in parallelo, ciò può portare a risultati imprevisti. Per implementare il supporto, assicuratevi che se IDT rileva questo evento, la logica di test indichi a tutti i casi di test in esecuzione di interrompersi, ripulisca le risorse temporanee e riporti un risultato del test a IDT. Per ulteriori informazioni sull'uscita anticipata in caso di guasto, consulta. Specificate il comportamento di uscita

`group-id` e `test-id`

Speciifica che IDT deve eseguire solo i gruppi di test o i casi di test selezionati.

I test runner possono utilizzare questi argomenti con il loro `run-suite` comando per specificare il seguente comportamento di esecuzione del test:

- Esegui tutti i test all'interno dei gruppi di test specificati.
- Esegui una selezione di test all'interno di un gruppo di test specificato.

Per supportare questi argomenti, la macchina a stati della suite di test deve includere un set specifico `RunTask` di `Choice` stati nella macchina a stati. Se non utilizzate una macchina a stati personalizzata, la macchina a stati IDT predefinita include gli stati richiesti e non è necessario intraprendere ulteriori azioni. Tuttavia, se utilizzate una macchina a stati personalizzata, usatela Esempio di macchina a stati: esegue gruppi di test selezionati dall'utente come esempio per aggiungere gli stati richiesti nella vostra macchina a stati.

Per ulteriori informazioni sui comandi CLI IDT, vedere. Esegui il debug ed esegui suite di test personalizzate

Scrivere registri degli eventi

Durante l'esecuzione del test, invii dati `stdout` e `stderr` scrivi registri degli eventi e messaggi di errore nella console. Per informazioni sul formato dei messaggi della console, vedereFormato dei messaggi della console.

Quando IDT termina l'esecuzione della suite di test, queste informazioni sono disponibili anche nel `test_manager.log` file che si trova nella *<devicetester-extract-location>*`/results/`*<execution-id>*`/logs` cartella.

È possibile configurare ogni test case in modo che scriva i log dell'esecuzione del test, inclusi i log del dispositivo sottoposto a test, nel *<group-id>*`_`*<test-id>* file che si trova nella cartella. *<device-tester-extract-location>*`/results/`*execution-id*`/logs` A tale scopo, recuperate il percorso del file di registro dal contesto IDT con la `testData.logFilePath` query, create un file in quel percorso e scrivete il contenuto desiderato. IDT aggiorna automaticamente il percorso in base al test case in esecuzione. Se scegliete di non creare il file di registro per un test case, non viene generato alcun file per quel test case.

È inoltre possibile configurare il file eseguibile di testo per creare file di registro aggiuntivi, se necessario, nella *<device-tester-extract-location>*`/logs` cartella. Ti consigliamo di specificare prefissi univoci per i nomi dei file di registro in modo che i file non vengano sovrascritti.

Segnala i risultati a IDT

IDT scrive i risultati dei test nei file `awsiotdevicetester_report.xml` e. *suite-name*`_report.xml` Questi file di report si trovano in*<device-tester-extract-location>*`/results/`*<execution-id>*`/`. Entrambi i report acquisiscono i risultati dell'esecuzione della suite di test. Per ulteriori informazioni sugli schemi utilizzati da IDT per questi report, vedere Esamina i risultati e i registri dei test IDT

Per compilare il contenuto del *suite-name*`_report.xml` file, è necessario utilizzare il `SendResult` comando per riportare i risultati dei test a IDT prima del termine dell'esecuzione del test. Se IDT non è in grado di individuare i risultati di un test, genera un errore per il test case. Il seguente estratto di Python mostra i comandi per inviare il risultato di un test a IDT:

```
request-variable = SendResultRequest(TestResult(result))
client.send_result(request-variable)
```

Se non riporti i risultati tramite l'API, IDT cerca i risultati dei test nella cartella test artifacts. Il percorso di questa cartella viene memorizzato nel `testData.testArtifactsPath` file nel contesto IDT. In questa cartella, IDT utilizza il primo file XML in ordine alfabetico che individua come risultato del test.

Se la logica del test produce risultati JUnit XML, è possibile scrivere i risultati del test in un file XML nella cartella artifacts per fornire i risultati direttamente a IDT anziché analizzarli e quindi utilizzare l'API per inviarli a IDT.

Se utilizzate questo metodo, assicuratevi che la logica del test riassuma accuratamente i risultati del test e formatti il file dei risultati nello stesso formato del file. *`suite-name`*`_report.xml` IDT non esegue alcuna convalida dei dati forniti, con le seguenti eccezioni:

- IDT ignora tutte le proprietà del tag. `testsuites` Al contrario, calcola le proprietà del tag in base ai risultati di altri gruppi di test riportati.
- All'interno `testsuites` deve esistere almeno un `testsuite` tag.

Poiché IDT utilizza la stessa cartella Artifacts per tutti i casi di test e non elimina i file dei risultati tra le esecuzioni dei test, questo metodo potrebbe anche portare a segnalazioni errate se IDT legge il file errato. Si consiglia di utilizzare lo stesso nome per il file dei risultati XML generato in tutti i casi di test per sovrascrivere i risultati di ogni test case e assicurarsi che siano disponibili i risultati corretti per l'uso da IDT. Sebbene nella suite di test sia possibile utilizzare un approccio misto alla reportistica, ovvero utilizzare un file di risultati XML per alcuni casi di test e inviare i risultati tramite l'API per altri, non consigliamo questo approccio.

Specificate il comportamento di uscita

Configura il tuo eseguibile testuale in modo che esca sempre con un codice di uscita pari a 0, anche se un test case riporta un errore o un errore. Utilizza codici di uscita diversi da zero solo per indicare che un test case non è stato eseguito o se l'eseguibile del test case non è in grado di comunicare alcun risultato a IDT. Quando IDT riceve un codice di uscita diverso da zero, indica che il test case ha riscontrato un errore che ne ha impedito l'esecuzione.

IDT potrebbe richiedere o aspettarsi che un test case smetta di funzionare prima del termine nei seguenti eventi. Utilizzate queste informazioni per configurare l'eseguibile del test case in modo da rilevare ciascuno di questi eventi dal test case:

Timeout

Si verifica quando un test case viene eseguito per un periodo più lungo del valore di timeout specificato nel `test.json` file. Se il test runner ha utilizzato l'`timeout-multiplier`argomento per specificare un moltiplicatore di timeout, IDT calcola il valore di timeout con il moltiplicatore.

Per rilevare questo evento, utilizzate la variabile di ambiente IDT_TEST_TIMEOUT. Quando un test runner avvia un test, IDT imposta il valore della variabile di ambiente IDT_TEST_TIMEOUT sul valore di timeout calcolato (in secondi) e passa la variabile all'eseguibile del test case. È possibile leggere il valore della variabile per impostare un timer appropriato.

Interrompere

Si verifica quando il test runner interrompe IDT. Ad esempio, premendo. Ctrl+C

Poiché i terminali propagano i segnali a tutti i processi secondari, nei casi di test è sufficiente configurare un gestore di segnali per rilevare i segnali di interruzione.

In alternativa, puoi interrogare periodicamente l'API per verificare il valore del `CancellationRequested` booleano nella risposta dell'API. `PollForNotifications` Quando IDT riceve un segnale di interruzione, imposta il valore del valore booleano su. `CancellationRequested` `true`

Smettila al primo errore

Si verifica quando un test case in esecuzione in parallelo al test case corrente ha esito negativo e il test runner ha utilizzato l'`stop-on-first-failure`argomento per specificare che IDT deve interrompersi in caso di errore.

Per rilevare questo evento, è possibile interrogare periodicamente l'API per verificare il valore del valore `CancellationRequested` booleano nella risposta dell'API. `PollForNotifications` Quando IDT rileva un errore ed è configurato per interrompersi al primo errore, imposta il valore del valore booleano su. `CancellationRequested` `true`

Quando si verifica uno di questi eventi, IDT attende 5 minuti per terminare l'esecuzione di tutti i test case attualmente in esecuzione. Se tutti i test case in esecuzione non si chiudono entro 5 minuti, IDT impone l'interruzione di ciascuno dei relativi processi. Se IDT non ha ricevuto i risultati dei test prima della fine dei processi, contrassegnerà i casi di test come scaduti. Come best practice, dovreste assicurarvi che i test case eseguano le seguenti azioni quando si verificano uno degli eventi:

1. Interrompi l'esecuzione della normale logica di test.

2. Pulisci tutte le risorse temporanee, come gli artefatti di test sul dispositivo sottoposto a test.
3. Segnala un risultato del test a IDT, ad esempio un errore o un fallimento del test.
4. Esci.

Usa il contesto IDT

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Quando IDT esegue una suite di test, la suite di test può accedere a una serie di dati che possono essere utilizzati per determinare l'esecuzione di ciascun test. Questi dati sono chiamati contesto IDT. Ad esempio, la configurazione dei dati utente fornita dai test runner in un `userdata.json` file viene resa disponibile alle suite di test nel contesto IDT.

Il contesto IDT può essere considerato un documento JSON di sola lettura. Le suite di test possono recuperare dati e scrivere dati nel contesto utilizzando tipi di dati JSON standard come oggetti, matrici, numeri e così via.

Schema contestuale

Il contesto IDT utilizza il seguente formato:

```
{
    "config": {
        <config-json-content>
        "timeoutMultiplier": timeout-multiplier
    },
    "device": {
        <device-json-device-element>
    },
    "devicePool": {
        <device-json-pool-element>
    },
    "resource": {
        "devices": [
```

```
            {
                <resource-json-device-element>
                "name": "<resource-name>"
            }
        ]
    },
    "testData": {
        "awsCredentials": {
            "awsAccessKeyId": "<access-key-id>",
            "awsSecretAccessKey": "<secret-access-key>",
            "awsSessionToken": "<session-token>"
        },
        "logFilePath": "/path/to/log/file"
    },
    "userData": {
        <userdata-json-content>
    }
}
```

`config`

Informazioni dal `config.json`file. Il `config` campo contiene anche il seguente campo aggiuntivo:

`config.timeoutMultiplier`

Il moltiplicatore per l'eventuale valore di timeout utilizzato dalla suite di test. Questo valore è specificato dal test runner dalla CLI IDT. Il valore predefinito è 1.

`device`

Informazioni sul dispositivo selezionato per l'esecuzione del test. Queste informazioni sono equivalenti all'elemento dell'`devices`array nel `device.json`file per il dispositivo selezionato.

`devicePool`

Informazioni sul pool di dispositivi selezionato per l'esecuzione del test. Queste informazioni sono equivalenti all'elemento dell'array del pool di dispositivi di primo livello definito nel `device.json` file per il pool di dispositivi selezionato.

`resource`

Informazioni sui dispositivi di risorse contenute nel `resource.json` file.

`resource.devices`

Queste informazioni sono equivalenti all'`devices`array definito nel `resource.json` file. Ogni `devices` elemento include il seguente campo aggiuntivo:

`resource.device.name`

Il nome del dispositivo di risorse. Questo valore è impostato sul `requiredResource.name` valore del `test.json` file.

`testData.awsCredentials`

Le AWS credenziali utilizzate dal test per connettersi al AWS cloud. Queste informazioni sono ottenute dal `config.json` file.

`testData.logFilePath`

Il percorso del file di registro in cui il test case scrive i messaggi di registro. La suite di test crea questo file se non esiste.

`userData`

Informazioni fornite dal test runner nel `userdata.json`file.

Accedere ai dati nel contesto

Puoi interrogare il contesto utilizzando la JSONPath notazione dei tuoi file JSON e del tuo file eseguibile di testo con and. `GetContextValue` `GetContextString` APIs La sintassi con cui le JSONPath stringhe accedono al contesto IDT varia come segue:

- In `suite.json` `etest.json`, si usa. `{{`*`query`*`}}` Cioè, non usate l'elemento root `$.` per iniziare l'espressione.
- In`statemachine.json`, si usa`{{$.`*`query`*`}}`.
- Nei comandi API, si utilizza *`query`* o`{{$.`*`query`*`}}`, a seconda del comando. Per ulteriori informazioni, consultate la documentazione in linea in. SDKs

La tabella seguente descrive gli operatori in una tipica espressione foobar JSONPath:

Operatore	Descrizione
`$`	Lelemento radice. Poiché il valore di contesto di primo livello per IDT è un oggetto, viene in genere utilizzato `$.` per avviare le query.
`.childName`	Accede all'elemento figlio con il nome `childName` di un oggetto. Se applicato a un array, restituisce un nuovo array con questo operatore applicato a ciascun elemento. Il nome dell'elemento distingue tra maiuscole e minuscole. Ad esempio, la query per accedere al `awsRegion` valore nell'`config`oggetto è`$.config.awsRegion` .
`[start:end]`	Filtra gli elementi da una matrice, recuperan do gli elementi che iniziano dall'`start`indice e risalgono all'`end`indice, entrambi inclusi.
`[index1, index2, ... , indexN]`	Filtra gli elementi da un array, recuperando gli elementi solo dagli indici specificati.
`[?(expr)]`	Filtra gli elementi di un array utilizzando l'espressione. `expr` Questa espressione deve restituire un valore booleano.

Per creare espressioni di filtro, utilizzate la seguente sintassi:

```
<jsonpath> | <value> operator <jsonpath> | <value>
```

In questa sintassi:

- `jsonpath`è un programma JSONPath che utilizza la sintassi JSON standard.
- `value`è qualsiasi valore personalizzato che utilizza la sintassi JSON standard.
- `operator`è uno dei seguenti operatori:
  - <(Meno di)

- <=(Minore o uguale a)
- ==(Uguale a)

  Se il valore JSONPath o nell'espressione è un array, un valore booleano o un valore di oggetto, questo è l'unico operatore binario supportato che è possibile utilizzare.
- >=(Maggiore o uguale a)
- >(Maggiore di)
- =~(Corrispondenza delle espressioni regolari). Per utilizzare questo operatore in un'espressione di filtro, il valore JSONPath o sul lato sinistro dell'espressione deve restituire una stringa e il lato destro deve essere un valore di modello che segue la RE2sintassi.

È possibile utilizzare JSONPath le interrogazioni nel formato {{*query*}} come stringhe segnaposto all'interno dei `environmentVariables` campi `args` e nei campi `test.json` dei file. `environmentVariables` `suite.json` IDT esegue una ricerca contestuale e popola i campi con il valore valutato della query. Ad esempio, nel `suite.json` file, è possibile utilizzare stringhe segnaposto per specificare i valori delle variabili di ambiente che cambiano con ogni test case e IDT popolerà le variabili di ambiente con il valore corretto per ogni test case. Tuttavia, quando utilizzate stringhe segnaposto nei `suite.json` file `test.json` e, alle vostre query valgono le seguenti considerazioni:

- È necessario che ogni occorrenza della `devicePool` chiave nella query sia scritta in lettere minuscole. Cioè, usa `devicepool` invece.
- Per gli array, è possibile utilizzare solo matrici di stringhe. Inoltre, gli array utilizzano un formato non standard. `item1, item2,...,itemN` Se l'array contiene solo un elemento, viene serializzato come`item`, rendendolo indistinguibile da un campo di stringa.
- Non è possibile utilizzare segnaposti per recuperare oggetti dal contesto.

Alla luce di queste considerazioni, consigliamo, quando possibile, di utilizzare l'API per accedere al contesto nella logica di test anziché le stringhe segnaposto nei file e nei file. `test.json` `suite.json` Tuttavia, in alcuni casi potrebbe essere più comodo utilizzare i JSONPath segnaposto per recuperare singole stringhe da impostare come variabili di ambiente.

Configura le impostazioni per i test runner

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Per eseguire suite di test personalizzate, i test runner devono configurare le proprie impostazioni in base alla suite di test che desiderano eseguire. Le impostazioni vengono specificate in base ai modelli di file di configurazione JSON presenti nella *<device-tester-extract-location>*`/configs/` cartella. Se necessario, i test runner devono anche impostare AWS le credenziali che IDT utilizzerà per connettersi al cloud. AWS

In qualità di scrittore di test, dovrai configurare questi file per eseguire il debug della tua suite di test. È necessario fornire istruzioni ai test runner in modo che possano configurare le seguenti impostazioni in base alle esigenze per eseguire le suite di test.

Configura dispositivo.json

Il `device.json` file contiene informazioni sui dispositivi su cui vengono eseguiti i test (ad esempio, indirizzo IP, informazioni di accesso, sistema operativo e architettura della CPU).

I test runner possono fornire queste informazioni utilizzando il seguente `device.json` file modello che si trova nella *<device-tester-extract-location>*`/configs/` cartella.

```
[
    {
        "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>",
                }
            }
        ]
    }
]
```

Tutti i campi che includono valori sono obbligatori, come descritto di seguito:

`id`

Un ID alfanumerico definito dall'utente che identifica in modo univoco una raccolta di dispositivi denominata un pool di dispositivi. I dispositivi che appartengono a un pool devono avere lo stesso hardware. Durante l'esecuzione di una suite di test, i dispositivi del pool vengono utilizzati per parallelizzare il carico di lavoro. Più dispositivi vengono utilizzati per eseguire diversi test.

`sku`

Un valore alfanumerico che identifica in modo univoco il dispositivo sottoposto a test. Lo SKU viene utilizzato per tracciare i dispositivi qualificati.

> **Note**
>
> Se desideri inserire la tua scheda nel AWS Partner Device Catalog, lo SKU che specifichi qui deve corrispondere allo SKU utilizzato nel processo di pubblicazione delle offerte.

`features`

Facoltativo. Un array contenente le caratteristiche supportate del dispositivo. Le funzionalità del dispositivo sono valori definiti dall'utente che configuri nella tua suite di test. È necessario fornire ai test runner informazioni sui nomi e sui valori delle funzionalità da includere nel `device.json` file. Ad esempio, se desiderate testare un dispositivo che funge da server MQTT per altri dispositivi, potete configurare la logica di test per convalidare livelli supportati specifici per una funzionalità denominata. `MQTT_QOS` I test runner forniscono questo nome di funzionalità e impostano il valore della funzionalità sui livelli QOS supportati dal proprio dispositivo. È possibile recuperare le informazioni fornite dal contesto IDT con la `devicePool.features` query o dal contesto della macchina a stati con la query. `pool.features`

`features.name`

Il nome della funzionalità.

`features.value`

I valori delle funzionalità supportate.

`features.configs`

Impostazioni di configurazione, se necessarie, per la funzionalità.

`features.config.name`

Il nome dell'impostazione di configurazione.

`features.config.value`

I valori di impostazione supportati.

`devices`

Una serie di dispositivi nel pool da testare. È richiesto almeno un dispositivo.

`devices.id`

Un identificativo univoco definito dall'utente del dispositivo sottoposto a test.

`connectivity.protocol`

Il protocollo di comunicazione utilizzato per comunicare con questo dispositivo. Ogni dispositivo in un pool deve utilizzare lo stesso protocollo.

Attualmente, gli unici valori supportati sono `ssh` e `uart` per i dispositivi fisici e `docker` per i contenitori Docker.

`connectivity.ip`

L'indirizzo IP del dispositivo sottoposto a test.

Questa proprietà si applica solo se `connectivity.protocol` è impostata su `ssh`.

`connectivity.port`

Facoltativo. Il numero di porta da utilizzare per le connessioni SSH.

Il valore predefinito è 22.

Questa proprietà si applica solo se `connectivity.protocol` è impostata su `ssh`.

`connectivity.auth`

Informazioni di autenticazione per la connessione.

Questa proprietà si applica solo se `connectivity.protocol` è impostata su `ssh`.

`connectivity.auth.method`

Il metodo di autorizzazione utilizzato per accedere a un dispositivo con un determinato protocollo di connettività.

I valori supportati sono:

- `pki`
- `password`

`connectivity.auth.credentials`

Le credenziali utilizzate per l'autenticazione.

`connectivity.auth.credentials.password`

La password utilizzata per l'accesso al dispositivo da testare.

Questo valore si applica solo se `connectivity.auth.method` è impostato su `password`.

`connectivity.auth.credentials.privKeyPath`

Il percorso completo alla chiave privata utilizzata per accedere al dispositivo sottoposto a test.

Questo valore si applica solo se `connectivity.auth.method` è impostato su `pki`.

`connectivity.auth.credentials.user`

Il nome utente per l'accesso al dispositivo sottoposto a test.

`connectivity.serialPort`

Facoltativo. La porta seriale a cui è collegato il dispositivo.

Questa proprietà si applica solo se `connectivity.protocol` è impostata su `uart`.

`connectivity.containerId`

L'ID contenitore o il nome del contenitore Docker in fase di test.

Questa proprietà si applica solo se `connectivity.protocol` è impostata su `docker`.

`connectivity.containerUser`

Facoltativo. Il nome da utente a utente all'interno del contenitore. Il valore predefinito è l'utente fornito nel Dockerfile.

Il valore predefinito è 22.

Questa proprietà si applica solo se `connectivity.protocol` è impostata su `docker`.

> **Note**
>
> Per verificare se i test runner configurano la connessione errata del dispositivo per un test, è possibile recuperarlo `pool.Devices[0].Connectivity.Protocol` dal contesto della macchina a stati e confrontarlo con il valore previsto in uno `Choice` stato. Se viene utilizzato un protocollo errato, stampa un messaggio utilizzando lo `LogMessage` stato e passa allo `Fail` stato.

In alternativa, è possibile utilizzare il codice di gestione degli errori per segnalare un errore di test per tipi di dispositivi errati.

(Facoltativo) Configura userdata.json

Il `userdata.json` file contiene tutte le informazioni aggiuntive richieste da una suite di test ma non specificate nel file. `device.json` Il formato di questo file dipende dal `userdata_scheme.json`file definito nella suite di test. Se sei uno scrittore di test, assicurati di fornire queste informazioni agli utenti che eseguiranno le suite di test che scrivi.

(Facoltativo) Configura resource.json

Il `resource.json` file contiene informazioni su tutti i dispositivi che verranno utilizzati come dispositivi di risorse. I dispositivi di risorse sono dispositivi necessari per testare determinate funzionalità di un dispositivo sottoposto a test. Ad esempio, per testare la funzionalità Bluetooth di un dispositivo, è possibile utilizzare un dispositivo di risorse per verificare che il dispositivo sia in grado di connettersi correttamente. I dispositivi di risorse sono opzionali e puoi richiedere tutti i dispositivi di risorse di cui hai bisogno. In qualità di autore del test, utilizzi il file test.json per definire le funzionalità dei dispositivi di risorse necessarie per un test. I test runner utilizzano quindi il `resource.json` file per fornire un pool di dispositivi di risorse dotati delle funzionalità richieste. Assicurati di fornire queste informazioni agli utenti che eseguiranno le suite di test che scrivi.

I test runner possono fornire queste informazioni utilizzando il seguente `resource.json` file modello che si trova nella *`<device-tester-extract-location>`*`/configs/` cartella.

```
[
    {
        "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>",
                }
            }
        ]
    }
]
```

Tutti i campi che includono valori sono obbligatori, come descritto di seguito:

`id`

Un ID alfanumerico definito dall'utente che identifica in modo univoco una raccolta di dispositivi denominata un pool di dispositivi. I dispositivi che appartengono a un pool devono avere lo stesso hardware. Durante l'esecuzione di una suite di test, i dispositivi del pool vengono utilizzati per parallelizzare il carico di lavoro. Più dispositivi vengono utilizzati per eseguire diversi test.

`features`

Facoltativo. Un array contenente le caratteristiche supportate del dispositivo. Le informazioni richieste in questo campo sono definite nei file test.json nella suite di test e determinano quali test eseguire e come eseguirli. Se la suite di test non richiede alcuna funzionalità, questo campo non è obbligatorio.

`features.name`

Il nome della funzionalità.

`features.version`

La versione della funzionalità.

`features.jobSlots`

Impostazione per indicare quanti test possono utilizzare il dispositivo contemporaneamente. Il valore predefinito è 1.

`devices`

Una serie di dispositivi nel pool da testare. È richiesto almeno un dispositivo.

`devices.id`

Un identificativo univoco definito dall'utente del dispositivo sottoposto a test.

`connectivity.protocol`

Il protocollo di comunicazione utilizzato per comunicare con questo dispositivo. Ogni dispositivo in un pool deve utilizzare lo stesso protocollo.

Attualmente, gli unici valori supportati sono `ssh` e `uart` per i dispositivi fisici e `docker` per i contenitori Docker.

`connectivity.ip`

L'indirizzo IP del dispositivo sottoposto a test.

Questa proprietà si applica solo se `connectivity.protocol` è impostata su `ssh`.

`connectivity.port`

Facoltativo. Il numero di porta da utilizzare per le connessioni SSH.

Il valore predefinito è 22.

Questa proprietà si applica solo se `connectivity.protocol` è impostata su `ssh`.

`connectivity.auth`

Informazioni di autenticazione per la connessione.

Questa proprietà si applica solo se `connectivity.protocol` è impostata su `ssh`.

`connectivity.auth.method`

Il metodo di autorizzazione utilizzato per accedere a un dispositivo con un determinato protocollo di connettività.

I valori supportati sono:

- `pki`
- `password`

`connectivity.auth.credentials`

Le credenziali utilizzate per l'autenticazione.

`connectivity.auth.credentials.password`

La password utilizzata per l'accesso al dispositivo da testare.

Questo valore si applica solo se `connectivity.auth.method` è impostato su `password`.

`connectivity.auth.credentials.privKeyPath`

Il percorso completo alla chiave privata utilizzata per accedere al dispositivo sottoposto a test.

Questo valore si applica solo se `connectivity.auth.method` è impostato su `pki`.

`connectivity.auth.credentials.user`

Il nome utente per l'accesso al dispositivo sottoposto a test.

`connectivity.serialPort`

Facoltativo. La porta seriale a cui è collegato il dispositivo.

Questa proprietà si applica solo se `connectivity.protocol` è impostata su `uart`.

`connectivity.containerId`

L'ID contenitore o il nome del contenitore Docker in fase di test.

Questa proprietà si applica solo se `connectivity.protocol` è impostata su `docker`.

`connectivity.containerUser`

Facoltativo. Il nome da utente a utente all'interno del contenitore. Il valore predefinito è l'utente fornito nel Dockerfile.

Il valore predefinito è 22.

Questa proprietà si applica solo se `connectivity.protocol` è impostata su `docker`.

(Facoltativo) Configura config.json

Il `config.json` file contiene informazioni di configurazione per IDT. In genere, i test runner non avranno bisogno di modificare questo file se non per fornire le proprie credenziali AWS utente per IDT e, facoltativamente, una regione. AWS Se vengono fornite AWS le credenziali con le autorizzazioni richieste, AWS IoT Device Tester raccoglie e invia le metriche di utilizzo a. AWS Si tratta di una funzionalità opzionale e viene utilizzata per migliorare la funzionalità IDT. Per ulteriori informazioni, consulta Metriche di utilizzo IDT.

I test runner possono configurare le proprie AWS credenziali in uno dei seguenti modi:

- File di credenziali

  IDT usa lo stesso file delle credenziali di AWS CLI. Per ulteriori informazioni, consulta l'argomento relativo ai file di configurazione e delle credenziali.

  La posizione del file delle credenziali varia in base al sistema operativo in uso:

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

  Le variabili di ambiente sono variabili gestite dal sistema operativo e utilizzate dai comandi di sistema. Le variabili definite durante una sessione SSH non sono disponibili dopo la chiusura della sessione. IDT può utilizzare le variabili di `AWS_SECRET_ACCESS_KEY` ambiente `AWS_ACCESS_KEY_ID` e per memorizzare le credenziali AWS

  Per impostare queste variabili su Linux, macOS o Unix, utilizza export:

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

Per impostare queste variabili su Windows, utilizza set:

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

Per configurare AWS le credenziali per IDT, i test runner modificano la `auth` sezione del `config.json` file che si trova nella cartella. *<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>"
        }
    }
}
]
```

Tutti i campi che includono valori sono obbligatori, come descritto di seguito:

> **Note**
>
> Tutti i percorsi di questo file sono definiti in relazione a. *<device-tester-extract-location>*

`log.location`

Il percorso della cartella dei registri in. *<device-tester-extract-location>*

`configFiles.root`

Il percorso della cartella che contiene i file di configurazione.

`configFiles.device`

Il percorso del `device.json` file.

`testPath`

Il percorso della cartella che contiene le suite di test.

`reportPath`

Il percorso della cartella che conterrà i risultati dei test dopo che IDT avrà eseguito una suite di test.

`awsRegion`

Facoltativo. La AWS regione che verranno utilizzate dalle suite di test. Se non è impostata, le suite di test utilizzeranno la regione predefinita specificata in ciascuna suite di test.

`auth.method`

Il metodo utilizzato da IDT per recuperare AWS le credenziali. I valori supportati sono `file` il recupero delle credenziali da un file di credenziali e il recupero delle credenziali utilizzando le variabili `environment` di ambiente.

`auth.credentials.profile`

Il profilo delle credenziali da utilizzare dal file delle credenziali. Questa proprietà si applica solo se `auth.method` è impostata su `file`.

Esegui il debug ed esegui suite di test personalizzate

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Dopo aver impostato la configurazione richiesta, IDT può eseguire la suite di test. Il tempo di esecuzione della suite di test completa dipende dall'hardware e dalla composizione della suite di test.

Per riferimento, sono necessari circa 30 minuti per completare l'intera suite di test di qualificazione FreerTOS su un Raspberry Pi 3B.

Mentre scrivi la tua suite di test, puoi usare IDT per eseguire la suite di test in modalità debug per controllare il codice prima di eseguirlo o fornirlo ai test runner.

Esegui IDT in modalità debug

Poiché le suite di test dipendono da IDT per interagire con i dispositivi, fornire il contesto e ricevere risultati, non è possibile semplicemente eseguire il debug delle suite di test in un IDE senza alcuna interazione IDT. A tale scopo, la CLI IDT fornisce `debug-test-suite` il comando che consente di eseguire IDT in modalità di debug. Eseguite il comando seguente per visualizzare le opzioni disponibili per: `debug-test-suite`

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

Quando eseguite IDT in modalità debug, IDT non avvia effettivamente la suite di test né esegue la macchina a stati; interagisce invece con l'IDE per rispondere alle richieste fatte dalla suite di test in esecuzione nell'IDE e stampa i log sulla console. IDT non scade e attende di uscire finché non viene interrotto manualmente. In modalità debug, IDT inoltre non esegue la macchina a stati e non genererà alcun file di report. Per eseguire il debug della suite di test, è necessario utilizzare l'IDE per fornire alcune informazioni che IDT di solito ottiene dai file JSON di configurazione. Assicurati di fornire le seguenti informazioni:

- Variabili di ambiente e argomenti per ogni test. IDT non leggerà queste informazioni da `test.json` `osuite.json`.
- Argomenti per selezionare i dispositivi di risorse. IDT non leggerà queste informazioni `datest.json`.

Per eseguire il debug delle tue suite di test, completa i seguenti passaggi:

1. Crea i file di configurazione delle impostazioni necessari per eseguire la suite di test. Ad esempio, se la tua suite di test richiede `device.json` `resource.jsonuser data.json`, e, assicurati di configurarli tutti secondo necessità.
2. Eseguite il comando seguente per mettere IDT in modalità debug e selezionare tutti i dispositivi necessari per eseguire il test.

   ```
   devicetester_[linux | mac | win_x86-64] debug-test-suite [options]
   ```

Dopo aver eseguito questo comando, IDT attende le richieste dalla suite di test e quindi risponde ad esse. IDT genera anche le variabili di ambiente necessarie per il processo di elaborazione dei casi per IDT Client SDK.

3. Nel tuo IDE, usa la `debug` configurazione `run` o per effettuare le seguenti operazioni:

   a. Imposta i valori delle variabili di ambiente generate da IDT.

   b. Imposta il valore di tutte le variabili o gli argomenti di ambiente che hai specificato nel file `test.json` and`suite.json`.

   c. Imposta i punti di interruzione secondo necessità.

4. Esegui la suite di test nel tuo IDE.

   Puoi eseguire il debug e rieseguire la suite di test tutte le volte che è necessario. IDT non scade in modalità di debug.

5. Dopo aver completato il debug, interrompi IDT per uscire dalla modalità di debug.

Comandi IDT CLI per eseguire test

La sezione seguente descrive i comandi IDT CLI:

IDT v4.0.0

`help`

Elenca le informazioni sul comando specificato.

`list-groups`

Elenca i gruppi in una determinata suite di test.

`list-suites`

Elenca le suite di test disponibili.

`list-supported-products`

Elenca i prodotti supportati per la tua versione di IDT, in questo caso le versioni FreeRTOS e le versioni della suite di test di qualificazione FreerTOS disponibili per la versione IDT corrente.

`list-test-cases`

Elenca i casi di test in un determinato gruppo di test. È supportata la seguente opzione:

- `group-id`. Il gruppo di test da cercare. Questa opzione è obbligatoria e deve specificare un singolo gruppo.

`run-suite`

Esegue una suite di test in un determinato pool di dispositivi. Di seguito sono riportate alcune opzioni di uso comune:

- `suite-id`. La versione della suite di test da eseguire. Se non specificato, IDT utilizza la versione più recente nella cartella `tests`.
- `group-id`. I gruppi di test da eseguire, sotto forma di elenco separato da virgole. Se non specificato, IDT esegue tutti i gruppi di test nella suite di test.
- `test-id`. I casi di test da eseguire, come elenco separato da virgole. Quando specificato, `group-id` deve specificare un singolo gruppo.
- `pool-id`. Il pool di dispositivi da testare. I test runner devono specificare un pool se hanno più pool di dispositivi definiti nel `device.json` file.
- `timeout-multiplier`. Configura IDT per modificare il timeout di esecuzione del test specificato nel `test.json` file per un test con un moltiplicatore definito dall'utente.
- `stop-on-first-failure`. Configura IDT per interrompere l'esecuzione al primo errore. Questa opzione deve essere utilizzata con `group-id` per eseguire il debug dei gruppi di test specificati.
- `userdata`. Imposta il file che contiene le informazioni sui dati utente necessarie per eseguire la suite di test. Questo è richiesto solo se `userdataRequired` è impostato su true nel `suite.json` file della suite di test.

Per ulteriori informazioni sulle opzioni `run-suite`, utilizzare l'opzione `help`:

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

`debug-test-suite`

Esegui la suite di test in modalità debug. Per ulteriori informazioni, consulta Esegui IDT in modalità debug.

Esamina i risultati e i registri dei test IDT

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Questa sezione descrive il formato in cui IDT genera i log della console e i report dei test.

Formato dei messaggi della console

AWS IoT Device Tester utilizza un formato standard per la stampa dei messaggi sulla console quando avvia una suite di test. Il seguente estratto mostra un esempio di messaggio di console generato da IDT.

```
time="2000-01-02T03:04:05-07:00" level=info msg=Using suite: MyTestSuite_1.0.0
executionId=9a52f362-1227-11eb-86c9-8c8590419f30
```

La maggior parte dei messaggi della console è composta dai seguenti campi:

`time`

Un timestamp ISO 8601 completo per l'evento registrato.

`level`

Il livello del messaggio per l'evento registrato. In genere, il livello del messaggio registrato è uno dei `info`warn, o. `error` IDT emette un `panic` messaggio `fatal` or se rileva un evento previsto che ne causa la chiusura anticipata.

`msg`

Il messaggio registrato.

`executionId`

Una stringa ID univoca per il processo IDT corrente. Questo ID viene utilizzato per distinguere le singole esecuzioni IDT.

I messaggi della console generati da una suite di test forniscono informazioni aggiuntive sul dispositivo sottoposto a test e sulla suite di test, sul gruppo di test e sui casi di test eseguiti da IDT. Il seguente estratto mostra un esempio di messaggio di console generato da una suite di test.

```
time="2000-01-02T03:04:05-07:00" level=info msg=Hello world! suiteId=MyTestSuite
groupId=myTestGroup testCaseId=myTestCase deviceId=my-device
executionId=9a52f362-1227-11eb-86c9-8c8590419f30
```

La parte specifica del messaggio della console di test contiene i seguenti campi:

`suiteId`

Il nome della suite di test attualmente in esecuzione.

`groupId`

L'ID del gruppo di test attualmente in esecuzione.

`testCaseId`

L'ID del test case attualmente in esecuzione.

`deviceId`

Un ID del dispositivo sottoposto a test utilizzato dal test case corrente.

Per stampare un riepilogo del test sulla console quando un IDT termina l'esecuzione di un test, è necessario includere uno `Reportstato` nella macchina a stati. Il riepilogo del test contiene informazioni sulla suite di test, i risultati dei test per ogni gruppo eseguito e le posizioni dei log e dei file di report generati. L'esempio seguente mostra un messaggio di riepilogo del test.

```
========== 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 schema del rapporto

`awsiotdevicetester_report.xml`è un rapporto firmato che contiene le seguenti informazioni:

- La versione di IDT.
- La versione della suite di test.
- La firma del rapporto e la chiave utilizzate per firmare il rapporto.
- Lo SKU del dispositivo e il nome del pool di dispositivi specificati nel `device.json` file.
- La versione del prodotto e le funzionalità del dispositivo testate.
- Il riepilogo aggregato dei risultati dei test. Queste informazioni sono le stesse contenute nel *`suite-name`*`_report.xml` file.

```
<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>
```

Il file `awsiotdevicetester_report.xml` contiene un tag `<awsproduct>` con le informazioni relative al prodotto sottoposto a test e le caratteristiche del prodotto che sono state convalidate dopo l'esecuzione di una suite di test.

Attributi utilizzati nel tag **`<awsproduct>`**

`name`

Il nome del prodotto sottoposto a test.

`version`

La versione del prodotto sottoposto a test.

`features`

Le caratteristiche convalidate. Le funzionalità contrassegnate come `required` sono necessarie per consentire alla suite di test di convalidare il dispositivo. Il seguente frammento di codice mostra come questa informazione viene visualizzata nel file `awsiotdevicetester_report.xml`.

```
<feature name="ssh" value="supported" type="required"></feature>
```

Le funzionalità contrassegnate come non `optional` sono richieste per la convalida. I seguenti snippet mostrano caratteristiche facoltative.

```
<feature name="hsi" value="supported" type="optional"></feature>
<feature name="mqtt" value="not-supported" type="optional"></feature>
```

Schema di report della suite di test

Il *suite-name*_Result.xml rapporto è in formato JUnit XML. Puoi eseguire l'integrazione in piattaforme di integrazione e distribuzione continue come Jenkins, Bambù e così via. Il rapporto contiene un riepilogo aggregato dei risultati del test.

```
<testsuites name="<suite-name> results" time="<run-duration>" tests="<number-of-test>"
 failures="<number-of-tests>" skipped="<number-of-tests>" errors="<number-of-tests>"
 disabled="0">
    <testsuite name="<test-group-id>" package="" tests="<number-of-tests>"
 failures="<number-of-tests>" skipped="<number-of-tests>" errors="<number-of-tests>"
 disabled="0">
        <!--success-->
        <testcase classname="<classname>" name="<name>" time="<run-duration>"/>
        <!--failure-->
        <testcase classname="<classname>" name="<name>" time="<run-duration>">
            <failure type="<failure-type>">
                reason
            </failure>
        </testcase>
        <!--skipped-->
        <testcase classname="<classname>" name="<name>" time="<run-duration>">
            <skipped>
                reason
            </skipped>
        </testcase>
        <!--error-->
        <testcase classname="<classname>" name="<name>" time="<run-duration>">
            <error>
                reason
            </error>
        </testcase>
    </testsuite>
</testsuites>
```

La sezione del rapporto in entrambe le sezioni awsiotdevicetester_report.xml o *suite-name*_report.xml elenca i test eseguiti e i risultati.

Il primo tag XML <testsuites> contiene il riepilogo dell'esecuzione dei test. Per esempio:

```
<testsuites name="MyTestSuite results" time="2299" tests="28" failures="0" errors="0"
 disabled="0">
```

Attributi utilizzati nel tag **`<testsuites>`**

`name`

Il nome della suite di test.

`time`

Il tempo impiegato, in secondi, per eseguire la suite di test.

`tests`

Il numero di test eseguiti.

`failures`

Il numero di test eseguiti ma non superati.

`errors`

Il numero di test che IDT non è stato in grado di eseguire.

`disabled`

Questo attributo non è utilizzato e si può ignorare.

In caso di esiti negativi o errori nei test, puoi identificare il test non riuscito esaminando i tag XML `<testsuites>`. I tag XML `<testsuite>` all'interno del tag `<testsuites>` mostrano il riepilogo dei risultati dei test per un gruppo di test. Per esempio:

```
<testsuite name="combination" package="" tests="1" failures="0" time="161" disabled="0"
 errors="0" skipped="0">
```

Il formato è simile al tag `<testsuites>`, ma con un attributo `skipped` che non viene utilizzato e che è possibile ignorare. All'interno di ogni tag XML `<testsuite>` ci sono tag `<testcase>` per ciascuno dei test eseguiti per un gruppo di test. Per esempio:

```
<testcase classname="Security Test" name="IP Change Tests" attempts="1"></testcase>
```

Attributi utilizzati nel tag **`<testcase>`**

`name`

Il nome del test.

`attempts`

Il numero di volte che IDT ha eseguito il test.

Quando un test non riesce o si verifica un errore, i tag `<failure>` o `<error>` vengono aggiunti al tag `<testcase>` con informazioni per la risoluzione dei problemi. Per esempio:

```
<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>
```

Metriche di utilizzo IDT

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Se fornisci AWS credenziali con le autorizzazioni richieste, AWS IoT Device Tester raccoglie e invia le metriche di utilizzo a. AWS Si tratta di una funzionalità opzionale e viene utilizzata per migliorare la funzionalità IDT. IDT raccoglie informazioni come le seguenti:

- L'ID AWS dell'account utilizzato per eseguire IDT
- I comandi IDT CLI utilizzati per eseguire i test
- La suite di test che viene eseguita
- Le suite di test nella *<device-tester-extract-location>* cartella
- Il numero di dispositivi configurati nel pool di dispositivi
- Nomi e tempi di esecuzione dei test case
- Informazioni sui risultati del test, ad esempio se i test sono stati superati, hanno avuto esito negativo, hanno riscontrato errori o sono stati saltati
- Caratteristiche del prodotto testate
- Comportamento di uscita IDT, ad esempio uscite impreviste o anticipate

Tutte le informazioni inviate da IDT vengono inoltre registrate in un `metrics.log` file nella cartella. *<device-tester-extract-location>*`/results/`*<execution-id>*`/` È possibile visualizzare il file di registro per visualizzare le informazioni raccolte durante un'esecuzione di test. Questo file viene generato solo se si sceglie di raccogliere le metriche di utilizzo.

Per disabilitare la raccolta delle metriche, non è necessario intraprendere ulteriori azioni. Semplicemente non archiviate AWS le vostre credenziali e, se avete AWS credenziali memorizzate, non configurate il file `config.jso` n per accedervi.

Configura le tue credenziali AWS

Se non disponi già di un AWS account, devi crearne uno. Se disponi già di un AWS account, devi semplicemente configurare le autorizzazioni richieste per il tuo account che consentano a IDT di inviare le metriche di utilizzo AWS a tuo nome.

Registrati per un Account AWS

Se non ne hai uno Account AWS, completa i seguenti passaggi per crearne uno.

Per iscriverti a un Account AWS

1. Apri la https://portal.aws.amazon.com/billing/registrazione.
2. Segui le istruzioni online.

   Parte della procedura di registrazione prevede la ricezione di una telefonata o di un messaggio di testo e l'immissione di un codice di verifica sulla tastiera del telefono.

   Quando ti iscrivi a un Account AWS, Utente root dell'account AWSviene creato un. L'utente root dispone dell'accesso a tutte le risorse e tutti i Servizi AWS nell'account. Come best practice di sicurezza, assegna l'accesso amministrativo a un utente e utilizza solo l'utente root per eseguire attività che richiedono l'accesso di un utente root.

AWS ti invia un'email di conferma dopo il completamento della procedura di registrazione. In qualsiasi momento, puoi visualizzare l'attività corrente del tuo account e gestirlo accedendo a https://aws.amazon.com/e scegliendo Il mio account.

Crea un utente con accesso amministrativo

Dopo esserti registrato Account AWS, proteggi Utente root dell'account AWS AWS IAM Identity Center, abilita e crea un utente amministrativo in modo da non utilizzare l'utente root per le attività quotidiane.

Proteggi i tuoi Utente root dell'account AWS

1. Accedi AWS Management Consolecome proprietario dell'account scegliendo Utente root e inserendo il tuo indirizzo Account AWS email. Nella pagina successiva, inserisci la password.

   Per informazioni sull'accesso utilizzando un utente root, consulta la pagina Signing in as the root user della Guida per l'utente di Accedi ad AWS .
2. Abilita l'autenticazione a più fattori (MFA) per l'utente root.

   Per istruzioni, consulta Abilitare un dispositivo MFA virtuale per l'utente Account AWS root (console) nella Guida per l'utente IAM.

Crea un utente con accesso amministrativo

1. Abilita Centro identità IAM.

   Per istruzioni, consulta Abilitazione di AWS IAM Identity Center nella Guida per l'utente di AWS IAM Identity Center .
2. In IAM Identity Center, assegna l'accesso amministrativo a un utente.

   Per un tutorial sull'utilizzo di IAM Identity Center directory come fonte di identità, consulta Configurare l'accesso utente con l'impostazione predefinita IAM Identity Center directory nella Guida per l'AWS IAM Identity Center utente.

Accesso come utente amministratore

- Per accedere con l'utente IAM Identity Center, utilizza l'URL di accesso che è stato inviato al tuo indirizzo e-mail quando hai creato l'utente IAM Identity Center.

  Per informazioni sull'accesso utilizzando un utente IAM Identity Center, consulta AWS Accedere al portale di accesso nella Guida per l'Accedi ad AWS utente.

Assegna l'accesso a ulteriori utenti

1. In IAM Identity Center, crea un set di autorizzazioni conforme alla best practice dell'applicazione di autorizzazioni con il privilegio minimo.

   Segui le istruzioni riportate nella pagina Creazione di un set di autorizzazioni nella Guida per l'utente di AWS IAM Identity Center .
2. Assegna al gruppo prima gli utenti e poi l'accesso con autenticazione unica (Single Sign-On).

   Per istruzioni, consulta Aggiungere gruppi nella Guida per l'utente di AWS IAM Identity Center .

Configura le autorizzazioni per IDT

In questo passaggio, configura le autorizzazioni utilizzate da IDT per eseguire test e raccogliere dati sull'utilizzo di IDT. Puoi utilizzare AWS Management Console or AWS Command Line Interface (AWS CLI) per creare una policy IAM e un utente per IDT, quindi allegare le policy all'utente.

- Per configurare le autorizzazioni per IDT (Console)
- Per configurare le autorizzazioni per IDT (AWS CLI)

Per configurare le autorizzazioni per IDT (Console)

Segui questi passaggi per utilizzare la console per configurare le autorizzazioni per IDT per FreerTOS.

1. Accedere alla console IAM.
2. Creare un criterio gestito dal cliente che concede le autorizzazioni per creare ruoli con autorizzazioni specifiche.

   a. Nel riquadro di navigazione, seleziona Policy e quindi Crea policy.

   b. Nella scheda JSON, sostituire il contenuto del segnaposto con la seguente policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot-device-tester:SendMetrics"
```

```
            ],
            "Resource": "*"
        }
    ]
}
```

c. Scegli Verifica policy.

d. Per Nome, immetti **`IDTUsageMetricsIAMPermissions`**. In Riepilogo, esaminare le autorizzazioni concesse dai criteri.

e. Scegliere Create Policy (Crea policy).

3. Crea un utente IAM e assegna le autorizzazioni all'utente.

   Per fornire l'accesso, aggiungi autorizzazioni agli utenti, gruppi o ruoli:

   - Utenti e gruppi in AWS IAM Identity Center:

     Crea un set di autorizzazioni. Segui le istruzioni riportate nella pagina Create a permission set (Creazione di un set di autorizzazioni) nella Guida per l'utente di AWS IAM Identity Center .

   - Utenti gestiti in IAM tramite un provider di identità:

     Crea un ruolo per la federazione delle identità. Segui le istruzioni riportate nella pagina Create a role for a third-party identity provider (federation) della Guida per l'utente IAM.

   - Utenti IAM:
     - Crea un ruolo che l'utente possa assumere. Segui le istruzioni riportate nella pagina Create a role for an IAM user della Guida per l'utente IAM.
     - (Non consigliato) Collega una policy direttamente a un utente o aggiungi un utente a un gruppo di utenti. Segui le istruzioni riportate nella pagina Aggiunta di autorizzazioni a un utente (console) nella Guida per l'utente IAM.

Per configurare le autorizzazioni per IDT (AWS CLI)

Segui questi passaggi per utilizzare per configurare le AWS CLI autorizzazioni per IDT for FreerTOS.

1. Sul tuo computer, installa e configura il AWS CLI se non è già installato. Segui la procedura descritta in Installazione di AWS CLI nella Guida AWS Command Line Interface per l'utente.

> **Note**
>
> AWS CLI È uno strumento open source che puoi utilizzare per interagire con AWS i servizi dalla tua shell a riga di comando.

2. Crea la seguente politica gestita dai clienti che conceda le autorizzazioni per gestire i ruoli IDT e 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**
   >
   > Questo passaggio include un esempio del prompt dei comandi di Windows perché utilizza una sintassi JSON diversa rispetto ai comandi del terminale Linux, macOS o Unix.

3. Crea un utente IAM e allega le autorizzazioni richieste da IDT per FreerTOS.

a. Crea un utente IAM.

```
aws iam create-user --user-name user-name
```

b. Allega la `IDTUsageMetricsIAMPermissions` policy che hai creato al tuo utente IAM. Sostituiscila *user-name* con il tuo nome utente IAM e *<account-id>* nel comando con l'ID del tuo AWS account.

```
aws iam attach-user-policy --user-name user-name --policy-arn
 arn:aws:iam::<account-id>:policy/IDTFreeRTOSIAMPermissions
```

4. Creare una chiave di accesso segreta per l'utente.

```
aws iam create-access-key --user-name user-name
```

Memorizzare l'output in una posizione sicura. Utilizzerai queste informazioni in seguito per configurare il file AWS delle credenziali.

Fornisci le AWS credenziali a IDT

Per consentire a IDT di accedere alle tue AWS credenziali e inviare le metriche a AWS, procedi come segue:

1. Archivia le AWS credenziali per il tuo utente IAM come variabili di ambiente o in un file di credenziali:

a. Per utilizzare le variabili di ambiente, esegui il comando seguente:

```
AWS_ACCESS_KEY_ID=access-key
AWS_SECRET_ACCESS_KEY=secret-access-key
```

b. Per utilizzare il file delle credenziali, aggiungete le seguenti informazioni al `.aws/credentials file:`

```
[profile-name]
aws_access_key_id=access-key
aws_secret_access_key=secret-access-key
```

2. Configura la `auth` sezione del `config.json` file. Per ulteriori informazioni, consulta (Facoltativo) Configura config.json.

Risoluzione dei problemi

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Ogni esecuzione della suite di test ha un ID di esecuzione univoco che viene utilizzato per creare una cartella denominata `results/`*`execution-id`* nella directory `results`. I log dei singoli gruppi di test sono disponibili nella directory `results/`*`execution-id`*`/logs`. Usa l'output della console IDT for FreerTOS per trovare l'id di esecuzione, l'id del test case e l'id del gruppo di test del test case che ha avuto esito negativo, quindi apri il file di registro per quel test case denominato. `results/`*`execution-id`*`/logs/`*`test_group_id`*`__`*`test_case_id`*`.log` Le informazioni nel file includono:

- Output completo dei comandi build e flash.
- Output dell'esecuzione di test.
- IDT più prolisso per l'output della console FreerTOS.

Per la risoluzione dei problemi consigliamo il seguente flusso di lavoro:

1. Se vedi l'errore "non *`user/role`* è autorizzato ad accedere a questa risorsa», assicurati di configurare le autorizzazioni come specificato in. ???
2. Leggere l'output della console per trovare le informazioni, ad esempio l'UUID dell'esecuzione e le attività attualmente in esecuzione.
3. Nel file `FRQ_Report.xml` cercare le istruzioni di errore risultanti da ciascun test. Questa directory contiene i log di esecuzione di ciascun gruppo di test.
4. Cerca nei file di registro sotto`/results/`*`execution-id`*`/logs`.
5. Esaminare una delle seguenti aree problematiche:

- Configurazione del dispositivo, ad esempio i file di configurazione JSON nella cartella `/configs/`.
- Interfaccia del dispositivo. Controllare i log per determinare l'interfaccia che ha generato l'errore.
- Strumenti del dispositivo. Verificare che le toolchain per le operazioni di compilazione e flashing del dispositivo siano installate e configurate correttamente.
- Assicurati di avere una versione pulita e clonata del codice sorgente di FreerTOS. Le versioni di FreerTOS sono etichettate in base alla versione di FreerTOS. Per clonare una versione specifica del codice, usa i seguenti comandi:

```
git clone --branch version-number https://github.com/aws/amazon-freertos.git
cd amazon-freertos
git submodule update --checkout --init --recursive
```

Risoluzione degli errori di configurazione del dispositivo

Quando si utilizza IDT per FreerTOS, è necessario disporre dei file di configurazione corretti prima di eseguire il file binario. Se ottieni errori di parsing e di configurazione, per prima cosa dovresti individuare e utilizzare un modello di configurazione appropriato per il tuo ambiente. Questi modelli si trovano nella directory *IDT_ROOT*`/configs`.

Se continui a riscontare problemi, consulta la seguente procedura di debug.

Dove devo cercare?

Per iniziare, leggi l'output della console per trovare le informazioni necessarie, ad esempio l'UUID dell'esecuzione, ovvero `execution-id` in questa documentazione.

Esamina quindi il file `FRQ_Report.xml` nella directory `/results/`*`execution-id`*. Questo file contiene tutti i casi di test eseguiti e i frammenti di errore per ogni fallimento. Per ottenere tutti i log di esecuzione, cerca il file `/results/`*`execution-id`*`/logs/`*`test_group_id`*`__`*`test_case_id`*`.log` per ciascun caso di test.

Codici di errore IDT

La tabella seguente spiega i codici di errore generati da IDT per FreerTOS:

Codice di errore	Nome del codice di errore	Possibile causa principale	Risoluzione dei problemi
201	InvalidInputError	I campi in `device.json` , `config.json` o `userdata.json` sono mancanti o in un formato errato.	Assicurati che i campi obbligatori non siano mancanti e che siano nel formato obbligatorio nei file elencati. Per ulteriori informazioni, consulta Preparazione dei test per la prima verifica della scheda del microcontrollore.
202	ValidationError	I campi in `device.json` , `config.json` o `userdata.json` contengono valori non validi.	Controllare il messaggio di errore a destra del codice di errore nel report: • AWS Regione non valida: specifica una AWS regione valida nel file. `config.json` Per ulteriori informazioni sulle AWS regioni, consulta Regioni ed endpoint. • AWS Credenziali non valide: imposta AWS credenziali valide sulla macchina di test (tramite le variabili di ambiente o il file

Codice di errore	Nome del codice di errore	Possibile causa principale	Risoluzione dei problemi
			delle credenziali). Verifica che il campo di autenticazione sia configurato correttamente. Per ulteriori informazioni, consulta ???.
203	CopySourceCodeError	Impossibile copiare il codice sorgente di FreerTOS nella directory specificata.	Verificare quanto segue: • Verificare che nel file `userdata.json` sia specificato un `sourcePath` valido. • Elimina la `build` cartella nella directory del codice sorgente di FreerTOS, se esiste. Per ulteriori informazioni, consulta Configurazione delle impostazioni di compilazione, flashing e test.

Codice di errore	Nome del codice di errore	Possibile causa principale	Risoluzione dei problemi
204	BuildSourceError	Impossibile compilare il codice sorgente di FreerTOS.	Verificare quanto segue: • Verifica che le informazioni contenute in `buildTool` nel file `userdata.json` siano corrette. • Se utilizzi `cmake` come strumento di compilazione, assicurati che `{{enableTests}}` sia specificato nel comando `buildTool` . Per ulteriori informazioni, consulta Configurazione delle impostazioni di compilazione, flashing e test. • Se hai estratto IDT per FreerTOS in un percorso di file sul tuo sistema che contiene spazi, ad esempio`C:\Users\My Name\Desktop\` , potresti

Codice di errore	Nome del codice di errore	Possibile causa principale	Risoluzione dei problemi
			aver bisogno di virgolette aggiuntive all'interno dei comandi di build per assicurarti che i percorsi vengano analizzati correttamente. La stessa cosa potrebbe essere necessaria per i comandi flash.
205	FlashOrRunTestError	IDT FreeRTOS non è in grado di eseguire il flashing o eseguire FreerTOS sul tuo DUT.	Verifica che le informazioni in `flashTool` nel file `userdata.json` siano corrette. Per ulteriori informazioni, consulta Configurazione delle impostazioni di compilazione, flashing e test.
206	StartEchoServerError	IDT FreerTOS non è in grado di avviare il server echo per i test o secure socket. WiFi	Verificare che le porte configurate in `echoServerConfiguration` nel file `userdata.json` non siano in uso o bloccate dalle impostazioni del firewall o della rete.

Debug degli errori di parsing

Talvolta un refuso in una configurazione JSON può causare errori di parsing. Nella maggior parte dei casi, il problema è dovuto all'omissione di parentesi, virgole o virgolette nel file JSON. IDT for FreerTOS esegue la convalida JSON e stampa le informazioni di debug. Inoltre indica la riga in cui si è verificato l'errore, il numero di riga e il numero di colonna dell'errore di sintassi. Queste informazioni dovrebbero essere sufficienti per aiutarti a correggere l'errore, ma se hai ancora problemi a localizzare l'errore, puoi eseguire la convalida manualmente nel tuo IDE, in un editor di testo come Atom o Sublime o tramite uno strumento online come. JSONLint

Errori di verifica dell'integrità durante il debug

Quando esegui il gruppo di RTOSIntegrity test Free e riscontri degli errori, assicurati innanzitutto di non aver modificato nessuno dei file di directory. *`freertos`* Se non l'hai fatto e continui a riscontrare problemi, assicurati di utilizzare il ramo corretto. Se esegui il `list-supported-products` comando di IDT, puoi trovare quale ramo taggato del *`freertos`* repository dovresti usare.

Se hai clonato il ramo con tag corretto del `freertos` repository e hai ancora problemi, assicurati di aver eseguito anche il comando. `submodule update` Il flusso di lavoro di clonazione per il `freertos` repository è il seguente.

```
git clone --branch version-number https://github.com/aws/amazon-freertos.git
cd amazon-freertos
git submodule update --checkout —init —recursive
```

L'elenco dei file che il correttore di integrità cerca si trova nel `checksums.json` file della directory. *`freertos`* Per qualificare una porta FreerTOS senza alcuna modifica ai file e alla struttura delle cartelle, assicurati che nessuno dei file elencati nelle sezioni " e `exhaustive` `minimal` " del `checksums.json` file sia stato modificato. Per eseguirlo con un SDK configurato, verifica che nessuno dei file nella sezione '`minimal`' sia stato modificato.

Se esegui IDT con un SDK e hai modificato alcuni file nella tua *`freertos`* directory, assicurati di configurare correttamente l'SDK nel file. `userdata` Altrimenti, il correttore di integrità verificherà tutti i file nella directory. *`freertos`*

Errori del gruppo di test di debug FullWiFi

Se si riscontrano errori nel gruppo di test e il FullWiFi test "`AFQP_WiFiConnectMultipleAP`" ha esito negativo, è possibile che entrambi i punti di accesso non si trovino nella stessa sottorete del

computer host su cui è in esecuzione IDT. Assicurati che entrambi i punti di accesso si trovino nella stessa sottorete del computer host su cui è in esecuzione IDT.

Debug dell'errore "parametro richiesto mancante"

Poiché vengono aggiunte nuove funzionalità a IDT per FreerTOS, potrebbero essere introdotte modifiche ai file di configurazione. L'utilizzo di un file di configurazione precedente potrebbe invalidare la tua configurazione. In questo caso, il file *`test_group_id__test_case_id`*`.log` nella directory `results/`*`execution-id`*`/logs` elenca in modo esplicito tutti i parametri mancanti. IDT for FreerTOS convalida gli schemi dei file di configurazione JSON per garantire che sia stata utilizzata l'ultima versione supportata.

Debug dell'errore "avvio del test non riuscito"

È possibile che si verifichino errori relativi a problemi in fase di avvio del test. Poiché le potenziali cause sono diverse, verifica che nelle seguenti aree non ci siano errori:

- Assicurati che il nome del pool incluso nel comando di esecuzione esista effettivamente. Questa informazione è riferita direttamente dal tuo file `device.json`.
- Verifica che i parametri di configurazione del dispositivo o dei dispositivi nel pool siano corretti.

Debug dell'errore "accesso non autorizzato alla risorsa"

Potresti vedere l'errore "non *`user/role`* è autorizzato ad accedere a questa risorsa» nell'output del terminale o nel file sottostante. `test_manager.log` `/results/`*`execution-id`*`/logs` Per risolvere questo problema, associare la policy `AWSIoTDeviceTesterForFreeRTOSFullAccess` gestita all'utente del test. Per ulteriori informazioni, consulta ???.

Debug degli errori di test di rete

Per i test basati sulla rete, IDT avvia un server echo che si collega a una porta non riservata sul computer host. Se riscontri errori dovuti a timeout o connessioni non disponibili nei test WiFi o secure socket, assicurati che la rete sia configurata per consentire il traffico verso le porte configurate nell'intervallo 1024 - 49151.

Il test Secure Sockets utilizza le porte 33333 e 33334 per impostazione predefinita. Per impostazione predefinita, i WiFi test utilizzano la porta 33335. Se queste tre porte sono in uso o bloccate da firewall o rete, è possibile scegliere di utilizzare per il test porte diverse in userdata.json. Per ulteriori

informazioni, consulta Configurazione delle impostazioni di compilazione, flashing e test. È possibile utilizzare i seguenti comandi per verificare se una porta specifica è in uso:

- Windows: `netsh advfirewall firewall show rule name=all | grep port`
- Linux: `sudo netstat -pan | grep port`
- macOS: `netstat -nat | grep port`

Errori di aggiornamento OTA a causa del payload appartenente alla stessa versione

Se i test case OTA falliscono perché la stessa versione è presente sul dispositivo dopo l'esecuzione di un'OTA, potrebbe essere dovuto al fatto che il sistema di compilazione (ad esempio cmake) non ha notato le modifiche di IDT al codice sorgente di FreerTOS e non ha creato un binario aggiornato. Ciò fa sì che OTA venga eseguito con lo stesso file binario attualmente presente sul dispositivo e che il test non riesca. Per risolvere gli errori di aggiornamento OTA, assicurarsi per prima cosa di utilizzare la versione più recente supportata del sistema di compilazione.

Errore del test OTA nel caso di test **`PresignedUrlExpired`**

Un prerequisito di questo test è che il tempo di aggiornamento OTA dovrebbe essere superiore a 60 secondi, altrimenti il test non riesce. In questo caso, nel log viene trovato il seguente messaggio di errore: "Test takes less than 60 seconds (url expired time) to finish. Please reach out to us." (Il test impiega meno di 60 secondi (tempo scadenza url) per terminare. Contattaci.)

Debug degli errori su interfaccia e porta del dispositivo

Questa sezione contiene informazioni sulle interfacce di dispositivo che IDT usa per connettersi ai tuoi dispositivi.

Piattaforme supportate

IDT supporta Linux, macOS e Windows. Le tre piattaforme hanno schemi di denominazione differenti per i dispositivi seriali a esse collegati:

- Linux: `/dev/tty*`
- macOS: `/dev/tty.*` o `/dev/cu.*`
- Windows: COM*

Per verificare la porta del tuo dispositivo:

- Per Linux e macOS, apri un terminale ed esegui `ls /dev/tty*`.
- Per macOS, apri un terminale ed esegui `ls /dev/tty.*` o `ls /dev/cu.*`.
- Per Windows, apri Device Manager ed espandi il gruppo di dispositivi seriali.

Per verificare quale dispositivo è collegato a una porta:

- Per Linux, verifica che il pacchetto `udev` sia installato, quindi esegui `udevadm info --name=PORT`. Questa utilità stampa le informazioni relative al driver del dispositivo. Tali informazioni consentono di verificare se si sta utilizzando la porta corretta.
- Per macOS, apri Launchpad e cerca **`System Information`**.
- Per Windows, apri Device Manager ed espandi il gruppo di dispositivi seriali.

Interfacce del dispositivo

Ogni dispositivo incorporato è diverso, il che significa che possono avere una o più porte seriali. Se sono collegati a un computer, i dispositivi dispongono comunemente di due porte:

- Una porta dati per le operazioni di flashing del dispositivo.
- Una porta di lettura per la lettura dell'output.

  Devi impostare la porta di lettura corretta nel file `device.json`. In caso contrario, la lettura dell'output dal dispositivo potrebbe restituire un errore.

  In presenza di più porte, assicurati di utilizzare la porta di lettura del dispositivo nel file `device.json`. Ad esempio, se colleghi un WRover dispositivo Espressif e le due porte ad esso assegnate sono e, utilizzate nel file. `/dev/ttyUSB0` `/dev/ttyUSB1` `/dev/ttyUSB1` `device.json`

  Per Windows, segui la stessa logica.

Lettura dei dati dal dispositivo

IDT for FreerTOS utilizza strumenti flash e di creazione di singoli dispositivi per specificare la configurazione delle porte. Se testando il dispositivo non ottieni un output, prova le seguenti impostazioni predefinite:

- Velocità in baud: 115200

- Bit di dati: 8
- Parità: nessuna
- Bit di stop: 1
- Controllo di flusso: nessuno

Queste impostazioni sono gestite da IDT per FreerTOS. Non è necessario impostarle manualmente. Tuttavia, è possibile utilizzare lo stesso metodo per leggere manualmente l'output del dispositivo. In Linux o macOS lo puoi fare utilizzando il comando `screen`. In Windows, è possibile utilizzare un programma come. TeraTerm

```
Screen: screen /dev/cu.usbserial 115200
```

```
TeraTerm: Use the above-provided settings to set the fields explicitly in
the GUI.
```

Problemi nella toolchain di sviluppo

In questa sezione vengono illustrati i problemi che possono verificarsi con la toolchain.

Code Composer Studio su Ubuntu

Le versioni più recenti di Ubuntu (17.10 e 18.04) hanno una versione del pacchetto `glibc` che non è compatibile con Code Composer Studio versioni 7.x. Consigliamo di installare Code Composer Studio versione 8.2 o successive.

I seguenti sono alcuni dei possibili sintomi di incompatibilità:

- FreerTOS non riesce a creare o eseguire il flashing sul tuo dispositivo.
- L'installer di Code Composer Studio potrebbe bloccarsi.
- Durante il processo di compilazione o flashing non viene visualizzato alcun output di log nella console.
- Il comando build tenta l'avvio in modalità GUI anche quando richiamato in modalità headless.

Registrazione

I log di IDT for FreerTOS sono collocati in un'unica posizione. Nella directory root di IDT, questi file sono disponibili in `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` sono i log più importanti da esaminare. `FRQ_Report.xml` contiene informazioni sui casi di test non riusciti con un messaggio di errore specifico. È quindi possibile utilizzare `logs/test_group_id__test_case_id.log` per analizzare più a fondo il problema per ottenere un contesto migliore.

Errori della console

Quando AWS IoT Device Tester viene eseguito, gli errori vengono segnalati alla console con brevi messaggi. Cerca in `results/execution-id/logs/test_group_id__test_case_id.log` ulteriori informazioni sull'errore.

Errori di log

Ogni esecuzione della suite di test ha un ID di esecuzione univoco che viene utilizzato per creare una cartella denominata `results/execution-id`. I log dei singoli casi di test sono disponibili nella directory `results/execution-id/logs`. Usa l'output della console IDT for FreerTOS per trovare l'id di esecuzione, l'id del test case e l'id del gruppo di test del test case che ha avuto esito negativo. Utilizzate quindi queste informazioni per trovare e aprire il file di registro per il test case denominato. `results/execution-id/logs/test_group_id__test_case_id.log` Le informazioni contenute in questo file includono l'output completo dei comandi di build e flash, l'output di esecuzione del test e l'output più dettagliato AWS IoT Device Tester della console.

Problemi con i bucket S3

Se si preme CTRL+C durante l'esecuzione di IDT, IDT avvierà un processo di pulizia. Parte di questa pulizia consiste nella rimozione delle risorse Amazon S3 che sono state create come parte dei test IDT. Se la pulizia non riesce a terminare, potresti riscontrare un problema a causa del numero eccessivo di bucket Amazon S3 creati. Ciò significa che la prossima volta che eseguirai IDT, i test inizieranno a fallire.

Se si preme CTRL+C per interrompere IDT, è necessario lasciarlo terminare il processo di pulizia per evitare questo problema. Puoi anche eliminare i bucket Amazon S3 dal tuo account che sono stati creati manualmente.

Risoluzione rei problemi relativi agli errori di timeout

Se si verificano errori di timeout durante l'esecuzione di una suite di test, aumentare il timeout specificando un fattore di moltiplicatore di timeout. Questo fattore viene applicato al valore di timeout predefinito. Qualsiasi valore configurato per questo flag deve essere maggiore o uguale a 1.0. Per usare il moltiplicatore di timeout, utilizzare il flag `--timeout-multiplier` durante l'esecuzione dei test.

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
```

Funzionalità e costi della rete cellulare AWS

Quando la `Cellular` funzionalità è impostata `Yes` nel tuo `device.JSON` file, FullSecureSockets utilizzerà EC2 le istanze t.micro per eseguire i test e ciò potrebbe comportare costi aggiuntivi per il tuo account. AWS Per ulteriori informazioni, consulta i EC2 prezzi di Amazon.

Politica di generazione dei report di qualificazione

I report sulle qualifiche vengono generati solo da versioni AWS IoT Device Tester (IDT) che supportano le versioni FreerTOS rilasciate negli ultimi due anni. Se avete domande sulla politica di supporto, contattateci. Supporto AWS

Politica di supporto AWS IoT Device Tester per FreerTOS

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

AWS IoT Device Tester for FreerTOS è uno strumento di automazione dei test per convalidare e qualificare i dispositivi FreeRTOS per l'inclusione nel Partner Device Catalog.AWS Ti consigliamo di utilizzare la versione più recente di FreerTOS AWS IoT e Device Tester per testare o qualificare i tuoi dispositivi. Supportiamo AWS IoT Device Tester per la versione più recente di FreerTOS e per le versioni FreeRTOS rilasciate nei sei mesi precedenti. Una versione AWS IoT Device Tester che supporta le versioni basate su LTS di FreerTOS è supportata per due anni. Attualmente, solo la versione AWS IoT Device Tester che supporta FreerTOS 202012.00 è supportata per due anni. L'ultima versione di FreerTOS è disponibile su. GitHub Per le versioni supportate di AWS IoT Device Tester, vedi. Versioni supportate di AWS IoT Device Tester per FreerTOS

Per ogni versione del framework IDT, sono supportate tre versioni della suite di test per la qualifica dei dispositivi.

Puoi anche utilizzare una qualsiasi delle versioni supportate di AWS IoT Device Tester con la versione corrispondente di FreerTOS per testare o qualificare i tuoi dispositivi. Anche se è possibile continuare a utilizzare Versioni IDT non supportate per FreerTOS, questi non riceveranno correzioni di bug o aggiornamenti.

In caso di domande sui criteri di supporto, contattare il supporto clienti AWS.

Sicurezza in AWS

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

La sicurezza del cloud AWS è la massima priorità. In qualità di AWS cliente, puoi beneficiare di un data center e di un'architettura di rete progettati per soddisfare i requisiti delle organizzazioni più sensibili alla sicurezza.

La sicurezza è una responsabilità condivisa tra AWS te e te. Il modello di responsabilità condivisa descrive questo come sicurezza del cloud e sicurezza nel cloud:

- Sicurezza del cloud: AWS è responsabile della protezione dell'infrastruttura che gestisce AWS i servizi nel AWS cloud. AWS ti fornisce anche servizi che puoi utilizzare in modo sicuro. L'efficacia della nostra sicurezza è regolarmente testata e verificata da revisori di terze parti come parte dei programmi di conformitàAWS. Per ulteriori informazioni sui programmi di conformità che si applicano a un AWS servizio, consulta AWS Services in Scope by Compliance Program.
- Sicurezza nel cloud: la tua responsabilità è determinata dal AWS servizio che utilizzi. L'utente è anche responsabile per altri fattori, tra cui la riservatezza dei dati, i requisiti dell'azienda e leggi e normative applicabili.

Questa documentazione ti aiuterà a capire come applicare il modello di responsabilità condivisa durante l'utilizzo AWS. Negli argomenti seguenti viene illustrato come eseguire la configurazione AWS per soddisfare gli obiettivi di sicurezza e conformità. Imparerai anche come utilizzare AWS i servizi che possono aiutarti a monitorare e proteggere AWS le tue risorse.

Per informazioni più approfondite sulla AWS IoT sicurezza, consulta Sicurezza e identità per AWS IoT.

Argomenti

- Identity and Access Management per FreerTOS
- Convalida della conformità

- Resilienza in AWS
- Sicurezza dell'infrastruttura in FreerTOS

Identity and Access Management per FreerTOS

AWS Identity and Access Management (IAM) è un software Servizio AWS che aiuta un amministratore a controllare in modo sicuro l'accesso alle risorse. AWS Gli amministratori IAM controllano chi può essere autenticato (effettuato l'accesso) e autorizzato (disporre delle autorizzazioni) per utilizzare le risorse FreerTOS. IAM è uno strumento Servizio AWS che puoi utilizzare senza costi aggiuntivi.

Argomenti

- Destinatari
- Autenticazione con identità
- Gestione dell'accesso con policy
- Come funziona FreerTOS con IAM
- Esempi di policy basate sull'identità per FreerTOS
- Risoluzione dei problemi relativi all'identità e all'accesso a FreerTOS

Destinatari

Il modo in cui usi AWS Identity and Access Management (IAM) varia a seconda del lavoro svolto in FreerTOS.

Utente del servizio: se utilizzi il servizio FreerTOS per svolgere il tuo lavoro, l'amministratore ti fornisce le credenziali e le autorizzazioni di cui hai bisogno. Man mano che utilizzi più funzionalità di FreerTOS per svolgere il tuo lavoro, potresti aver bisogno di autorizzazioni aggiuntive. La comprensione della gestione dell'accesso ti consente di richiedere le autorizzazioni corrette all'amministratore. Se non riesci ad accedere a una funzionalità di FreerTOS, consulta. Risoluzione dei problemi relativi all'identità e all'accesso a FreerTOS

Amministratore del servizio: se sei responsabile delle risorse FreeRTOS della tua azienda, probabilmente hai pieno accesso a FreerTOS. Il tuo compito è determinare a quali funzionalità e risorse FreerTOS devono accedere gli utenti del servizio. Devi inviare le richieste all'amministratore IAM per cambiare le autorizzazioni degli utenti del servizio. Esamina le informazioni contenute in

questa pagina per comprendere i concetti di base relativi a IAM. Per saperne di più su come la tua azienda può utilizzare IAM con FreerTOS, consulta. Come funziona FreerTOS con IAM

Amministratore IAM: se sei un amministratore IAM, potresti voler conoscere i dettagli su come scrivere policy per gestire l'accesso a FreerTOS. Per visualizzare esempi di policy basate sull'identità di FreerTOS che puoi utilizzare in IAM, vedi. Esempi di policy basate sull'identità per FreerTOS

Autenticazione con identità

L'autenticazione è il modo in cui accedi utilizzando le tue credenziali di identità. AWS Devi essere autenticato (aver effettuato l' Utente root dell'account AWS accesso AWS) come utente IAM o assumendo un ruolo IAM.

Puoi accedere AWS come identità federata utilizzando le credenziali fornite tramite una fonte di identità. AWS IAM Identity Center Gli utenti (IAM Identity Center), l'autenticazione Single Sign-On della tua azienda e le tue credenziali di Google o Facebook sono esempi di identità federate. Se accedi come identità federata, l'amministratore ha configurato in precedenza la federazione delle identità utilizzando i ruoli IAM. Quando accedi AWS utilizzando la federazione, assumi indirettamente un ruolo.

A seconda del tipo di utente, puoi accedere al AWS Management Console o al portale di AWS accesso. Per ulteriori informazioni sull'accesso a AWS, vedi Come accedere al tuo Account AWS nella Guida per l'Accedi ad AWS utente.

Se accedi a AWS livello di codice, AWS fornisce un kit di sviluppo software (SDK) e un'interfaccia a riga di comando (CLI) per firmare crittograficamente le tue richieste utilizzando le tue credenziali. Se non utilizzi AWS strumenti, devi firmare tu stesso le richieste. Per ulteriori informazioni sul metodo consigliato per la firma delle richieste, consulta Signature Version 4 AWS per le richieste API nella Guida per l'utente IAM.

A prescindere dal metodo di autenticazione utilizzato, potrebbe essere necessario specificare ulteriori informazioni sulla sicurezza. Ad esempio, ti AWS consiglia di utilizzare l'autenticazione a più fattori (MFA) per aumentare la sicurezza del tuo account. Per ulteriori informazioni, consulta Autenticazione a più fattori nella Guida per l'utente di AWS IAM Identity Center e Utilizzo dell'autenticazione a più fattori (MFA)AWS in IAM nella Guida per l'utente IAM.

Account AWS utente root

Quando si crea un account Account AWS, si inizia con un'identità di accesso che ha accesso completo a tutte Servizi AWS le risorse dell'account. Questa identità è denominata utente Account

AWS root ed è accessibile effettuando l'accesso con l'indirizzo e-mail e la password utilizzati per creare l'account. Si consiglia vivamente di non utilizzare l'utente root per le attività quotidiane. Conserva le credenziali dell'utente root e utilizzale per eseguire le operazioni che solo l'utente root può eseguire. Per un elenco completo delle attività che richiedono l'accesso come utente root, consulta la sezione Attività che richiedono le credenziali dell'utente root nella Guida per l'utente IAM.

Identità federata

Come procedura consigliata, richiedi agli utenti umani, compresi gli utenti che richiedono l'accesso come amministratore, di utilizzare la federazione con un provider di identità per accedere Servizi AWS utilizzando credenziali temporanee.

Un'identità federata è un utente dell'elenco utenti aziendale, di un provider di identità Web AWS Directory Service, della directory Identity Center o di qualsiasi utente che accede utilizzando le Servizi AWS credenziali fornite tramite un'origine di identità. Quando le identità federate accedono Account AWS, assumono ruoli e i ruoli forniscono credenziali temporanee.

Per la gestione centralizzata degli accessi, consigliamo di utilizzare AWS IAM Identity Center. Puoi creare utenti e gruppi in IAM Identity Center oppure puoi connetterti e sincronizzarti con un set di utenti e gruppi nella tua fonte di identità per utilizzarli su tutte le tue applicazioni. Account AWS Per ulteriori informazioni su IAM Identity Center, consulta Cos'è IAM Identity Center? nella Guida per l'utente di AWS IAM Identity Center .

Utenti e gruppi IAM

Un utente IAM è un'identità interna Account AWS che dispone di autorizzazioni specifiche per una singola persona o applicazione. Ove possibile, consigliamo di fare affidamento a credenziali temporanee invece di creare utenti IAM con credenziali a lungo termine come le password e le chiavi di accesso. Tuttavia, se si hanno casi d'uso specifici che richiedono credenziali a lungo termine con utenti IAM, si consiglia di ruotare le chiavi di accesso. Per ulteriori informazioni, consulta la pagina Rotazione periodica delle chiavi di accesso per casi d'uso che richiedono credenziali a lungo termine nella Guida per l'utente IAM.

Un gruppo IAM è un'identità che specifica un insieme di utenti IAM. Non è possibile eseguire l'accesso come gruppo. È possibile utilizzare gruppi per specificare le autorizzazioni per più utenti alla volta. I gruppi semplificano la gestione delle autorizzazioni per set di utenti di grandi dimensioni. Ad esempio, potresti avere un gruppo denominato IAMAdminse concedere a quel gruppo le autorizzazioni per amministrare le risorse IAM.

Gli utenti sono diversi dai ruoli. Un utente è associato in modo univoco a una persona o un'applicazione, mentre un ruolo è destinato a essere assunto da chiunque ne abbia bisogno. Gli utenti dispongono di credenziali a lungo termine permanenti, mentre i ruoli forniscono credenziali temporanee. Per ulteriori informazioni, consulta Casi d'uso per utenti IAM nella Guida per l'utente IAM.

Ruoli IAM

Un ruolo IAM è un'identità interna all'utente Account AWS che dispone di autorizzazioni specifiche. È simile a un utente IAM, ma non è associato a una persona specifica. Per assumere temporaneamente un ruolo IAM in AWS Management Console, puoi passare da un ruolo utente a un ruolo IAM (console). Puoi assumere un ruolo chiamando un'operazione AWS CLI o AWS API o utilizzando un URL personalizzato. Per ulteriori informazioni sui metodi per l'utilizzo dei ruoli, consulta Utilizzo di ruoli IAM nella Guida per l'utente IAM.

I ruoli IAM con credenziali temporanee sono utili nelle seguenti situazioni:

- Accesso utente federato: per assegnare le autorizzazioni a una identità federata, è possibile creare un ruolo e definire le autorizzazioni per il ruolo. Quando un'identità federata viene autenticata, l'identità viene associata al ruolo e ottiene le autorizzazioni da esso definite. Per ulteriori informazioni sulla federazione dei ruoli, consulta Create a role for a third-party identity provider (federation) nella Guida per l'utente IAM. Se utilizzi IAM Identity Center, configura un set di autorizzazioni. IAM Identity Center mette in correlazione il set di autorizzazioni con un ruolo in IAM per controllare a cosa possono accedere le identità dopo l'autenticazione. Per informazioni sui set di autorizzazioni, consulta Set di autorizzazioni nella Guida per l'utente di AWS IAM Identity Center .
- Autorizzazioni utente IAM temporanee: un utente IAM o un ruolo può assumere un ruolo IAM per ottenere temporaneamente autorizzazioni diverse per un'attività specifica.
- Accesso multi-account: è possibile utilizzare un ruolo IAM per permettere a un utente (un principale affidabile) con un account diverso di accedere alle risorse nell'account. I ruoli sono lo strumento principale per concedere l'accesso multi-account. Tuttavia, con alcuni Servizi AWS, è possibile allegare una policy direttamente a una risorsa (anziché utilizzare un ruolo come proxy). Per informazioni sulle differenze tra ruoli e policy basate su risorse per l'accesso multi-account, consulta Accesso a risorse multi-account in IAM nella Guida per l'utente IAM.
- Accesso a più servizi: alcuni Servizi AWS utilizzano le funzionalità di altri Servizi AWS. Ad esempio, quando effettui una chiamata in un servizio, è normale che quel servizio esegua applicazioni in Amazon EC2 o archivi oggetti in Amazon S3. Un servizio può eseguire questa

operazione utilizzando le autorizzazioni dell'entità chiamante, utilizzando un ruolo di servizio o utilizzando un ruolo collegato al servizio.

- Sessioni di accesso inoltrato (FAS): quando utilizzi un utente o un ruolo IAM per eseguire azioni AWS, sei considerato un principale. Quando si utilizzano alcuni servizi, è possibile eseguire un'operazione che attiva un'altra operazione in un servizio diverso. FAS utilizza le autorizzazioni del principale che chiama an Servizio AWS, combinate con la richiesta Servizio AWS per effettuare richieste ai servizi downstream. Le richieste FAS vengono effettuate solo quando un servizio riceve una richiesta che richiede interazioni con altri Servizi AWS o risorse per essere completata. In questo caso è necessario disporre delle autorizzazioni per eseguire entrambe le azioni. Per i dettagli delle policy relative alle richieste FAS, consulta Forward access sessions.
- Ruolo di servizio: un ruolo di servizio è un ruolo IAM che un servizio assume per eseguire operazioni per tuo conto. Un amministratore IAM può creare, modificare ed eliminare un ruolo di servizio dall'interno di IAM. Per ulteriori informazioni, consulta la sezione Create a role to delegate permissions to an Servizio AWS nella Guida per l'utente IAM.
- Ruolo collegato al servizio: un ruolo collegato al servizio è un tipo di ruolo di servizio collegato a un. Servizio AWS Il servizio può assumere il ruolo per eseguire un'azione per tuo conto. I ruoli collegati al servizio vengono visualizzati nel tuo account Account AWS e sono di proprietà del servizio. Un amministratore IAM può visualizzare le autorizzazioni per i ruoli collegati ai servizi, ma non modificarle.

- Applicazioni in esecuzione su Amazon EC2: puoi utilizzare un ruolo IAM per gestire le credenziali temporanee per le applicazioni in esecuzione su un' EC2 istanza e che AWS CLI effettuano richieste AWS API. Questa soluzione è preferibile alla memorizzazione delle chiavi di accesso all'interno dell' EC2 istanza. Per assegnare un AWS ruolo a un' EC2 istanza e renderlo disponibile per tutte le sue applicazioni, create un profilo di istanza collegato all'istanza. Un profilo di istanza contiene il ruolo e consente ai programmi in esecuzione sull' EC2 istanza di ottenere credenziali temporanee. Per ulteriori informazioni, consulta Utilizzare un ruolo IAM per concedere le autorizzazioni alle applicazioni in esecuzione su EC2 istanze Amazon nella IAM User Guide.

Gestione dell'accesso con policy

Puoi controllare l'accesso AWS creando policy e collegandole a AWS identità o risorse. Una policy è un oggetto AWS che, se associato a un'identità o a una risorsa, ne definisce le autorizzazioni. AWS valuta queste politiche quando un principale (utente, utente root o sessione di ruolo) effettua una richiesta. Le autorizzazioni nelle policy determinano l'approvazione o il rifiuto della richiesta. La maggior parte delle politiche viene archiviata AWS come documenti JSON. Per ulteriori informazioni

sulla struttura e sui contenuti dei documenti delle policy JSON, consulta Panoramica delle policy JSON nella Guida per l'utente IAM.

Gli amministratori possono utilizzare le policy AWS JSON per specificare chi ha accesso a cosa. In altre parole, quale principale può eseguire operazioni su quali risorse e in quali condizioni.

Per impostazione predefinita, utenti e ruoli non dispongono di autorizzazioni. Per concedere agli utenti l'autorizzazione a eseguire operazioni sulle risorse di cui hanno bisogno, un amministratore IAM può creare policy IAM. L'amministratore può quindi aggiungere le policy IAM ai ruoli e gli utenti possono assumere i ruoli.

Le policy IAM definiscono le autorizzazioni relative a un'operazione, a prescindere dal metodo utilizzato per eseguirla. Ad esempio, supponiamo di disporre di una policy che consente l'operazione `iam:GetRole`. Un utente con tale policy può ottenere informazioni sul ruolo dall' AWS Management Console AWS CLI, dall'o dall' AWS API.

Policy basate sull'identità

Le policy basate su identità sono documenti di policy di autorizzazione JSON che è possibile allegare a un'identità (utente, gruppo di utenti o ruolo IAM). Tali policy definiscono le operazioni che utenti e ruoli possono eseguire, su quali risorse e in quali condizioni. Per informazioni su come creare una policy basata su identità, consulta Definizione di autorizzazioni personalizzate IAM con policy gestite dal cliente nella Guida per l'utente IAM.

Le policy basate su identità possono essere ulteriormente classificate come policy inline o policy gestite. Le policy inline sono integrate direttamente in un singolo utente, gruppo o ruolo. Le politiche gestite sono politiche autonome che puoi allegare a più utenti, gruppi e ruoli nel tuo Account AWS. Le politiche gestite includono politiche AWS gestite e politiche gestite dai clienti. Per informazioni su come scegliere tra una policy gestita o una policy inline, consulta Scelta fra policy gestite e policy inline nella Guida per l'utente IAM.

Policy basate sulle risorse

Le policy basate su risorse sono documenti di policy JSON che è possibile collegare a una risorsa. Esempi di policy basate sulle risorse sono le policy di attendibilità dei ruoli IAM e le policy dei bucket Amazon S3. Nei servizi che supportano policy basate sulle risorse, gli amministratori dei servizi possono utilizzarli per controllare l'accesso a una risorsa specifica. Quando è collegata a una risorsa, una policy definisce le operazioni che un principale può eseguire su tale risorsa e a quali condizioni. È necessario specificare un principale in una policy basata sulle risorse. I principali possono includere account, utenti, ruoli, utenti federati o. Servizi AWS

Le policy basate sulle risorse sono policy inline che si trovano in tale servizio. Non puoi utilizzare le policy AWS gestite di IAM in una policy basata sulle risorse.

Elenchi di controllo degli accessi () ACLs

Le liste di controllo degli accessi (ACLs) controllano quali principali (membri dell'account, utenti o ruoli) dispongono delle autorizzazioni per accedere a una risorsa. ACLs sono simili alle politiche basate sulle risorse, sebbene non utilizzino il formato del documento di policy JSON.

Amazon S3 e Amazon VPC sono esempi di servizi che supportano. AWS WAF ACLs Per ulteriori informazioni ACLs, consulta la panoramica della lista di controllo degli accessi (ACL) nella Amazon Simple Storage Service Developer Guide.

Altri tipi di policy

AWS supporta tipi di policy aggiuntivi e meno comuni. Questi tipi di policy possono impostare il numero massimo di autorizzazioni concesse dai tipi di policy più comuni.

- Limiti delle autorizzazioni: un limite delle autorizzazioni è una funzionalità avanzata nella quale si imposta il numero massimo di autorizzazioni che una policy basata su identità può concedere a un'entità IAM (utente o ruolo IAM). È possibile impostare un limite delle autorizzazioni per un'entità. Le autorizzazioni risultanti sono l'intersezione delle policy basate su identità dell'entità e i relativi limiti delle autorizzazioni. Le policy basate su risorse che specificano l'utente o il ruolo nel campo `Principal`sono condizionate dal limite delle autorizzazioni. Un rifiuto esplicito in una qualsiasi di queste policy sostituisce l'autorizzazione. Per ulteriori informazioni sui limiti delle autorizzazioni, consulta Limiti delle autorizzazioni per le entità IAM nella Guida per l'utente IAM.
- Politiche di controllo del servizio (SCPs): SCPs sono politiche JSON che specificano le autorizzazioni massime per un'organizzazione o un'unità organizzativa (OU) in. AWS Organizations AWS Organizations è un servizio per il raggruppamento e la gestione centralizzata di più di proprietà dell' Account AWS azienda. Se abiliti tutte le funzionalità di un'organizzazione, puoi applicare le politiche di controllo del servizio (SCPs) a uno o tutti i tuoi account. L'SCP limita le autorizzazioni per le entità presenti negli account dei membri, inclusa ciascuna di esse. Utente root dell'account AWS Per ulteriori informazioni su Organizations and SCPs, consulta le politiche di controllo dei servizi nella Guida AWS Organizations per l'utente.
- Politiche di controllo delle risorse (RCPs): RCPs sono politiche JSON che puoi utilizzare per impostare le autorizzazioni massime disponibili per le risorse nei tuoi account senza aggiornare le politiche IAM allegate a ciascuna risorsa di tua proprietà. L'RCP limita le autorizzazioni per le risorse negli account dei membri e può influire sulle autorizzazioni effettive per le identità,

incluse le Utente root dell'account AWS, indipendentemente dal fatto che appartengano o meno all'organizzazione. Per ulteriori informazioni su Organizations e RCPs, incluso un elenco di Servizi AWS tale supporto RCPs, vedere Resource control policies (RCPs) nella Guida per l'AWS Organizations utente.

- Policy di sessione: le policy di sessione sono policy avanzate che vengono trasmesse come parametro quando si crea in modo programmatico una sessione temporanea per un ruolo o un utente federato. Le autorizzazioni della sessione risultante sono l'intersezione delle policy basate su identità del ruolo o dell'utente e le policy di sessione. Le autorizzazioni possono anche provenire da una policy basata su risorse. Un rifiuto esplicito in una qualsiasi di queste policy sostituisce l'autorizzazione. Per ulteriori informazioni, consulta Policy di sessione nella Guida per l'utente IAM.

Più tipi di policy

Quando più tipi di policy si applicano a una richiesta, le autorizzazioni risultanti sono più complicate da comprendere. Per scoprire come si AWS determina se consentire o meno una richiesta quando sono coinvolti più tipi di policy, consulta la logica di valutazione delle policy nella IAM User Guide.

Come funziona FreerTOS con IAM

Prima di utilizzare IAM per gestire l'accesso a FreerTOS, scopri quali funzionalità IAM sono disponibili per l'uso con FreerTOS.

Funzionalità IAM che puoi usare con FreerTOS

Funzionalità IAM	Supporto FreerTOS
Policy basate su identità	Sì
Policy basate su risorse	No
Azioni di policy	Sì
Risorse relative alle policy	Sì
Chiavi di condizione della policy (specifica del servizio)	Sì
ACLs	No

Funzionalità IAM	Supporto FreerTOS
ABAC (tag nelle policy)	Parziale
Credenziali temporanee	Sì
Autorizzazioni del principale	Sì
Ruoli di servizio	Sì
Ruoli collegati al servizio	No

Per avere una visione di alto livello di come FreerTOS e AWS altri servizi funzionano con la maggior parte delle funzionalità IAM, AWS consulta i servizi che funzionano con IAM nella IAM User Guide.

Politiche basate sull'identità per FreerTOS

Supporta le policy basate su identità: sì

Le policy basate su identità sono documenti di policy di autorizzazione JSON che è possibile allegare a un'identità (utente, gruppo di utenti o ruolo IAM). Tali policy definiscono le operazioni che utenti e ruoli possono eseguire, su quali risorse e in quali condizioni. Per informazioni su come creare una policy basata su identità, consulta Definizione di autorizzazioni personalizzate IAM con policy gestite dal cliente nella Guida per l'utente IAM.

Con le policy basate su identità di IAM, è possibile specificare quali operazioni e risorse sono consentite o respinte, nonché le condizioni in base alle quali le operazioni sono consentite o respinte. Non è possibile specificare l'entità principale in una policy basata sull'identità perché si applica all'utente o al ruolo a cui è associato. Per informazioni su tutti gli elementi utilizzabili in una policy JSON, consulta Guida di riferimento agli elementi delle policy JSON IAM nella Guida per l'utente di IAM.

Esempi di policy basate sull'identità per FreerTOS

Per visualizzare esempi di politiche basate sull'identità di FreerTOS, vedere. Esempi di policy basate sull'identità per FreerTOS

Politiche basate sulle risorse all'interno di FreerTOS

Supporta le policy basate su risorse: no

Le policy basate su risorse sono documenti di policy JSON che è possibile collegare a una risorsa. Esempi di policy basate sulle risorse sono le policy di attendibilità dei ruoli IAM e le policy dei bucket Amazon S3. Nei servizi che supportano policy basate sulle risorse, gli amministratori dei servizi possono utilizzarli per controllare l'accesso a una risorsa specifica. Quando è collegata a una risorsa, una policy definisce le operazioni che un principale può eseguire su tale risorsa e a quali condizioni. È necessario specificare un principale in una policy basata sulle risorse. I principali possono includere account, utenti, ruoli, utenti federati o. Servizi AWS

Per consentire l'accesso multi-account, puoi specificare un intero account o entità IAM in un altro account come principale in una policy basata sulle risorse. L'aggiunta di un principale multi-account a una policy basata sulle risorse rappresenta solo una parte della relazione di trust. Quando il principale e la risorsa sono diversi Account AWS, un amministratore IAM dell'account affidabile deve inoltre concedere all'entità principale (utente o ruolo) l'autorizzazione ad accedere alla risorsa. L'autorizzazione viene concessa collegando all'entità una policy basata sull'identità. Tuttavia, se una policy basata su risorse concede l'accesso a un principale nello stesso account, non sono richieste ulteriori policy basate su identità. Per ulteriori informazioni, consulta Accesso a risorse multi-account in IAM nella Guida per l'utente IAM.

Azioni politiche per FreerTOS

Supporta le operazioni di policy: si

Gli amministratori possono utilizzare le policy AWS JSON per specificare chi ha accesso a cosa. In altre parole, quale principale può eseguire operazioni su quali risorse, e in quali condizioni.

L'elemento `Action`di una policy JSON descrive le operazioni che è possibile utilizzare per consentire o negare l'accesso a un criterio. Le azioni politiche in genere hanno lo stesso nome dell'operazione AWS API associata. Ci sono alcune eccezioni, ad esempio le operazioni di sola autorizzazione che non hanno un'operazione API corrispondente. Esistono anche alcune operazioni che richiedono più operazioni in una policy. Queste operazioni aggiuntive sono denominate operazioni dipendenti.

Includi le operazioni in una policy per concedere le autorizzazioni a eseguire l'operazione associata.

Per visualizzare un elenco di azioni FreerTOS, vedere Azioni definite da FreerTOS nel Service Authorization Reference.

Le azioni politiche in FreerTOS utilizzano il seguente prefisso prima dell'azione:

```
awes
```

Per specificare più operazioni in una sola istruzione, occorre separarle con la virgola.

```
"Action": [
      "awes:action1",
      "awes:action2"
         ]
```

Per visualizzare esempi di politiche basate sull'identità di FreerTOS, vedere. Esempi di policy basate sull'identità per FreerTOS

Risorse politiche per FreerTOS

Supporta le risorse di policy: sì

Gli amministratori possono utilizzare le policy AWS JSON per specificare chi ha accesso a cosa. In altre parole, quale principale può eseguire operazioni su quali risorse, e in quali condizioni.

L'elemento JSON `Resource`della policy specifica l'oggetto o gli oggetti ai quali si applica l'operazione. Le istruzioni devono includere un elemento `Resource`o un elemento `NotResource`. Come best practice, specifica una risorsa utilizzando il suo nome della risorsa Amazon (ARN). È possibile eseguire questa operazione per operazioni che supportano un tipo di risorsa specifico, note come autorizzazioni a livello di risorsa.

Per le azioni che non supportano le autorizzazioni a livello di risorsa, ad esempio le operazioni di elenco, utilizza un carattere jolly (*) per indicare che l'istruzione si applica a tutte le risorse.

```
"Resource": "*"
```

Per visualizzare un elenco dei tipi di risorse FreeRTOS e ARNs relativi, vedere Resources defined by FreerTOS nel Service Authorization Reference. Per sapere con quali azioni è possibile specificare l'ARN di ciascuna risorsa, vedere Azioni definite da FreerTOS.

Per visualizzare esempi di politiche basate sull'identità di FreerTOS, vedere. Esempi di policy basate sull'identità per FreerTOS

Chiavi delle condizioni delle policy per FreerTOS

Supporta le chiavi di condizione delle policy specifiche del servizio: sì

Gli amministratori possono utilizzare le policy AWS JSON per specificare chi ha accesso a cosa. In altre parole, quale principale può eseguire operazioni su quali risorse, e in quali condizioni.

L'elemento `Condition`(o blocco `Condition`) consente di specificare le condizioni in cui un'istruzione è in vigore. L'elemento `Condition`è facoltativo. È possibile compilare espressioni condizionali che utilizzano operatori di condizione, ad esempio uguale a o minore di, per soddisfare la condizione nella policy con i valori nella richiesta.

Se specifichi più elementi `Condition`in un'istruzione o più chiavi in un singolo elemento `Condition`, questi vengono valutati da AWS utilizzando un'operazione `AND`logica. Se si specificano più valori per una singola chiave di condizione, AWS valuta la condizione utilizzando un'operazione logica. `OR` Tutte le condizioni devono essere soddisfatte prima che le autorizzazioni dell'istruzione vengano concesse.

È possibile anche utilizzare variabili segnaposto quando specifichi le condizioni. Ad esempio, è possibile autorizzare un utente IAM ad accedere a una risorsa solo se è stata taggata con il relativo nome utente IAM. Per ulteriori informazioni, consulta Elementi delle policy IAM: variabili e tag nella Guida per l'utente di IAM.

AWS supporta chiavi di condizione globali e chiavi di condizione specifiche del servizio. Per visualizzare tutte le chiavi di condizione AWS globali, consulta le chiavi di contesto delle condizioni AWS globali nella Guida per l'utente IAM.

Per visualizzare un elenco delle chiavi di condizione di FreeRTOS, consulta Condition keys for FreerTOS nel Service Authorization Reference. Per sapere con quali azioni e risorse puoi usare una chiave di condizione, vedi Azioni definite da FreerTOS.

Per visualizzare esempi di politiche basate sull'identità di FreerTOS, vedere. Esempi di policy basate sull'identità per FreerTOS

ACLs in FreerTOS

Supporti ACLs: no

Le liste di controllo degli accessi (ACLs) controllano quali principali (membri dell'account, utenti o ruoli) dispongono delle autorizzazioni per accedere a una risorsa. ACLs sono simili alle politiche basate sulle risorse, sebbene non utilizzino il formato del documento di policy JSON.

ABAC con FreerTOS

Supporta ABAC (tag nelle policy): parzialmente

Il controllo dell'accesso basato su attributi (ABAC) è una strategia di autorizzazione che definisce le autorizzazioni in base agli attributi. In AWS, questi attributi sono chiamati tag. Puoi allegare tag a entità IAM (utenti o ruoli) e a molte AWS risorse. L'assegnazione di tag alle entità e alle risorse è il primo passaggio di ABAC. In seguito, vengono progettate policy ABAC per consentire operazioni quando il tag dell'entità principale corrisponde al tag sulla risorsa a cui si sta provando ad accedere.

La strategia ABAC è utile in ambienti soggetti a una rapida crescita e aiuta in situazioni in cui la gestione delle policy diventa impegnativa.

Per controllare l'accesso basato su tag, fornisci informazioni sui tag nell'elemento condizione di una policy utilizzando le chiavi di condizione `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`o `aws:TagKeys`.

Se un servizio supporta tutte e tre le chiavi di condizione per ogni tipo di risorsa, il valore per il servizio è Yes (Sì). Se un servizio supporta tutte e tre le chiavi di condizione solo per alcuni tipi di risorsa, allora il valore sarà Parziale.

Per ulteriori informazioni su ABAC, consulta Definizione delle autorizzazioni con autorizzazione ABAC nella Guida per l'utente IAM. Per visualizzare un tutorial con i passaggi per l'impostazione di ABAC, consulta Utilizzo del controllo degli accessi basato su attributi (ABAC) nella Guida per l'utente di IAM.

Utilizzo di credenziali temporanee con FreerTOS

Supporta le credenziali temporanee: sì

Alcune Servizi AWS non funzionano quando si accede utilizzando credenziali temporanee. Per ulteriori informazioni, incluse quelle che Servizi AWS funzionano con credenziali temporanee, consulta la sezione relativa alla Servizi AWS compatibilità con IAM nella IAM User Guide.

Stai utilizzando credenziali temporanee se accedi AWS Management Console utilizzando qualsiasi metodo tranne nome utente e password. Ad esempio, quando accedi AWS utilizzando il link Single Sign-On (SSO) della tua azienda, tale processo crea automaticamente credenziali temporanee. Le credenziali temporanee vengono create in automatico anche quando accedi alla console come utente e poi cambi ruolo. Per ulteriori informazioni sullo scambio dei ruoli, consulta Passaggio da un ruolo utente a un ruolo IAM (console) nella Guida per l'utente IAM.

È possibile creare manualmente credenziali temporanee utilizzando l'API or. AWS CLI AWS È quindi possibile utilizzare tali credenziali temporanee per accedere. AWS AWS consiglia di generare dinamicamente credenziali temporanee anziché utilizzare chiavi di accesso a lungo termine. Per ulteriori informazioni, consulta Credenziali di sicurezza provvisorie in IAM.

Autorizzazioni principali multiservizio per FreerTOS

Supporta l'inoltro delle sessioni di accesso (FAS): sì

Quando utilizzi un utente o un ruolo IAM per eseguire azioni AWS, sei considerato un principale. Quando si utilizzano alcuni servizi, è possibile eseguire un'operazione che attiva un'altra operazione in un servizio diverso. FAS utilizza le autorizzazioni del principale che chiama an Servizio AWS, in combinazione con la richiesta Servizio AWS per effettuare richieste ai servizi downstream. Le richieste FAS vengono effettuate solo quando un servizio riceve una richiesta che richiede interazioni con altri Servizi AWS o risorse per essere completata. In questo caso è necessario disporre delle autorizzazioni per eseguire entrambe le azioni. Per i dettagli delle policy relative alle richieste FAS, consulta Forward access sessions.

Ruoli di servizio per FreerTOS

Supporta i ruoli di servizio: sì

Un ruolo di servizio è un ruolo IAM che un servizio assume per eseguire operazioni per tuo conto. Un amministratore IAM può creare, modificare ed eliminare un ruolo di servizio dall'interno di IAM. Per ulteriori informazioni, consulta la sezione Create a role to delegate permissions to an Servizio AWS nella Guida per l'utente IAM.

> **Warning**
>
> La modifica delle autorizzazioni per un ruolo di servizio potrebbe interrompere la funzionalità di FreerTOS. Modifica i ruoli di servizio solo quando FreerTOS fornisce una guida per farlo.

Ruoli collegati ai servizi per FreerTOS

Supporta i ruoli collegati ai servizi: no

Un ruolo collegato al servizio è un tipo di ruolo di servizio collegato a un. Servizio AWS Il servizio può assumere il ruolo per eseguire un'azione per tuo conto. I ruoli collegati al servizio vengono visualizzati nel tuo account Account AWS e sono di proprietà del servizio. Un amministratore IAM può visualizzare le autorizzazioni per i ruoli collegati ai servizi, ma non modificarle.

Per ulteriori informazioni su come creare e gestire i ruoli collegati ai servizi, consulta Servizi AWS supportati da IAM. Trova un servizio nella tabella che include un `Yes` nella colonna Service-linked

role (Ruolo collegato ai servizi). Scegli il collegamento Sì per visualizzare la documentazione relativa al ruolo collegato ai servizi per tale servizio.

Esempi di policy basate sull'identità per FreerTOS

Per impostazione predefinita, gli utenti e i ruoli non hanno il permesso di creare o modificare risorse FreerTOS. Inoltre, non possono eseguire attività utilizzando AWS Management Console, AWS Command Line Interface (AWS CLI) o AWS API. Per concedere agli utenti l'autorizzazione a eseguire operazioni sulle risorse di cui hanno bisogno, un amministratore IAM può creare policy IAM. L'amministratore può quindi aggiungere le policy IAM ai ruoli e gli utenti possono assumere i ruoli.

Per informazioni su come creare una policy basata su identità IAM utilizzando questi documenti di policy JSON di esempio, consulta Creazione di policy IAM (console) nella Guida per l'utente IAM.

Per i dettagli sulle azioni e sui tipi di risorse definiti da FreeRTOS, incluso il formato di per ciascuno ARNs dei tipi di risorse, vedere Azioni, risorse e chiavi di condizione per FreerTOS nel Service Authorization Reference.

Argomenti

- Best practice per le policy
- Utilizzo della console FreerTOS
- Consentire agli utenti di visualizzare le loro autorizzazioni

Best practice per le policy

Le politiche basate sull'identità determinano se qualcuno può creare, accedere o eliminare risorse FreerTOS nel tuo account. Queste azioni possono comportare costi aggiuntivi per l' Account AWS. Quando crei o modifichi policy basate su identità, segui queste linee guida e raccomandazioni:

- Inizia con le policy AWS gestite e passa alle autorizzazioni con privilegi minimi: per iniziare a concedere autorizzazioni a utenti e carichi di lavoro, utilizza le politiche gestite che concedono le autorizzazioni per molti casi d'uso comuni.AWS Sono disponibili nel tuo. Account AWS Ti consigliamo di ridurre ulteriormente le autorizzazioni definendo politiche gestite dai AWS clienti specifiche per i tuoi casi d'uso. Per ulteriori informazioni, consulta Policy gestite da AWSo Policy gestite da AWS per le funzioni dei processi nella Guida per l'utente IAM.
- Applica le autorizzazioni con privilegio minimo: quando imposti le autorizzazioni con le policy IAM, concedi solo le autorizzazioni richieste per eseguire un'attività. È possibile farlo definendo le azioni

che possono essere intraprese su risorse specifiche in condizioni specifiche, note anche come autorizzazioni con privilegi minimi. Per ulteriori informazioni sull'utilizzo di IAM per applicare le autorizzazioni, consulta Policy e autorizzazioni in IAM nella Guida per l'utente IAM.

- Condizioni d'uso nelle policy IAM per limitare ulteriormente l'accesso: per limitare l'accesso a operazioni e risorse è possibile aggiungere una condizione alle tue policy. Ad esempio, è possibile scrivere una condizione di policy per specificare che tutte le richieste devono essere inviate utilizzando SSL. Puoi anche utilizzare le condizioni per concedere l'accesso alle azioni del servizio se vengono utilizzate tramite uno specifico Servizio AWS, ad esempio AWS CloudFormation. Per ulteriori informazioni, consulta la sezione Elementi delle policy JSON di IAM: condizione nella Guida per l'utente IAM.
- Utilizzo di IAM Access Analyzer per convalidare le policy IAM e garantire autorizzazioni sicure e funzionali: IAM Access Analyzer convalida le policy nuove ed esistenti in modo che aderiscano alla sintassi della policy IAM (JSON) e alle best practice di IAM. IAM Access Analyzer offre oltre 100 controlli delle policy e consigli utili per creare policy sicure e funzionali. Per ulteriori informazioni, consulta Convalida delle policy per il Sistema di analisi degli accessi IAM nella Guida per l'utente IAM.
- Richiedi l'autenticazione a più fattori (MFA): se hai uno scenario che richiede utenti IAM o un utente root nel Account AWS tuo, attiva l'MFA per una maggiore sicurezza. Per richiedere la MFA quando vengono chiamate le operazioni API, aggiungi le condizioni MFA alle policy. Per ulteriori informazioni, consulta Protezione dell'accesso API con MFA nella Guida per l'utente IAM.

Per maggiori informazioni sulle best practice in IAM, consulta Best practice di sicurezza in IAM nella Guida per l'utente di IAM.

Utilizzo della console FreerTOS

Per accedere alla console FreerTOS, devi disporre di un set minimo di autorizzazioni. Queste autorizzazioni devono permetterti di elencare e visualizzare i dettagli sulle risorse FreerTOS presenti nel tuo. Account AWS Se crei una policy basata sull'identità più restrittiva rispetto alle autorizzazioni minime richieste, la console non funzionerà nel modo previsto per le entità (utenti o ruoli) associate a tale policy.

Non è necessario consentire autorizzazioni minime per la console agli utenti che effettuano chiamate solo verso o l' AWS CLI API. AWS Al contrario, concedi l'accesso solo alle operazioni che corrispondono all'operazione API che stanno cercando di eseguire.

Per garantire che utenti e ruoli possano ancora utilizzare la console FreeRTOS, collega anche FreeRTOS o la policy gestita alle *ConsoleAccess* entità *ReadOnly* AWS . Per ulteriori informazioni, consulta Aggiunta di autorizzazioni a un utente nella Guida per l'utente IAM.

Consentire agli utenti di visualizzare le loro autorizzazioni

Questo esempio mostra in che modo è possibile creare una policy che consente agli utenti IAM di visualizzare le policy inline e gestite che sono cpllegate alla relativa identità utente. Questa politica include le autorizzazioni per completare questa azione sulla console o utilizzando programmaticamente l'API o. AWS CLI AWS

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
```

```
}
```

Risoluzione dei problemi relativi all'identità e all'accesso a FreerTOS

Utilizza le seguenti informazioni per aiutarti a diagnosticare e risolvere i problemi più comuni che potresti incontrare quando lavori con FreerTOS e IAM.

Argomenti

- Non sono autorizzato a eseguire un'azione in FreerTOS
- Non sono autorizzato a eseguire iam: PassRole
- Voglio consentire a persone esterne a me di accedere Account AWS alle mie risorse FreerTOS

Non sono autorizzato a eseguire un'azione in FreerTOS

Se ricevi un errore che indica che non sei autorizzato a eseguire un'operazione, le tue policy devono essere aggiornate per poter eseguire l'operazione.

L'errore di esempio seguente si verifica quando l'utente IAM `mateojackson` prova a utilizzare la console per visualizzare i dettagli relativi a una risorsa *`my-example-widget`* fittizia ma non dispone di autorizzazioni `awes:`*`GetWidget`* fittizie.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 awes:GetWidget on resource: my-example-widget
```

In questo caso, la policy per l'utente `mateojackson` deve essere aggiornata per consentire l'accesso alla risorsa *`my-example-widget`* utilizzando l'azione `awes:`*`GetWidget`*.

Se hai bisogno di aiuto, contatta il tuo AWS amministratore. L'amministratore è la persona che ti ha fornito le credenziali di accesso.

Non sono autorizzato a eseguire iam: PassRole

Se ricevi un messaggio di errore indicante che non sei autorizzato a eseguire l'`iam:PassRole`azione, le tue policy devono essere aggiornate per consentirti di passare un ruolo a FreerTOS.

Alcuni Servizi AWS consentono di passare un ruolo esistente a quel servizio invece di creare un nuovo ruolo di servizio o un ruolo collegato al servizio. Per eseguire questa operazione, è necessario disporre delle autorizzazioni per trasmettere il ruolo al servizio.

Il seguente errore di esempio si verifica quando un utente IAM `marymajor` di nome tenta di utilizzare la console per eseguire un'azione in FreerTOS. Tuttavia, l'azione richiede che il servizio disponga delle autorizzazioni concesse da un ruolo di servizio. Mary non dispone delle autorizzazioni per passare il ruolo al servizio.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In questo caso, le policy di Mary devono essere aggiornate per poter eseguire l'operazione `iam:PassRole`.

Se hai bisogno di aiuto, contatta il tuo AWS amministratore. L'amministratore è la persona che ti ha fornito le credenziali di accesso.

Voglio consentire a persone esterne a me di accedere Account AWS alle mie risorse FreerTOS

È possibile creare un ruolo con il quale utenti in altri account o persone esterne all'organizzazione possono accedere alle tue risorse. È possibile specificare chi è attendibile per l'assunzione del ruolo. Per i servizi che supportano politiche basate sulle risorse o liste di controllo degli accessi (ACLs), puoi utilizzare tali politiche per concedere alle persone l'accesso alle tue risorse.

Per ulteriori informazioni, consulta gli argomenti seguenti:

- Per sapere se FreerTOS supporta queste funzionalità, consulta. Come funziona FreerTOS con IAM
- Per scoprire come fornire l'accesso alle tue risorse su tutto Account AWS ciò che possiedi, consulta Fornire l'accesso a un utente IAM in un altro Account AWS di tua proprietà nella IAM User Guide.
- Per scoprire come fornire l'accesso alle tue risorse a terze parti Account AWS, consulta Fornire l'accesso a soggetti Account AWS di proprietà di terze parti nella Guida per l'utente IAM.
- Per informazioni su come fornire l'accesso tramite la federazione delle identità, consulta Fornire l'accesso a utenti autenticati esternamente (Federazione delle identità) nella Guida per l'utente IAM.

- Per informazioni sulle differenze di utilizzo tra ruoli e policy basate su risorse per l'accesso multi-account, consulta Accesso a risorse multi-account in IAM nella Guida per l'utente IAM.

Convalida della conformità

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

Per sapere se un Servizio AWS programma rientra nell'ambito di specifici programmi di conformità, consulta Servizi AWS la sezione Scope by Compliance Program Servizi AWS e scegli il programma di conformità che ti interessa. Per informazioni generali, consulta Programmi di AWS conformità Programmi di di .

È possibile scaricare report di audit di terze parti utilizzando AWS Artifact. Per ulteriori informazioni, consulta Scaricamento dei report in AWS Artifact .

La vostra responsabilità di conformità durante l'utilizzo Servizi AWS è determinata dalla sensibilità dei dati, dagli obiettivi di conformità dell'azienda e dalle leggi e dai regolamenti applicabili. AWS fornisce le seguenti risorse per contribuire alla conformità:

- Governance e conformità per la sicurezza: queste guide all'implementazione di soluzioni illustrano considerazioni relative all'architettura e i passaggi per implementare le funzionalità di sicurezza e conformità.
- Riferimenti sui servizi conformi ai requisiti HIPAA: elenca i servizi HIPAA idonei. Non tutti Servizi AWS sono idonei alla normativa HIPAA.
- AWS Risorse per la per la conformità: questa raccolta di cartelle di lavoro e guide potrebbe essere valida per il tuo settore e la tua località.
- AWS Guide alla conformità dei clienti: comprendi il modello di responsabilità condivisa attraverso la lente della conformità. Le guide riassumono le migliori pratiche per la protezione Servizi AWS e mappano le linee guida per i controlli di sicurezza su più framework (tra cui il National Institute of Standards and Technology (NIST), il Payment Card Industry Security Standards Council (PCI) e l'International Organization for Standardization (ISO)).

- Valutazione delle risorse con regole nella Guida per gli AWS Config sviluppatori: il AWS Config servizio valuta la conformità delle configurazioni delle risorse alle pratiche interne, alle linee guida e alle normative del settore.
- AWS Security Hub— Ciò Servizio AWS fornisce una visione completa dello stato di sicurezza interno. AWS La Centrale di sicurezza utilizza i controlli di sicurezza per valutare le risorse AWS e verificare la conformità agli standard e alle best practice del settore della sicurezza. Per un elenco dei servizi e dei controlli supportati, consulta la pagina Documentazione di riferimento sui controlli della Centrale di sicurezza.
- Amazon GuardDuty: Servizio AWS rileva potenziali minacce ai tuoi carichi di lavoro Account AWS, ai contenitori e ai dati monitorando l'ambiente alla ricerca di attività sospette e dannose. GuardDuty può aiutarti a soddisfare vari requisiti di conformità, come lo standard PCI DSS, soddisfacendo i requisiti di rilevamento delle intrusioni imposti da determinati framework di conformità.
- AWS Audit Manager— Ciò Servizio AWS consente di verificare continuamente l' AWS utilizzo per semplificare la gestione del rischio e la conformità alle normative e agli standard di settore.

Resilienza in AWS

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

L'infrastruttura AWS globale è costruita attorno a AWS regioni e zone di disponibilità. AWS forniscono più zone di disponibilità fisicamente separate e isolate che sono connesse tramite reti altamente ridondanti, a bassa latenza e con throughput elevato. Con le zone di disponibilità, è possibile progettare e gestire applicazioni e database che eseguono il failover automatico tra zone di disponibilità senza interruzioni. Le zone di disponibilità sono più disponibili, tolleranti ai guasti e scalabili rispetto alle infrastrutture tradizionali a data center singolo o multiplo.

Per ulteriori informazioni su AWS regioni e zone di disponibilità, vedere Infrastruttura AWS globale.

Sicurezza dell'infrastruttura in FreerTOS

> **Important**
>
> Questa è una versione archiviata della FreeRTOS User Guide da utilizzare con la versione FreeRTOS 202012.00. Per l'ultima versione di questo documento, consulta la FreerTOS User Guide.

AWS i servizi gestiti sono protetti dalle procedure di sicurezza di rete AWS globali descritte nel white paper Amazon Web Services: Overview of Security Processes.

Utilizzi chiamate API AWS pubblicate per accedere ai AWS servizi attraverso la rete. I client devono supportare Transport Layer Security (TLS) 1.2 o versioni successive. È consigliabile TLS 1.3 o versioni successive. I client devono, inoltre, supportare le suite di crittografia con PFS (Perfect Forward Secrecy), ad esempio Ephemeral Diffie-Hellman (DHE) o Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). La maggior parte dei sistemi moderni, come Java 7 e versioni successive, supporta tali modalità.

Inoltre, le richieste devono essere firmate utilizzando un ID chiave di accesso e una chiave di accesso segreta associata a un principale IAM. In alternativa, è possibile utilizzare AWS Security Token Service (AWS STS) per generare le credenziali di sicurezza temporanee per sottoscrivere le richieste.

Le traduzioni sono generate tramite traduzione automatica. In caso di conflitto tra il contenuto di una traduzione e la versione originale in Inglese, quest'ultima prevarrà.