aws

Guida per l'utente

FreeRTOS

FreeRTOS: Guida per l'utente

Table of Contents

Cos'è FreerTOS?

Sviluppato in collaborazione con le principali società di chip del mondo per un periodo di 15 anni e ora scaricato ogni 170 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. È possibile scaricare il codice sorgente, apportare modifiche o miglioramenti o segnalare problemi sul GitHub sito all'indirizzo https://github.com/FreeRTOS/FreeRTOS.

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). Per visualizzare i sorgenti di markdown per la documentazione, consultate https://github.com/awsdocs/. aws-freertos-docs È rilasciato sotto licenza Creative Commons (CC BY-ND).

Scaricare il codice sorgente di FreerTOS

Scarica gli ultimi pacchetti FreeRTOS e Long Term Support (LTS) dalla pagina Download su freertos.org.

Controllo delle versioni FreerTOS

Le singole librerie utilizzano numeri di versione in stile x.y.z, simili alle versioni semantiche. x è il numero della versione principale, y il numero della versione secondaria e, a partire dal 2022, z è un numero di patch. Prima del 2022, z era un numero di release specifico, che richiedeva che le prime librerie LTS avessero un numero di patch del tipo «x.y.z LTS Patch 2".

I pacchetti di librerie utilizzano numeri di versione con datario in stile yyyymm.x. yyyy è l'anno, mm il mese e x un numero di sequenza opzionale che mostra l'ordine di rilascio entro il mese. Nel caso

del pacchetto LTS, x è un numero di patch sequenziale per quella versione LTS. Le singole librerie contenute in un pacchetto corrispondono alla versione più recente di quella libreria in quella data. Per il pacchetto LTS, è l'ultima versione patch delle librerie LTS originariamente rilasciata come versione LTS in quella data.

Supporto a lungo termine FreerTOS

Le versioni di FreerTOS Long Term Support (LTS) ricevono correzioni di bug di sicurezza e critici (se necessario) per almeno due anni dopo il loro rilascio. Con questa manutenzione continua, puoi incorporare correzioni di bug durante un ciclo di sviluppo e distribuzione senza le costose interruzioni dell'aggiornamento alle nuove versioni principali delle librerie FreerTOS.

Con FreerTOS LTS, ottieni il set completo di librerie necessarie per creare prodotti IoT e integrati connessi in modo sicuro. LTS aiuta a ridurre i costi di manutenzione e test associati all'aggiornamento delle librerie sui dispositivi già in produzione.

FreerTOS LTS include il kernel FreerTOS e le librerie IoT: FreerTOS+TCP, CoreMQTT, CoreHTTP, CorePKCS11, CoreJSON, OTA, Jobs e Device Shadow. AWS IoT AWS IoT AWS IoT Device Defender AWS IoT Per ulteriori informazioni, consulta le librerie FreerTOS LTS.

Piano di manutenzione esteso FreerTOS

AWSoffre anche FreerTOS Extended Maintenance Plan (EMP), che fornisce patch di sicurezza e correzioni di bug critici sulla versione FreerTOS Long Term Support (LTS) scelta per un massimo di dieci anni aggiuntivi. Con FreeRTOS EMP, i tuoi dispositivi di lunga durata basati su FreerTOS possono contare su una versione che ha stabilità delle funzionalità e riceve aggiornamenti di sicurezza per anni. Ricevi notifiche tempestive delle prossime patch sulle librerie FreerTOS, così puoi pianificare l'implementazione delle patch di sicurezza sui tuoi dispositivi Internet of Things (IoT).

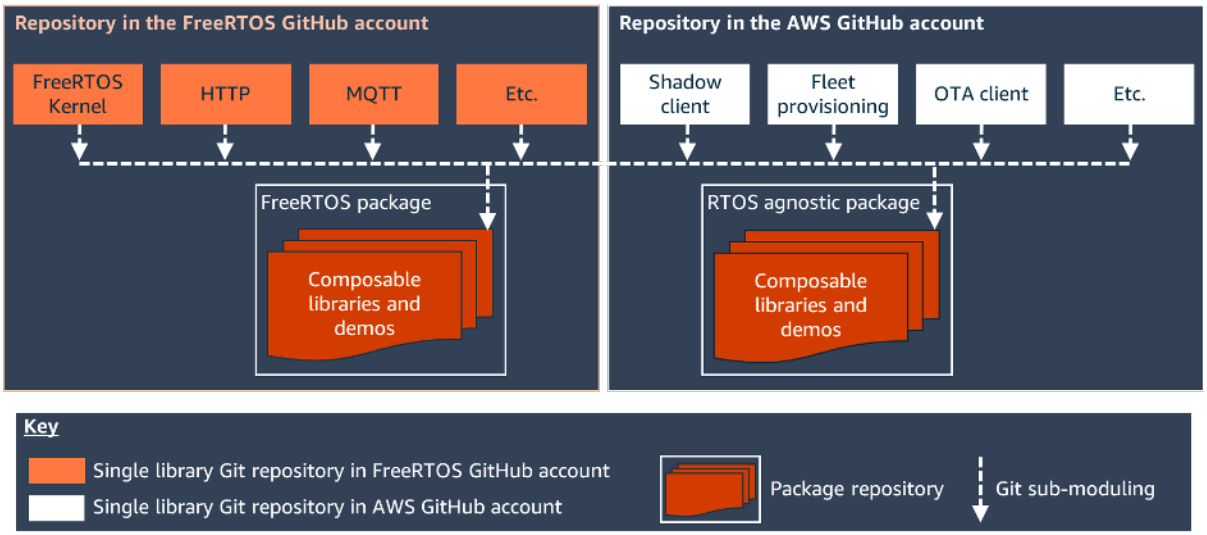
Per saperne di più su FreerTOS EMP, consulta la pagina Caratteristiche.

Architettura FreerTOS

FreerTOS contiene due tipi di repository, repository a libreria singola e repository di pacchetti. Ogni singolo repository di librerie contiene il codice sorgente di una libreria senza progetti o esempi di compilazione. I repository di pacchetti contengono più librerie e possono contenere progetti preconfigurati che dimostrano l'uso della libreria.

Sebbene gli archivi di pacchetti contengano più librerie, non contengono copie di tali librerie. Invece, i repository di pacchetti fanno riferimento alle librerie che contengono come sottomoduli git. L'uso dei sottomoduli garantisce l'esistenza di un'unica fonte di verità per ogni singola libreria.

I repository git delle singole librerie sono suddivisi tra due GitHub organizzazioni. I repository contenenti librerie specifiche di FreerTOS (come FreeRTOS+TCP) o librerie generiche (come CoreMQTT, che è indipendente dal cloud perché funziona con qualsiasi broker MQTT) si trovano nell'organizzazione FreerTOS. GitHub Nell'organizzazione sono presenti repository contenenti librerie specifiche (come il client di aggiornamento). AWS IoT AWS IoT over-the-air AWS GitHub Il diagramma seguente illustra la struttura.



Piattaforme hardware certificate FreeRTOS

Le seguenti piattaforme hardware sono qualificate per FreerTOS:

- Kit di provisioning Zero Touch ATECC608A per AWS IoT
- Kit di sviluppo Cypress CYW943907AEVAL1F
- Kit di sviluppo Cypress CYW954907AEVAL1F
- Kit Cypress CY8CKIT-064S0S2-4343W
- Espressif ESP32-C DevKit
- Espressif ESP-WROVER-KIT
- Espressif ESP-WROOM-32SE
- Espresso ESP32-S2-Saola-1
- Kit di connettività IoT Infineon XMC4800

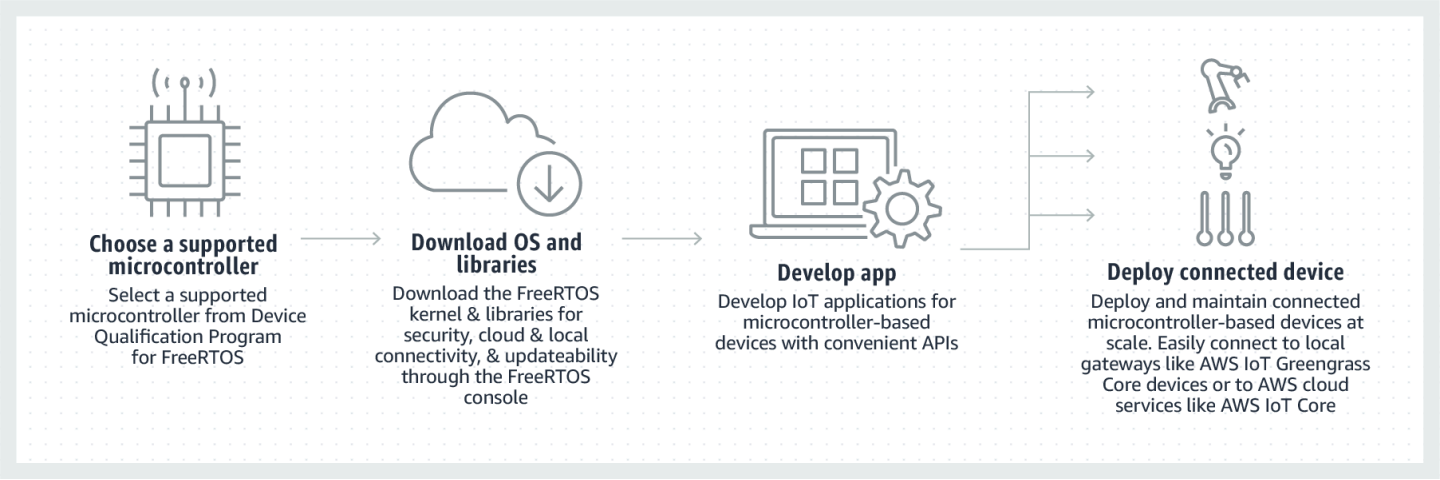
- Marvell MW320 AWS IoT Starter Kit
- Marvell MW322 AWS IoT Starter Kit
- MediaTek Kit di sviluppo MT7697Hx
- Pacchetto Microchip Curiosity PIC32MZEF
- Nordic nRF52840-DK
- NuMaker-IoT-M487
- Modulo IoT NXP LPC54018
- Soluzione di sicurezza OPTIGA Trust X
- Modulo IoT Renesas RX65N RSK
- Nodo IoT STMicroelectronics STM32L4 Discovery Kit
- Texas Instruments CC3220SF-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 distribuiti possono connettersi al servizio AWS IoT o AWS IoT Greengrass come parte di una soluzione IoT completa.



Risorse aggiuntive

Queste risorse possono essere utili per l'utente.

- Per ulteriore documentazione su FreerTOS, vedere freertos.org.
- Per domande su FreeRTOS per il team di ingegneri di FreeRTOS, puoi aprire un problema nella pagina FreeRTOS. GitHub
- Per domande tecniche su FreerTOS, consulta i FreerTOS Community Forum.
- Per ulteriori informazioni sulla connessione dei dispositivi aAWS IoT, consulta Device Provisioning nella Device Guide. AWS IoT Core
- Per il supporto tecnico perAWS, consulta AWSil Centro assistenza.
- Per domande su AWS fatturazione, servizi relativi all'account, eventi, abusi o altri problemi relativiAWS, consulta la pagina Contattaci.

Aspetti fondamentali del kernel FreeRTOS

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.
- Support per il multiprocessing simmetrico (SMP) su microcontrollori multi-core.

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. I task bloccati non richiedono lunghe operazioni di manutenzione periodica. irect-to-task Le notifiche D consentono una segnalazione rapida delle attività, praticamente senza sovraccarico di RAM. Possono essere utilizzati nella maggior parte degli scenari di interattività 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, consulta 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).

Le API che agiscono sulle code permettono 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, come direct-to-task 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.

irect-to-task Notifiche D

Le notifiche dei task consentono l'interazione tra task e la relativa sincronizzazione con le interrupt service routine (ISR), 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 del buffer di flusso 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.

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`.

Supporto SMP (Symmetric Multiprocessing)

Il supporto SMP nel kernel FreeRTOS consente a un'istanza del kernel FreeRTOS di pianificare le attività su più core di processore identici. Le architetture di base devono essere identiche e condividere la stessa memoria.

Modifica delle applicazioni per utilizzare il kernel FreeRTOS-SMP

L'API FreeRTOS rimane sostanzialmente la stessa tra le versioni single-core e SMP, ad eccezione di queste API aggiuntive. Pertanto, un'applicazione scritta per la versione single-core di FreeRTOS deve essere compilata con la versione SMP con uno sforzo minimo o nullo. Tuttavia, potrebbero esserci alcuni problemi funzionali, perché alcune ipotesi valide per le applicazioni single-core potrebbero non essere più valide per le applicazioni multi-core.

Un presupposto comune è che un'attività con priorità inferiore non possa essere eseguita mentre è in esecuzione un'attività con priorità più alta. Sebbene ciò fosse vero su un sistema single-core, non è più vero per i sistemi multi-core perché più attività possono essere eseguite contemporaneamente. Se l'applicazione si basa su priorità relative delle attività per garantire l'esclusione reciproca, potrebbe ottenere risultati imprevisti in un ambiente multicore.

Un altro presupposto comune è che gli ISR non possano essere eseguiti contemporaneamente tra loro o con altre attività. Questo non è più vero in un ambiente multi-core. L'autore dell'applicazione deve garantire un'adeguata esclusione reciproca durante l'accesso ai dati condivisi tra attività e ISR.

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.

IL`FreeRTOSConfig.h` file per la demo e le applicazioni di test di FreeRTOS si trovano in*`freertos`*/`vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/FreeRTOSConfig.h` e*`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 Device SDK for Embedded C

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

SDK per dispositivi AWS IoT per Embedded C (C-SDK) è una raccolta di file di origine C sotto la licenza open source MIT che è possibile utilizzare nelle applicazioni integrate per la connessione sicura dei dispositivi IoT ad AWS IoT Core. Include un client MQTT, un client HTTP, un parser JSON,AWS IoT Device Shadow,AWS IoT Jobs, il provisioning delAWS IoT parco istanze eAWS IoT Device Defender le librerie. 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 vincoli di risorse che richiedono un runtime del linguaggio C ottimizzato. Puoi utilizzare l'SDK su qualsiasi sistema operativo e ospitarlo su qualsiasi tipo di processore (ad esempio MCU e MPU). Tuttavia, se i tuoi dispositivi dispongono di memoria e risorse di elaborazione sufficienti, ti consigliamo di utilizzare uno dei AWS IoTDevice SDK di livello superiore.

Per ulteriori informazioni, consulta gli argomenti seguenti:

- AWS IoT Device SDK for Embedded C
- AWS IoTDevice SDK per EmbedC SDK per EmbedC SDK per GitHub
- File Readme dell'SDK di dispositivo AWS IoT per Embedded C
- SDK per dispositivo AWS IoT per Embedded C Esempi

I/O comune

Le API IO comuni fungono da livelli di astrazione hardware (HAL) che forniscono un'interfaccia comune tra driver e codice applicativo di livello superiore. FreeRTOS Common IO fornisce una serie di API standard per l'accesso a dispositivi seriali comuni su schede di riferimento supportate; le implementazioni di queste API non sono incluse. Queste API comuni comunicano e interagiscono con queste periferiche e consentono al codice di funzionare tra le piattaforme. Senza Common IO, la scrittura di codice per funzionare con dispositivi di basso livello è specifica del fornitore di silicio.

> **Note**
>
> FreeRTOS non richiede implementazioni delle API Common IO per funzionare, ma tenterà di utilizzare le API Common IO come modo per interfacciarsi con le periferiche specifiche su una scheda basata su microcontrollore anziché con le API specifiche del fornitore.

In generale, i driver dei dispositivi sono indipendenti dal sistema operativo sottostante e sono specifici per una determinata configurazione hardware. L'HAL estrae i dettagli del funzionamento di un driver specifico e fornisce un'API uniforme per controllare tali dispositivi. È possibile utilizzare le stesse API per accedere a vari driver di dispositivo su più schede di riferimento basate su microcontrollori (MCU).

Libraries (Librerie)

Attualmente, FreeRTOS fornisce due librerie IO comuni: Common IO - basic e Common IO - BLE.

Common IO - base

Panoramica

Common IO - basic fornisce API che gestiscono le periferiche e le funzioni I/O di base che si possono trovare sulle schede basate su MCU. L'archivio Common IO - basic è disponibile su. GitHub

Periferiche supportate

- AGGIUNGERE
- GPIO

- I2C
- PWM
- SPI
- UART
- Cane da guardia
- Flash
- RTC
- RIFIUTARE
- Reimposta
- I2S
- Contatore delle prestazioni
- Informazioni sulla piattaforma hardware

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

  Trasferisci i dati tra primario e secondario, il che significa che la scrittura e la lettura avvengono contemporaneamente.

Documentazione di riferimento dell'API

Per un riferimento completo alle API, consulta Common IO - Basic API reference.

IO comune - BLE

Panoramica

Common IO - BLE fornisce un'astrazione dallo stack Bluetooth Low Energy del produttore. Fornisce le seguenti interfacce che possono essere utilizzate per controllare il dispositivo ed eseguire operazioni GAP e GATT. L'archivio Common IO - BLE è disponibile su. GitHub

Gestione dispositivi Bluetooth:

Ciò fornisce un'interfaccia per controllare il dispositivo Bluetooth, eseguire operazioni di rilevamento del dispositivo e altre attività relative alla connettività.

Gestore adattatori BLE:

Ciò fornisce un'interfaccia per le funzioni dell'API GAP specifiche di BLE.

Gestore adattatori Bluetooth Classic:

Ciò fornisce un'interfaccia per controllare le funzionalità classiche di BT di un dispositivo.

Server GATT:

Ciò fornisce un'interfaccia per utilizzare la funzionalità del server Bluetooth GATT.

Cliente GATT:

Ciò fornisce un'interfaccia per utilizzare la funzionalità client Bluetooth GATT.

Interfaccia di connessione A2DP:

Ciò fornisce un'interfaccia per il profilo A2DP Source per il dispositivo locale.

Documentazione di riferimento dell'API

Per un riferimento completo all'API, consulta il riferimento API Common IO - BLE.

I/O comune per Amazon Common Software

Le API Common IO fanno parte delle implementazioni richieste da Amazon Common Software for Devices, in particolare per essere implementate in un vendor device porting kit (DPK).

Che cos'è ACS?

Amazon Common Software (ACS) for Devices è un software che velocizza l'integrazione degli SDK Amazon Device sui tuoi dispositivi. ACS fornisce un livello di integrazione API unificato, componenti

preconvalidati ed efficienti in termini di memoria per funzioni comuni come connettività, un kit di portabilità dei dispositivi (DPK) e suite di test a più livelli.

Programma di qualificazione

Il programma di qualificazione Amazon Common Software for Devices verifica che una build dell'ACS DPK (Device Porting Kit) eseguita su una specifica scheda di sviluppo basata su microcontrollore sia compatibile con le best practice pubblicate dal programma e sufficientemente robusta da superare i test obbligatori ACS specificati dal programma di qualificazione.

I fornitori qualificati nell'ambito di questo programma sono elencati nella pagina Fornitori di chipset ACS.

Per informazioni sulla qualificazione, contatta ACS for Devices.

Nozioni di base su FreeRTOS

Argomenti:

- Guida introduttivaAWS IoT e FreeRTOS con Quick Connect
- Esplorazione di FreeRTOS
- Scopri come creare unAWS IoT prodotto sicuro e robusto
- Sviluppa il tuo prodottoAWS IoT applicativo

Guida introduttivaAWS IoT e FreeRTOS con Quick Connect

Per esplorare rapidamenteAWS IoT, inizia con AWSQuick Connect Demos. Le demo Quick Connect sono semplici da configurare e connettere una scheda certificata FreeRTOS fornita da un partner AWS IoT.

Segui il tutorial AWS IoTintroduttivo per una migliore comprensione della consoleAWS IoT e dellaAWS IoT console. Puoi modificare il codice sorgente della demo fornito con le demo Quick Connect utilizzando il sistema di compilazione e gli strumenti della scheda scelta per connetterti al tuoAWS account. Il flusso di dati dallaAWS IoT console del tuo account è ora visibile.

Esplorazione di FreeRTOS

Una volta capito comeAWS IoT lavorare insieme un dispositivo IoT, puoi iniziare a esplorare le librerie FreeRTOS e le librerie LTS (Long-Term Support).

Alcune librerie comunemente utilizzate per iAWS IoT dispositivi basati su FreeRTOS sono:

- Kernel FreeRTOS
- Core MQTT
- AWS IoTvia etere (OTA)

Visita freertos.org per la documentazione tecnica e le demo specifiche della libreria.

Scopri come creare unAWS IoT prodotto sicuro e robusto

Consulta AWS IoTle integrazioni FreeRTOS in evidenza per conoscere le migliori pratiche per rendere il software dei dispositivi IoT più sicuro e robusto. Queste integrazioni IoT FreeRTOS sono progettate per una maggiore sicurezza utilizzando una combinazione di software FreeRTOS e una scheda fornita dai partner con funzionalità di sicurezza hardware. Usali in produzione così come sono o usali come modello per i tuoi progetti.

Sviluppa il tuo prodottoAWS IoT applicativo

Segui questi passaggi per creare un progetto applicativo per il tuoAWS IoT prodotto:

1. Scarica la versione più recente di FreeRTOS o Long Term Support (LTS) da freertos.org o clona dal GitHub repository FreeRTOS-LTS. Puoi anche integrare le librerie FreeRTOS richieste nel tuo progetto dalla toolchain del fornitore MCU, se disponibile.
2. Segui la guida FreeRTOS Porting per creare un progetto, configurare l'ambiente di sviluppo e integrare le librerie FreeRTOS nel tuo progetto. Usa il GitHub repository FreeRTOS-Libraries-Integration-tests per convalidare il porting.

AWS IoT Device Testerper FreeRTOS

L'IDT for FreeRTOS è uno strumento per qualificare la velocità di trasmissione dei dati con il sistema operativo FreeRTOS. Il tester del dispositivo (IDT) apre innanzitutto una connessione USB o UART a un dispositivo. Quindi lampeggia un'immagine di FreeRTOS configurato per testare la funzionalità del dispositivo in varie condizioni. AWS IoT Device Testerle suite sono estensibili e IDT viene utilizzato per l'orchestrazione dei AWS IoT test dei clienti.

IDT for FreeRTOS viene eseguito su un computer host (Windows, macOS o Linux) collegato al dispositivo da testare. IDT configura e orchestra i casi di test e aggrega i risultati. Inoltre offre un'interfaccia a riga di comando per gestire l'esecuzione dei test.

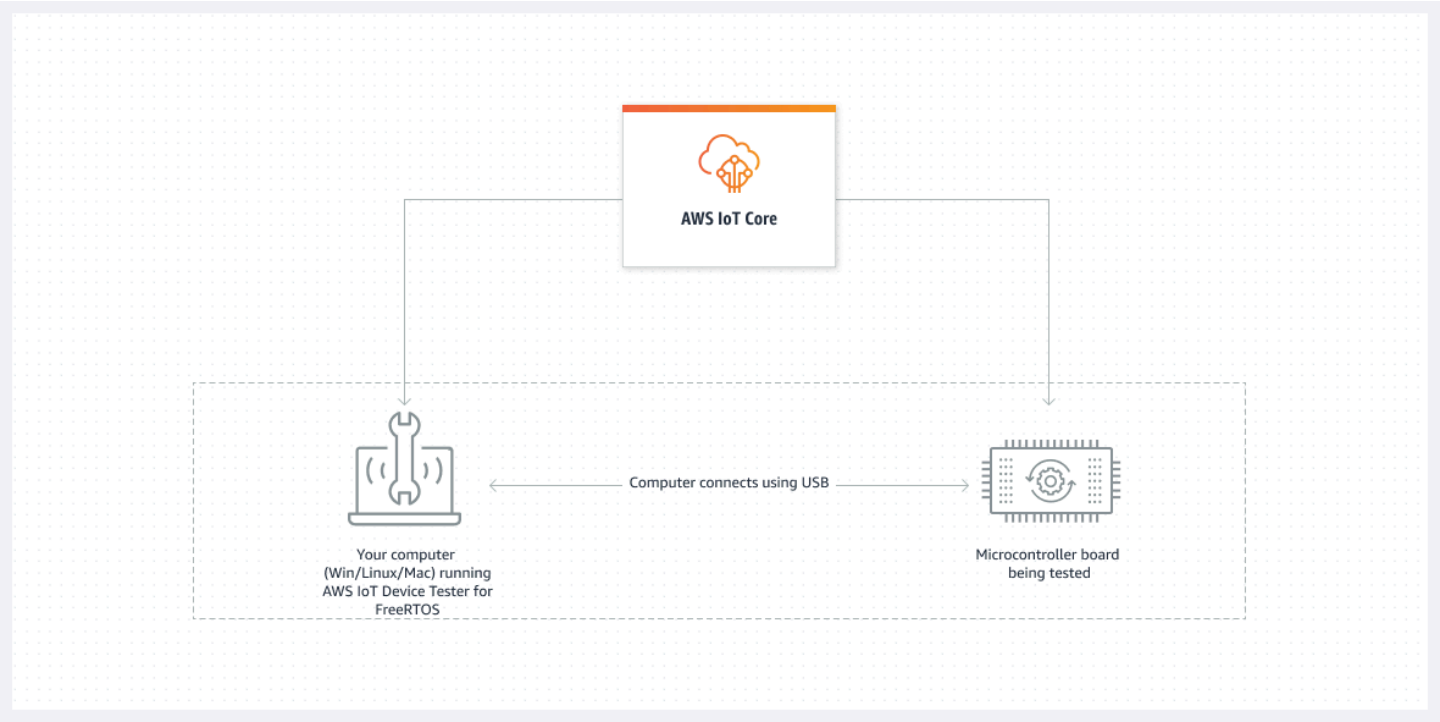
Suite di qualificazione FreeRTOS

IDT for FreeRTOS verifica la porta di FreeRTOS sul microcontrollore e se può comunicare efficacemente con AWS IoT FreeRTOS in modo affidabile e sicuro. In particolare, verifica se le interfacce del livello di porting per le librerie FreeRTOS sono implementate correttamente. Esegue inoltre test end-to-end con AWS IoT Core. Ad esempio, verifica se la tua scheda è in grado di inviare e ricevere messaggi MQTT ed elaborarli correttamente.

La suite FreeRTOS qualification (FRQ) 2.0 utilizza i casi di test di FreeRTOS-Libraries-Integration-Tests e Device Advisor definiti nella FreeRTOS Qualification Guide.

IDT for FreeRTOS genera rapporti di test che puoi inviare a AWS Partner Network (APN) per l'inclusione dei tuoi dispositivi FreeRTOS nel Partner Device Catalog. AWS Per ulteriori informazioni, consulta AWS Device Qualification Program.

Il diagramma seguente mostra la configurazione dell'infrastruttura di test per la qualificazione di FreeRTOS.



IDT for FreeRTOS organizza le risorse di 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 casi di test relativi a una particolare funzionalità, come la messaggistica BLE e MQTT.

Per ulteriori informazioni, consulta Versioni della suite di test.

Suite di test personalizzate

IDT for FreeRTOS combina una configurazione standardizzata e un formato dei risultati con un ambiente di suite di test. Questo ambiente consente di sviluppare suite di test personalizzate per i dispositivi e il software del dispositivo. Puoi aggiungere test personalizzati per la tua convalida interna o fornirli ai tuoi clienti per la verifica del dispositivo.

La modalità di configurazione delle suite di test personalizzate determina le configurazioni di impostazione che è necessario fornire agli utenti per eseguire le suite di test personalizzate. Per ulteriori informazioni, consulta Usa IDT per sviluppare ed eseguire le tue suite di test.

Versioni supportate diAWS IoT Device Testerper FreerTOS

Questo argomento elenca le versioni supportate diAWS IoT Device Testerper FreerTOS. Come best practice, ti consigliamo di utilizzare l'ultima versione di IDT for FreerTOS che supporti la versione di destinazione di FreerTOS. Ogni versione di IDT per FreerTOS ha una o più versioni corrispondenti di FreerTOS che supporta. Ti consigliamo di scaricare una nuova versione di IDT per FreerTOS quando viene rilasciata una nuova versione di FreeRTOS.

Scaricando il software, l'utente accettaAWS IoT Device TesterContratto di licenza contenuto nell'archivio dei download.

> **Note**
>
> Quando si utilizzaAWS IoT Device Testerper FreerTOS, ti consigliamo di eseguire l'aggiornamento all'ultima patch della versione più recente di FreeRTOS-LTS.

> **Important**
>
> A partire da ottobre 2022,AWS IoT Device TesterperAWS IoTFreerTOS Qualification (FRQ) 1.0 non genera report di qualificazione firmati. Non puoi qualificarti come nuovoAWS IoTDispositivi FreerTOS da elencare nelAWSCatalogo di dispositivi per i partnerattraverso ilAWSProgramma di qualificazione dei dispositiviutilizzando le versioni IDT FRQ 1.0. Sebbene non sia possibile qualificare i dispositivi FreerTOS utilizzando IDT FRQ 1.0, è possibile continuare a testare i dispositivi FreerTOS con FRQ 1.0. Si consiglia di utilizzareIDT FRQ 2.0per qualificare ed elencare i dispositivi FreerTOS nelAWSCatalogo di dispositivi per i partner.

Versione più recente di AWS IoT Device Tester for FreeRTOS

Usa i seguenti link per scaricare le ultime versioni di IDT per FreerTOS.

Versione più recente di AWS IoT Device Tester for FreeRTOS

AWS IoT Device Tester version (Versione Python)	versioni della suite di test	Versioni FreerTOS supportate	Link per il download	Data di uscita	Note di rilascio
IDT v4.9.0	FRQ_2.5.0	• 202112,00 • 202212,00 • 202212,01 • Tutte le patch di FreerTOS 202210-LTS che utilizzano le librerie FreerTOS LTS.	• Linux • macOS • finestre	04 aprile 2023	• Supporta i test contro FreerTOS202112,20 e tutte le patch di FreerTOS202210-LTSche utilizza le librerie FreerTOS. VediREADME.md per ulteriori informazioni. Devi includere la versione patch per FreeRTOS-LTS nel tuo`manifest.yml` . • Tempo di esecuzione migliorato dei test OTA E2E. • Limita il numero di dispositi

AWS IoT Device Tester version (Versione Python)	versioni della suite di test	Versioni FreerTOS supportate	Link per il download	Data di uscita	Note di rilascio
					vi elencati in `device.json` on a 1. • Miglioramenti e correzioni di bug minori.

> **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.

Versioni IDT precedenti per FreerTOS

Sono supportate anche le seguenti versioni precedenti di IDT per FreerTOS.

Versioni precedenti diAWS IoT Device Testerper FreerTOS

AWS IoT Device Tester version (Versione Python)	versioni della suite di test	Versioni FreerTOS supportate	Link per il download	Data di uscita	Note di rilascio
IDT v4.8.1	FRQ_2.4.0	• 202112,00 • 202212,00	• Linux • macOS	23.01.23	• VediLEGGIMI.MDper

AWS IoT Device Tester version (Versione Python)	versioni della suite di test	Versioni FreerTOS supportate	Link per il download	Data di uscita	Note di rilascio
		• 202212.01 • Tutte le patch di FreerTOS 202210-LTS che utilizzano le librerie FreerTOS LTS.	• finestre		ulteriori informazioni. Devi includere la versione patch per FreeRTOS-LTS nel tuo`manifest.yml` . • Miglioramenti e correzioni di bug minori.

AWS IoT Device Tester version (Versione Python)	versioni della suite di test	Versioni FreerTOS supportate	Link per il download	Data di uscita	Note di rilascio
IDT versione 4.6.0	FRQ_2.3.0	• 202112,00 • 202212,00 • 202212,01 • 202210-LTS che utilizzano le librerie FreerTOS LTS.	• Linux • macOS • finestre	16.11.2022	• VediLEGGIMI.MDper ulteriori informazioni. Devi includere la versione patch per FreeRTOS-LTS nel tuo`manifest.yml` . • Per ulteriori informazioni su cosa è incluso nei FreerTOS202210-LT Sversione, vedi ilchangelog.mdfile su GitHub. • Aggiunge la possibilità di configurare ed eseguireA

AWS IoT Device Tester version (Versione Python)	versioni della suite di test	Versioni FreerTOS supportate	Link per il download	Data di uscita	Note di rilascio
					WS IoT Device Testerper FreerTOS tramite un'interfaccia utente basata sul web. VediUsa l'interfaccia utente IDT for FreeRTOS per eseguire la suite di qualificazione FreeRTOS 2.0 (FRQ 2.0)per iniziare. • Aggiunge un'opzione per conservare le copie modificate del codice

AWS IoT Device Tester version (Versione Python)	versioni della suite di test	Versioni FreerTOS supportate	Link per il download	Data di uscita	Note di rilascio
					sorgente creato e utilizzato in fase di esecuzione e per il debug post-test. Per ulteriori informazioni, consulta Configurazione delle impostazioni di compilazione, flashing e test. • Aggiunge il supporto IDT Client SDK per Java. Per ulteriori informazioni sull'IDT Client SDK, vedereUsa IDT per

AWS IoT Device Tester version (Versione Python)	versioni della suite di test	Versioni FreerTOS supportate	Link per il download	Data di uscita	Note di rilascio
					sviluppare ed eseguire le tue suite di test.

AWS IoT Device Tester version (Versione Python)	versioni della suite di test	Versioni FreerTOS supportate	Link per il download	Data di uscita	Note di rilascio
IDT versione 4.5.11	FRQ_2.2.0	• 202112,00 • 202212,00 • 202212,01 • 202210-LTS che utilizzano le librerie FreerTOS LTS.	• Linux • macOS • finestre	14.10.2022	• VediLEGGIMI.MDper ulteriori informazioni. Devi includere la versione patch per FreeRTOS-LTS nel tuo`manifest.yml` . • Per ulteriori informazioni su cosa è incluso nei FreerTOS202210-LT Sversione, vedi ilchangelog.mdfile su GitHub. • Miglioramenti e correzioni di bug minori.

Per ulteriori informazioni, consulta Politica di supporto per AWS IoT Device Tester for FreeRTOS.

Versioni IDT non supportate per FreeRTOS

Questa sezione elenca le versioni non supportate di IDT for FreeRTOS. Le versioni non supportate non ricevono correzioni di bug o aggiornamenti. Per ulteriori informazioni, consulta Politica di supporto per AWS IoT Device Tester for FreeRTOS.

Le seguenti versioni di IDT-FreeRTOS non sono più supportate.

Versioni non supportate di AWS IoT Device Tester for FreeRTOS

AWS IoT Device Tester version (Versione Python)	Versioni della suite di test	Versioni FreeRTOS supportate	Data di uscita	Note di rilascio
IDT v4.5.10	FRQ_2.1.4	• 12.00 2021 • 202012-LTS che utilizzano le librerie FreeRTOS LTS.	02/09/2022	• Per ulteriori informazioni su ciò che è incluso nella versione FreeRTOS 202012-LTS, consulta il file ChangeLog.md su. GitHub • È stato risolto un problema che interessava il gruppo di `OTA End to End` test. • Rimosso `FullTransportInterfacePlainText` dalla

AWS IoT Device Tester version (Versione Python)	Versioni della suite di test	Versioni FreeRTOS supportate	Data di uscita	Note di rilascio
				partecipa zione alle gare di qualifica zione. Il testo normale può ancora essere eseguito come gruppo di test di sviluppo utilizzando il `-\-group-id` flag. • Migliorata la registrazione e la leggibilità della console e dell'output dei file. • Miglioramenti e correzioni di bug minori.

AWS IoT Device Tester version (Versione Python)	Versioni della suite di test	Versioni FreeRTOS supportate	Data di uscita	Note di rilascio
IDT v4.5.9	FRQ_2.1.3	• 12.00 2021 • 202012.04 -LTS che utilizzano le librerie LTS di FreeRTOS.	2022.08.17	• Per ulteriori informazioni su ciò che è incluso nella versione FreeRTOS 202012.04 -LTS, consulta il file ChangeLog .md su. GitHub • È stato risolto un problema che interessava il gruppo di `FreeRTOSIntegrity` test. • Gruppo `FullCloudIoT` di test aggiornato rimuovendo il test case «MQTT Connect Exponenti al Backoff Retries».

AWS IoT Device Tester version (Versione Python)	Versioni della suite di test	Versioni FreeRTOS supportate	Data di uscita	Note di rilascio
				• Miglioramenti e correzioni di bug minori.

AWS IoT Device Tester version (Versione Python)	Versioni della suite di test	Versioni FreeRTOS supportate	Data di uscita	Note di rilascio
IDT v4.5.6	FRQ_2.1.2	• 12.00 2021 • 202012.04 -LTS che utilizzano le librerie LTS di FreeRTOS.	2022.06.29	• Per ulteriori informazioni su ciò che è incluso nella versione FreeRTOS 202012.04 -LTS, consulta il file ChangeLog .md su. GitHub • Aggiunge un nuovo gruppo di test `FullCloudIoT` con cui mette alla prova la schedaAWS IoT Core Device Advisor. • È stato risolto un problema che riguardava i casi di test OTA E2E. • Miglioramenti e correzioni di bug minori.

AWS IoT Device Tester version (Versione Python)	Versioni della suite di test	Versioni FreeRTOS supportate	Data di uscita	Note di rilascio
IDT v4.5.5	FRQ_2.1.1	• 12.00 2021 • 202012.04 -LTS che utilizzano le librerie LTS di FreeRTOS.	06/06/2022	• Per ulteriori informazioni su ciò che è incluso nella versione FreeRTOS 202012.04 -LTS, consulta il file ChangeLog .md su. GitHub • Aggiunge un nuovo gruppo di test `FullCloud IoT` con cui mette alla prova la schedaAWS IoT Core Device Advisor. • È stato risolto un problema che riguardava i casi di test FreeRTOSVersion e FreeRTOSIntegrity.

AWS IoT Device Tester version (Versione Python)	Versioni della suite di test	Versioni FreeRTOS supportate	Data di uscita	Note di rilascio
				• Miglioramenti e correzioni di bug minori.
IDT v4.5.5	FRQ_2.1.0	• 2021-07,00 • 12.00 2021 • 202012.04 -LTS che utilizzano le librerie LTS di FreeRTOS.	31/05/2022	• Per ulteriori informazioni su ciò che è incluso nella versione FreeRTOS 202012.04 -LTS, consulta il file ChangeLog .md su. GitHub • Aggiunge un nuovo gruppo di test `FullCloud IoT` con cui mette alla prova la schedaAWS IoT Core Device Advisor. • Miglioramenti e correzioni di bug minori.

AWS IoT Device Tester version (Versione Python)	Versioni della suite di test	Versioni FreeRTOS supportate	Data di uscita	Note di rilascio
IDT v4.5.4	FRQ_2.0.0	• 12.00 2021 • 202012.04 -LTS che utilizzano le librerie LTS di FreeRTOS.	2022.05.09	• Per ulteriori informazioni sugli elementi inclusi nella versione FreeRTOS 202012.04 -LTS, consulta il file ChangeLog .md all'indir izzo. GitHub • Rimuove l'obbligo di qualificare le schede utilizzando solo le versioni di Amazon FreeRTOS dal repositor y. `aws/amazo n-freerto s` GitHub • Miglioramenti e correzioni di bug minori.

AWS IoT Device Tester version (Versione Python)	Versioni della suite di test	Versioni FreeRTOS supportate	Data di uscita	Note di rilascio
IDT v4.5.2	FRQ_1.6.2	2021-07,00	25/01/2022	• Per ulteriori informazioni sugli elementi inclusi nella versione FreeRTOS 202107.00, consulta il file ChangeLog .md all'indirizzo. GitHub • Implementa il nuovo orchestratore di test IDT per la configurazione di suite di test personalizzate. Per ulteriori informazioni, consulta Configurazione dell'orchestratore di test IDT. • Miglioramenti e correzioni di bug minori.

AWS IoT Device Tester version (Versione Python)	Versioni della suite di test	Versioni FreeRTOS supportate	Data di uscita	Note di rilascio
IDT v4.0.3	FRQ_1.5.1	2020-12,00	2021.07.30	• Support per la qualificazione di dispositivi con credenziali bloccate su un modulo di sicurezza hardware. • Miglioramenti e correzioni di bug minori.

AWS IoT Device Tester version (Versione Python)	Versioni della suite di test	Versioni FreeRTOS supportate	Data di uscita	Note di rilascio
IDT v4.3.0	FRQ_1.6.1	2021-07,00	2021.07.26	• Per ulteriori informazioni sugli elementi inclusi nella versione FreeRTOS 202107.00, consulta il file ChangeLog .md all'indirizzo. GitHub • Aggiunge la possibilità di configurare ed eseguire AWS IoT Device Tester FreeRTOS tramite un'interfaccia utente basata sul Web. Vedi Usa l'interfaccia utente IDT for FreerTOS per eseguire la suite di qualificazione FreerTOS per iniziare.

AWS IoT Device Tester version (Versione Python)	Versioni della suite di test	Versioni FreeRTOS supportate	Data di uscita	Note di rilascio

IDT v4.1.0	FRQ_1.6.0	2021-07,00	2021.07.21	• Per ulteriori informazioni sugli elementi inclusi nella versione FreeRTOS 202107.00, consulta il file ChangeLog .md all'indirizzo. GitHub
				• Rimuove i seguenti casi di test dalla qualifica OTA:
				  • Agente OTA
				  • Nome del file mancante di OTA mancante
				  • Numero massimo di blocchi configurato OTA
				• Rimuove il gruppo di `Both` test OTA Dataplane

AWS IoT Device Tester version (Versione Python)	Versioni della suite di test	Versioni FreeRTOS supportate	Data di uscita	Note di rilascio
				dalla qualificazione OTA. Nel device.json file, la `OTADataPlaneProtocol` configurazione ora accetta solo `HTTP` o `MQTT` come valori supportati. • Implementa le seguenti modifiche alla `freertosFileConfiguration` configurazione nel userdata.json file per le modifiche al codice sorgente di FreeRTOS: • Cambia il nome del file specificato per

AWS IoT Device Tester version (Versione Python)	Versioni della suite di test	Versioni FreeRTOS supportate	Data di uscita	Note di rilascio
				`otaAgentTestsConfig` e `otaAgentDemosConfig` da `aws_ota_agent_config.h` a `ota_config.h`. • Aggiunge una nuova configurazione `otaDemosConfig` opzionale per specificare il percorso del file al nuovo `ota_demo_config.h` file. • Aggiunge un nuovo campo `testStartDelayms` `userdata.json` per

AWS IoT Device Tester version (Versione Python)	Versioni della suite di test	Versioni FreeRTOS supportate	Data di uscita	Note di rilascio
				specificare un ritardo tra il momento in cui un dispositivo viene lampeggiato per eseguire un gruppo di test FreeRTOS e l'avvio dei test. Il valore deve essere espresso in millisecondi. Questo ritardo può essere utilizzato per dare a IDT la possibilità di connettersi in modo che nessun output di test venga perso.

AWS IoT Device Tester version (Versione Python)	Versioni della suite di test	Versioni FreeRTOS supportate	Data di uscita	Note di rilascio
IDT v4.0.1	FRQ_1.4.1	2020-12,00	2021.01.19	• Per ulteriori informazioni sugli elementi inclusi nella versione FreeRTOS 202012.00, consulta il file ChangeLog .md in. GitHub • Introduce casi di test OTA (Over-the-air) E2E (end-to-end) aggiuntivi. • 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

AWS IoT Device Tester version (Versione Python)	Versioni della suite di test	Versioni FreeRTOS supportate	Data di uscita	Note di rilascio
				FreeRTOS utilizzando la connettività cellulare. • Risolve un bug nella configurazione del server echo. • Consente di sviluppare ed eseguire suite di test personalizzate utilizzando AWS IoT Device Tester for FreeRTOS. Per ulteriori informazioni, consulta Usa IDT per sviluppare ed eseguire le tue suite di test. • Fornisce applicazioni IDT con firma in codice, quindi non è necessario concedere

AWS IoT Device Tester version (Versione Python)	Versioni della suite di test	Versioni FreeRTOS supportate	Data di uscita	Note di rilascio
				autorizza zioni quando lo esegui in Windows o macOS. • Perfezion amento della logica di analisi dei risultati del test BLE.

AWS IoT Device Tester version (Versione Python)	Versioni della suite di test	Versioni FreeRTOS supportate	Data di uscita	Note di rilascio
IDT v3.4.0	FRQ_1.3.0	01.01.2021	05/11/2020	• Per ulteriori dettagli, consulta il file ChangeLog .md in. GitHub • È stato corretto un bug in cui 'RSA' non era un'opzione di configura zione PKCS11 valida. • È stato corretto un bug per cui i bucket Amazon S3 non venivano puliti correttam ente dopo i test OTA. • Aggiornamenti per supportare i nuovi casi di test all'interno del gruppo di test FullMQTT.

AWS IoT Device Tester version (Versione Python)	Versioni della suite di test	Versioni FreeRTOS supportate	Data di uscita	Note di rilascio
IDT v3.3.0	FRQ_1.2.0	20207.00	2020.09.17	• Per ulteriori dettagli, consulta il file ChangeLog .md in. GitHub • Nuovi test end-to-end per convalidare la funzionalità di sospensione e ripresa degli aggiornamenti di 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. • `--update-idt` Parametro aggiunto al `run-suite` comando. È possibile utilizzar

AWS IoT Device Tester version (Versione Python)	Versioni della suite di test	Versioni FreeRTOS supportate	Data di uscita	Note di rilascio
				e questa opzione per impostare la risposta per la richiesta di aggiornamento IDT. • `--update-managed-policy` Parametro aggiunto al `run-suite` comando. È possibile utilizzare questa opzione per impostare la risposta per la richiesta di aggiornamento delle policy gestite. • Miglioramenti interni e correzioni di bug, tra cui: • Per gli aggiornamenti automatici

AWS IoT Device Tester version (Versione Python)	Versioni della suite di test	Versioni FreeRTOS supportate	Data di uscita	Note di rilascio
				della suite di test, miglioramenti all'aggiornamento dei file di configurazione.

AWS IoT Device Tester version (Versione Python)	Versioni della suite di test	Versioni FreeRTOS supportate	Data di uscita	Note di rilascio

IDT v3.0.2	FRQ_1.0.1	202002.00		• Per ulteriori informazioni, consulta il file ChangeLog .md in. GitHub
				• Aggiunge l'aggiorn amento automatico delle suite di test in IDT. IDT può ora scaricare le suite di test più recenti disponibili per la tua versione 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

AWS IoT Device Tester version (Versione Python)	Versioni della suite di test	Versioni FreeRTOS supportate	Data di uscita	Note di rilascio
				impostando un flag all'avvio di IDT. Utilizzare l'opzione `-u flag` dove *flag* può essere "y" per scaricare sempre o "n" per utilizzare 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

AWS IoT Device Tester version (Versione Python)	Versioni della suite di test	Versioni FreeRTOS supportate	Data di uscita	Note di rilascio
				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`

AWS IoT Device Tester version (Versione Python)	Versioni della suite di test	Versioni FreeRTOS supportate	Data di uscita	Note di rilascio
				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` comando: elenca i casi di test disponibili in un gruppo di test. • Aggiunge l'`test-id`opzione per il `run-suite` comando: utilizza questa opzione per eseguire

AWS IoT Device Tester version (Versione Python)	Versioni della suite di test	Versioni FreeRTOS supportate	Data di uscita	Note di rilascio
				singoli casi di test in un gruppo di test.

AWS IoT Device Tester version (Versione Python)	Versioni della suite di test	Versioni FreeRTOS supportate	Data di uscita	Note di rilascio
IDT v1.7.1	FRQ_1.0.0	202002.00		• Per ulteriori dettagli, consulta il file ChangeLog .md in. GitHub • Supporta il metodo di firma del codice personalizzato per i casi di test end-to-end over-the-air (OTA) in modo da poter utilizzare comandi e script di firma del codice personalizzati per firmare i payload OTA. • Aggiunge un controllo preliminare per le porte seriali prima dell'inizio dei test. I test avranno esito negativo rapidamen

AWS IoT Device Tester version (Versione Python)	Versioni della suite di test	Versioni FreeRTOS supportate	Data di uscita	Note di rilascio
				te 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 release richiedono autorizzazioni aggiuntive, le aggiungiamo a questa

AWS IoT Device Tester version (Versione Python)	Versioni della suite di test	Versioni FreeRTOS supportate	Data di uscita	Note di rilascio
				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` .

AWS IoT Device Tester version (Versione Python)	Versioni della suite di test	Versioni FreeRTOS supportate	Data di uscita	Note di rilascio
IDT v1.6.2	FRQ_1.0.0	202002.00		• Supporta test opzionali per OTA su HTTPS per qualificare le schede di sviluppo FreeRTOS. • Supporta l'endpoint AWS IoT ATS nei test. • Supporta la funzionalità di informare gli utenti sull'ulti ma versione IDT prima dell'inizio della suite di test.

AWS IoT Device Tester version (Versione Python)	Versioni della suite di test	Versioni FreeRTOS supportate	Data di uscita	Note di rilascio
IDT v1.5.2	FRQ_1.0.0	201910.00		• 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 del moltiplicatore di timeout per aumentare i timeout, utile quando si risolvono errori relativi al timeout. • Aggiunta correzione di bug per l'analisi dei log.

AWS IoT Device Tester version (Versione Python)	Versioni della suite di test	Versioni FreeRTOS supportate	Data di uscita	Note di rilascio
				• Supporta l'endpoint iot ats nei test.
IDT v1.4.1	FRQ_1.0.0	201908.00		• Aggiunto il supporto per la nuova libreria PKCS11 e gli aggiornamenti dei casi di test. • Introdotti i codici di errore fruibili. Per ulteriori informazioni, consulta Codici di errore IDT. • È stata aggiornata la policy IAM utilizzata per eseguire IDT.

AWS IoT Device Tester version (Versione Python)	Versioni della suite di test	Versioni FreeRTOS supportate	Data di uscita	Note di rilascio
IDT v1.3.2	FRQ_1.0.0	2019-06.00		• 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	FRQ_1.0.0	• FreeRTOS v1.4.8 • FreeRTOS v1.4.9		Aggiunto il supporto per testare i dispositivi FreeRTOS con il sistema di build CMAKE.
IDT-FreeRTOS v1.1	FRQ_1.0.0			
IDT-FreeRTOS v1.0	FRQ_1.0.0			

Scarica IDT per FreerTOS

Questo argomento descrive le opzioni per scaricare IDT per FreerTOS. È possibile utilizzare uno dei seguenti collegamenti per il download del software oppure seguire le istruzioni per scaricare IDT a livello di codice.

> **Important**
>
> A partire da ottobre 2022,AWS IoT Device TesterperAWS IoTFreerTOS Qualification (FRQ) 1.0 non genera report di qualificazione firmati. Non puoi qualificarti di nuovoAWS IoTDispositivi FreerTOS da elencare nelAWSCatalogo dei dispositivi dei partnerattraverso laAWSProgramma di qualificazione dei dispositiviutilizzando le versioni IDT FRQ 1.0. Sebbene non sia possibile qualificare i dispositivi FreerTOS utilizzando IDT FRQ 1.0, è possibile continuare a testare i dispositivi FreerTOS con FRQ 1.0. Ti consigliamo di utilizzareIDT FRQ 2.0per qualificare ed elencare i dispositivi FreerTOS nelAWSCatalogo dei dispositivi dei partner.

Argomenti

- Scarica IDT manualmente
- Scarica IDT a livello di programmazione

Scaricando il software, l'utente accettaAWS IoT Device TesterContratto di licenza contenuto nell'archivio dei download.

> **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.

Scarica IDT manualmente

Questo argomento elenca le versioni supportate di IDT per FreerTOS. Come procedura ottimale, si consiglia di utilizzare la versione più recente diAWS IoT Device Testerche supporta la versione di destinazione di FreerTOS. Le nuove versioni di FreerTOS potrebbero richiedere il download di una

nuova versione diAWS IoT Device Tester. Riceverai una notifica quando inizi un'esecuzione di test seAWS IoT Device Testernon è compatibile con la versione di FreerTOS che stai utilizzando.

Per informazioni, consultare Versioni supportate diAWS IoT Device Testerper FreerTOS.

Scarica IDT a livello di programmazione

IDT fornisce un'operazione API che puoi utilizzare per recuperare un URL da cui scaricare IDT a livello di codice. Puoi utilizzare questa operazione API anche per verificare se disponi della versione più recente di IDT. Questa operazione API ha il seguente endpoint.

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

Per chiamare questa operazione API, devi disporre dell'autorizzazione per eseguire la**`iot-device-tester:LatestIdt`**azione. Includi il tuoAWSfirma, con`iot-device-tester`come nome del servizio

Richiesta API

HostOs — Il sistema operativo del computer host. Seleziona una delle opzioni seguenti:

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

TestSuiteType — Il tipo di suite di test. Scegli la seguente opzione:

`FR`— IDT per FreerTOS

ProductVersion

(Opzionale) La versione di FreerTOS. Il servizio restituisce l'ultima versione compatibile di IDT per quella versione di FreerTOS. Se non specifichi questa opzione, il servizio restituisce la versione più recente di IDT.

Risposta API

La risposta API ha il formato seguente. La`DownloadURL`include un file zip.

```
{
    "Success": True or False,
```

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

Esempi

È possibile fare riferimento ai seguenti esempi per scaricare IDT a livello di codice. Questi esempi utilizzano credenziali memorizzate in`AWS_ACCESS_KEY_ID`e`AWS_SECRET_ACCESS_KEY`le variabili di ambiente. Per seguire le migliori pratiche di sicurezza, non memorizzate le credenziali nel codice.

Example

Esempio: download utilizzando cURL versione 7.75.0 o successiva (Mac e Linux)

Se hai cURL versione 7.75.0 o successiva, puoi usare`aws-sigv4`flag per firmare la richiesta API. Questo esempio utilizzajqper analizzare l'URL di download dalla risposta.

> **Warning**
>
> La`aws-sigv4`flag richiede che i parametri di interrogazione della richiesta curl GET siano nell'ordine diHostOs/ProductVersion/TestSuiteTypeoHostOs/TestSuiteType. Gli ordini non conformi comporteranno un errore nell'ottenere firme non corrispondenti per la stringa canonica dall'API Gateway.
> Se il parametro opzionaleProductVersionè incluso, è necessario utilizzare una versione del prodotto supportata come documentato inVersioni supportate diAWS IoT Device Testerper FreerTOS.

- Sostituisci*`Stati Uniti-West-2`*con il tuoRegione AWS. Per l'elenco dei codici regionali, vedereEndpoint regionali.
- Sostituisci*`linux`*con il sistema operativo della macchina host.
- Sostituisci*`202107,00`*con la tua versione di FreerTOS.

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

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

Example

Esempio: download utilizzando una versione precedente di cURL (Mac e Linux)

Puoi utilizzare il seguente comando cURL conAWSfirma che firmi e calcoli. Per ulteriori informazioni su come firmare e calcolare unAWSfirma, vediFirmaAWSle richieste API.

- Sostituisci*linux*con il sistema operativo della macchina host.
- Sostituisci*Timestamp*con la data e l'ora, ad esempio**20220210T004606Z**.
- Sostituisci*Data*con la data, ad esempio**20220210**.
- Sostituisci*AWSRegion*con il tuoRegione AWS. Per l'elenco dei codici regionali, vedereEndpoint regionali.
- Sostituisci*AWSSignature*con ilAWSfirmache generi.

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

Example

Esempio: download utilizzando uno script Python

Questo esempio usa Pythonrichiestebiblioteca. Questo esempio è adattato dall'esempio Python aFirma unAWSRichiesta APInelAWSRiferimento generale.

- Sostituisci*Stati Uniti-West-2*con la tua regione. Per l'elenco dei codici regionali, vedereEndpoint regionali.

- Sostituisci*linux*con il sistema operativo della macchina host.

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

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

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

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

# Read AWS access key from env. variables or configuration file. Best practice is NOT
# to embed credentials in code.
access_key = os.environ.get('AWS_ACCESS_KEY_ID')
```

```
secret_key = os.environ.get('AWS_SECRET_ACCESS_KEY')
if access_key is None or secret_key is None:
    print('No access key is available.')
    sys.exit()

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

# ************* TASK 1: CREATE A CANONICAL REQUEST *************
# http://docs.aws.amazon.com/general/latest/gr/sigv4-create-canonical-request.html
# Step 1 is to define the verb (GET, POST, etc.)--already done.
# Step 2: Create canonical URI--the part of the URI from domain to query
# string (use '/' if no path)
canonical_uri = '/latestidt'
# Step 3: Create the canonical query string. In this example (a GET request),
# request parameters are in the query string. Query string values must
# be URL-encoded (space=%20). The parameters must be sorted by name.
# For this example, the query string is pre-formatted in the request_parameters
 variable.
canonical_querystring = request_parameters
# Step 4: Create the canonical headers and signed headers. Header names
# must be trimmed and lowercase, and sorted in code point order from
# low to high. Note that there is a trailing \n.
canonical_headers = 'host:' + host + '\n' + 'x-amz-date:' + amzdate + '\n'
# Step 5: Create the list of signed headers. This lists the headers
# in the canonical_headers list, delimited with ";" and in alpha order.
# Note: The request can include any headers; canonical_headers and
# signed_headers lists those that you want to be included in the
# hash of the request. "Host" and "x-amz-date" are always required.
signed_headers = 'host;x-amz-date'
# Step 6: Create payload hash (hash of the request body content). For GET
# requests, the payload is an empty string ("").
payload_hash = hashlib.sha256(('').encode('utf-8')).hexdigest()
# Step 7: Combine elements to create canonical request
canonical_request = method + '\n' + canonical_uri + '\n' + canonical_querystring + '\n'
 + canonical_headers + '\n' + signed_headers + '\n' + payload_hash

# ************* TASK 2: CREATE THE STRING TO SIGN*************
# Match the algorithm to the hashing algorithm you use, either SHA-1 or
# SHA-256 (recommended)
algorithm = 'AWS4-HMAC-SHA256'
credential_scope = datestamp + '/' + region + '/' + service + '/' + 'aws4_request'
```

```
string_to_sign = algorithm + '\n' +  amzdate + '\n' +  credential_scope + '\n' +
 hashlib.sha256(canonical_request.encode('utf-8')).hexdigest()
# ************* TASK 3: CALCULATE THE SIGNATURE *************
# Create the signing key using the function defined above.
signing_key = getSignatureKey(secret_key, datestamp, region, service)
# Sign the string_to_sign using the signing_key
signature = hmac.new(signing_key, (string_to_sign).encode('utf-8'),
 hashlib.sha256).hexdigest()

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

# ************* SEND THE REQUEST *************
request_url = endpoint + '?' + canonical_querystring
print('\nBEGIN REQUEST++++++++++++++++++++++++++++++++++++')
print('Request URL = ' + request_url)
response = requests.get(request_url, headers=headers)
print('\nRESPONSE++++++++++++++++++++++++++++++++++++')
print('Response code: %d\n' % response.status_code)
print(response.text)

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

Usa IDT con la suite di qualificazione FreeRTOS 2.0 (FRQ 2.0)

La suite di qualificazione FreeRTOS 2.0 è una versione aggiornata della suite di qualificazione FreeRTOS. Consigliamo agli sviluppatori di utilizzare FRQ 2.0 perché consiste in casi di test pertinenti per qualificare i dispositivi che eseguono le librerie FreeRTOS Long Term Support (LTS).

IDT for FreeRTOS verifica la porta di FreeRTOS sul microcontrollore e se comunica efficacemente con. AWS IoT In particolare, verifica le interfacce del livello di porting con le librerie di FreeRTOS e se i repository di test FreeRTOS sono implementati correttamente. Esegue inoltre test end-to-end con AWS IoT Core. I test eseguiti da IDT per FreeRTOS sono definiti nel repository FreeRTOS. GitHub

IDT for FreeRTOS esegue i test come applicazioni incorporate che lampeggiano sul dispositivo microcontrollore sottoposto a test. Le immagini binarie dell'applicazione includono FreeRTOS, le interfacce FreeRTOS con portabilità e i driver dei dispositivi di scheda. Lo scopo dei test è verificare che le interfacce FreeRTOS trasferite funzionino correttamente sui driver del dispositivo.

IDT for FreeRTOS genera rapporti di test che puoi inviare per AWS IoT far sì che il tuo hardware sia elencato nel AWS Partner Device Catalog. Per ulteriori informazioni, consulta AWS Device Qualification Program.

IDT for FreeRTOS viene eseguito su un computer host (Windows, macOS o Linux) collegato al dispositivo in fase di test. IDT configura e orchestra i casi di test e aggrega i risultati. Fornisce inoltre un'interfaccia a riga di comando per gestire l'esecuzione dei test.

Per testare il tuo dispositivo, IDT for FreeRTOS crea risorse come AWS IoT oggetti, gruppi FreeRTOS, funzioni Lambda. Per creare queste risorse, IDT for FreeRTOS utilizza le AWS credenziali configurate in per effettuare chiamate API `config.json` per conto dell'utente. 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.

Argomenti

- Informazioni su risultati e log

Prerequisiti

Questa sezione descrive i prerequisiti per testare i microcontrollori con. AWS IoT Device Tester

Preparazione della qualificazione FreeRTOS

> **Note**
>
> AWS IoT Device Testerfor FreeRTOS consiglia vivamente di utilizzare l'ultima versione di patch della versione più recente di FreeRTOS-LTS.

IDT for FRQ 2.0 è una qualifica per FreeRTOS. Prima di eseguire IDT FRQ 2.0 per la qualificazione, devi completare la Qualifica della scheda nella Guida alle qualifiche di FreeRTOS. Per portare le librerie, testarle e configurarle*`manifest.yml`*, vedete Porting the FreeRTOS library nella FreeRTOS Porting Guide. FRQ 2.0 contiene un processo diverso per la qualificazione. Consulta le ultime modifiche alla qualificazione nella guida alla qualificazione di FreeRTOS per i dettagli.

Il repository FreeRTOS-Libraries-Integration-Tests deve essere presente per l'esecuzione di IDT. Consulta il README.md su come clonare e trasferire questo repository nel tuo progetto sorgente. FreeRTOS-Libraries-Integration-Tests deve includere il `manifest.yml` localizzato nella radice del progetto, affinché IDT possa essere eseguito.

> **Note**
>
> IDT dipende dall'implementazione del repository dei test di. `UNITY_OUTPUT_CHAR` I registri di uscita del test e i registri del dispositivo non devono interlacciarsi tra loro. Vedi la sezione Implementazione delle macro di registrazione della libreria nella FreeRTOS Porting Guide per ulteriori dettagli.

Scarica IDT FreeRTOS

Ogni versione di FreeRTOS ha una versione corrispondente di IDT for FreeRTOS per eseguire i test di qualificazione. Scarica la versione appropriata di IDT for FreeRTOS dalle versioni supportate di AWS IoT Device Tester for FreeRTOS.

Estrai IDT for FreeRTOS in una posizione del file system in cui disponi delle autorizzazioni di lettura e scrittura. Poiché Microsoft Windows ha un limite di caratteri per la lunghezza del percorso, estrai IDT for FreeRTOS in una directory principale come or. `C:\` `D:\`

> **Note**
>
> Più utenti non devono eseguire IDT da una posizione condivisa, come una directory NFS o una cartella condivisa di rete Windows. Ciò comporterà arresti anomali o danneggiamento dei dati. Si consiglia di estrarre il pacchetto IDT in un'unità locale.

Scarica Git

IDT deve avere Git installato come prerequisito per garantire l'integrità del codice sorgente.

Segui le istruzioni nella GitHubguida per installare Git. Per verificare la versione attualmente installata di Git, inserisci il comando `git --version` nel terminale.

> **Warning**
>
> IDT usa Git per allinearsi allo stato pulito o sporco di una directory. Se Git non è installato, i gruppi di `FreeRTOSIntegrity` test falliranno o non verranno eseguiti come previsto. Se IDT restituisce un errore come `git executable not found` o`git command not found`, installa o reinstalla Git e riprova.

Creazione e configurazione di un account AWS

> **Note**
>
> La suite completa di qualifiche IDT è supportata solo nelle seguenti aree Regioni AWS
>
> - Stati Uniti orientali (Virginia settentrionale)
> - Stati Uniti occidentali (Oregon)
> - Asia Pacifico (Tokyo)
> - Europa (Irlanda)

Per testare il tuo dispositivo, IDT for FreeRTOS crea risorse come AWS IoT oggetti, gruppi FreeRTOS e funzioni Lambda. Per creare tali risorse, IDT for FreeRTOS richiede la creazione e la configurazione di un AWS account e una politica IAM che conceda a IDT for FreeRTOS il permesso di accedere alle risorse per conto dell'utente durante l'esecuzione dei test.

I passaggi seguenti servono a creare e configurare il tuo AWS account.

1. Se disponi già di un account AWS, passa alla fase successiva. Altrimenti crea un AWSaccount.
2. Segui la procedura descritta in Creazione di ruoli IAM. Non aggiungere autorizzazioni o politiche in questo momento.
3. Per eseguire i test di qualificazione OTA, vai al passaggio 4. Altrimenti vai al passaggio 5.
4. Allega la politica in linea delle autorizzazioni IAM OTA al tuo ruolo IAM.

   a.

   > **Important**
   >
   > Il modello di policy seguente concede l'autorizzazione IDT per creare ruoli, policy e collegare policy ai ruoli. IDT for FreeRTOS utilizza queste autorizzazioni per i test che creano ruoli. Sebbene il modello di policy non fornisca privilegi di amministratore all'utente, le autorizzazioni possono essere utilizzate per ottenere l'accesso da amministratore al tuo AWS account.

   b. Segui i passaggi seguenti per assegnare le autorizzazioni necessarie al tuo ruolo IAM:

      i. Nella pagina Autorizzazioni, scegli Aggiungi autorizzazioni.

      ii. Scegli Crea politica in linea.

      iii. Scegliere la scheda JSON e copiare le seguenti autorizzazioni nella casella di testo JSON . Usa il modello nella sezione La maggior parte delle regioni se non ti trovi nella regione della Cina. Se ti trovi nella regione della Cina, usa il modello nelle regioni di Pechino e Ningxia.

      Most Regions

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iotdeviceadvisor:*",
            "Resource": [
```

```
                "arn:aws:iotdeviceadvisor:*:*:suiterun/*/*",
                "arn:aws:iotdeviceadvisor:*:*:suitedefinition/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::*:role/idt*",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService":
"iotdeviceadvisor.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "execute-api:Invoke*",
                "iam:ListRoles",
                "iot:Connect",
                "iot:CreateJob",
                "iot:DeleteJob",
                "iot:DescribeCertificate",
                "iot:DescribeEndpoint",
                "iot:DescribeJobExecution",
                "iot:DescribeJob",
                "iot:DescribeThing",
                "iot:GetPolicy",
                "iot:ListAttachedPolicies",
                "iot:ListCertificates",
                "iot:ListPrincipalPolicies",
                "iot:ListThingPrincipals",
                "iot:ListThings",
                "iot:Publish",
                "iot:UpdateThingShadow",
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:DescribeLogGroups",
                "logs:DescribeLogStreams",
                "logs:PutLogEvents",
                "logs:PutRetentionPolicy"
            ],
            "Resource": "*"
```

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

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

Beijing and Ningxia Regions

Il modello di policy seguente può essere utilizzato nelle Regioni di Pechino e Ningxia.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreatePolicy",
                "iam:DetachRolePolicy",
                "iam:DeleteRolePolicy",
                "iam:DeletePolicy",
                "iam:CreateRole",
                "iam:DeleteRole",
                "iam:AttachRolePolicy"
            ],
            "Resource": [
                "arn:aws-cn:iam::*:policy/idt*",
                "arn:aws-cn:iam::*:role/idt*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ssm:GetParameters"
            ],
            "Resource": [
                "arn:aws-cn:ssm:*::parameter/aws/service/ami-amazon-
linux-latest/amzn2-ami-hvm-x86_64-gp2"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeInstances",
                "ec2:RunInstances",
                "ec2:CreateSecurityGroup",
                "ec2:CreateTags",
                "ec2:DeleteTags"
            ],
```

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

iv. Al termine, seleziona Review policy (Rivedi policy).

v. Inserisci IDTFreeRtoSiamPermissions come nome della policy.

vi. Scegli Create Policy (Crea policy).

5. Collega AWSIoTDeviceTesterForFreeRTOSFullAccessal tuo ruolo IAM.

   a. Per allegare le autorizzazioni necessarie al tuo ruolo IAM:

      i. Nella pagina Autorizzazioni, scegli Aggiungi autorizzazioni.

ii. Scegli Collega policy.

iii. Cerca la AWSIoTDeviceTesterForFreeRTOSFullAccesspolitica. Seleziona la casella.

b. Scegli Add Permissions (Aggiungi autorizzazioni).

6. Esporta credenziali per IDT. Per i dettagli, consulta Ottenere le credenziali dei ruoli IAM per l'accesso alla CLI.

Policy gestita di AWS IoT Device Tester

La policy `AWSIoTDeviceTesterForFreeRTOSFullAccess` gestita contiene le seguenti AWS IoT Device Tester autorizzazioni per il controllo della versione, le funzionalità di aggiornamento auto e la raccolta di metriche.

- `iot-device-tester:SupportedVersion`

  Concede AWS IoT Device Tester l'autorizzazione a recuperare l'elenco dei prodotti, delle suite di test e delle versioni IDT supportate.

- `iot-device-tester:LatestIdt`

  Concede AWS IoT Device Tester il permesso di recuperare l'ultima versione IDT disponibile per il download.

- `iot-device-tester:CheckVersion`

  Concede AWS IoT Device Tester l'autorizzazione a verificare la compatibilità delle versioni per IDT, suite di test e prodotti.

- `iot-device-tester:DownloadTestSuite`

  Concede AWS IoT Device Tester l'autorizzazione a scaricare gli aggiornamenti della suite di test.

- `iot-device-tester:SendMetrics`

  Concede AWS l'autorizzazione a raccogliere metriche sull'utilizzo AWS IoT Device Tester interno.

(Facoltativo) Installazione dell'AWS Command Line Interface

In alternativa, puoi utilizzare l'AWS CLI per eseguire alcune operazioni. Se non è AWS CLI installato, segui le istruzioni in Installare il AWS CLI.

Configura AWS CLI per la AWS regione che desideri utilizzare eseguendo aws configure da una riga di comando. Per informazioni sulle AWS regioni che supportano IDT for FreeRTOS, vedi AWSRegioni ed endpoint. Per ulteriori informazioni, aws configure vedere Configurazione rapida con aws configure.

Preparazione dei test per la prima verifica della scheda del microcontrollore

Puoi usare IDT for FreeRTOS per testare la tua implementazione delle librerie FreeRTOS. Dopo aver effettuato il porting delle librerie FreeRTOS per i driver dei dispositivi della scheda, usale per AWS IoT Device Tester eseguire i test di qualificazione sulla scheda del microcontrollore.

Aggiungi livelli di porting della libreria e implementa un repository di test FreeRTOS

Per portare FreeRTOS per il tuo dispositivo, consulta la FreeRTOS Porting Guide. Quando si implementa il repository di test FreeRTOS e si esegue il porting dei livelli di FreeRTOS, è necessario fornire percorsi a ciascuna libreria, incluso `manifest.yml` il repository dei test. Questo file si troverà nella directory principale del codice sorgente. Consulta le istruzioni del file manifest per i dettagli.

Configura le credenziali AWS

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. AWSLe credenziali valide sono specificate nel file *`devicetester_extract_location`*`/devicetester_freertos_`*`[win|mac|linux]`*`/configs/config.json` di configurazione.

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

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

L'`auth`attributo del `config.json` file ha un campo metodo che controlla AWS l'autenticazione e può essere dichiarato come file o ambiente. L'impostazione del campo su ambiente estrae le AWS

credenziali dalle variabili di ambiente della macchina host. L'impostazione del campo su file importa un profilo specificato dal file di `.aws/credentials` 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 for FreeRTOS per testare un singolo dispositivo o più dispositivi in un pool. Per accelerare il processo di qualificazione, IDT for FreeRTOS può testare dispositivi con le stesse specifiche in parallel. Lo strumento utilizza un metodo Round Robin per eseguire un gruppo di test differente su ciascun dispositivo di un pool.

Il `device.json` file ha un array al livello superiore. Ogni attributo dell'array è un nuovo pool di dispositivi. Ogni pool di dispositivi ha un attributo di array di dispositivi, che ha più dispositivi dichiarati. Nel modello è presente un pool di dispositivi e solo un dispositivo in quel pool di dispositivi. È possibile aggiungere uno o più dispositivi a un pool di dispositivi modificando la sezione `devices` del modello `device.json` nella cartella `configs`.

> **Note**
>
> Tutti i dispositivi dello stesso pool devono avere le stesse specifiche tecniche e lo stesso SKU. Per abilitare le build parallel 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. È necessario fare riferimento al percorso del codice sorgente nel comando build o flash utilizzando la `testdata.sourcePath` variabile. IDT for FreeRTOS sostituisce questa variabile con un percorso temporaneo del codice sorgente copiato. Per ulteriori informazioni, consulta IDT per variabili FreeRTOS.

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

```
            {
              "name": "serialNo",
              "value": "serialNo-value"
            }
          ]
        }
      ]
    }
]
```

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 che utilizzi nella procedura di pubblicazione dell'offerta.

`features`

Un array contenente le funzionalità supportate dal dispositivo. AWS IoT Device Testerutilizza queste informazioni per selezionare i test di qualificazione da eseguire.

I valori supportati sono:

`Wifi`

Indica se la scheda dispone di funzionalità Wi-Fi.

`Cellular`

Indica se la scheda dispone di funzionalità cellulari.

PKCS11

Indica l'algoritmo di crittografia della chiave pubblica supportato dalla scheda. PKCS11 è obbligatorio per la qualifica. I valori supportati sono `ECC`RSA, e`Both`. `Both`indica che la scheda supporta `ECC` sia`RSA`.

KeyProvisioning

Indica il metodo di scrittura di un certificato client X.509 attendibile sulla scheda.

I valori validi sono `Import`O`nboard`, `Both` e`No`. `O`n`boardBoth`, oppure `No` per la qualificazione è richiesta la fornitura di chiavi. `Import`da sola non è un'opzione valida per la qualificazione.

- Usalo `Import` solo se la tua scheda consente l'importazione di chiavi private. `Import`La selezione non è una configurazione valida per la qualificazione e deve essere utilizzata solo a scopo di test, in particolare con i casi di test PKCS11. `O`n`board`, `Both` o `No` è richiesto per la qualificazione.
- Usalo `O`n`board` se la tua scheda supporta le chiavi private integrate (ad esempio, se il tuo dispositivo ha un elemento sicuro o se preferisci generare una key pair e un certificato del tuo dispositivo). 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`.
- Usalo `Both` se la tua scheda supporta sia l'importazione di chiavi private che la generazione di chiavi integrate per la fornitura di chiavi.
- Usalo `No` se la tua scheda non supporta la fornitura di chiavi. `No`è un'opzione valida solo se il dispositivo è anche preconfigurato.

OTA

Indica se la scheda supporta la funzionalità di aggiornamento over-the-air (OTA). L'attributo `O`t`aDataPlaneProtocol` indica quale protocollo di piano dei dati OTA supporta il dispositivo. Per la qualificazione è richiesta una OTA con protocollo dataplane HTTP o MQTT. Per saltare l'esecuzione dei test OTA durante il test, imposta la funzionalità OTA su `No` e l'`O`t`aDataPlaneProtocol`attributo su. `None` Questa non sarà una corsa di qualificazione.

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.serialPort`

La porta seriale del computer host utilizzato per connettersi ai dispositivi da testare.

`devices.secureElementConfig.PublicKeyAsciiHexFilePath`

Obbligatorio se la tua tavola NON `PublicDeviceCertificateArn` è `pre-provisioned` o non è fornita. Poiché `Onboard` è un tipo obbligatorio di Key Provisioning, questo campo è attualmente obbligatorio per il gruppo di test FullTransportInterface TLS. Se il dispositivo è`pre-provisioned`, `PublicKeyAsciiHexFilePath` è opzionale e non è necessario che sia incluso.

Il blocco seguente è un percorso assoluto del file che contiene la chiave pubblica in byte esadecimali estratta dalla chiave `Onboard` privata.

```
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 la chiave pubblica in formato esadecimale per generare il file esadecimale.

Per generare il file esadecimale da una chiave pubblica .der, inserisci il seguente xxd comando:

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

Se la tua chiave pubblica è in formato.pem, puoi estrarre le intestazioni e i piè di pagina codificati in base64 e decodificarli in formato binario. Quindi, codifichi in formato esadecimale la stringa binaria per generare il file esadecimale.

Per generare un file esadecimale per una chiave pubblica .pem, procedi come segue:

1. Esegui il seguente base64 comando per rimuovere l'intestazione e il piè di pagina base64 dalla chiave pubblica. La chiave decodificata, denominata`base64key`, viene quindi emessa nel file: `pubkey.der`

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

2. Esegui il seguente xxd comando per `pubkey.der` convertire in formato esadecimale. La chiave risultante viene salvata come *`outFile`*

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

`devices.secureElementConfig.PublicDeviceCertificateArn`

L'ARN del certificato derivante dall'elemento sicuro su cui è stato caricato. AWS IoT Core Per informazioni sul caricamento del certificato suAWS IoT Core, consulta i certificati client X.509 nella Guida per gli AWS IoT sviluppatori.

`devices.secureElementConfig.SecureElementSerialNumber`

(Facoltativo) Il numero di serie dell'elemento sicuro. Il numero di serie viene utilizzato facoltativamente per creare certificati di dispositivo per la fornitura di chiavi JITR.

`devices.secureElementConfig.preProvisioned`

(Facoltativo) Imposta su «Sì» se il dispositivo dispone di un elemento sicuro preconfigurato con credenziali bloccate, che non può importare, creare o distruggere oggetti. Se questo attributo è impostato su Sì, è necessario fornire le etichette pkcs11 corrispondenti.

`devices.secureElementConfig.pkcs11JITPCodeVerifyRootCertSupport`

(Facoltativo) Impostare su Sì se l'implementazione CorePKCS11 del dispositivo supporta l'archiviazione per JITP. Ciò consentirà il test JITP durante il `codeverify` test del core PKCS 11 e richiede la fornitura della chiave di verifica del codice, del certificato JITP e delle etichette PKCS 11 del certificato principale.

`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

IDT for FreeRTOS crea e invia automaticamente i test sulla scheda. Per abilitare ciò, devi configurare IDT per eseguire i comandi build e flash per il tuo 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`. Il seguente esempio JSON mostra come configurare IDT for FreeRTOS per testare più dispositivi:

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

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

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

```
}
```

Il seguente elenca gli attributi utilizzati in `userdata.json`:

`sourcePath`

Il percorso alla radice del codice sorgente di FreeRTOS portato.

`retainModifiedSourceDirectories`

(Facoltativo) Verifica se conservare le directory di origine modificate utilizzate durante la compilazione e il flashing per scopi di debug. Se impostata su`true`, le directory di origine modificate sono denominate RetainedSrc e si trovano nelle cartelle del registro dei risultati di ogni gruppo di test eseguito. Se non è incluso, il campo predefinito è. `false`

`freeRTOSTestParamConfigPath`

Il percorso del `test_param_config.h` file per l'integrazione con FreeRTOS-Libraries-Integration-Tests. Questo file deve utilizzare la variabile `{{testData.sourcePath}}` segnaposto per renderlo relativo alla radice del codice sorgente. AWS IoT Device Testerutilizza i parametri di questo file per configurare i test.

`freeRTOSTestExecutionConfigPath`

Il percorso del `test_execution_config.h` file per l'integrazione con FreeRTOS-Libraries-Integration-Tests. Questo file deve utilizzare la variabile `{{testData.sourcePath}}` segnaposto per renderlo relativo alla radice del repository. AWS IoT Device Testerusa questo file per controllare quali test devono essere eseguiti.

`freeRTOSVersion`

La versione di FreeRTOS, inclusa la versione patch utilizzata nell'implementazione. Vedi Versioni supportate di AWS IoT Device Tester for FreeRTOS per le versioni di FreeRTOS compatibili con FreeRTOS. AWS IoT Device Tester

`buildTool`

Il comando per compilare 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}}`. Usa il `{{config.idtRootPath}}` segnaposto per fare riferimento a uno script di compilazione relativo al percorso AWS IoT Device Tester principale.

`flashTool`

Il comando per eseguire il flashing di un'immagine sul dispositivo. Tutti i riferimenti al percorso del codice sorgente nel comando flash devono essere sostituiti dalla AWS IoT Device Tester variabile`{{testData.sourcePath}}`. Utilizzate il `{{config.idtRootPath}}` segnaposto per fare riferimento a uno script flash relativo al percorso AWS IoT Device Tester principale.

> **Note**
>
> La nuova struttura dei test di integrazione con FRQ 2.0 non richiede variabili di percorso come `{{enableTests}}` e`{{buildImageName}}`. I test OTA End to End vengono eseguiti con i modelli di configurazione forniti nel repository GitHubFreeRTOS-Libraries-Integration-Tests. Se i file nel GitHub repository sono presenti nel progetto sorgente principale, il codice sorgente non viene modificato tra un test e l'altro. Se è necessaria un'immagine di build diversa per OTA End to End, è necessario creare questa immagine nello script di compilazione e specificarla nel `userdata.json` file specificato sotto`otaConfiguration`.

`testStartDelayms`

Specifica quanti millisecondi aspetterà il test runner di FreeRTOS prima di iniziare a eseguire i test. Ciò può essere utile se il dispositivo sottoposto a test inizia a emettere importanti informazioni di test prima che IDT abbia la possibilità di connettersi e avviare la registrazione a causa di problemi di rete o di altro tipo di latenza. Questo valore è applicabile solo ai gruppi di test FreeRTOS e non ad altri gruppi di test che non utilizzano il test runner FreeRTOS, come i test OTA. Se ricevi un errore relativo al 10 previsto ma ne ricevi 5, questo campo deve essere impostato su 5000.

`echoServerConfiguration`

La configurazione per configurare il server echo per il test TLS. Questo campo è obbligatorio.

`keyGenerationMethod`

Il server echo è configurato con questa opzione. Le opzioni sono EC o RSA.

`serverPort`

Il numero della porta sulla quale viene eseguito il server echo.

`otaConfiguration`

La configurazione per i test OTA PAL e OTA E2E. Questo campo è obbligatorio.

`otaE2EFirmwarePath`

Percorso dell'immagine del contenitore OTA utilizzata da IDT per i test OTA End to End.

`otaPALCertificatePath`

Il percorso verso il certificato per il test OTA PAL sul dispositivo. Viene utilizzato per verificare la firma. Ad esempio, ecdsa-sha256-signer.crt.pem.

`deviceFirmwarePath`

Il percorso del nome in codice rigido per l'avvio dell'immagine del firmware. Se il dispositivo NON utilizza il file system per l'avvio del firmware, specifica questo campo come `'NA'`. Se il dispositivo utilizza il file system per l'avvio del firmware, specifica il percorso o il nome dell'immagine di avvio del firmware.

`codeSigningConfiguration`

`signingMethod`

Il metodo di firma del codice. I valori possibili sono AWS o Personalizzato.

> **Note**
>
> Per le Regioni di Pechino e Ningxia, usa Personalizzato. AWSla firma del codice non è supportata in quella regione.

`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, utilizza il nome della risorsa Amazon (ARN) per il certificato attendibile caricato nel Certificate Name

(ARN AWS Certificate Manager. Per il metodo di firma del codice personalizzato, utilizza il percorso assoluto del file del certificato del firmatario. Per informazioni sulla creazione di un certificato affidabile, consulta Creare un certificato di firma con codice.

`untrustedSignerCertificate`

L'ARN o il percorso del file per un secondo certificato utilizzato in alcuni test OTA come certificato non affidabile. Per informazioni sulla creazione di un certificato, consulta Creare un certificato di firma con codice.

`signerCertificateFileName`

Il nome del file del certificato di firma del codice sul dispositivo. Questo valore deve corrispondere al nome del file fornito durante l'esecuzione del `aws acm import-certificate` comando.

`compileSignerCertificate`

Valore booleano che determina lo stato del certificato di verifica della firma. I valori validi sono `true` e `false`.

Imposta questo valore su true se il certificato di verifica della firma del firmatario del codice non è fornito o lampeggia. Deve essere compilato nel progetto. AWS IoT Device Testerrecupera il certificato affidabile e lo compila. `aws_codesigner_certificate.h`

`signerPlatform`

L'algoritmo di firma e hash che AWS Code Signer utilizza durante la creazione dell'attività 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`

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.

`pkcs11LabelConfiguration`

La configurazione delle etichette PKCS11 richiede almeno un set di etichette di etichetta del certificato del dispositivo, etichetta a chiave pubblica e etichetta a chiave privata per eseguire i gruppi di test PKCS11. Le etichette PKCS11 richieste si basano sulla configurazione del dispositivo nel `device.json` file. Se la preimpostazione è impostata su Sì`device.json`, le etichette richieste devono essere una delle seguenti, a seconda della funzionalità scelta per la funzionalità PKCS11.

- `PreProvisionedEC`
- `PreProvisionedRSA`

Se pre-provisioned è impostato su No in`device.json`, le etichette richieste sono:

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

Le tre etichette seguenti sono obbligatorie solo se si seleziona Sì per `pkcs11JITPCodeVerifyRootCertSupport` nel `device.json` file.

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

I valori di questi campi devono corrispondere ai valori definiti nella FreeRTOS Porting Guide.

`pkcs11LabelDevicePrivateKeyForTLS`

(Facoltativo) Questa etichetta viene utilizzata per l'etichetta PKCS #11 della chiave privata. Per i dispositivi con supporto integrato e di importazione per la fornitura di chiavi, questa etichetta viene utilizzata per i test. Questa etichetta può essere diversa da quella definita per il caso prestabilito. Se il provisioning delle chiavi è impostato su No e il provisioning predefinito è impostato su Sì, in`device.json`, questo non sarà definito.

`pkcs11LabelDevicePublicKeyForTLS`

(Facoltativo) Questa etichetta viene utilizzata per l'etichetta PKCS #11 della chiave pubblica. Per i dispositivi con supporto integrato e di importazione per la fornitura di chiavi, questa

etichetta viene utilizzata per i test. Questa etichetta può essere diversa da quella definita per il caso prestabilito. Se il provisioning delle chiavi è impostato su No e il provisioning predefinito è impostato su Sì, in`device.json`, questo non sarà definito.

`pkcs11LabelDeviceCertificateForTLS`

(Facoltativo) Questa etichetta viene utilizzata per l'etichetta PKCS #11 del certificato del dispositivo. Per i dispositivi con supporto integrato e di importazione per la fornitura di chiavi, questa etichetta verrà utilizzata per i test. Questa etichetta può essere diversa da quella definita per il caso prestabilito. Se il provisioning delle chiavi è impostato su No e il provisioning predefinito è impostato su Sì, in`device.json`, questo non sarà definito.

`pkcs11LabelPreProvisionedECDevicePrivateKeyForTLS`

(Facoltativo) Questa etichetta viene utilizzata per l'etichetta PKCS #11 della chiave privata. Per i dispositivi con elementi sicuri o limitazioni hardware, questo avrà un'etichetta diversa per conservare AWS IoT le credenziali. Se il tuo dispositivo supporta il pre-provisioning con una chiave EC, fornisci questa etichetta. Quando PreProvisioned è impostato su Sì`device.json`, è necessario fornire questa etichetta o entrambe. `pkcs11LabelPreProvisionedRSADevicePrivateKeyForTLS` Questa etichetta può essere diversa da quella definita per i casi di bordo e di importazione.

`pkcs11LabelPreProvisionedECDevicePublicKeyForTLS`

(Facoltativo) Questa etichetta viene utilizzata per l'etichetta PKCS #11 della chiave pubblica. Per i dispositivi con elementi sicuri o limitazioni hardware, questo avrà un'etichetta diversa per conservare AWS IoT le credenziali. Se il tuo dispositivo supporta il pre-provisioning con una chiave EC, fornisci questa etichetta. Quando PreProvisioned è impostato su Sì`device.json`, è necessario fornire questa etichetta o entrambe. `pkcs11LabelPreProvisionedRSADevicePublicKeyForTLS` Questa etichetta può essere diversa da quella definita per i casi di bordo e di importazione.

`pkcs11LabelPreProvisionedECDeviceCertificateForTLS`

(Facoltativo) Questa etichetta viene utilizzata per l'etichetta PKCS #11 del certificato del dispositivo. Per i dispositivi con elementi sicuri o limitazioni hardware, questo avrà un'etichetta diversa per conservare AWS IoT le credenziali. Se il tuo dispositivo supporta il pre-provisioning con una chiave EC, fornisci questa etichetta. Quando PreProvisioned è impostato su Sì`device.json`, è necessario fornire questa etichetta o entrambe. `pkcs11LabelPreProvisionedRSADeviceCertificateForTLS` Questa etichetta può essere diversa da quella definita per i casi di bordo e di importazione.

pkcs11LabelPreProvisionedRSADevicePrivateKeyForTLS

(Facoltativo) Questa etichetta viene utilizzata per l'etichetta PKCS #11 della chiave privata. Per i dispositivi con elementi sicuri o limitazioni hardware, questo avrà un'etichetta diversa per conservare AWS IoT le credenziali. Se il tuo dispositivo supporta il pre-provisioning con una chiave RSA, fornisci questa etichetta. Quando PreProvisioned è impostato su Yes in`device.json`, è necessario fornire questa etichetta o entrambe. `pkcs11LabelPreProvisionedECDevicePrivateKeyForTLS`

pkcs11LabelPreProvisionedRSADevicePublicKeyForTLS

(Facoltativo) Questa etichetta viene utilizzata per l'etichetta PKCS #11 della chiave pubblica. Per i dispositivi con elementi sicuri o limitazioni hardware, questo avrà un'etichetta diversa per conservare AWS IoT le credenziali. Se il tuo dispositivo supporta il pre-provisioning con una chiave RSA, fornisci questa etichetta. Quando PreProvisioned è impostato su Yes in`device.json`, è necessario fornire questa etichetta o entrambe. `pkcs11LabelPreProvisionedECDevicePublicKeyForTLS`

pkcs11LabelPreProvisionedRSADeviceCertificateForTLS

(Facoltativo) Questa etichetta viene utilizzata per l'etichetta PKCS #11 del certificato del dispositivo. Per i dispositivi con elementi sicuri o limitazioni hardware, questo avrà un'etichetta diversa per conservare AWS IoT le credenziali. Se il tuo dispositivo supporta il pre-provisioning con una chiave RSA, fornisci questa etichetta. Quando PreProvisioned è impostato su Yes in`device.json`, è necessario fornire questa etichetta o entrambe. `pkcs11LabelPreProvisionedECDeviceCertificateForTLS`

pkcs11LabelCodeVerifyKey

(Facoltativo) Questa etichetta viene utilizzata per l'etichetta PKCS #11 della chiave di verifica del codice. Se il tuo dispositivo dispone del supporto di archiviazione PKCS #11 del certificato JITP, della chiave di verifica del codice e del certificato root, fornisci questa etichetta. Quando `pkcs11JITPCodeVerifyRootCertSupport` in `device.json` è impostato su Sì, è necessario fornire questa etichetta.

pkcs11LabelJITPCertificate

(Facoltativo) Questa etichetta viene utilizzata per l'etichetta PKCS #11 del certificato JITP. Se il tuo dispositivo dispone del supporto di archiviazione PKCS #11 del certificato JITP, della chiave di verifica del codice e del certificato root, fornisci questa etichetta. Quando `pkcs11JITPCodeVerifyRootCertSupport` in `device.json` è impostato su Sì, è necessario fornire questa etichetta.

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 Testerconsente di fare riferimento alle informazioni sul dispositivo in flash e di creare comandi utilizzando JsonPath. Utilizzando JsonPath espressioni semplici, puoi recuperare le informazioni richieste specificate nel tuo `device.json` file.

Variabili di percorso

IDT for 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.

`{{device.connectivity.serialPort}}`

Si estende alla porta seriale.

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

Estende al numero seriale del dispositivo.

`{{config.idtRootPath}}`

Si espande fino al percorso AWS IoT Device Tester principale.

Usa l'interfaccia utente IDT for FreeRTOS per eseguire la suite di qualificazione FreeRTOS 2.0 (FRQ 2.0)

AWS IoT Device Testerper FreeRTOS (IDT per FreeRTOS) include un'interfaccia utente (UI) basata sul Web in cui è possibile interagire con l'applicazione a riga di comando IDT e i relativi file di configurazione. Utilizzi l'interfaccia utente IDT for FreeRTOS per creare una nuova configurazione o modificarne una esistente per il tuo dispositivo. Puoi anche usare l'interfaccia utente per chiamare l'applicazione IDT ed eseguire i test FreeRTOS sul tuo dispositivo.

Per informazioni su come utilizzare la riga di comando per eseguire test di qualificazione, consultaPreparazione dei test per la prima verifica della scheda del microcontrollore.

Questa sezione descrive i prerequisiti per l'interfaccia utente IDT for FreeRTOS e come eseguire i test di qualificazione dall'interfaccia utente.

Argomenti

Prerequisiti

Per eseguire i test tramite l'interfaccia utenteAWS IoT Device Tester (IDT) for FreeRTOS, è necessario completare i prerequisiti nellaPrerequisiti pagina per IDT FreeRTOS Qualification (FRQ) 2.x.

ConfigurazioneAWS delle credenziali

È necessario configurare le credenziali utente IAM per l'AWSutente creato inCreazione e configurazione di un account AWS. Puoi specificare le credenziali in uno dei due modi seguenti:

- In un file di credenziali
- Come variabili di ambiente

ConfigurareAWS le credenziali con un 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 utilizzato:

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

AggiungiAWS le tue credenziali al`credentials` file nel seguente formato:

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

> **Note**
>
> Se non si utilizza il`default`AWS profilo, è necessario specificare il nome del profilo nell'interfaccia utente IDT for FreeRTOS. Per ulteriori informazioni sui profili, consulta.

ConfigurazioneAWS delle credenziali con variabili di ambiente

Le variabili di ambiente sono variabili gestite dal sistema operativo e utilizzate dai comandi di sistema. Non vengono salvati se si chiude la sessione SSH. L'interfaccia utente IDT per FreeRTOS utilizza le variabili di`AWS_SECRET_ACCESS_KEY` ambiente`AWS_ACCESS_KEY_ID` e per archiviareAWS le credenziali.

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
```

Apri l'interfaccia utente IDT per FreeRTOS

Per aprire l'interfaccia utente IDT for FreeRTOS

1. Scarica una versione IDT supportata per FreeRTOS. Quindi estrai l'archivio scaricato in una directory per la quale hai i permessi di lettura e scrittura.
2. Accedere alla directory di installazione di IDT for FreeRTOS:

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

3. Esegui il comando seguente per aprire l'interfaccia IDT per FreeRTOS:

Linux

```
.devicetester_ui_linux_x86-64
```

Windows

```
./devicetester_ui_win_x64-64
```

macOS

```
./devicetester_ui_mac_x86-64
```

> **Note**
>
> In macOS, per consentire al sistema di eseguire l'interfaccia utente, vai su Preferenze di Sistema -> Sicurezza e privacy. Quando si eseguono i test, potrebbe essere necessario eseguire questa operazione altre tre volte. questo

L'interfaccia utente IDT per FreeRTOS si apre nel browser predefinito. Le ultime tre versioni principali dei seguenti browser supportano l'interfaccia utente:

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

> **Note**
>
> Per un'esperienza migliore, consigliamo Google Chrome o Mozilla Firefox per accedere all'interfaccia utente IDT for FreeRTOS. Microsoft Internet Explorer non è supportato dall'interfaccia utente.

> **Important**
>
> È necessario configurare leAWS credenziali prima di aprire l'interfaccia utente. Se non hai configurato le tue credenziali, chiudi la finestra del browser dell'interfaccia utente di IDT for FreeRTOS, segui i passaggi indicatiConfigurazioneAWS delle credenziali, quindi riapri l'interfaccia utente di IDT for FreeRTOS.

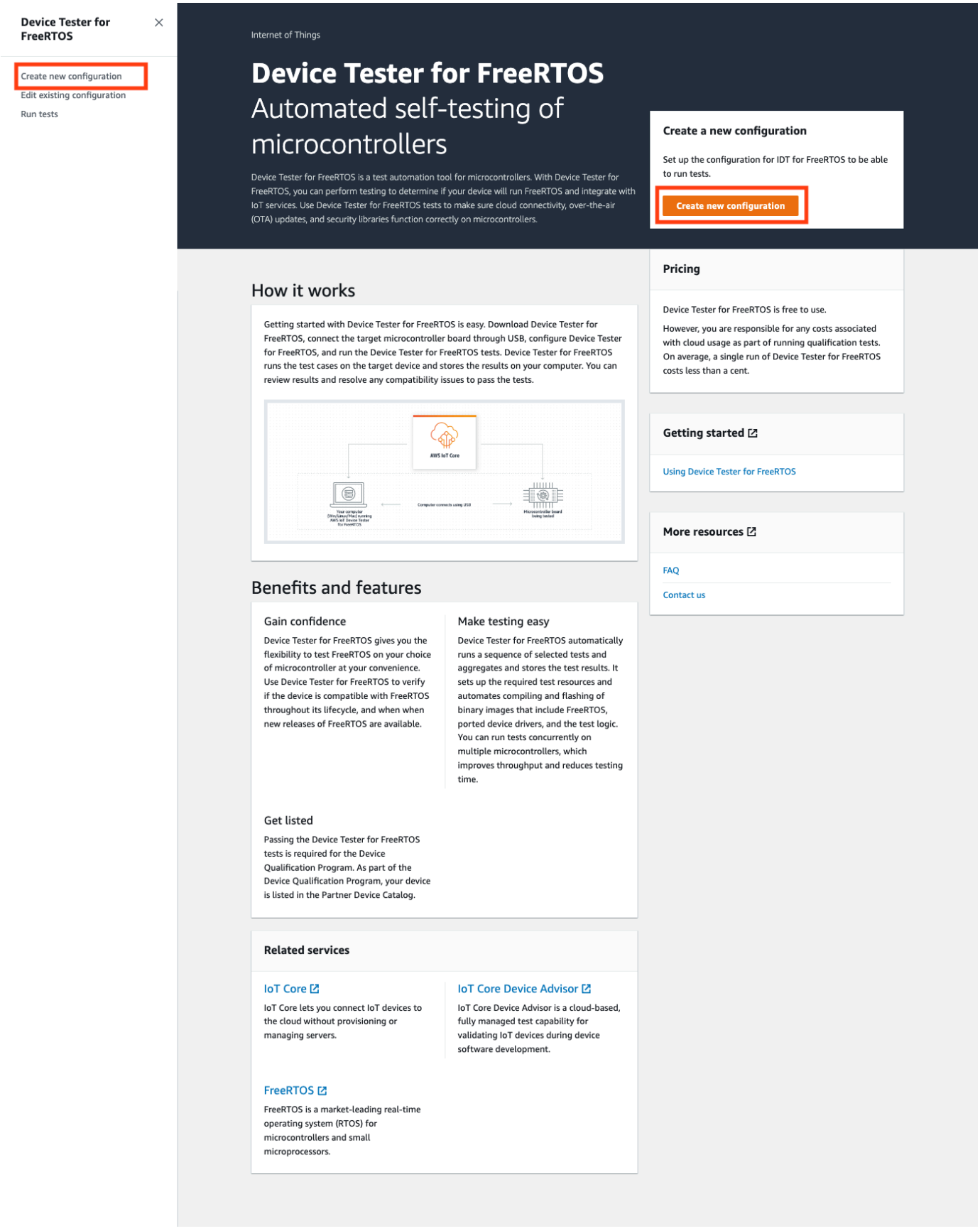
Crea una nuova configurazione

Se sei un utente alle prime armi, devi creare una nuova configurazione per configurare i file di configurazione JSON necessari per IDT for FreeRTOS per eseguire i test. È quindi possibile eseguire test o modificare la configurazione creata.

Per esempi di`userdata.json` file`config.jsondevice.json`, e, vederePreparazione dei test per la prima verifica della scheda del microcontrollore.

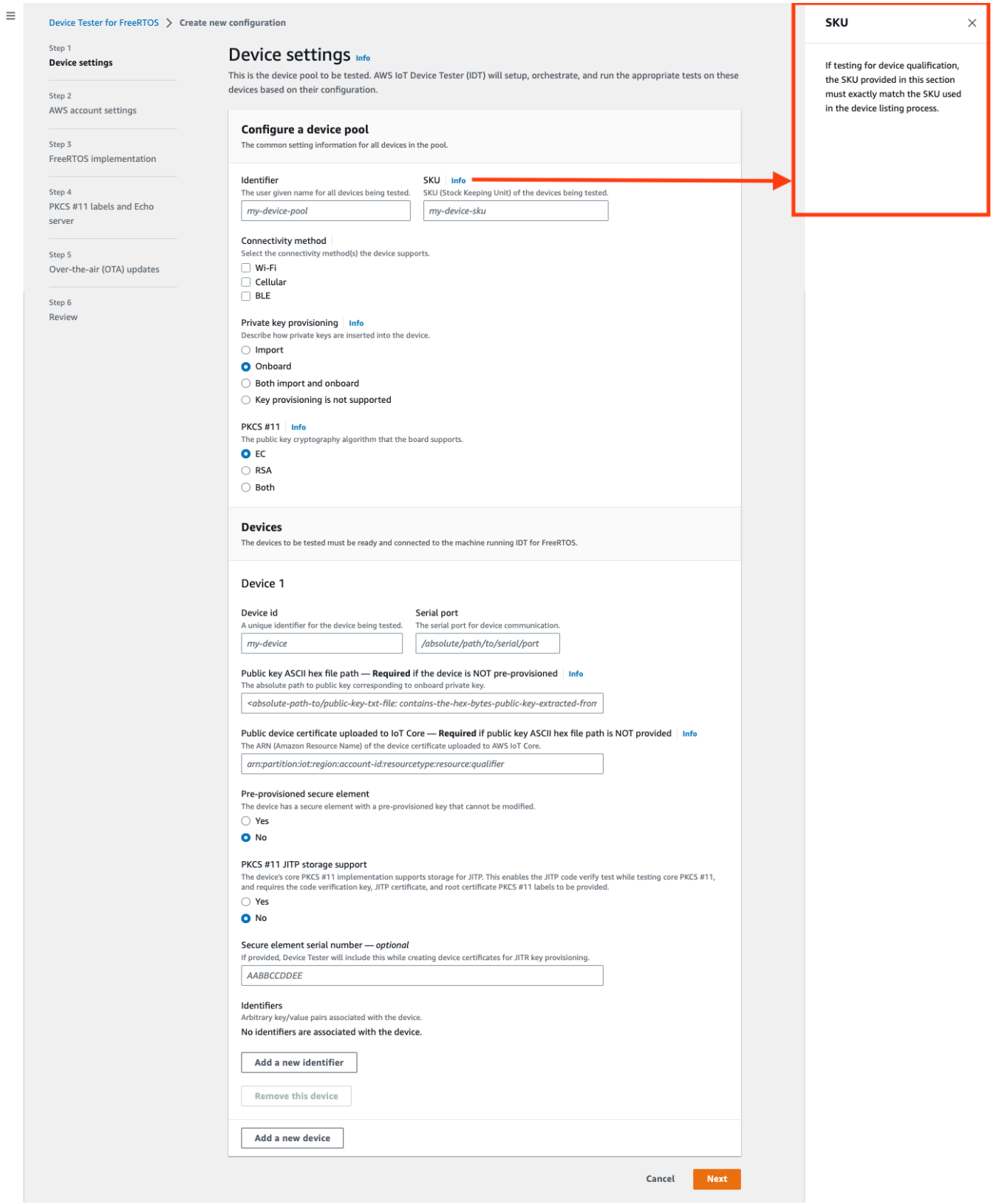
Per creare una nuova configurazione

1. Nell'interfaccia utente di IDT for FreeRTOS, apri il menu di navigazione e scegli Crea nuova configurazione.

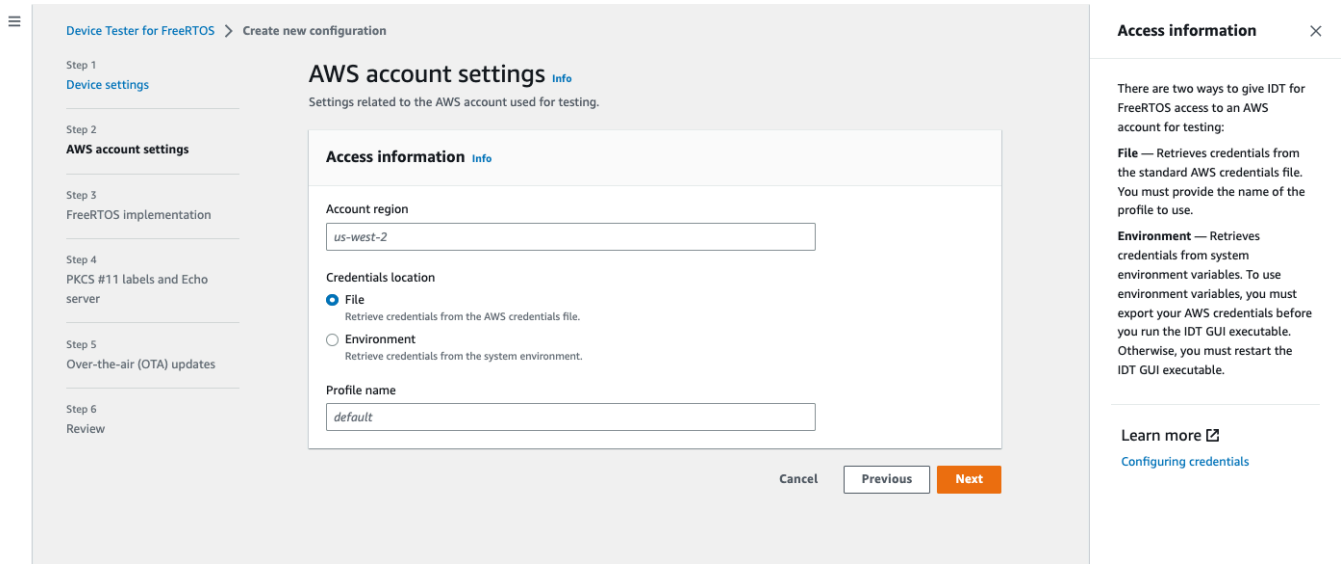

Device Tester for FreeRTOS
Create new configuration
Edit existing configuration
Run tests
Internet of Things
Device Tester for FreeRTOS
Automated self-testing of microcontrollers
Device Tester for FreeRTOS is a test automation tool for microcontrollers. With Device Tester for FreeRTOS, you can perform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

2. Segui la procedura guidata di configurazione per inserire le impostazioni di configurazione IDT utilizzate per eseguire i test di qualificazione. La procedura guidata configura le seguenti impostazioni nei file di configurazione JSON che si trovano nella*devicetester-extract-location*/config directory.

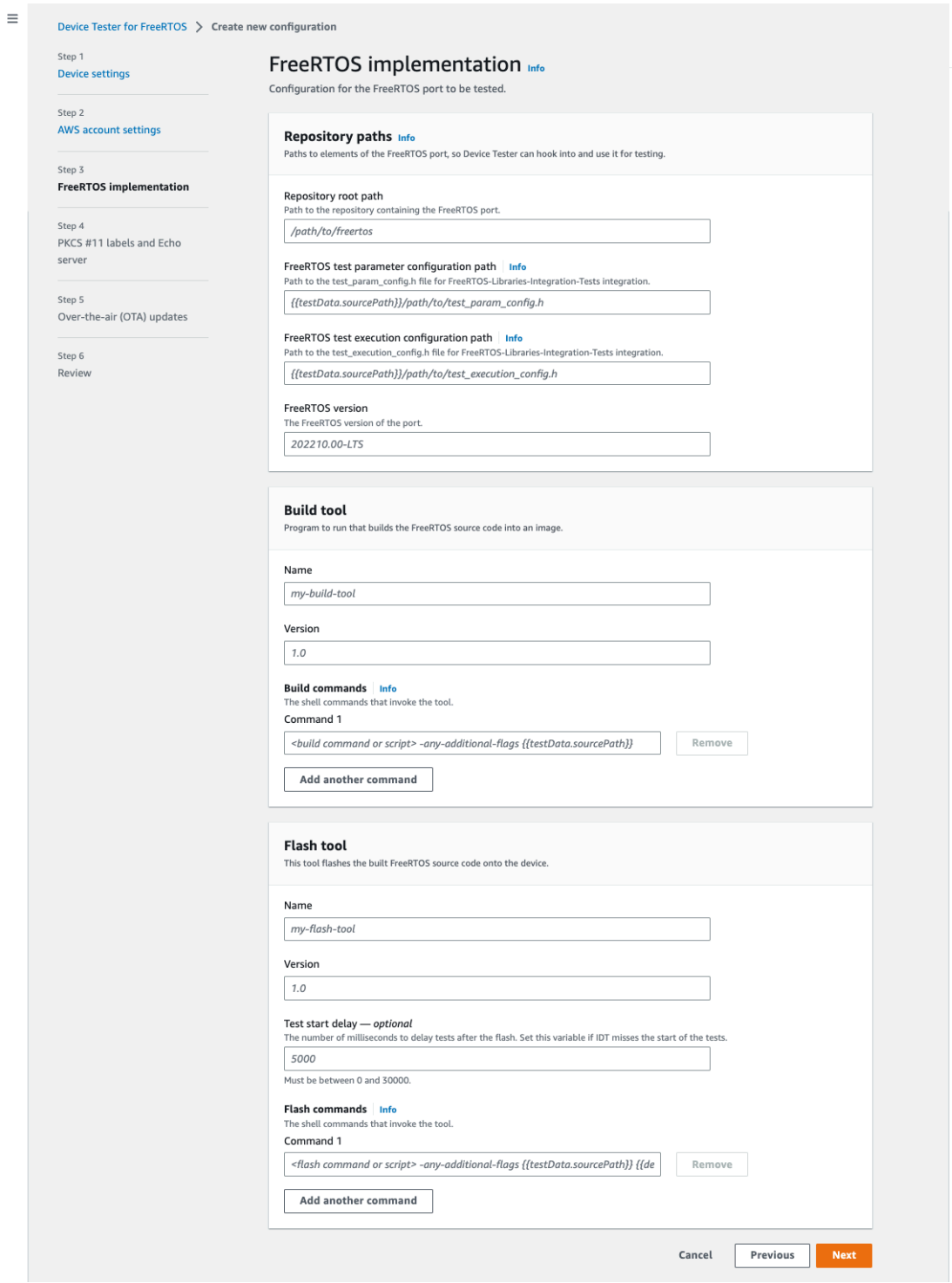
   - Impostazioni del dispositivo: le impostazioni del pool di dispositivi per i dispositivi da testare. Queste impostazioni sono configurate neisku campiid and e il blocco dei dispositivi per il pool di dispositivi nelconfig.json file.


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

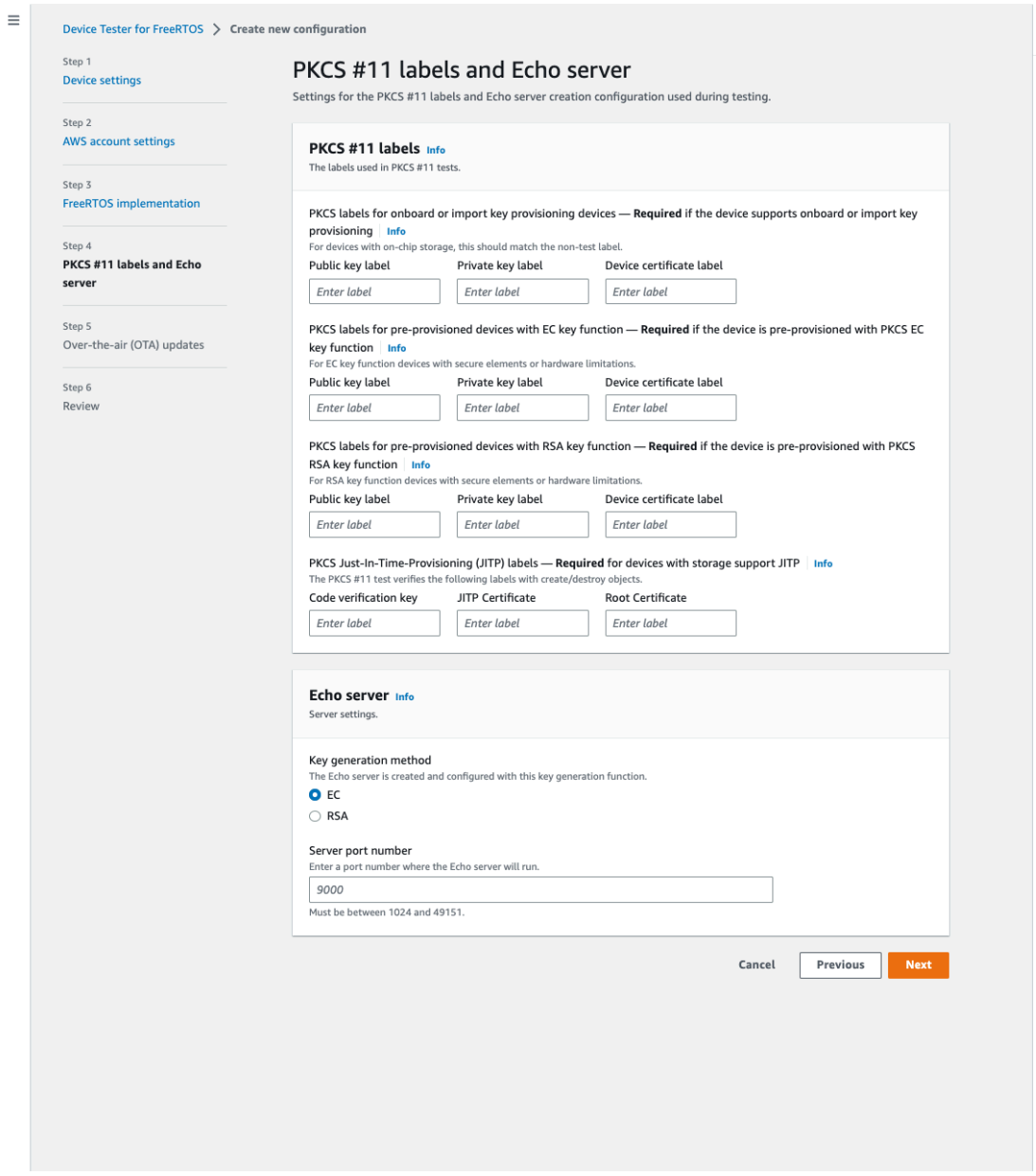
- AWSimpostazioni dell'account: leAccount AWS informazioni utilizzate da IDT for FreeRTOS per creareAWS risorse durante le esecuzioni di test. Queste impostazioni sono configurate nel`config.json` file.



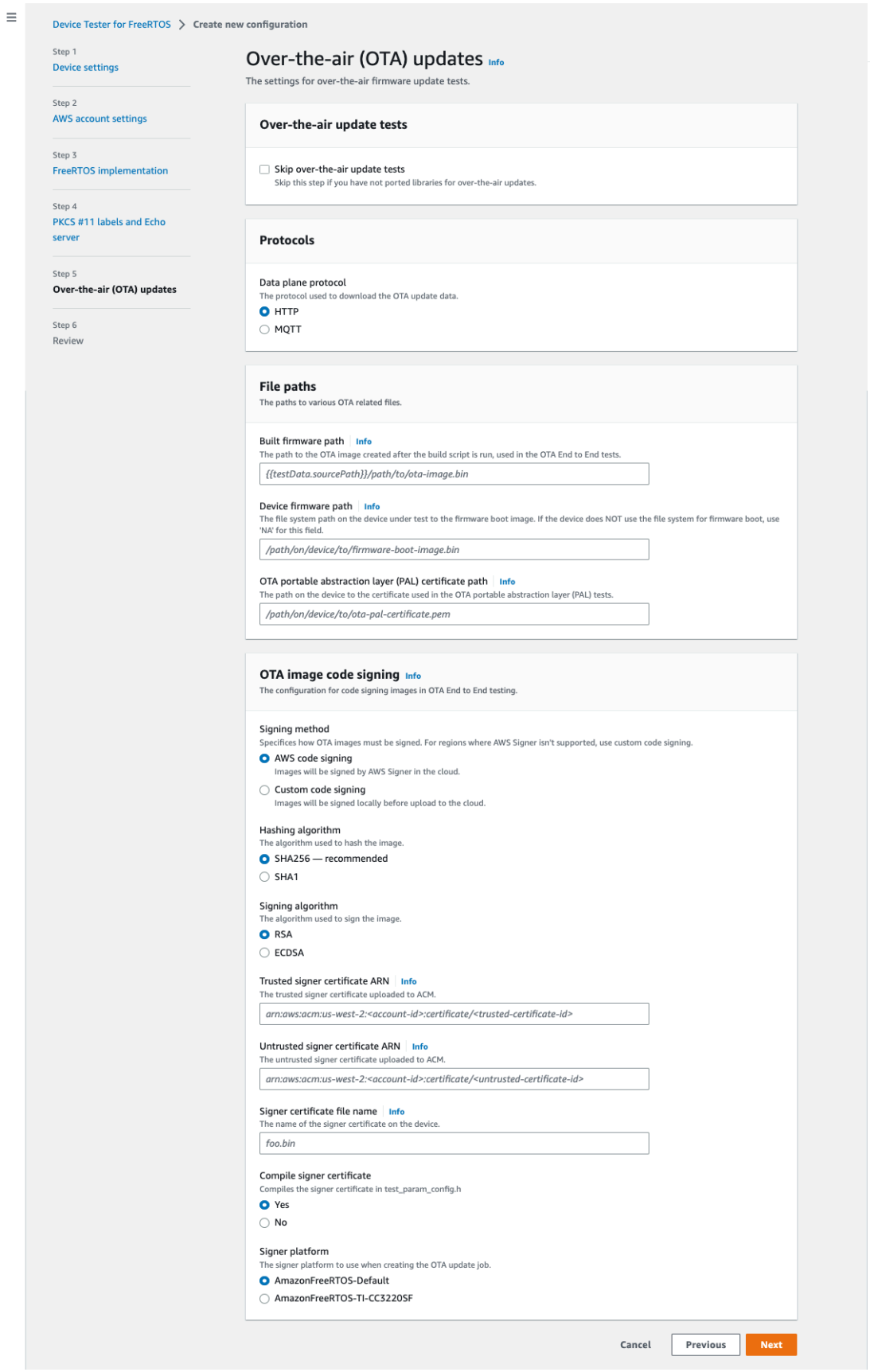
- Implementazione FreeRTOS: il percorso assoluto del repository FreeRTOS e del codice trasferito e la versione di FreeRTOS su cui si desidera eseguire IDT FRQ. I percorsi dei file di intestazione dell'esecuzione e della configurazione dei parametri dal`FreeRTOS-Libraries-Integration-Tests` GitHub repository. I comandi build e flash per l'hardware che consentono a IDT di compilare e flashare automaticamente i test sulla scheda. Queste impostazioni sono configurate nel`userdata.json` file.



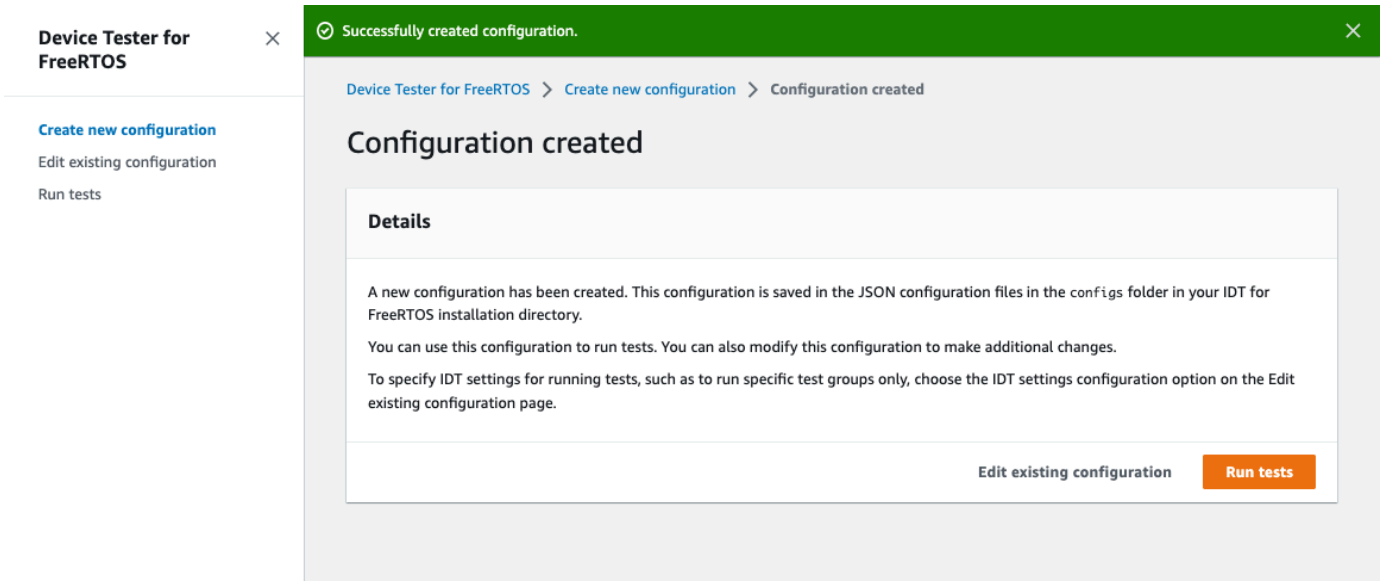
- Etichette PKCS #11 e server Echo: le etichette PKCS #11 che corrispondono alle chiavi fornite nell'hardware in base alla funzionalità chiave e al metodo di fornitura delle chiavi. Le impostazioni di configurazione del server echo per i test dell'interfaccia di trasporto. Queste impostazioni sono configurate neidevice.json fileuserdata.json and.



- Aggiornamenti Over-the-air (OTA): le impostazioni che controllano i test di funzionalità OTA. Queste impostazioni sono configurate nel`features` blocco dei`userdata.json` file`device.json` and.


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

3. Nella pagina Revisione, verifica le informazioni di configurazione.



Dopo aver esaminato la configurazione, per eseguire i test di qualificazione, scegli Esegui test.

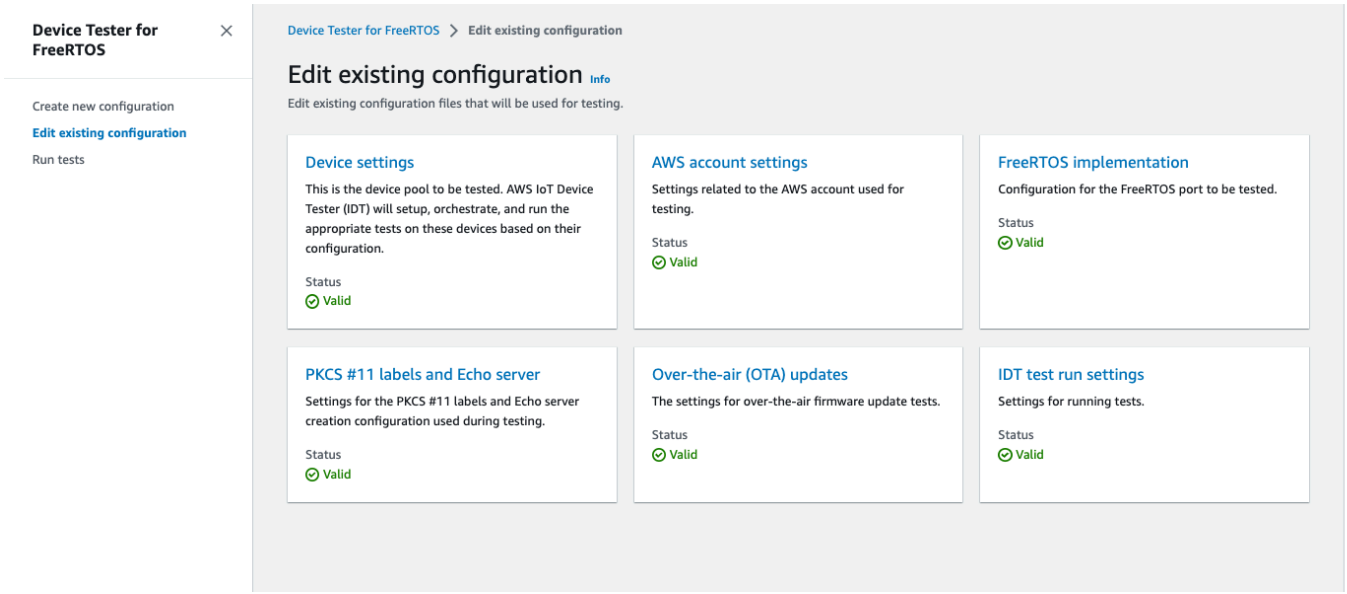
Modifica di una configurazione esistente

Se hai già impostato i file di configurazione per IDT for FreeRTOS, puoi utilizzare l'interfaccia utente IDT for FreeRTOS per modificare la configurazione esistente. I file di configurazione esistenti devono trovarsi nella*devicetester-extract-location*`/config` directory.

Per modificare una configurazione

1. Nell'interfaccia utente di IDT for FreeRTOS, apri il menu di navigazione e scegli Modifica configurazione esistente.

   La dashboard di configurazione mostra informazioni sulle impostazioni di configurazione esistenti. Se una configurazione non è corretta o non è disponibile, lo stato di tale configurazione è`Error validating configuration`.

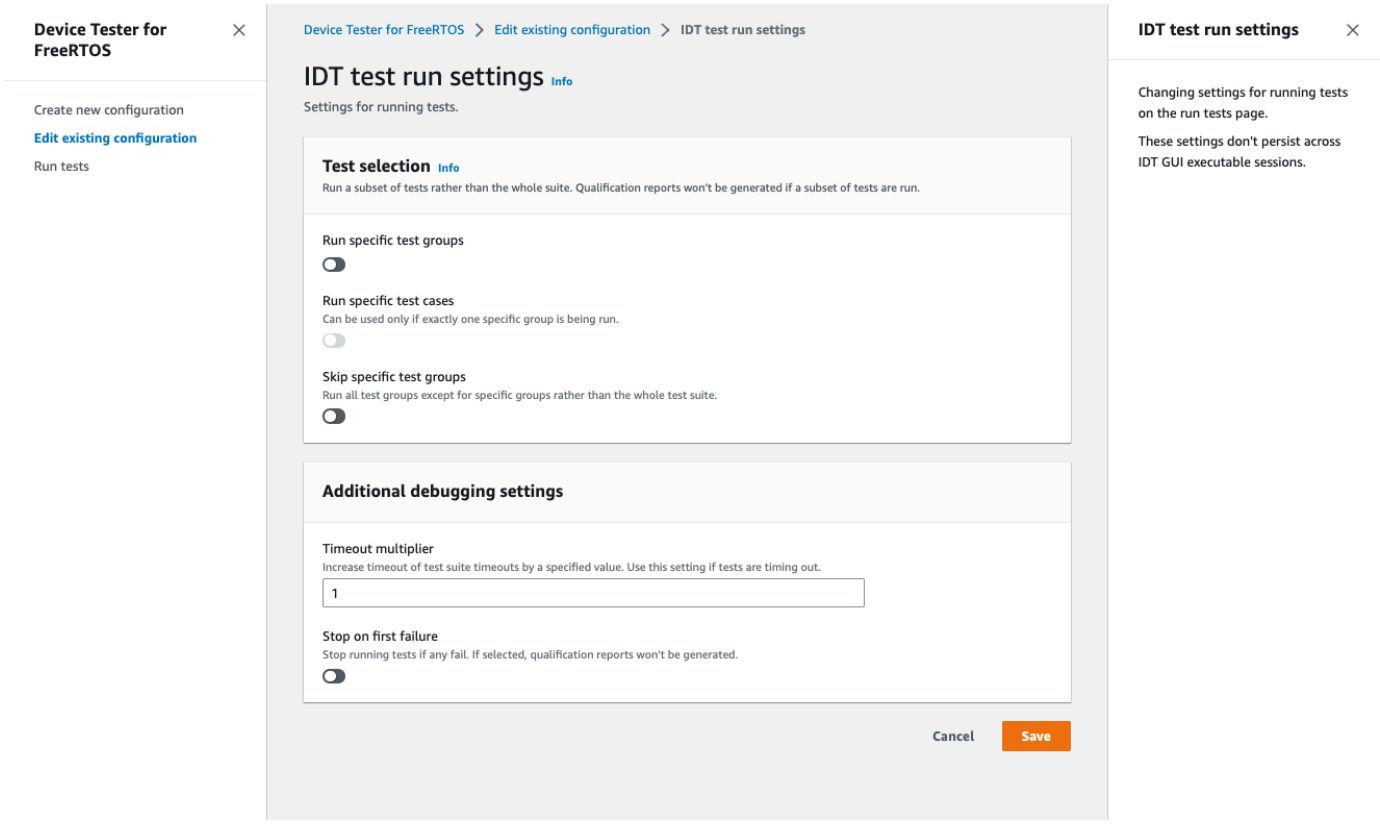


2. Per modificare un'impostazione di configurazione esistente, completa la procedura seguente:

   a. Scegli il nome di un'impostazione di configurazione per aprire la relativa pagina delle impostazioni.

   b. Modifica le impostazioni, quindi scegli Salva per rigenerare il file di configurazione corrispondente.

3. Per modificare le impostazioni dell'esecuzione del test IDT for FreeRTOS, scegli le impostazioni dell'esecuzione del test IDT nella vista di modifica:



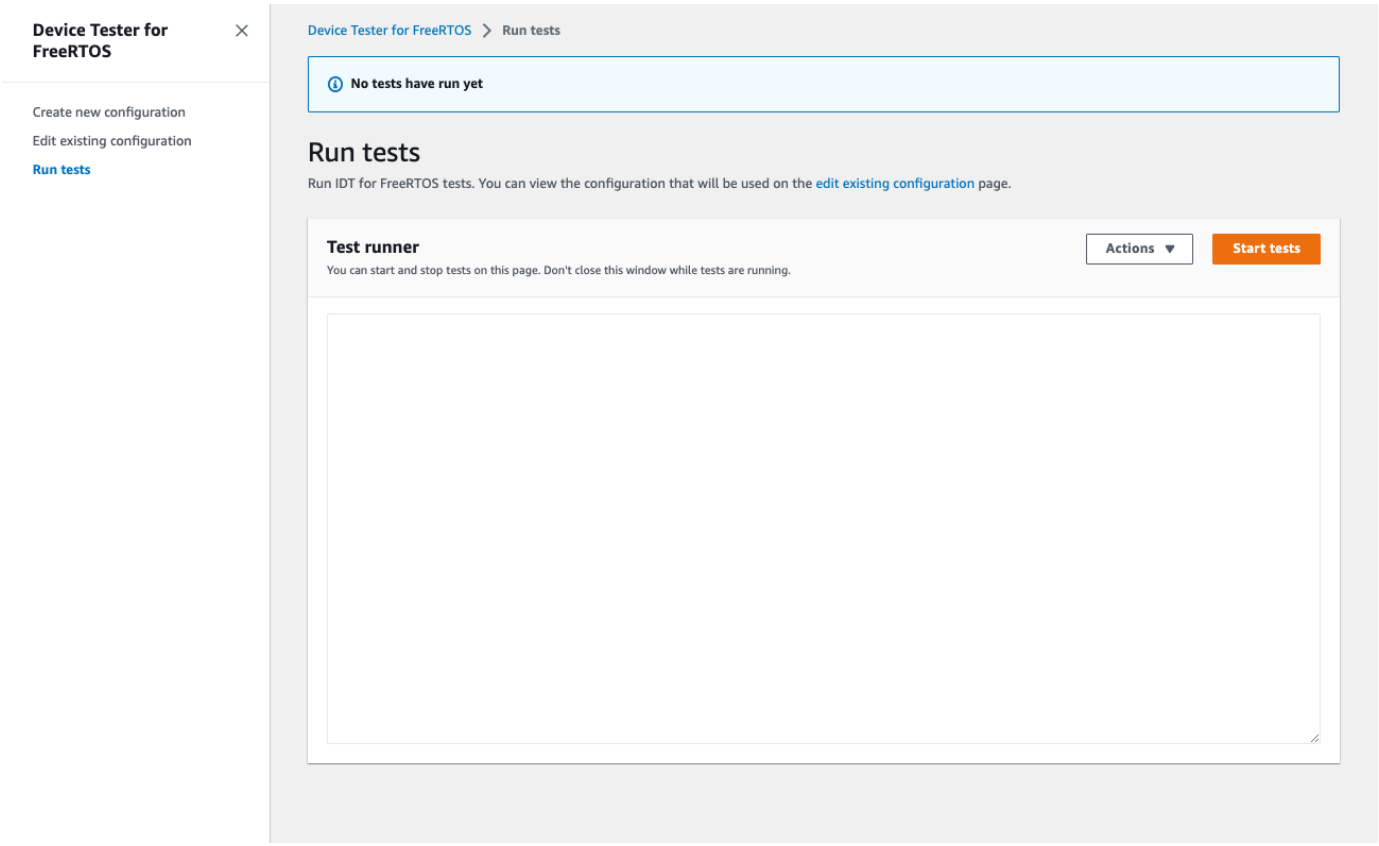
Dopo aver completato la modifica della configurazione, verifica che tutte le impostazioni di configurazione superino la convalida. Se lo stato di ciascuna impostazione di configurazione è`Valid`, puoi eseguire i test di qualificazione con questa configurazione.

Esegui test di qualificazione

Dopo aver creato una configurazione per l'interfaccia utente IDT for FreeRTOS, puoi eseguire i test di qualificazione.

Per eseguire test di qualificazione

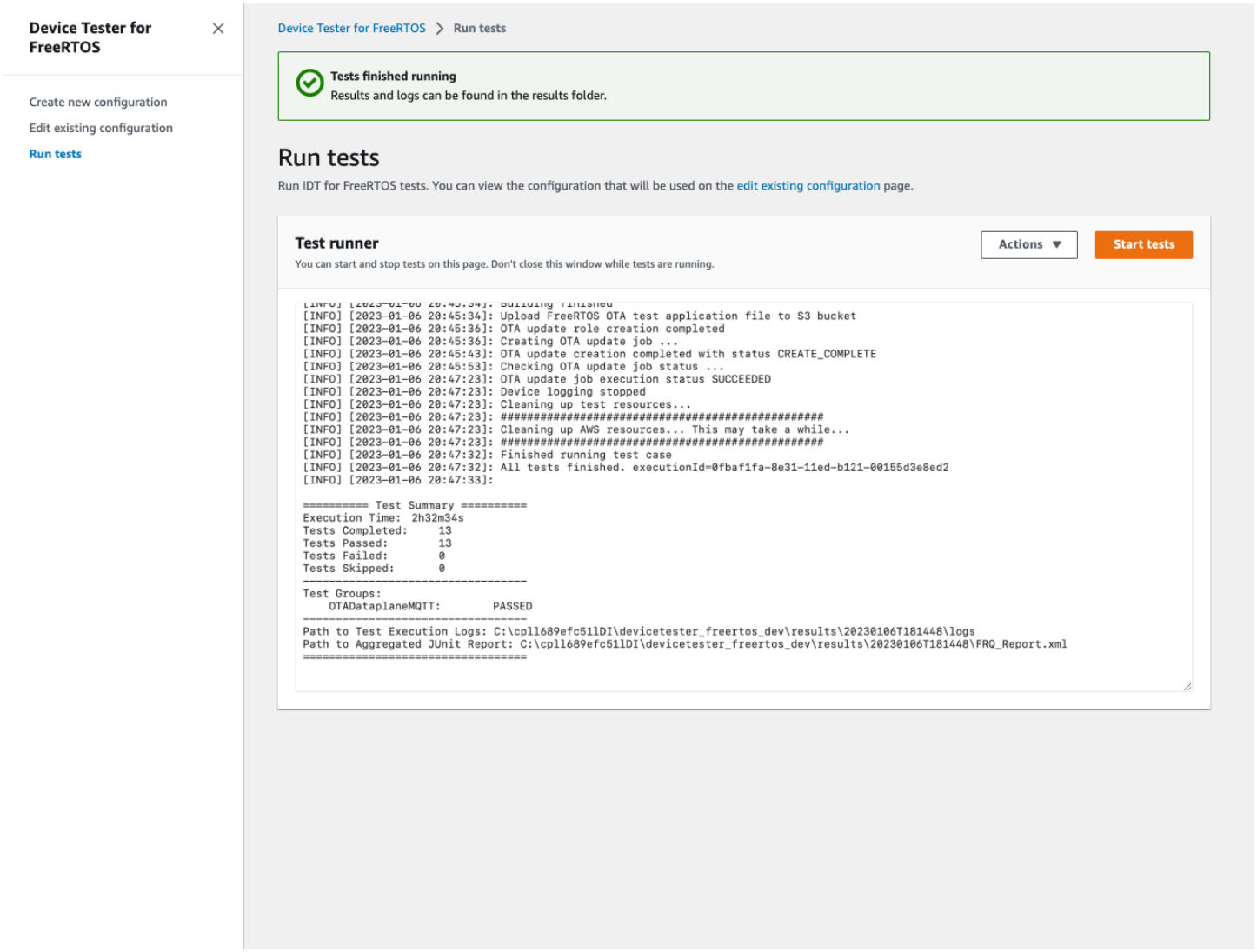
1. Nel menu di navigazione, scegliere Esegui test.
2. Scegli Avvia test per avviare l'esecuzione del test. Per impostazione predefinita, vengono eseguiti tutti i test applicabili per la configurazione del dispositivo. IDT per FreeRTOS genera un rapporto di qualificazione al termine di tutti i test.



IDT per FreeRTOS esegue i test di qualificazione. Quindi visualizza il riepilogo dell'esecuzione del test e gli eventuali errori nella console Test runner. Dopo aver completato l'esecuzione del test, è possibile visualizzare i risultati del test e i registri dalle seguenti posizioni:

- I risultati del test si trovano nella*devicetester-extract-location*`/results/`*execution-id* directory.
- I log del test si trovano nella*devicetester-extract-location*`/results/`*execution-id*`/logs` directory.

Per ulteriori informazioni sui risultati del test e sui log, consultaInformazioni su risultati e log.



Esecuzione della suite di qualificazione FreeRTOS 2.0

Usa 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 v4.5.2 and later

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

Esegue una suite di test in un determinato pool di dispositivi. Il file `userdata.json` si deve trovare nella directory *`devicetester_extract_location`*`/devicetester_freertos_`*`[win|mac|linux]`*`/configs/`.

> **Note**
>
> Se stai eseguendo 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_1.99.0  \
    --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.

Opzioni della riga di comando IDT per 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.

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 FreeRTOSVersion --
test-id FreeRTOSVersion
```

h

Utilizza l'opzione di aiuto per ulteriori informazioni sulle opzioni `run-suite`.

Example

Esempio

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

Comandi IDT per FreeRTOS

Il comando IDT for FreeRTOS supporta le operazioni seguenti:

IDT v4.5.2 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 FreeRTOSVersion --
test-id FreeRTOSVersion
```

> **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.

Informazioni su risultati e log

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 sul 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 casi di test suddivisi per le librerie sottoposte a test in base alle caratteristiche del dispositivo.

`FRQ_Report.xml` è un report in formato JUnit XML format standard. È possibile integrarlo nelle piattaforme CI/CD quali 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 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. Ad esempio:

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

Attributi utilizzati nel **`<testsuites>`** tag

`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 stato in grado di eseguire.

`disabled`

Questo attributo non è utilizzato e si può ignorare.

Se non ci sono errori o errori nei test case, il 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 dell'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="FreeRTOSVersion" package="" tests="1" failures="0"
time="2" 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. Ad esempio:

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

Attributi utilizzati nel **`<awsproduct>`** tag

`name`

Il nome del prodotto sottoposto a test.

`version`

La versione del prodotto sottoposto a test.

`features`

Le caratteristiche convalidate. Le caratteristiche contrassegnate come `required` sono necessarie per inviare la scheda per la qualifica. Il frammento seguente mostra come appare nel `awsiotdevicetester_report.xml` file.

```
<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 ci sono 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 AWSPartner 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. Ad 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. Ad esempio:

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

Attributi utilizzati nel **`<testcase>`** tag

`name`

Il nome del caso di test.

`attempts`

Il numero di volte in cui IDT for 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. Ad esempio:

```
<testcase classname="FRQ FreeRTOSVersion" name="FreeRTOSVersion">
    <failure type="Failure">Reason for the test case failure</failure>
    <error>Reason for the test case execution error</error>
</testcase>
```

Per ulteriori informazioni, consulta Risoluzione dei problemi.

Visualizzazione dei registri di

Puoi trovare i log che IDT for FreeRTOS genera dall'esecuzione dei test in. *devicetester-extract-location*`/results/`*execution-id*`/logs` Vengono generate due serie di log:

- `test_manager.log`

  Contiene i log generati da IDT for FreeRTOS (ad esempio, la configurazione relativa ai log e la generazione di report).

- *test_group_id*`/`*test_case_id*`/`*test_case_id*`.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 con la suite di qualificazione FreerTOS 1.0 (FRQ 1.0)

> **Important**
>
> A ottobre 2022, AWS IoT Device Tester per AWS IoT FreerTOS Qualification (FRQ) 1.0 non genera report di qualificazione firmati. Non è possibile qualificare nuovi dispositivi AWS IoT FreerTOS da inserire nel Partner Device Catalog tramite AWS il Device Qualification Program utilizzando AWS le versioni IDT FRQ 1.0. Sebbene non sia possibile qualificare i dispositivi FreerTOS utilizzando IDT FRQ 1.0, è possibile continuare a testare i dispositivi FreerTOS

con FRQ 1.0. Ti consigliamo di utilizzare IDT FRQ 2.0 per qualificare ed elencare i dispositivi FreerTOS nel Partner Device Catalog. AWS

Puoi utilizzare la qualificazione IDT for FreerTOS per verificare che il sistema operativo FreeRTOS funzioni localmente sul tuo dispositivo e con cui possa comunicare. 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.

Argomenti

- Prerequisiti

- Preparazione dei test per la prima verifica della scheda del microcontrollore
- Usa l'interfaccia utente IDT for FreerTOS per eseguire la suite di qualificazione FreerTOS
- Esecuzione dei test Bluetooth Low Energy
- Esecuzione della suite di qualificazione FreerTOS
- Informazioni su risultati e log

Prerequisiti

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 diAWS IoT Device Testerper 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\`

Considerazioni per la qualificazione 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 TesterIDT) 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 diAWS IoT Device Testerper 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.

Creazione e configurazione di un account AWS

Registrarsi per creare un Account AWS

Se non disponi di un Account AWS, completa la procedura seguente per crearne uno.

Per registrarsi a un Account AWS

1. Apri la pagina all'indirizzo https://portal.aws.amazon.com/billing/signup.
2. Segui le istruzioni online.

   Nel corso della procedura di registrazione riceverai una telefonata, durante la quale sarà necessario inserire un codice di verifica attraverso la tastiera del telefono.

   Durante la registrazione di un Account AWS, viene creato un Utente root dell'account AWS. L'utente root dispone dell'accesso a tutte le risorse e tutti i Servizi AWSnell'account. Come best practice di sicurezza, assegna l'accesso amministrativo a un utente amministrativo e utilizza solo l'utente root per eseguire attività che richiedono l'accesso di un utente root.

Al termine del processo di registrazione, riceverai un'e-mail di conferma da AWS. È possibile visualizzare l'attività corrente dell'account e gestire l'account in qualsiasi momento accedendo all'indirizzo https://aws.amazon.com/ e selezionando Il mio account.

Creazione di un utente amministratore

Dopo aver effettuato la registrazione di un Account AWS, proteggi Utente root dell'account AWS, abilita AWS IAM Identity Center e crea un utente amministratore in modo da non utilizzare l'utente root per le attività quotidiane.

Protezione dell'Utente root dell'account AWS

1. Accedi alla AWS Management Console come proprietario dell'account scegliendo Utente root e immettendo l'indirizzo email del Account AWS. Nella pagina successiva, inserisci la password.

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

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

Creazione di un utente amministratore

1. Abilita IAM Identity Center

   Per istruzioni, consulta Abilitazione di AWS IAM Identity Center nella Guida per l'utente di AWS IAM Identity Center.
2. In Centro identità AWS IAM, assegna l'accesso amministrativo a un utente amministrativo.

   Per un tutorial sull'utilizzo di IAM Identity Center directory come origine di identità, consulta Configure user access with the default IAM Identity Center directory nella Guida per l'utente di AWS IAM Identity Center.

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 Accedere al portale di accesso AWS nella Guida per l'utente Accedi ad AWS.

Policy gestita di AWS IoT Device Tester

La policy `AWSIoTDeviceTesterForFreeRTOSFullAccess` gestita contiene le seguenti AWS IoT Device Tester autorizzazioni per il controllo della versione, le funzionalità di aggiornamento automatico e la raccolta di metriche.

- `iot-device-tester:SupportedVersion`

  Concede AWS IoT Device Tester l'autorizzazione a recuperare l'elenco dei prodotti supportati, delle suite di test e delle versioni IDT.
- `iot-device-tester:LatestIdt`

  Concede AWS IoT Device Tester l'autorizzazione a recuperare l'ultima versione IDT disponibile per il download.
- `iot-device-tester:CheckVersion`

  Concede AWS IoT Device Tester l'autorizzazione a verificare la compatibilità delle versioni per IDT, suite di test e prodotti.
- `iot-device-tester:DownloadTestSuite`

  Concede AWS IoT Device Tester l'autorizzazione a scaricare gli aggiornamenti della suite di test.
- `iot-device-tester:SendMetrics`

  Concede AWS l'autorizzazione a raccogliere metriche sull'AWS IoT Device Testerutilizzo interno.

(Facoltativo) Installazione dell'AWS Command Line Interface

In alternativa, puoi utilizzare l'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

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.

Configurazione delle credenziali AWS

Devi configurare le tue AWS credenziali per AWS IoT Device Tester comunicare con il Cloud. AWS Per ulteriori informazioni, consulta Configurare AWS le credenziali e la regione per lo sviluppo. AWSLe 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, consulta 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"
                    }
                ]
            },
            {
                "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 Testerutilizza queste informazioni per selezionare i test di qualificazione da eseguire.

I valori supportati sono:

`TCP/IP`

Indica se la scheda supporta uno stack TCP/IP e, in caso positivo, se è supportato su chip (MCU) o se viene eseguito l'offload su un altro modulo. Il protocollo TCP/IP è obbligatorio per la qualifica.

`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 dotata di funzionalità di 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 istanze AWS t2.micro EC2 e ciò potrebbe comportare costi aggiuntivi per l'account. Per ulteriori informazioni, consulta Prezzi di Amazon EC2.

`TLS`

Indica se la scheda supporta TLS. TLS è obbligatorio per la qualifica.

`PKCS11`

Indica l'algoritmo di crittografia della chiave pubblica supportato dalla scheda. PKCS11 è obbligatorio per la qualifica. 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.

> **Note**
>
> A partire da IDT v4.1.0, `OtaDataPlaneProtocol` accetta solo `HTTP` e `MQTT` come valori supportati.

`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 demo/test FreerTOS.

`devices.secureElementConfig.preProvisioned`

(Facoltativo) Imposta su «Sì» se il dispositivo dispone di un elemento sicuro predisposto 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": [
            "{{config.idtRootPath}}/relative-path-to/build-parallel.sh
{{testData.sourcePath}} {{enableTests}}"
        ]
    },
    "flashTool": {
        "name": "your-flash-tool-name",
        "version": "your-flash-tool-version",
        "command": [
            "/{{config.idtRootPath}}/relative-path-to/flash-parallel.sh
{{testData.sourcePath}} {{device.connectivity.serialPort}} {{buildImageName}}"
        ],
        "buildImageInfo" : {
            "testsImageName": "tests-image-name",
            "demosImageName": "demos-image-name"
        }
    },
    "testStartDelayms": 0,
    "clientWifiConfig": {
        "wifiSSID": "ssid",
        "wifiPassword": "password",
        "wifiSecurityType": "eWiFiSecurityOpen | eWiFiSecurityWEP | eWiFiSecurityWPA |
eWiFiSecurityWPA2 | eWiFiSecurityWPA3"
    },
    "testWifiConfig": {
        "wifiSSID": "ssid",
        "wifiPassword": "password",
        "wifiSecurityType": "eWiFiSecurityOpen | eWiFiSecurityWEP | eWiFiSecurityWPA |
eWiFiSecurityWPA2 | eWiFiSecurityWPA3"
    },
    //**********
    //This section is used to start echo server based on server certificate generation
method,
    //When certificateGenerationMethod is set as Automatic specify the eccCurveFormat
to generate certifcate and key based on curve format,
    //When certificateGenerationMethod is set as Custom specify the certificatePath and
PrivateKeyPath to be used to start echo server
    //**********
```

```
    "echoServerCertificateConfiguration": {
      "certificateGenerationMethod": "Automatic | Custom",
      "customPath": {
          "clientCertificatePath":"/path/to/clientCertificate",
          "clientPrivateKeyPath": "/path/to/clientPrivateKey",
          "serverCertificatePath":"/path/to/serverCertificate",
          "serverPrivateKeyPath": "/path/to/serverPrivateKey"
      },
    "eccCurveFormat": "P224 | P256 | P384 | P521"
    },
    "echoServerConfiguration": {
        "securePortForSecureSocket": 33333, // Secure tcp port used by SecureSocket
 test. Default value is 33333. Ensure that the port configured isn't blocked by the
 firewall or your corporate network
       "insecurePortForSecureSocket": 33334, // Insecure tcp port used by SecureSocket
 test. Default value is 33334. Ensure that the port configured isn't blocked by the
 firewall or your corporate network
       "insecurePortForWiFi": 33335 // Insecure tcp port used by Wi-Fi test. Default
 value is 33335. Ensure that the port configured isn't blocked by the firewall or your
 corporate network
    },
    "otaConfiguration": {
        "otaFirmwareFilePath": "{{testData.sourcePath}}/relative-path-to/ota-image-
generated-in-build-process",
       "deviceFirmwareFileName": "ota-image-name-on-device",
       "otaDemoConfigFilePath": "{{testData.sourcePath}}/relative-path-to/ota-demo-
config-header-file",
       "codeSigningConfiguration": {
           "signingMethod": "AWS | Custom",
           "signerHashingAlgorithm": "SHA1 | SHA256",
           "signerSigningAlgorithm": "RSA | ECDSA",
           "signerCertificate": "arn:partition:service:region:account-
id:resource:qualifier | /absolute-path-to/signer-certificate-file",
           "signerCertificateFileName": "signerCertificate-file-name",
           "compileSignerCertificate": boolean,
           // ***********Use signerPlatform if you choose aws for
 signingMethod***************
           "signerPlatform": "AmazonFreeRTOS-Default | AmazonFreeRTOS-TI-CC3220SF",
           "untrustedSignerCertificate": "arn:partition:service:region:account-
id:resourcetype:resource:qualifier",
           // ***********Use signCommand if you choose custom for
 signingMethod***************
           "signCommand": [
```

```
                "/absolute-path-to/sign.sh {{inputImageFilePath}}
 {{outputSignatureFilePath}}"
            ]
        }
    },
    // ***********Remove the section below if you're not configuring
 CMake**************
    "cmakeConfiguration": {
        "boardName": "board-name",
        "vendorName": "vendor-name",
        "compilerName": "compiler-name",
        "frToolchainPath": "/path/to/freertos/toolchain",
        "cmakeToolchainPath": "/path/to/cmake/toolchain"
    },
    "freertosFileConfiguration": {
        "required": [
            {
                "configName": "pkcs11Config",
                "filePath": "{{testData.sourcePath}}/vendors/vendor-name/boards/board-
name/aws_tests/config_files/core_pkcs11_config.h"
            },
            {
                "configName": "pkcs11TestConfig",
                "filePath": "{{testData.sourcePath}}/vendors/vendor-name/boards/board-
name/aws_tests/config_files/iot_test_pkcs11_config.h"
            }
        ],
        "optional": [
            {
                "configName": "otaAgentTestsConfig",
                "filePath": "{{testData.sourcePath}}/vendors/vendor-name/boards/board-
name/aws_tests/config_files/ota_config.h"
            },
            {
                "configName": "otaAgentDemosConfig",
                "filePath": "{{testData.sourcePath}}/vendors/vendor-name/boards/board-
name/aws_demos/config_files/ota_config.h"
            },
            {
                "configName": "otaDemosConfig",
                "filePath": "{{testData.sourcePath}}/vendors/vendor-name/boards/board-
name/aws_demos/config_files/ota_demo_config.h"
            }
        ]
```

```
    }
}
```

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 usando un SDK, allora 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}}` Utilizzate il `{{config.idtRootPath}}` segnaposto per fare riferimento al percorso IDT assoluto o relativo.

testStartDelayms

Speciifica quanti millisecondi aspetterà il test runner FreerTOS prima di iniziare a eseguire i test. Ciò può essere utile se il dispositivo sottoposto a test inizia a emettere importanti informazioni di test prima che IDT abbia la possibilità di connettersi e avviare la registrazione a causa della rete o di altra latenza. Il valore massimo consentito è 30000 ms (30 secondi). Questo valore è applicabile solo ai gruppi di test FreerTOS e non applicabile ad altri gruppi di test che non utilizzano il test runner FreerTOS, come i test OTA.

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. Utilizzate il segnaposto per fare riferimento al percorso IDT assoluto o relativo. `{{sdkPath}}` `{{config.idtRootPath}}`

`buildImageInfo`

`testsImageName`

Il nome del file prodotto dal comando build durante la creazione dei test dalla cartella. *`freertos-source`*`/tests`

`demosImageName`

Il nome del file prodotto dal comando build durante la creazione dei test dalla *`freertos-source`*`/demos` cartella.

`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 codice AWS, utilizza l'Amazon Resource Name (ARN) per il certificato attendibile 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 nome del file del certificato di firma del codice sul dispositivo. Questo valore deve corrispondere al nome di file fornito durante l'esecuzione del `aws acm import-certificate` comando.

Per ulteriori informazioni, consulta Creazione di un certificato di firma del codice.

`compileSignerCertificate`

Imposta su `true` se il certificato di verifica della firma del firmatario del codice non è stato sottoposto a provisioning o flashing, in modo che venga compilato nel progetto. AWS IoT Device Tester recupera il certificato attendibile e lo compila in `aws_codesigner_certifiate.h`.

`untrustedSignerCertificate`

L'ARN o il percorso del file per un secondo certificato utilizzato in alcuni test OTA come certificato non attendibile. Per ulteriori informazioni sulla creazione di un certificato, consulta Creare un certificato di firma del codice.

`signerPlatform`

L'algoritmo di firma e hash che AWS Code Signer utilizza durante la creazione dell'attività 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.

`cmakeConfiguration`

Configurazione CMake [opzionale]

> **Note**
>
> Per eseguire i casi di test di CMake, devi specificare il nome della scheda, il nome del fornitore e `frToolchainPath` oppure `compilerName`. Puoi anche fornire, `cmakeToolchainPath` se disponi di un percorso personalizzato, alla toolchain di CMake.

`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`

Percorso completo del 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 sono stati spostati i file di configurazione, ad esempio PKCS11, 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 casi di test di CMake, devi specificare il nome della scheda, il nome del fornitore e `afrToolchainPath` oppure `compilerName`. Puoi anche specificare `cmakeToolchainPath` se disponi di un percorso personalizzato per la toolchain CMake.

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 Testerconsente 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.

`{{config.idtRootPath}}`

Si espande fino al percorso principale. AWS IoT Device Tester Questa variabile sostituisce il percorso assoluto per IDT quando viene utilizzata dai comandi build e flash.

Usa l'interfaccia utente IDT for FreerTOS per eseguire la suite di qualificazione FreerTOS

A partire da IDT v4.3.0, for AWS IoT Device Tester FreerTOS (IDT-FreeRTOS) include un'interfaccia utente basata sul Web che consente di interagire con l'eseguibile della riga di comando IDT e i relativi file di configurazione. È possibile utilizzare l'interfaccia utente IDT-FreeRTOS per creare una nuova configurazione per eseguire test IDT o per modificare una configurazione esistente. È inoltre possibile utilizzare l'interfaccia utente per richiamare l'eseguibile IDT ed eseguire test.

L'interfaccia utente IDT-FreeRTOS offre le seguenti funzioni:

- Semplifica la configurazione dei file di configurazione per i test IDT-FreeRTOS.
- Semplifica l'utilizzo di IDT-FreeRTOS per eseguire test di qualificazione.

Per informazioni sull'utilizzo della riga di comando per eseguire i test di qualificazione, vedere. Preparazione dei test per la prima verifica della scheda del microcontrollore

Questa sezione descrive i prerequisiti per l'utilizzo dell'interfaccia utente IDT-FreeRTOS e mostra come iniziare a eseguire test di qualificazione nell'interfaccia utente.

Argomenti

Prerequisiti

Questa sezione descrive i prerequisiti per testare i microcontrollori con. AWS IoT Device Tester

Argomenti

Utilizza un browser Web supportato

L'interfaccia utente IDT-FreeRTOS supporta i seguenti browser Web.

Browser	Versione
Google Chrome	Le ultime tre versioni principali
Mozilla Firefox	Le ultime tre versioni principali
Microsoft Edge	Le ultime tre versioni principali
Apple Safari per macOS	Le ultime tre versioni principali

Ti consigliamo di utilizzare Google Chrome o Mozilla Firefox per un'esperienza migliore.

> **Note**
>
> L'interfaccia utente IDT-FreeRTOS non supporta Microsoft Internet Explorer.

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 diAWS IoT Device Testerper 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\`

Considerazioni per la qualificazione 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 TesterIDT) 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 diAWS IoT Device Testerper 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**
>
> Si consiglia di estrarre il pacchetto IDT in un'unità locale. Consentire a più utenti di eseguire IDT da una posizione condivisa, ad esempio una directory NFS o una cartella condivisa di rete Windows, potrebbe causare la mancata risposta del sistema o il danneggiamento dei dati.

Creazione e configurazione di un account AWS

Registrarsi per creare un Account AWS

Se non disponi di un Account AWS, completa la procedura seguente per crearne uno.

Per registrarsi a un Account AWS

1. Apri la pagina all'indirizzo https://portal.aws.amazon.com/billing/signup.

2. Segui le istruzioni online.

   Nel corso della procedura di registrazione riceverai una telefonata, durante la quale sarà necessario inserire un codice di verifica attraverso la tastiera del telefono.

   Durante la registrazione di un Account AWS, viene creato un Utente root dell'account AWS. L'utente root dispone dell'accesso a tutte le risorse e tutti i Servizi AWSnell'account. Come best practice di sicurezza, assegna l'accesso amministrativo a un utente amministrativo e utilizza solo l'utente root per eseguire attività che richiedono l'accesso di un utente root.

Al termine del processo di registrazione, riceverai un'e-mail di conferma da AWS. È possibile visualizzare l'attività corrente dell'account e gestire l'account in qualsiasi momento accedendo all'indirizzo https://aws.amazon.com/ e selezionando Il mio account.

Creazione di un utente amministratore

Dopo aver effettuato la registrazione di un Account AWS, proteggi Utente root dell'account AWS, abilita AWS IAM Identity Center e crea un utente amministratore in modo da non utilizzare l'utente root per le attività quotidiane.

Protezione dell'Utente root dell'account AWS

1. Accedi alla AWS Management Console come proprietario dell'account scegliendo Utente root e immettendo l'indirizzo email del Account AWS. Nella pagina successiva, inserisci la password.

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

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

Creazione di un utente amministratore

1. Abilita IAM Identity Center

   Per istruzioni, consulta Abilitazione di AWS IAM Identity Center nella Guida per l'utente di AWS IAM Identity Center.
2. In Centro identità AWS IAM, assegna l'accesso amministrativo a un utente amministrativo.

Per un tutorial sull'utilizzo di IAM Identity Center directory come origine di identità, consulta Configure user access with the default IAM Identity Center directory nella Guida per l'utente di AWS IAM Identity Center.

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 Accedere al portale di accesso AWS nella Guida per l'utente Accedi ad AWS.

Policy gestita di AWS IoT Device Tester

Per consentire l'esecuzione del tester del dispositivo e la raccolta delle metriche, la policy gestita contiene le seguenti autorizzazioni: `AWSIoTDeviceTesterForFreeRTOSFullAccess`

- `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 versione più recente AWS IoT Device Tester disponibile per il download.
- `iot-device-tester:CheckVersion`

  Concede l'autorizzazione per verificare la compatibilità di una combinazione di versioni di prodotto, suite di test e AWS IoT Device Tester.
- `iot-device-tester:DownloadTestSuite`

  Concede l'autorizzazione ad AWS IoT Device Tester per scaricare le suite di test.
- `iot-device-tester:SendMetrics`

  Concede l'autorizzazione a pubblicare i dati dei parametri di utilizzo di AWS IoT Device Tester.

Guida introduttiva all'interfaccia utente IDT-FreeRTOS

Questa sezione mostra come utilizzare l'interfaccia utente IDT-FreeRTOS per creare o modificare la configurazione, quindi mostra come eseguire i test.

Argomenti

Configurazione delle credenziali AWS

È necessario configurare le credenziali per l'AWSutente in cui è stato creato. Creazione e configurazione di un account AWS Puoi specificare le credenziali in uno dei due modi seguenti:

- In un file di credenziali
- Come variabili di ambiente

Configura AWS le credenziali con un 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 a seconda del sistema operativo in uso:

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

Aggiungi AWS le tue credenziali al `credentials` file nel seguente formato:

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

> **Note**
>
> Se non usi il `default` AWS profilo, assicurati di specificare il nome del profilo nell'interfaccia utente IDT-FreeRTOS. Per ulteriori informazioni sui profili, vedete Profili denominati.

Configura AWS le credenziali con variabili di ambiente

Le variabili di ambiente sono variabili gestite dal sistema operativo e utilizzate dai comandi di sistema. Non vengono salvate se si chiude la sessione SSH. L'interfaccia utente IDT-FreeRTOS utilizza 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>
```

Apri l'interfaccia utente IDT-FreeRTOS

Per aprire l'interfaccia utente di IDT-FreeRTOS

1. Scarica una versione IDT-FreeRTOS supportata ed estrai l'archivio scaricato in una posizione del tuo file system in cui disponi delle autorizzazioni di lettura e scrittura.
2. Esegui il seguente comando per accedere alla directory di installazione di IDT-FreeRTOS:

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

3. Esegui il seguente comando per aprire l'interfaccia utente IDT-FreeRTOS:

   Linux

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

Windows

```
./devicetestergui_win_x64-64
```

macOS

```
./devicetestergui_mac_x86-64
```

> **Note**
>
> Su Mac, per consentire al sistema di eseguire l'interfaccia utente, vai su Preferenze di Sistema -> Sicurezza e privacy. Quando esegui i test, potresti dover eseguire questa operazione altre tre volte.

L'interfaccia utente IDT-FreeRTOS si apre nel browser predefinito. Per informazioni sui browser supportati, vedere. Utilizza un browser Web supportato

Creare una nuova configurazione

Se sei un utente alle prime armi, devi creare una nuova configurazione per configurare i file di configurazione JSON richiesti da IDT-FreeRTOS per eseguire i test. È quindi possibile eseguire test o modificare la configurazione creata.

Per esempi dei `userdata.json` file, `econfig.json`,`device.json`,,,,,, vediPreparazione dei test per la prima verifica della scheda del microcontrollore. Per un esempio del `resource.json` file utilizzato solo per eseguire i test Bluetooth Low Energy (BLE), vedereEsecuzione dei test Bluetooth Low Energy.

Per creare una nuova configurazione

1. Nell'interfaccia utente IDT-FreeRTOS, apri il menu di navigazione, quindi scegli Crea nuova configurazione.

> **Important**
>
> È necessario configurare le AWS credenziali prima di aprire l'interfaccia utente. Se non hai configurato le tue credenziali, chiudi la finestra del browser dell'interfaccia utente

IDT-FreeRTOS, segui i passaggi indicati e riapri l'interfaccia utente di IDT-FreeRTOS. Configurazione delle credenziali AWS

2. Segui la procedura guidata di configurazione per accedere alle impostazioni di configurazione IDT utilizzate per eseguire i test di qualificazione. La procedura guidata configura le seguenti impostazioni nei file di configurazione JSON che si trovano nella directory. *devicetester-extract-location*/config

   - AWSsettings: le Account AWS informazioni che IDT-FreeRTOS utilizza per creare AWS risorse durante le esecuzioni dei test. Queste impostazioni sono configurate nel file. `config.json`
   - Repository FreerTOS: il percorso assoluto verso il repository FreerTOS e il codice portato e il tipo di qualifica che desideri eseguire. Queste impostazioni sono configurate nel file. `userdata.json`

     Devi portare FreerTOS sul tuo dispositivo prima di poter eseguire i test di qualificazione. Per ulteriori informazioni, consulta la FreerTOS Porting Guide
   - Build and flash: i comandi build e flash per l'hardware che consentono a IDT di creare e eseguire test flash sulla scheda in modo automatico. Queste impostazioni sono configurate nel file. `userdata.json`
   - Dispositivi: le impostazioni del pool di dispositivi per i dispositivi da testare. Queste impostazioni sono configurate nei `sku` campi `id` e nel `devices` blocco per il pool di dispositivi nel `device.json` file.
   - Rete: le impostazioni per testare il supporto delle comunicazioni di rete per i dispositivi. Queste impostazioni sono configurate nel `features` blocco del `device.json` file e nei `testWifiConfig` blocchi `clientWifiConfig` and del `userdata.json` file.
   - Echo server: le impostazioni di configurazione del server echo per i test Secure Socket. Queste impostazioni sono configurate nel file. `userdata.json`

     Per ulteriori informazioni sulla configurazione del server echo, vederehttps://docs.aws.amazon.com/freertos/latest/portingguide/afr-echo-server.html.
   - CMake — (Facoltativo) Le impostazioni per eseguire i test di funzionalità di compilazione di CMake. Questa configurazione è richiesta solo se utilizzi CMake come sistema di compilazione. Queste impostazioni sono configurate nel `userdata.json` file.

- BLE: le impostazioni per eseguire i test di funzionalità Bluetooth Low Energy. Queste impostazioni sono configurate nel `features` blocco del `device.json` file e nel `resource.json` file.
- OTA: le impostazioni per eseguire i test di funzionalità OTA. Queste impostazioni sono configurate nel `features` blocco del `device.json` file e nel `userdata.json` file.

3. Nella pagina Revisione, verifica le informazioni di configurazione.

Dopo aver esaminato la configurazione, per eseguire i test di qualificazione, scegli Esegui test.

Modifica una configurazione esistente

Se hai già impostato i file di configurazione per IDT, puoi utilizzare l'interfaccia utente IDT-FreeRTOS per modificare la configurazione esistente. Assicuratevi che i file di configurazione esistenti siano disponibili nella directory. *`devicetester-extract-location`*`/config`

Per modificare una nuova configurazione

1. Nell'interfaccia utente IDT-FreeRTOS, apri il menu di navigazione, quindi scegli Modifica configurazione esistente.

   La dashboard di configurazione mostra informazioni sulle impostazioni di configurazione esistenti. Se una configurazione è errata o non disponibile, lo stato di tale configurazione è`Error validating configuration`.
2. Per modificare un'impostazione di configurazione esistente, completare i seguenti passaggi:

   a. Scegli il nome di un'impostazione di configurazione per aprire la relativa pagina delle impostazioni.

   b. Modifica le impostazioni, quindi scegli Salva per rigenerare il file di configurazione corrispondente.

Dopo aver modificato la configurazione, verifica che tutte le impostazioni di configurazione superino la convalida. Se lo stato di ogni impostazione di configurazione è`Valid`, puoi eseguire i test di qualificazione utilizzando questa configurazione.

Esegui test di qualificazione

Dopo aver creato una configurazione per IDT-FreeRTOS, puoi eseguire i test di qualificazione.

Per eseguire test di qualificazione

1. Convalida la configurazione.
2. Nel menu di navigazione, scegli Esegui test.
3. Per iniziare l'esecuzione del test, scegli Avvia test.

IDT-FreeRTOS esegue i test di qualificazione e visualizza il riepilogo dei test eseguiti e gli eventuali errori nella console Test runner. Una volta completata l'esecuzione del test, è possibile visualizzare i risultati e i registri del test dalle seguenti posizioni:

- I risultati dei test si trovano nella *devicetester-extract-location*/results/*execution-id* directory.
- I registri dei test si trovano nella *devicetester-extract-location*/results/*execution-id*/logs directory.

Per ulteriori informazioni sui risultati e sui registri dei test, vedere. Informazioni su risultati e log

Esecuzione dei test Bluetooth Low Energy

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 è stato estratto AWS IoT Device Tester, creare 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

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 suY, IDT interromperà l'esecuzione del test se rileva che è disponibile una versione più recente. Se questo parametro è impostato suN, 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 suY, IDT interromperà l'esecuzione del test se rileva che la politica gestita non lo è. up-to-date Se questo parametro è impostato suN, 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

Esempio

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

Esempio

```
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

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.

Informazioni su risultati e log

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`Questo è il rapporto sul test di qualificazione che invii per inserire il tuo dispositivo nel Partner Device AWS Catalog. AWS 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` è un report in formato JUnit XML format standard. È possibile integrarlo nelle piattaforme CI/CD quali 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 AWSPartner 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 usati 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

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 che forniscono a IDT informazioni su come eseguire la suite di test.
- File eseguibili di test utilizzati da IDT per eseguire casi di test.
- File aggiuntivi necessari per eseguire i test.

Completa i seguenti passaggi di base per creare test IDT personalizzati:

1. Crea file di configurazione per la tua suite di test.
2. Crea eseguibili per 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

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
  - Ultima versione 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 per vedere vari esempi di eseguibili di test case e file di configurazione dei test.

> **Note**
>
> La suite di test di esempio include il codice sorgente Python. Non includete informazioni sensibili nel codice della suite di 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

Una suite di test combina quanto segue:

- Eseguibili di test che contengono la logica di test
- File di configurazione 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
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
  - Ultima versione 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

La tua suite di test deve contenere i seguenti file di configurazione richiesti:

File richiesti

`suite.json`

Contiene informazioni sulla suite di test. Per informazioni, consulta 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. Per informazioni, consulta 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. Per informazioni, consulta 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

Questa sezione descrive i formati in cui create i file di configurazione da includere quando scrivete una suite di test personalizzata.

File di configurazione richiesti

`suite.json`

Contiene informazioni sulla suite di test. Per informazioni, consulta 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. Per informazioni, consulta 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. Per informazioni, consulta Configura test.json.

File di configurazione opzionali

`test_orchestrator.yaml` o **`state_machine.json`**

Definisce come vengono eseguiti i test quando IDT esegue la suite di test. Configura test_orchestrator.yamlSse.

> **Note**
>
> A partire da IDT v4.5.2, si utilizza il `test_orchestrator.yaml` file per definire il flusso di lavoro di test. Nelle versioni precedenti di IDT, si utilizza il file. `state_machine.json` Per informazioni sulla macchina a stati, vedereConfigurare la macchina a stati IDT.

`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. Per informazioni, consulta Configura userdata_schema.json.

I file di configurazione vengono inseriti nel file *<custom-test-suite-folder>* come illustrato qui.

```
<custom-test-suite-folder>
### suite
    ### suite.json
    ### test_orchestrator.yaml
    ### 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 e non deve contenere caratteri di sottolineatura ()_. 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 un file eseguibile per il 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 e non deve contenere caratteri di sottolineatura ()_. 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 un file eseguibile per il 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 test_orchestrator.yaml

Un orchestrator di test 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 un orchestratore di test definito dall'utente, IDT genererà un orchestratore di test per voi.

L'orchestrator di test predefinito 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 dell'IDT test orchestrator, consulta. Configura l'orchestrator di test 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 del test orchestrator.

Configura l'orchestrator di test IDT

A partire da IDT v4.5.2, IDT include un nuovo componente di test orchestrator. Il test orchestrator è un componente IDT che controlla il flusso di esecuzione della suite di test e genera il rapporto di test dopo che IDT ha terminato l'esecuzione di tutti i test. Il test orchestrator determina la selezione dei test e l'ordine in cui i test vengono eseguiti in base a regole definite dall'utente.

Se la tua suite di test non include un orchestratore di test definito dall'utente, IDT genererà un orchestrator di test per te.

L'orchestrator di test predefinito 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.

Il test orchestrator sostituisce la macchina a stati IDT. Ti consigliamo vivamente di utilizzare il test orchestrator per sviluppare le tue suite di test anziché la macchina a stati IDT. Il test orchestrator offre le seguenti funzionalità migliorate:

- Utilizza un formato dichiarativo rispetto al formato imperativo utilizzato dalla macchina a stati IDT. Ciò consente di specificare quali test eseguire e quando eseguirli.
- Gestisce la gestione di gruppi specifici, la generazione di report, la gestione degli errori e il tracciamento dei risultati in modo da non dover gestire manualmente queste azioni.
- Utilizza il formato YAML, che supporta i commenti per impostazione predefinita.
- Richiede l'80% di spazio su disco in meno rispetto al test orchestrator per definire lo stesso flusso di lavoro.
- Aggiunge la convalida preliminare al test per verificare che la definizione del flusso di lavoro non contenga ID di test errati o dipendenze circolari.

Formato dell'orchestratore di test

È possibile utilizzare il seguente modello per configurare il proprio file: *custom-test-suite-folder*`/suite/test_orchestrator.yaml`

```
Aliases:
  string: context-expression

ConditionalTests:
  - Condition: context-expression
    Tests:
      - test-descriptor

Order:
  - - group-descriptor
    - group-descriptor

Features:
  - Name: feature-name
    Value: support-description
    Condition: context-expression
    Tests:
        - test-descriptor
    OneOfTests:
        - test-descriptor
```

```
    IsRequired: boolean
```

Tutti i campi che includono valori sono obbligatori, come descritto di seguito:

`Aliases`

Facoltativo. Stringhe definite dall'utente che mappano alle espressioni di contesto. Gli alias consentono di generare nomi descrittivi per identificare le espressioni di contesto nella configurazione del test orchestrator. Ciò è particolarmente utile se state creando espressioni di contesto complesse o espressioni da utilizzare in più posizioni.

È possibile utilizzare le espressioni di contesto per archiviare query di contesto che consentono di accedere ai dati da altre configurazioni IDT. Per ulteriori informazioni, consulta Accedere ai dati nel contesto.

Example

Esempio

```
Aliases:
    FizzChosen: "'{{$pool.features[?(@.name == 'Fizz')].value[0]}}' == 'yes'"
    BuzzChosen: "'{{$pool.features[?(@.name == 'Buzz')].value[0]}}' == 'yes'"
    FizzBuzzChosen: "'{{$aliases.FizzChosen}}' && '{{$aliases.BuzzChosen}}'"
```

`ConditionalTests`

Facoltativo. Un elenco di condizioni e i casi di test corrispondenti che vengono eseguiti quando ogni condizione è soddisfatta. Ogni condizione può avere più casi di test; tuttavia, è possibile assegnare un determinato test case a una sola condizione.

Per impostazione predefinita, IDT esegue qualsiasi test case che non è assegnato a una condizione in questo elenco. Se non specificate questa sezione, IDT esegue tutti i gruppi di test nella suite di test.

Ogni elemento dell'`ConditionalTests`elenco include i seguenti parametri:

`Condition`

Un'espressione di contesto che restituisce un valore booleano. Se il valore valutato è vero, IDT esegue i casi di test specificati nel parametro. `Tests`

`Tests`

L'elenco dei descrittori di test.

Ogni descrittore di test utilizza l'ID del gruppo di test e uno o più ID dei test case per identificare i singoli test da eseguire da uno specifico gruppo di test. Il descrittore di test utilizza il seguente formato:

```
GroupId: group-id
CaseIds: [test-id, test-id] # optional
```

Example

Esempio

L'esempio seguente utilizza espressioni di contesto generiche che è possibile definire come`Aliases`.

```
ConditionalTests:
    - Condition: "{{$aliases.Condition1}}"
      Tests:
          - GroupId: A
          - GroupId: B
    - Condition: "{{$aliases.Condition2}}"
      Tests:
          - GroupId: D
    - Condition: "{{$aliases.Condition1}} || {{$aliases.Condition2}}"
      Tests:
          - GroupId: C
```

In base alle condizioni definite, IDT seleziona i gruppi di test come segue:

- Se `Condition1` è vero, IDT esegue i test nei gruppi di test A, B e C.
- Se `Condition2` è vero, IDT esegue i test nei gruppi di test C e D.

`Order`

Facoltativo. L'ordine in cui eseguire i test. L'ordine dei test viene specificato a livello di gruppo di test. Se non specificate questa sezione, IDT esegue tutti i gruppi di test applicabili in ordine casuale. Il valore di `Order` è un elenco di elenchi di descrittori di gruppo. Qualsiasi gruppo di test

in `Order` cui non fai parte dell'elenco può essere eseguito in parallelo con qualsiasi altro gruppo di test elencato.

Ogni elenco di descrittori di gruppo contiene uno o più descrittori di gruppo e identifica l'ordine in cui eseguire i gruppi specificati in ciascun descrittore. È possibile utilizzare i seguenti formati per definire i singoli descrittori di gruppo:

- *`group-id`*— L'ID del gruppo di un gruppo di test esistente.
- `[`*`group-id`*`, `*`group-id`*`]`—Elenco di gruppi di test che possono essere eseguiti in qualsiasi ordine l'uno rispetto all'altro.
- `"*"`—Wild card. Equivale all'elenco di tutti i gruppi di test che non sono già specificati nell'elenco corrente dei descrittori di gruppo.

Il valore per `Order` deve inoltre soddisfare i seguenti requisiti:

- Gli ID dei gruppi di test specificati in un descrittore di gruppo devono esistere nella suite di test.
- Ogni elenco di descrittori di gruppo deve includere almeno un gruppo di test.
- Ogni elenco di descrittori di gruppo deve contenere ID di gruppo univoci. Non è possibile ripetere un ID di gruppo di test all'interno di singoli descrittori di gruppo.
- Un elenco di descrittori di gruppo può avere al massimo un descrittore di gruppo con caratteri jolly. Il descrittore di gruppo con caratteri jolly deve essere il primo o l'ultimo elemento dell'elenco.

Example

Esempio

Per una suite di test che contiene i gruppi di test A, B, C, D ed E, il seguente elenco di esempi mostra diversi modi per specificare che IDT deve prima eseguire il gruppo di test A, quindi eseguire il gruppo di test B e quindi eseguire i gruppi di test C, D ed E in qualsiasi ordine.

- ```
  Order:
      - - A
        - B
        - [C, D, E]
  ```
- ```
  Order:
      - - A
        - B
  ```

```
      - "*"
```

- ```
  Order:
      - - A
        - B

      - - B
        - C

      - - B
        - D

      - - B
        - E
  ```

`Features`

Facoltativo. L'elenco delle funzionalità del prodotto che desideri che IDT aggiunga al `awsiotdevicetester_report.xml` file. Se non specifichi questa sezione, IDT non aggiungerà alcuna funzionalità del prodotto al rapporto.

Una funzionalità del prodotto è costituita da informazioni definite dall'utente su criteri specifici che un dispositivo potrebbe soddisfare. Ad esempio, la funzionalità del prodotto MQTT può indicare che il dispositivo pubblica correttamente i messaggi MQTT. In`awsiotdevicetester_report.xml`, le funzionalità del prodotto sono impostate come o come un valore personalizzato definito dall'utente `supported`o `not-supported`, in base al superamento dei test specificati.

Ogni elemento dell'`Features`elenco è composto dai seguenti parametri:

`Name`

Il nome della funzionalità.

`Value`

Facoltativo. Il valore personalizzato che desideri utilizzare nel rapporto anzichè`supported`. Se questo valore non è specificato, Based IDT imposta il valore della funzionalità su `supported` o in `not-supported` base ai risultati del test. Se testate la stessa funzionalità con condizioni diverse, potete utilizzare un valore personalizzato per ogni istanza di quella funzionalità nell'`Features`elenco e IDT concatena i valori delle caratteristiche per le condizioni supportate. Per ulteriori informazioni, consulta la pagina
  ```

`Condition`

Un'espressione di contesto che restituisce un valore booleano. Se il valore valutato è vero, IDT aggiunge la funzionalità al rapporto di test dopo aver terminato l'esecuzione della suite di test. Se il valore valutato è falso, il test non è incluso nel rapporto.

`Tests`

Facoltativo. L'elenco dei descrittori dei test. Tutti i test specificati in questo elenco devono essere superati affinché la funzionalità sia supportata.

Ogni descrittore di test in questo elenco utilizza l'ID del gruppo di test e uno o più ID dei test case per identificare i singoli test da eseguire da uno specifico gruppo di test. Il descrittore di test utilizza il seguente formato:

```
GroupId: group-id
CaseIds: [test-id, test-id] # optional
```

È necessario specificare una `Tests` o `OneOfTests` per ciascuna funzionalità nell'`Features`elenco.

`OneOfTests`

Facoltativo. L'elenco dei descrittori dei test. Almeno uno dei test specificati in questo elenco deve essere superato affinché la funzionalità sia supportata.

Ogni descrittore di test in questo elenco utilizza l'ID del gruppo di test e uno o più ID dei test case per identificare i singoli test da eseguire da un gruppo di test specifico. Il descrittore di test utilizza il seguente formato:

```
GroupId: group-id
CaseIds: [test-id, test-id] # optional
```

È necessario specificare una `Tests` o `OneOfTests` per ciascuna funzionalità nell'`Features`elenco.

`IsRequired`

Il valore booleano che definisce se la funzionalità è richiesta nel rapporto di test. Il valore predefinito è `false`.

Contesto dell'orchestratore di test

Il contesto di test orchestrator è un documento JSON di sola lettura che contiene dati disponibili per l'orchestrator di test durante l'esecuzione. Il contesto del test orchestrator è accessibile solo dal test orchestrator 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 del test orchestrator utilizza il seguente formato:

```
{
    "pool": {
        <device-json-pool-element>
    },
    "userData": {
        <userdata-json-content>
    },
    "config": {
        <config-json-content>
    }
}
```

`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`

Informazioni contenute nel `config.json` fascicolo.

È possibile interrogare il contesto utilizzando la notazione JsonPath. La sintassi per le query JSONPath nelle definizioni di stato è. `{{query}}` Quando accedi ai dati dal contesto di test orchestrator, assicurati che ogni valore restituisca una stringa, un numero o un valore booleano.

Per ulteriori informazioni sull'utilizzo della notazione JsonPath per accedere ai dati dal contesto, consulta. Usa il contesto IDT

Configurare la macchina a stati IDT

> **Important**
>
> A partire da IDT v4.5.2, questa macchina a stati è obsoleta. Ti consigliamo vivamente di utilizzare il nuovo orchestrator di test. Per ulteriori informazioni, consulta Configura l'orchestrator di test IDT.

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. *`l'oggetto state-name contiene la definizione di uno stato valido mappato al nome dello stato.`*

L'`States`oggetto deve includere gli stati and. `Succeed` `Fail` 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 matrice di ID dei test case del gruppo specificato in. `TestGroup` In base ai valori di `TestGroup` e`TestCases`, IDT determina il comportamento di esecuzione del test nel modo seguente:

- Quando `TestCases` vengono specificati entrambi gli `TestGroup` E, 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 gli ID dei singoli test case, 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 degli ID dei test case 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 `e`true, `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 ID dei gruppi di test. Tutti i test all'interno di ogni gruppo di test specificato devono essere superati affinché la funzionalità sia supportata.

`OneOfGroups`

Facoltativo. Una serie di ID dei gruppi di test. 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 ID dei test case. 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 singoli ID di test case, imposta `ResultVar` su*`group-id_test-id`*`_passed`.
- Se non hai specificato gli ID dei singoli test case, imposta `ResultVar` su*`group-id`*`_passed`.

Lo `AddProductFeatures` stato verifica i risultati dei test nel modo seguente:

- Se non è stato specificato alcun ID del test case, il risultato per ogni gruppo di test viene determinato dal valore della *`group-id`*`_passed` variabile nel contesto della macchina a stati.
- Se avete specificato gli ID dei test case, 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 un 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 ID di gruppo di test validi 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. Si tratta di uno stato finale per la 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 di ID di gruppi di test specifici.

`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 ID specifici dei test case.

`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 la notazione JsonPath. La sintassi per le query JSONPath nelle definizioni di stato è. `{{$.query}}` È possibile utilizzare le query JsonPath come

stringhe segnaposto in alcuni stati. IDT sostituisce le stringhe segnaposto con il valore della query JsonPath valutata dal contesto. 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 notazione JsonPath per accedere ai dati dal contesto, consulta. 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 un file eseguibile per il test case IDT

È possibile creare e inserire il file eseguibile del 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

Gli IDT Client SDK consentono di semplificare il modo in cui si scrive la logica di test nell'eseguibile di test con comandi API che è possibile utilizzare per interagire con IDT e i dispositivi sottoposti a test. IDT attualmente fornisce i seguenti SDK:

- SDK del client IDT per Python
- SDK del client IDT per Go
- SDK del client IDT per Java

Questi SDK si 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 il file 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

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 invece di analizzarli e quindi utilizzare l'API per inviarli a IDT.

Se utilizzi questo metodo, assicurati 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

Quando IDT esegue una suite di test, la suite di test può accedere a un set 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,
        "idtRootPath": <path/to/IDT/root>
    },
    "device": {
        <device-json-device-element>
    },
    "devicePool": {
        <device-json-pool-element>
    },
    "resource": {
        "devices": [
            {
                <resource-json-device-element>
                "name": "<resource-name>"
            }
        ]
    },
    "testData": {
        "awsCredentials": {
            "awsAccessKeyId": "<access-key-id>",
            "awsSecretAccessKey": "<secret-access-key>",
            "awsSessionToken": "<session-token>"
        },
        "logFilePath": "/path/to/log/file"
    },
    "userData": {
        <userdata-json-content>
    }
}
```

`config`

Informazioni dal `config.json`file. Il `config` campo contiene anche i seguenti campi aggiuntivi:

`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.

`config.idRootPath`

Questo valore è un segnaposto per il valore assoluto del percorso di IDT durante la configurazione del file. `userdata.json` Viene utilizzato dai comandi build e flash.

`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 notazione JsonPath dai tuoi file di configurazione e dal tuo file di testo eseguibile con le `GetContextValue` API and. `GetContextString` La sintassi per le stringhe JsonPath per accedere al contesto IDT varia come segue:

- In `suite.json` e, si usa. `test.json` {{*query*}} Cioè, non usate l'elemento root `$.` per iniziare l'espressione.
- `Instatemachine.json`, si usa{{`$.`*query*}}.
- Nei comandi API, si utilizza *query* o{{`$.`*query*}}, a seconda del comando. Per ulteriori informazioni, consulta la documentazione in linea negli SDK.

La tabella seguente descrive gli operatori in una tipica espressione JsonPath di foobar:

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, recuperando 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 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 JsonPath o il valore dell'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 JSONPath o il valore sul lato sinistro dell'espressione deve restituire una stringa e il lato destro deve essere un valore di modello che segue la sintassi RE2.

È possibile utilizzare le query JsonPath nel formato {{*query*}} come stringhe segnaposto all'interno dei campi and nei file `args` e `environmentVariables` all'interno dei campi nei file. `test.json` `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 interamente 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 segnaposti JsonPath per recuperare singole stringhe da impostare come variabili di ambiente.

Configura le impostazioni per i test runner

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 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>",
                "pairedResource": "<device-id>", //used for no-op protocol
                "connectivity": {
                    "protocol": "ssh | uart | docker | no-op",
                    // ssh
                    "ip": "<ip-address>",
                    "port": <port-number>,
                    "publicKeyPath": "<public-key-path>",
                    "auth": {
                        "method": "pki | password",
                        "credentials": {
                            "user": "<user-name>",
                            // pki
                            "privKeyPath": "/path/to/private/key",

                            // password
                            "password": "<password>",
                        }
                    },

                    // uart
                    "serialPort": "<serial-port>",

                    // docker
                    "containerId": "<container-id>",
                    "containerUser": "<container-user-name>",
                }
            }
        ]
    }
]
```

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 nella procedura di pubblicazione.

`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.

`devices.pairedResource`

Un identificatore univoco definito dall'utente per un dispositivo di risorse. Questo valore è necessario quando si testano i dispositivi utilizzando il protocollo di `no-op` connettività.

`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, `docker` per i contenitori Docker e `no-op` per i dispositivi che non dispongono di una connessione diretta con la macchina host IDT ma richiedono un dispositivo di risorse come middleware fisico per comunicare con la macchina host.

Per i dispositivi non operativi, si configura l'ID del dispositivo di risorsa in. `devices.pairedResource` È inoltre necessario specificare questo ID nel `resource.json` file. Il dispositivo associato deve essere un dispositivo fisicamente associato al dispositivo in prova. Dopo che IDT ha identificato e si è connesso al dispositivo di risorse associato, IDT non si connetterà ad altri dispositivi di risorse in base alle funzionalità descritte nel file. `test.json`

`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.publicKeyPath`

Facoltativo. Il percorso completo della chiave pubblica utilizzata per autenticare le connessioni al dispositivo in esame. Quando si specifica il`publicKeyPath`, IDT convalida la chiave

pubblica del dispositivo quando stabilisce una connessione SSH al dispositivo in prova. Se questo valore non è specificato, IDT crea una connessione SSH, ma non convalida la chiave pubblica del dispositivo.

Ti consigliamo vivamente di specificare il percorso della chiave pubblica e di utilizzare un metodo sicuro per recuperare questa chiave pubblica. Per i client SSH standard basati sulla riga di comando, la chiave pubblica viene fornita nel file. `known_hosts` Se si specifica un file con chiave pubblica separato, questo file deve utilizzare lo stesso formato del `known_hosts` file, ovvero. `ip-address key-type public-key`

`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>,
                    "publicKeyPath": "<public-key-path>",
                    "auth": {
                        "method": "pki | password",
                        "credentials": {
                            "user": "<user-name>",
                            // pki
                            "privKeyPath": "/path/to/private/key",

                            // password
```

```
                        "password": "<password>",
                    }
                },

                // uart
                "serialPort": "<serial-port>",

                // docker
                "containerId": "<container-id>",
                "containerUser": "<container-user-name>",
            }
        }
    ]
}
]
```

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.publicKeyPath`

Facoltativo. Il percorso completo della chiave pubblica utilizzata per autenticare le connessioni al dispositivo in esame. Quando si specifica il`publicKeyPath`, IDT convalida la chiave pubblica del dispositivo quando stabilisce una connessione SSH al dispositivo in prova. Se questo valore non è specificato, IDT crea una connessione SSH, ma non convalida la chiave pubblica del dispositivo.

Ti consigliamo vivamente di specificare il percorso della chiave pubblica e di utilizzare un metodo sicuro per recuperare questa chiave pubblica. Per i client SSH standard basati sulla riga di comando, la chiave pubblica viene fornita nel file. `known_hosts` Se si specifica un file con chiave pubblica separato, questo file deve utilizzare lo stesso formato del `known_hosts` file, ovvero. `ip-address key-type public-key`

`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 logs nel file *< 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

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 di 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 il test orchestrator; 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 fino a quando non viene interrotto manualmente. In modalità debug, IDT inoltre non esegue il test orchestrator 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 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 di debug.

Esamina i risultati e i registri dei test IDT

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 Testerutilizza 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.

```
[INFO] [2000-01-02 03:04:05]: 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.

```
[INFO] [2000-01-02 03:04:05]: Hello world! suiteId=MyTestSuitegroupId=myTestGroup
 testCaseId=myTestCase deviceId=my-
deviceexecutionId=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.

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 Testerschema 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 **`<awsproduct>`** tag

`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 affinché la suite di test possa 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 report *`suite-name`*`_Result.xml` è in formato XML JUnit. 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 **`<testsuites>`** tag

`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 **`<testcase>`** tag

`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

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 cartella *< device-tester-extract-location >*
- 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 `config.json` file per accedervi.

Registrarsi per creare un Account AWS

Se non disponi di un Account AWS, completa la procedura seguente per crearne uno.

Per registrarsi a un Account AWS

1. Apri la pagina all'indirizzo https://portal.aws.amazon.com/billing/signup.
2. Segui le istruzioni online.

   Nel corso della procedura di registrazione riceverai una telefonata, durante la quale sarà necessario inserire un codice di verifica attraverso la tastiera del telefono.

   Durante la registrazione di un Account AWS, viene creato un Utente root dell'account AWS. L'utente root dispone dell'accesso a tutte le risorse e tutti i Servizi AWSnell'account. Come best practice di sicurezza, assegna l'accesso amministrativo a un utente amministrativo e utilizza solo l'utente root per eseguire attività che richiedono l'accesso di un utente root.

Al termine del processo di registrazione, riceverai un'e-mail di conferma da AWS. È possibile visualizzare l'attività corrente dell'account e gestire l'account in qualsiasi momento accedendo all'indirizzo https://aws.amazon.com/ e selezionando Il mio account.

Creazione di un utente amministratore

Dopo aver effettuato la registrazione di un Account AWS, proteggi Utente root dell'account AWS, abilita AWS IAM Identity Center e crea un utente amministratore in modo da non utilizzare l'utente root per le attività quotidiane.

Protezione dell'Utente root dell'account AWS

1. Accedi alla AWS Management Console come proprietario dell'account scegliendo Utente root e immettendo l'indirizzo email del Account AWS. Nella pagina successiva, inserisci la password.

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

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

Creazione di un utente amministratore

1. Abilita IAM Identity Center

   Per istruzioni, consulta Abilitazione di AWS IAM Identity Center nella Guida per l'utente di AWS IAM Identity Center.
2. In Centro identità AWS IAM, assegna l'accesso amministrativo a un utente amministrativo.

   Per un tutorial sull'utilizzo di IAM Identity Center directory come origine di identità, consulta Configure user access with the default IAM Identity Center directory nella Guida per l'utente di AWS IAM Identity Center.

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 Accedere al portale di accesso AWS nella Guida per l'utente Accedi ad AWS.

Per fornire l'accesso, aggiungi autorizzazioni ai tuoi 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 Creating a role for a third-party identity provider (federation) (Creazione di un ruolo per un provider di identità di terze parti [federazione]) nella Guida per l'utente di IAM.

- Utenti IAM:
  - Crea un ruolo che l'utente possa assumere. Per istruzioni, consulta la pagina Creating a role for an IAM user (Creazione di un ruolo per un utente IAM) nella Guida per l'utente di 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 di IAM.

Fornisci le AWS credenziali a IDT

Per consentire a IDT di accedere alle tue AWS credenziali e inviare le metriche aAWS, 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 seguente comando:

   ```
   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.

AWS IoT Device Testerper le versioni della suite di test FreeRTOS

IDT for FreeRTOS organizza le risorse di 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 il set di singoli test relativi a una particolare funzionalità, come la messaggistica BLE e MQTT.

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 per AWS IoT Device Tester for FreeRTOS.

Puoi scaricare le suite di test utilizzando il comando `upgrade-test-suite`. In alternativa, puoi utilizzare il parametro opzionale `-upgrade-test-suite` *`flag`* quando avvii 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.

Risoluzione dei problemi

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 non è riuscito, 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 dettagliato per l'output della console FreeRTOS.

Per la risoluzione dei problemi consigliamo il seguente flusso di lavoro:

1. Se viene visualizzato l'errore indicante che l'*`utente/ruolo`* non è autorizzato ad accedere alla risorsa, assicurarsi di configurare le autorizzazioni come specificato in Creazione e configurazione di un account AWS.
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.

- Per FRQ 1.x.x assicuratevi che sia disponibile 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 seguente tabella 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`	Assicurati che i campi obbligatori non siano mancanti e che

Codice di errore	Nome del codice di errore	Possibile causa principale	Risoluzione dei problemi
		sono mancanti o in un formato errato.	siano nel formato obbligatorio nei file elencati. Per ulteriori informazioni, consulta Preparazione dei test per la prima verifica della scheda del microcontrollore.

Codice di errore	Nome del codice di errore	Possibile causa principale	Risoluzione dei problemi
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: • Non validoAWS Regione: specifica una regione validaAWS regione nella tua`config.json` on fascicolo. Per ulteriori informazioni suAWSregioni, vediRegioni ed endpoint. • Non validoAWS credenziali - Imposta validaAWS credenziali sulla macchina di prova (tramite le variabili di ambiente o il file delle credenziali). Verifica che il campo di autenticazione sia configurato correttamente. Per ulteriori informazioni, consulta Creazione e configurazione di un account AWS.

Codice di errore	Nome del codice di errore	Possibile causa principale	Risoluzione dei problemi
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. • Eliminare il `build` cartella nella directory del codice sorgente di FreeRTOS, se esiste. Per ulteriori informazioni, consulta Configurazione delle impostazioni di compilazione, flashing e test. • Windows ha un limite di caratteri per i nomi dei percorsi dei file. Un nome di percorso di file lungo causerà un errore.

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 for FreeRTOS in un percorso di file sul tuo sistema che contiene spazi, ad esempio`C:\Users\My Name\Desktop`

Codice di errore	Nome del codice di errore	Possibile causa principale	Risoluzione dei problemi
			\, potresti aver bisogno di virgolett e aggiuntive all'inter no dei comandi di compilazione per assicurarti che i percorsi vengano analizzati correttam ente. 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 Configura zione delle impostazi oni di compilazione, flashing e test.
206	StartEchoServerError	IDT FreeRTOS non è in grado di avviare il server echo perWiFio test di prese sicure.	Verificare che le porte configura te in `echoServe rConfiguration` nel file `userdata. json` non siano in uso o bloccate dalle impostazioni del firewall o della rete.

Errori di analisi dei file di configurazione di debug

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 continui ad avere difficoltà a individuare l'errore, puoi eseguire la convalida manualmente nel tuo IDE, utilizzando un editor di testo come Atom o Sublime, oppure con uno strumento online come JSONLint.

Errori di analisi dei risultati dei test di debug

Durante l'esecuzione di un gruppo di test daTest di integrazione delle librerie TOS di FreeRTOS, ad esempioFullTransportInterfaceTLS, PKCS11_core completo, PKCS11_onboard_ECC completo, PKCS11_onboard_RSA completo, Full PKCS11_PreProvisioned_ECC, PKCS completo 11_PreProvisioned_RSA, oppureOtaCore, IDT for FreeRTOS analizza i risultati del test dal dispositivo di test con la connessione seriale. A volte, uscite seriali aggiuntive sul dispositivo possono interferire con l'analisi dei risultati del test.

Nel caso sopra menzionato, vengono emessi strani motivi di errore del test, come stringhe provenienti da uscite di dispositivi non correlate. Il file di registro del case di test IDT for FreeRTOS (che include tutti gli output seriali che IDT for FreeRTOS ha ricevuto durante il test) può mostrare quanto segue:

```
<unrelated device output>
TEST(Full_PKCS11_Capabilities, PKCS11_Capabilities)<unrelated device output>
<unrelated device output>
 PASS
```

Nell'esempio precedente, l'uscita del dispositivo non correlata impedisce a IDT for FreeRTOS di rilevare il risultato del test, che èPASSARE.

Controlla quanto segue per garantire test ottimali.

- Assicurati che le macro di registrazione utilizzate sul dispositivo siano thread-safe. VediImplementazione delle macro di registrazione della libreriaper ulteriori informazioni.
- Assicurati che ci siano un numero minimo di uscite per la connessione seriale durante i test. Le uscite di altri dispositivi possono essere un problema anche se le macro di registrazione sono

correttamente thread-safe, poiché i risultati del test verranno emessi in chiamate separate durante il test.

Un registro dei casi di test IDT per FreeRTOS mostrerebbe idealmente un output ininterrotto dei risultati del test come di seguito:

```
---------STARTING TESTS---------
TEST(Full_OTA_PAL, otaPal_CloseFile_ValidSignature) PASS
TEST(Full_OTA_PAL, otaPal_CloseFile_InvalidSignatureBlockWritten) PASS
-----------------------
2 Tests 0 Failures 0 Ignored
```

Errori di debug nel controllo dell'integrità

Se si utilizza la versione FRQ 1.x.x di FreeRTOS, si applicano i seguenti controlli di integrità.

Quando esegui il gruppo di test FreeRTOSIntegrity e riscontri degli errori, assicurati innanzitutto di non aver modificato nessuno dei*freertos*file di directory. Se non l'hai fatto e continui a riscontrare problemi, assicurati di utilizzare il ramo corretto. Se esegui IDT`list-supported-products`comando, puoi trovare quale ramo etichettato del*freertos*repo che dovresti usare.

Se hai clonato il ramo con tag corretto di`freertos`repo e hai ancora problemi, assicurati di aver eseguito anche il`submodule update`comando. Il flusso di lavoro di clonazione per`freertos`il repo 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 ricercati dal correttore di integrità si trova nel`checksums.json`file nel tuo*freertos*rubrica. Per qualificare una porta FreeRTOS senza alcuna modifica ai file e alla struttura delle cartelle, assicurati che nessuno dei file elencati nel '`exhaustive`'e'`minimal`'sezioni del`checksums.json`il file è stato modificato. Per eseguire con un SDK configurato, verifica che nessuno dei file sotto '`minimal`'la sezione è stata modificata.

Se esegui IDT con un SDK e hai modificato alcuni file nel tuo*freertos*directory, quindi assicurati di configurare correttamente il tuo SDK nel tuo`userdata`fascicolo. In caso contrario, il correttore di integrità verificherà tutti i file nel*freertos*rubrica.

DebuggingFullWiFierrori nei gruppi di test

Se si utilizza FRQ 1.x.x e si verificano errori nelFullWiFigruppo di test e il»`AFQP_WiFiConnectMultipleAP`«il test fallisce, ciò potrebbe essere dovuto al fatto che entrambi i punti di accesso non si trovano nella stessa sottorete del computer host che esegue IDT. Assicurati che entrambi i punti di accesso si trovino nella stessa sottorete del computer host che esegue IDT.

Debug dell'errore "parametro richiesto mancante"

Poiché vengono aggiunte nuove funzionalità a IDT for FreeRTOS, è possibile che vengano 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.

Effettuare il debug di un errore «impossibile avviare il test»

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.

Effettuare il debug di un errore «impossibile trovare i risultati di inizio del test»

Potresti visualizzare degli errori quando IDT tenta di analizzare i risultati emessi dal dispositivo sottoposto a test. Esistono diverse cause possibili, quindi controlla la correttezza delle seguenti aree:

- Assicurati che il dispositivo sottoposto a test abbia una connessione stabile alla macchina host. Puoi controllare il file di registro per un test che mostri questi errori per vedere cosa sta ricevendo IDT.
- Se usi FRQ 1.x.x e il dispositivo sottoposto a test è connesso tramite una rete lenta o un'altra interfaccia, o non vedi il flag «---------STARTING TESTS---------» in un registro del gruppo di test

FreeRTOS insieme agli altri output del gruppo di test FreeRTOS, puoi provare ad aumentare il valore di`testStartDelayms`nella configurazione dei dati utente. Per ulteriori informazioni, consulta Configurazione delle impostazioni di compilazione, flashing e test.

Eseguire il debug di un errore «Test fallito: __ risultati attesi ma ___»

È possibile che vengano visualizzati errori che indicano un errore del test durante il test. Il test prevede un certo numero di risultati e non lo vede durante il test. Alcuni test FreeRTOS vengono eseguiti prima che IDT veda l'output del dispositivo. Se vedi questo errore, puoi provare ad aumentare il valore di`testStartDelayms`nel tuodati utenteconfigurazione. Per ulteriori informazioni, consulta Configurazione delle impostazioni di compilazione, flashing e test.

Il debug di un «_____» è stato deselezionato a causa di`ConditionalTests`vincoli» errore

Ciò significa che stai eseguendo un test su un pool di dispositivi incompatibile con il test. Questo può accadere con i test OTA E2E. Ad esempio, durante l'esecuzione del`OTADataplaneMQTT`gruppo di test e nel tuo`device.json`file di configurazione, hai scelto OTA come`NoOTADataPlaneProtocol`come`HTTP`. Il gruppo di test scelto per l'esecuzione deve corrispondere al tuo`device.json`selezioni di capacità.

Eseguire il debug di un timeout IDT durante il monitoraggio dell'output del dispositivo

IDT può scadere per una serie di motivi. Se si verifica un timeout durante la fase di monitoraggio dell'output del dispositivo di un test e puoi visualizzare i risultati all'interno del registro dei casi del test IDT, significa che i risultati sono stati analizzati in modo errato da IDT. Uno dei motivi potrebbero essere i messaggi di registro interlacciati nel bel mezzo dei risultati del test. In tal caso, si prega di fare riferimento alGuida al porting di FreeRTOSper ulteriori dettagli su come devono essere configurati i log UNITY.

Un altro motivo per cui si verifica un timeout durante il monitoraggio dell'output del dispositivo potrebbe essere il riavvio del dispositivo dopo un singolo errore del test TLS. Il dispositivo esegue quindi l'immagine lampeggiata e causa un ciclo infinito che viene visualizzato nei registri. In tal caso, assicurati che il dispositivo non si riavvii dopo un errore del test.

Debug dell'errore "accesso non autorizzato alla risorsa"

Potresti vedere un errore indicante che l'*`utente/ruolo`* non è autorizzato ad accedere alla risorsa nell'output del terminale o nel file `test_manager.log` sotto `/results/`*`execution-id`*`/logs`. Per

risolvere questo problema, associare la policy `AWSIoTDeviceTesterForFreeRTOSFullAccess` gestita all'utente del test. Per ulteriori informazioni, consulta Creazione e configurazione di un account AWS.

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 si verificano errori dovuti a timeout o connessioni non disponibili nelWiFio test di socket sicuri, assicurati che la tua rete sia configurata per consentire il traffico verso porte configurate nell'intervallo 1024 - 49151.

Il test Secure Sockets utilizza le porte 33333 e 33334 per impostazione predefinita. LaWiFitests utilizza la porta 33335 per impostazione predefinita. 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 tuo 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 dispositivo Espressif WRover e le due porte a esso assegnate sono `/dev/ttyUSB0` e `/dev/ttyUSB1`, usa `/dev/ttyUSB1` nel file `device.json`.

Per Windows, segui la stessa logica.

Lettura dei dati dal dispositivo

IDT for FreeRTOS utilizza la build dei singoli dispositivi e gli strumenti flash 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 for 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, puoi usare un programma comeTeraTerm.

```
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:

- Errore di compilazione o flashing di FreeRTOS sul 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 registri IDT for FreeRTOS vengono 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

QuandoAWS IoT Device Testerviene eseguito, i guasti 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 non è riuscito. Quindi usa queste informazioni per trovare e aprire il file di registro per quel 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 build e flash, l'output dell'esecuzione del test e altro ancoraAWS IoT Device Testeruscita console.

Problemi con il bucket S3

Se si premeCTRL+Cdurante l'esecuzione di IDT, IDT avvierà un processo di pulizia. Parte di questa pulizia consiste nel rimuovere le 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 della creazione di troppi bucket Amazon S3. Ciò significa che la prossima volta che eseguirai IDT, i test inizieranno a fallire.

Se si premeCTRL+Cper interrompere IDT, è necessario lasciarlo completare il processo di pulizia per evitare questo problema. Puoi anche eliminare dal tuo account i bucket Amazon S3 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 cellulare eAWSaccuse

Quando il`Cellular`la funzionalità è impostata su`Yes`nel tuo`device.JSON`fascicolo,FullSecureSocketsutilizzerà le istanze t.micro EC2 per eseguire i test e ciò potrebbe comportare costi aggiuntivi per il tuoAWSconto. Per ulteriori informazioni, consulta Prezzi di Amazon EC2.

Politica di generazione di report sulle qualifiche

I report sulle qualifiche vengono generati solo daAWS IoT Device TesterVersioni (IDT) che supportano le versioni di FreeRTOS rilasciate negli ultimi due anni. In caso di domande sulla politica di supporto, si prega di contattareAWS Support.

AWSPolitica gestita perAWS IoT Device Tester

Una policy gestita da AWS è una policy autonoma creata e amministrata da AWS. Le policy gestite da AWS sono progettate per fornire autorizzazioni per molti casi d'uso comuni in modo da poter iniziare ad assegnare autorizzazioni a utenti, gruppi e ruoli.

Ricorda che le policy gestite da AWS potrebbero non concedere autorizzazioni con privilegi minimi per i tuoi casi d'uso specifici perché possono essere utilizzate da tutti i clienti AWS. Consigliamo pertanto di ridurre ulteriormente le autorizzazioni definendo policy gestite dal cliente specifiche per i tuoi casi d'uso.

Non è possibile modificare le autorizzazioni definite nelle policy gestite da AWS. Se AWS aggiorna le autorizzazioni definite in una policy gestita da AWS, l'aggiornamento riguarda tutte le identità principali (utenti, gruppi e ruoli) a cui è collegata la policy. È molto probabile che AWS aggiorni una policy gestita da AWS quando viene lanciato un nuovo Servizio AWS o nuove operazioni API diventano disponibili per i servizi esistenti.

Per ulteriori informazioni, consultare Policy gestite da AWS nella Guida per l'utente di IAM.

Argomenti

- AWSpolitica gestita:AWSIoTDeviceTesterForFreeRTOSFullAccess
- Aggiornamenti di AWS IoT Device Tester alle policy gestite da AWS

AWSpolitica gestita:AWSIoTDeviceTesterForFreeRTOSFullAccess

`LaAWSIoTDeviceTesterForFreeRTOSFullAccess`la politica gestita contiene quanto segueAWS IoT Device Testerautorizzazioni per il controllo della versione, le funzionalità di aggiornamento automatico e la raccolta di metriche.

Dettagli dell'autorizzazione

Questa policy include le seguenti autorizzazioni:

- `iot-device-tester:SupportedVersion`

  SovvenzioniAWS IoT Device Testerautorizzazione a recuperare l'elenco dei prodotti supportati, delle suite di test e delle versioni IDT.
- `iot-device-tester:LatestIdt`

  SovvenzioniAWS IoT Device Testerautorizzazione a recuperare l'ultima versione IDT disponibile per il download.
- `iot-device-tester:CheckVersion`

  SovvenzioniAWS IoT Device Testerautorizzazione a verificare la compatibilità delle versioni per IDT, suite di test e prodotti.
- `iot-device-tester:DownloadTestSuite`

  SovvenzioniAWS IoT Device Testerautorizzazione a scaricare gli aggiornamenti della suite di test.
- `iot-device-tester:SendMetrics`

  SovvenzioniAWSautorizzazione a raccogliere metriche suAWS IoT Device Testeruso interno.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::*:role/idt-*",
            "Condition": {
                "StringEquals": {
```

```
                    "iam:PassedToService": "iot.amazonaws.com"
                }
            }
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": [
                "iot:DeleteThing",
                "iot:AttachThingPrincipal",
                "iot:DeleteCertificate",
                "iot:GetRegistrationCode",
                "iot:CreatePolicy",
                "iot:UpdateCACertificate",
                "s3:ListBucket",
                "iot:DescribeEndpoint",
                "iot:CreateOTAUpdate",
                "iot:CreateStream",
                "signer:ListSigningJobs",
                "acm:ListCertificates",
                "iot:CreateKeysAndCertificate",
                "iot:UpdateCertificate",
                "iot:CreateCertificateFromCsr",
                "iot:DetachThingPrincipal",
                "iot:RegisterCACertificate",
                "iot:CreateThing",
                "iam:ListRoles",
                "iot:RegisterCertificate",
                "iot:DeleteCACertificate",
                "signer:PutSigningProfile",
                "s3:ListAllMyBuckets",
                "signer:ListSigningPlatforms",
                "iot-device-tester:SendMetrics",
                "iot-device-tester:SupportedVersion",
                "iot-device-tester:LatestIdt",
                "iot-device-tester:CheckVersion",
                "iot-device-tester:DownloadTestSuite"
            ],
            "Resource": "*"
        },
        {
            "Sid": "VisualEditor2",
            "Effect": "Allow",
            "Action": [
```

```
                "iam:GetRole",
                "signer:StartSigningJob",
                "acm:GetCertificate",
                "signer:DescribeSigningJob",
                "s3:CreateBucket",
                "execute-api:Invoke",
                "s3:DeleteBucket",
                "s3:PutBucketVersioning",
                "signer:CancelSigningProfile"
            ],
            "Resource": [
                "arn:aws:execute-api:us-east-1:098862408343:9xpmnvs5h4/prod/POST/
metrics",
                "arn:aws:signer:*:*:/signing-profiles/*",
                "arn:aws:signer:*:*:/signing-jobs/*",
                "arn:aws:iam::*:role/idt-*",
                "arn:aws:acm:*:*:certificate/*",
                "arn:aws:s3:::idt-*",
                "arn:aws:s3:::afr-ota*"
            ]
        },
        {
            "Sid": "VisualEditor3",
            "Effect": "Allow",
            "Action": [
                "iot:DeleteStream",
                "iot:DeleteCertificate",
                "iot:AttachPolicy",
                "iot:DetachPolicy",
                "iot:DeletePolicy",
                "s3:ListBucketVersions",
                "iot:UpdateCertificate",
                "iot:GetOTAUpdate",
                "iot:DeleteOTAUpdate",
                "iot:DescribeJobExecution"
            ],
            "Resource": [
                "arn:aws:s3:::afr-ota*",
                "arn:aws:iot:*:*:thinggroup/idt*",
                "arn:aws:iam::*:role/idt-*"
            ]
        },
        {
            "Sid": "VisualEditor4",
```

```
            "Effect": "Allow",
            "Action": [
                "iot:DeleteCertificate",
                "iot:AttachPolicy",
                "iot:DetachPolicy",
                "s3:DeleteObjectVersion",
                "iot:DeleteOTAUpdate",
                "s3:PutObject",
                "s3:GetObject",
                "iot:DeleteStream",
                "iot:DeletePolicy",
                "s3:DeleteObject",
                "iot:UpdateCertificate",
                "iot:GetOTAUpdate",
                "s3:GetObjectVersion",
                "iot:DescribeJobExecution"
            ],
            "Resource": [
                "arn:aws:s3:::afr-ota*/*",
                "arn:aws:s3:::idt-*/*",
                "arn:aws:iot:*:*:policy/idt*",
                "arn:aws:iam::*:role/idt-*",
                "arn:aws:iot:*:*:otaupdate/idt*",
                "arn:aws:iot:*:*:thing/idt*",
                "arn:aws:iot:*:*:cert/*",
                "arn:aws:iot:*:*:job/*",
                "arn:aws:iot:*:*:stream/*"
            ]
        },
        {
            "Sid": "VisualEditor5",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::afr-ota*/*",
                "arn:aws:s3:::idt-*/*"
            ]
        },
        {
            "Sid": "VisualEditor6",
            "Effect": "Allow",
```

```
            "Action": [
                "iot:CancelJobExecution"
            ],
            "Resource": [
                "arn:aws:iot:*:*:job/*",
                "arn:aws:iot:*:*:thing/idt*"
            ]
        },
        {
            "Sid": "VisualEditor7",
            "Effect": "Allow",
            "Action": [
                "ec2:TerminateInstances"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:instance/*"
            ],
            "Condition": {
                "StringEquals": {
                    "ec2:ResourceTag/Owner": "IoTDeviceTester"
                }
            }
        },
        {
            "Sid": "VisualEditor8",
            "Effect": "Allow",
            "Action": [
                "ec2:AuthorizeSecurityGroupIngress",
                "ec2:DeleteSecurityGroup"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:security-group/*"
            ],
            "Condition": {
                "StringEquals": {
                    "ec2:ResourceTag/Owner": "IoTDeviceTester"
                }
            }
        },
        {
            "Sid": "VisualEditor9",
            "Effect": "Allow",
            "Action": [
                "ec2:RunInstances"
```

```
            ],
            "Resource": [
                "arn:aws:ec2:*:*:instance/*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/Owner": "IoTDeviceTester"
                }
            }
        },
        {
            "Sid": "VisualEditor10",
            "Effect": "Allow",
            "Action": [
                "ec2:RunInstances"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:image/*",
                "arn:aws:ec2:*:*:security-group/*",
                "arn:aws:ec2:*:*:volume/*",
                "arn:aws:ec2:*:*:key-pair/*",
                "arn:aws:ec2:*:*:placement-group/*",
                "arn:aws:ec2:*:*:snapshot/*",
                "arn:aws:ec2:*:*:network-interface/*",
                "arn:aws:ec2:*:*:subnet/*"
            ]
        },
        {
            "Sid": "VisualEditor11",
            "Effect": "Allow",
            "Action": [
                "ec2:CreateSecurityGroup"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:security-group/*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/Owner": "IoTDeviceTester"
                }
            }
        },
        {
            "Sid": "VisualEditor12",
```

```
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeInstances",
                "ec2:DescribeSecurityGroups",
                "ssm:DescribeParameters",
                "ssm:GetParameters"
            ],
            "Resource": "*"
        },
        {
            "Sid": "VisualEditor13",
            "Effect": "Allow",
            "Action": [
                "ec2:CreateTags"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:security-group/*",
                "arn:aws:ec2:*:*:instance/*"
            ],
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "aws:TagKeys": [
                        "Owner"
                    ]
                },
                "StringEquals": {
                    "ec2:CreateAction": [
                        "RunInstances",
                        "CreateSecurityGroup"
                    ]
                }
            }
        }
    ]
}
```

Aggiornamenti di AWS IoT Device Tester alle policy gestite da AWS

Puoi visualizzare i dettagli sugli aggiornamenti diAWSpolitiche gestite perAWS IoT Device Testerdal momento in cui questo servizio ha iniziato a tracciare queste modifiche.

Versione	Modifica	Descrizione	Data
7 (Più recente)	Ristrutturato ile`ec2:CreateTags` condizioni.	Rimozione dell'utilizzo di`ForAnyValues` .	14/06/2023
6	Rimosso`freertos:ListHardwarePlatforms` dalla politica.	Rimozione delle autorizzazioni poiché questa azione è obsoleta a partire dal 1° marzo 2023.	02/06/2023
5	Sono state aggiunte le autorizzazioni per eseguire i test del server echo utilizzando EC2.	Questo serve per avviare e arrestare un'istanza EC2 in 'AWSconti.	15/12/2020
4	Aggiunto `iot:CancelJobExecution` .	Questa autorizzazione annulla i lavori OTA.	17/07/2020
3	Sono state aggiunte le seguenti autorizzazioni: • `iot-device-tester:DownloadTestSuite` , • `iot-device-tester:CheckVersion` , • `iot-device-tester:LatestIdt` , • `iot-device-tester:`	• `iot-device-tester:DownloadTestSuite` — SovvenzioniAWS IoT Device Testerautorizzazione a scaricare gli aggiornamenti della suite di test, • `iot-device-tester:CheckVersion` — SovvenzioniAWS IoT Device Testerautorizzazio	23/03/2020

Versione	Modifica	Descrizione	Data
	`Supported Version` .	ne a verificare la compatibilità delle versioni per IDT, suite di test e prodotti, • `iot-device-tester: LatestIdt` — Sovvenzioni AWS IoT Device Tester autorizzazione a recuperare l'ultima versione IDT disponibile per il download, • `iot-device-tester: SupportedVersion` — Sovvenzioni AWS IoT Device Tester autorizzazione a recuperare l'elenco dei prodotti supportati, delle suite di test e delle versioni IDT.	
2	Aggiunta `iot-device-tester: SendMetrics` autorizzazioni.	Sovvenzioni AWS autorizzazione a raccogliere metriche su AWS IoT Device Tester uso interno.	18/02/2020
1	Versione iniziale.		12/02/2020

Politica di supporto per AWS IoT Device Tester for FreeRTOS

> **Important**
>
> A partire da ottobre 2022, AWS IoT Device Tester for AWS IoT FreeRTOS Qualification (FRQ) 1.0 non genera rapporti di qualificazione firmati. Non è possibile qualificare nuovi dispositivi AWS IoT FreeRTOS da inserire nel AWSPartner Device Catalog tramite il Device Qualification Program utilizzando le AWS versioni IDT FRQ 1.0. Sebbene non sia possibile qualificare i dispositivi FreeRTOS utilizzando IDT FRQ 1.0, è possibile continuare a testare i dispositivi FreeRTOS con FRQ 1.0. Ti consigliamo di utilizzare IDT FRQ 2.0 per qualificare ed elencare i dispositivi FreeRTOS nel Partner Device Catalog. AWS

AWS IoT Device Testerfor FreeRTOS è uno strumento di automazione dei test per convalidare la porta FreeRTOS sui dispositivi. Inoltre, puoi qualificare i tuoi dispositivi FreeRTOS ed elencarli nel AWSPartner Device Catalog. Il AWS IoT Device Tester for FreeRTOS supporta la convalida e la qualificazione delle librerie FreeRTOS Long Term Supported (LTS) disponibili su FreeRTOS/FreeRTOS e la linea principale di FreeRTOS disponibile GitHub su FreeRTOS/FreeRTOS. Ti consigliamo di utilizzare le versioni più recenti di FreeRTOS e di FreeRTOS AWS IoT Device Tester per convalidare e qualificare i tuoi dispositivi.

Per FreeRTOS-LTS, IDT supporta la convalida e la qualificazione della versione LTS di FreeRTOS 202210. Vedi qui per ulteriori informazioni sulle versioni LTS di FreeRTOS e sulla loro tempistica di manutenzione. Una volta terminato il periodo di supporto di queste versioni LTS, puoi continuare la convalida, ma IDT non genererà un rapporto che ti consentirà di presentare il tuo dispositivo per l'idoneità.

Per i FreeRTOS principali disponibili su FreeRTOS/FreeRTOS, supportiamo la convalida e la qualificazione di tutte le versioni rilasciate negli ultimi sei mesi, o delle due versioni precedenti di FreeRTOS se rilasciate a più di sei mesi di distanza. Vedi qui per le versioni attualmente supportate. Per le versioni non supportate di FreeRTOS, puoi continuare la convalida, ma IDT non genererà un rapporto, che ti consentirà di inviare il tuo dispositivo per l'idoneità.

Consulta Versioni supportate diAWS IoT Device Testerper FreerTOS le ultime versioni IDT e FreeRTOS supportate. Puoi utilizzare una qualsiasi delle versioni supportate di AWS IoT Device Tester con la versione corrispondente di FreeRTOS per testare o qualificare il tuo dispositivo. Se continui a utilizzare ilVersioni IDT non supportate per FreeRTOS, non riceverai le correzioni di bug o gli aggiornamenti più recenti.

Per domande sulla politica di supporto, contatta l'assistenza AWS clienti.

Sicurezza dell'AWS

Per AWS, la sicurezza del cloud ha la massima priorità. In quanto cliente AWS, è possibile trarre vantaggio da un'architettura di data center e di rete progettata per soddisfare i requisiti delle organizzazioni più esigenti a livello di sicurezza.

La sicurezza è una responsabilità condivisa tra te e AWS. Il modello di responsabilità condivisa descrive questo come sicurezza del cloud e sicurezza nel cloud:

- La sicurezza del cloud: AWS è responsabile della protezione dell'infrastruttura che gestisce i servizi AWS nel cloud AWS. AWS fornisce inoltre servizi che puoi utilizzare in sicurezza. L'efficacia della nostra sicurezza è regolarmente testata e verificata da revisori di terze parti come parte dei programmi di conformità AWS. Per ulteriori informazioni sui programmi di conformità che si applicano a un servizio AWS, consulta Servizi coperti dal programma di conformità AWS.
- Sicurezza nel cloud: la responsabilità è determinata dal servizio AWS che viene utilizzato. 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 consente di comprendere come applicare il modello di responsabilità condivisa quando si usa AWS. I seguenti argomenti illustrano come configurare l'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 sicurezza di AWS IoT, consulta Sicurezza e identità per AWS IoT.

Argomenti

Identity and Access Management per FreerTOS

AWS Identity and Access Management (IAM) è un Servizio AWSche consente agli amministratori di 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 un Servizio AWS il cui uso non comporta costi aggiuntivi.

Argomenti

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 la procedura di accesso ad AWS con le credenziali di identità. Devi essere autenticato (connesso a AWS) come utente root Utente root dell'account AWS, come utente IAM o assumere un ruolo IAM.

Puoi accedere ad AWS come identità federata utilizzando le credenziali fornite attraverso un'origine di identità. Gli utenti AWS IAM Identity Center (Centro identità IAM), l'autenticazione Single Sign-On (SSO) dell'azienda e le 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. Se accedi ad AWS tramite la federazione, assumi indirettamente un ruolo.

A seconda del tipo di utente, puoi accedere alla AWS Management Console o al portale di accesso AWS. Per ulteriori informazioni sull'accesso ad AWS, consulta la sezione Come accedere al tuo Account AWS nella Guida per l'utente di Accedi ad AWS.

Se accedi ad AWS in modo programmatico, AWS fornisce un Software Development Kit (SDK) e un'interfaccia a riga di comando (CLI) per firmare crittograficamente le richieste utilizzando le tue credenziali. Se non utilizzi gli strumenti AWS, devi firmare le richieste personalmente. Per ulteriori informazioni sulla firma delle richieste, consulta Firma delle richieste AWS nella Guida per l'utente IAM.

A prescindere dal metodo di autenticazione utilizzato, potrebbe essere necessario specificare ulteriori informazioni sulla sicurezza. AWS consiglia ad esempio di utilizzare l'autenticazione a più fattori (MFA) per aumentare la sicurezza dell'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) in AWS nella Guida per l'utente di IAM.

Utente root di un Account AWS

Quando crei un Account AWS, inizi con una singola identità di accesso che ha accesso completo a tutti i Servizi AWS e le risorse nell'account. Tale identità è detta utente root Account AWS ed è possibile accedervi 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 utilizzarle 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 di IAM.

Identità federata

Come best practice, richiedi agli utenti umani, compresi quelli che richiedono l'accesso di amministratore, di utilizzare la federazione con un provider di identità per accedere a Servizi AWSutilizzando credenziali temporanee.

Un'identità federata è un utente della directory degli utenti aziendali, un provider di identità Web, AWS Directory Service, la directory Identity Center o qualsiasi utente che accede ai Servizi AWS utilizzando le credenziali fornite tramite un'origine di identità. Quando le identità federate accedono agli Account AWS, assumono ruoli e i ruoli forniscono credenziali temporanee.

Per la gestione centralizzata degli accessi, consigliamo di utilizzare AWS IAM Identity Center. È possibile creare utenti e gruppi in IAM Identity Center oppure connettersi e sincronizzarsi con un gruppo di utenti e gruppi nell'origine di identità per utilizzarli in tutte le applicazioni e gli Account AWS. Per ulteriori informazioni sul Centro identità IAM, consulta Cos'è Centro identità IAM? nella Guida per l'utente di AWS IAM Identity Center.

Utenti e gruppi IAM

Un utente IAM è una identità all'interno del tuo 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, per 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 di 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, è possibile avere un gruppo denominato Amministratori IAM e concedere a tale 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 Quando creare un utente IAM (invece di un ruolo) nella Guida per l'utente di IAM.

Ruoli IAM

Un ruolo IAM è un'identità all'interno di un Account AWS che dispone di autorizzazioni specifiche. È simile a un utente IAM, ma non è associato a una persona specifica. È possibile assumere temporaneamente un ruolo IAM nella AWS Management Console mediante lo scambio di ruoli. È possibile assumere un ruolo chiamando un'azione AWS CLI o API AWS oppure utilizzando un URL personalizzato. Per ulteriori informazioni sui metodi per l'utilizzo dei ruoli, consulta Utilizzo di ruoli IAM nella Guida per l'utente di 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 Creazione di un ruolo per un provider di identità di terza parte nella Guida per l'utente di 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 ulteriori 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, per alcuni dei Servizi AWS, è possibile collegare 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 Differenza tra i ruoli IAM e le policy basate su risorse nella Guida per l'utente di IAM.
- Accesso multi-servizio: alcuni Servizi AWS utilizzano funzionalità in altri Servizi AWS. Ad esempio, quando effettui una chiamata in un servizio, è comune che tale 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.
  - Inoltro delle sessioni di accesso (FAS): quando si utilizza un utente o un ruolo IAM per eseguire operazioni in AWS, tale utente o ruolo viene considerato un principale. Quando si utilizzano alcuni servizi, è possibile eseguire un'operazione che attiva un'altra azione in un servizio diverso. FAS utilizza le autorizzazioni del principale che effettua la chiamata a un Servizio

AWS, combinate con il Servizio AWS richiedente, per effettuare richieste a servizi a valle. Le richieste FAS vengono effettuate solo quando un servizio riceve una richiesta che necessita di interazioni con altri Servizi AWS o risorse per essere portata a termine. In questo caso è necessario disporre delle autorizzazioni per eseguire entrambe le operazioni. Per i dettagli delle policy relative alle richieste FAS, consulta la pagina Forward access sessions.

- Ruolo di servizio: un ruolo di servizio è un ruolo IAM assunto da un servizio per eseguire operazioni per conto dell'utente. Un amministratore IAM può creare, modificare ed eliminare un ruolo di servizio dall'interno di IAM. Per ulteriori informazioni, consulta la sezione Creazione di un ruolo per delegare le autorizzazioni a un Servizio AWS nella Guida per l'utente di 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 ai servizi sono visualizzati nell'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: è possibile utilizzare un ruolo IAM per gestire credenziali temporanee per le applicazioni in esecuzione su un'istanza EC2 che eseguono richieste di AWS CLIo dell'API AWS. Ciò è preferibile all'archiviazione delle chiavi di accesso nell'istanza EC2. Per assegnare un ruolo AWS a un'istanza EC2, affinché sia disponibile per tutte le relative applicazioni, puoi creare un profilo dell'istanza collegato all'istanza. Un profilo dell'istanza contiene il ruolo e consente ai programmi in esecuzione sull'istanza EC2 di ottenere le credenziali temporanee. Per ulteriori informazioni, consulta Utilizzo di un ruolo IAM per concedere autorizzazioni ad applicazioni in esecuzione su istanze di Amazon EC2 nella Guida per l'utente di IAM.

Per informazioni sull'utilizzo dei ruoli IAM, consulta Quando creare un ruolo IAM (invece di un utente) nella Guida per l'utente di IAM.

Gestione dell'accesso con policy

Per controllare l'accesso a AWS è possibile creare policy e collegarle a identità o risorse AWS. Una policy è un oggetto in AWS che, quando associato a un'identità o a una risorsa, ne definisce le autorizzazioni. AWS valuta queste policy quando un principale IAM (utente, utente root o sessione ruolo) effettua una richiesta. Le autorizzazioni nelle policy determinano l'approvazione o il rifiuto della richiesta. La maggior parte delle policy viene archiviata in AWSsotto forma di 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 di IAM.

Gli amministratori possono utilizzare le policy AWSJSON per specificare l'accesso ai diversi elementi. In altre parole, quale principale può eseguire azioni 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 azioni sulle risorse di cui hanno bisogno, un amministratore IAM può creare policy IAM. Successivamente l'amministratore può 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'azione `iam:GetRole`. Un utente con tale policy può ottenere informazioni sul ruolo dalla AWS Management Console, la AWS CLI o l'API AWS.

Policy basate su 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 azioni che utenti e ruoli possono eseguire, su quali risorse e in quali condizioni. Per informazioni su come creare una policy basata su identità, consulta Creazione di policy IAM nella Guida per l'utente di IAM.

Le policy basate su identità possono essere ulteriormente classificate come policy inline o policy gestite. Le policy inline sono incorporate direttamente in un singolo utente, gruppo o ruolo. Le policy gestite sono policy autonome che possono essere collegate a più utenti, gruppi e ruoli in Account AWS. Le policy gestite includono le policy gestite da AWS e le policy gestite dal cliente. 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 di IAM.

Policy basate su risorse

Le policy basate su risorse sono documenti di policy JSON che è possibile allegare a una risorsa. Gli esempi più comuni di policy basate su 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 utilizzarle per controllare l'accesso a una risorsa specifica. Quando è allegata a una risorsa, una policy definisce le azioni 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 è possibile utilizzare le policy gestite da AWS da IAM in una policy basata su risorse.

Liste di controllo degli accessi (ACL)

Le liste di controllo degli accessi (ACL) controllano quali principali (membri, utenti o ruoli dell'account) hanno le autorizzazioni per accedere a una risorsa. Le ACL sono simili alle policy basate sulle risorse, sebbene non utilizzino il formato del documento di policy JSON.

Amazon S3, AWS WAF e Amazon VPC sono esempi di servizi che supportano le ACL. Per maggiori informazioni sulle ACL, consulta Panoramica delle liste di controllo degli accessi (ACL) nella Guida per gli sviluppatori di Amazon Simple Storage Service.

Altri tipi di policy

AWS supporta altri tipi di policy 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 funzione 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 di IAM.
- Policy di controllo dei servizi (SCP): le SCP sono policy JSON che specificano il numero massimo di autorizzazioni per un'organizzazione o unità organizzativa (OU) in AWS Organizations. AWS Organizationsè un servizio per il raggruppamento e la gestione centralizzata degli Account AWSmultipli di proprietà dell'azienda. Se abiliti tutte le funzionalità in un'organizzazione, puoi applicare le policy di controllo dei servizi (SCP) a uno o tutti i tuoi account. La SCP limita le autorizzazioni per le entità negli account membri, compreso ogni Utente root dell'account AWS. Per ulteriori informazioni su organizzazioni e policy SCP, consulta la pagina sulle Policy di controllo dei servizi nella Guida per l'utente di AWS Organizations.
- 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 di IAM.

Più tipi di policy

Quando più tipi di policy si applicano a una richiesta, le autorizzazioni risultanti sono più complicate da comprendere. Per informazioni su come AWS determina se consentire una richiesta quando sono coinvolti più tipi di policy, consulta Logica di valutazione delle policy nella Guida per l'utente di IAM.

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
Liste di controllo degli accessi (ACL)	No
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, AWSconsulta 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 azioni che utenti e ruoli possono eseguire, su quali risorse e in quali condizioni. Per informazioni su come creare una policy basata su identità, consulta Creazione di policy IAM nella Guida per l'utente di 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 allegare a una risorsa. Gli esempi più comuni di policy basate su 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 utilizzarle per controllare l'accesso a una risorsa specifica. Quando è allegata a una risorsa, una policy definisce le azioni 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 l'entità principale e la risorsa si trovano in diversi Account AWS, un amministratore IAM nell'account attendibile deve concedere all'entità principale (utente o ruolo) anche l'autorizzazione per 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 Differenza tra i ruoli IAM e le policy basate su risorse nella Guida per l'utente di IAM.

Azioni politiche per FreerTOS

Supporta le azioni di policy	Sì

Gli amministratori possono utilizzare le policy JSON AWS per specificare gli accessi ai diversi elementi. Cioè, quale principale può eseguire azioni 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 di policy hanno spesso lo stesso nome dell'operazione API AWS. Ci sono alcune eccezioni, ad esempio le azioni 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 JSON AWS per specificare gli accessi ai diversi elementi. Cioè, 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'azione. 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). Puoi eseguire questa operazione per azioni 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 dei relativi ARN, consulta 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 JSON AWS per specificare gli accessi ai diversi elementi. Cioè, quale principale può eseguire azioni 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. Puoi 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 specifichi più valori per una singola chiave di condizione, AWS valuta la condizione utilizzando un'operazione `OR` logica. Tutte le condizioni devono essere soddisfatte prima che le autorizzazioni dell'istruzione vengano concesse.

Puoi anche utilizzare variabili segnaposto quando specifichi le condizioni. Ad esempio, puoi 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 per il servizio. Per visualizzare tutte le chiavi di condizione globali di AWS, consulta Chiavi di contesto delle condizioni globali di AWS nella Guida per l'utente di 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

ACL in FreerTOS

Supporta le ACL	No

Le liste di controllo degli accessi (ACL) controllano quali principali (membri, utenti o ruoli dell'account) hanno le autorizzazioni ad accedere a una risorsa. Le ACL sono simili alle policy basate su risorse, sebbene non utilizzino il formato del documento di policy JSON.

ABAC con FreerTOS

Supporta ABAC (tag nelle policy)	Parziale

Il controllo dell'accesso basato su attributi (ABAC) è una strategia di autorizzazione che definisce le autorizzazioni in base agli attributi. In AWS, tali attributi sono denominati tag. È possibile collegare dei tag alle entità IAM (utenti o ruoli) e a numerose risorse AWS. 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 Che cos'è ABAC? nella Guida per l'utente di 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

Alcuni Servizi AWS non funzionano quando si accede utilizzando credenziali temporanee. Per ulteriori informazioni, inclusi i Servizi AWS che funzionano con le credenziali temporanee, consulta Servizi AWS supportati da IAM nella Guida per l'utente IAM.

Le credenziali temporanee sono utilizzate se si accede alla AWS Management Console utilizzando qualsiasi metodo che non sia la combinazione di nome utente e password. Ad esempio, quando accedi ad AWS utilizzando il collegamento Single Sign-On (SSO) della tua azienda, tale processo crea in automatico 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 Cambio di un ruolo (console) nella Guida per l'utente di IAM.

È possibile creare manualmente credenziali temporanee utilizzando la AWS CLIo l'API AWS. È quindi possibile utilizzare tali credenziali temporanee per accedere ad AWS. AWSconsiglia di generare le

credenziali temporanee dinamicamente anziché utilizzare chiavi di accesso a lungo termine. Per ulteriori informazioni, consulta Credenziali di sicurezza provvisorie in IAM.

Autorizzazioni principali multiservizio per FreerTOS

Supporta sessioni di accesso diretto (FAS)	Sì

Quando si utilizza un utente o un ruolo IAM per eseguire operazioni in AWS, si viene considerati un principale. Quando si utilizzano alcuni servizi, è possibile eseguire un'azione che attiva un'altra azione in un servizio diverso. FAS utilizza le autorizzazioni del principale che effettua la chiamata a un Servizio AWS, combinate con il Servizio AWS richiedente, per effettuare richieste a servizi a valle. Le richieste FAS vengono effettuate solo quando un servizio riceve una richiesta che necessita di interazioni con altri Servizi AWS o risorse per essere portata a termine. In questo caso è necessario disporre delle autorizzazioni per eseguire entrambe le operazioni. Per i dettagli delle policy relative alle richieste FAS, consulta la pagina 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 Creazione di un ruolo per delegare le autorizzazioni a un Servizio AWS nella Guida per l'utente di 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 ai servizi è un tipo di ruolo di servizio che è collegato a un Servizio AWS. Il servizio può assumere il ruolo per eseguire un'operazione per tuo conto. I ruoli collegati ai servizi sono visualizzati nell'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 sono in grado di eseguire attività utilizzando la AWS Management Console, l'AWS Command Line Interface (AWS CLI) o l'API AWS. Per concedere agli utenti l'autorizzazione per 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 nella Guida per l'utente di IAM.

Per i dettagli sulle azioni e sui tipi di risorse definiti da FreeRTOS, incluso il formato degli ARN per ciascuno 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 operazioni possono comportare costi aggiuntivi per l'Account AWS. Quando crei o modifichi policy basate su identità, segui queste linee guida e raccomandazioni:

- Nozioni di base sulle policy gestite da AWSe passaggio alle autorizzazioni con privilegio minimo: per le informazioni di base su come concedere autorizzazioni a utenti e carichi di lavoro, utilizza le policy gestite da AWSche concedono le autorizzazioni per molti casi d'uso comuni. Sono disponibili nel tuo Account AWS. Ti consigliamo pertanto di ridurre ulteriormente le autorizzazioni definendo policy gestite dal cliente di AWSspecifiche per i tuoi casi d'uso. Per ulteriori informazioni, consulta Policy gestite da AWS o Policy gestite da AWS per le funzioni dei processi nella Guida per l'utente IAM.
- Applica le autorizzazioni con privilegi minimi: quando imposti le autorizzazioni con le policy IAM, concedi solo le autorizzazioni richieste per eseguire un'attività. Puoi 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 di IAM.
- Condizioni d'uso nelle policy IAM per limitare ulteriormente l'accesso: per limitare l'accesso a operazioni e risorse puoi 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 inoltre utilizzare le condizioni per concedere l'accesso alle operazioni di servizio, ma solo 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 di 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 IAM Access Analyzer nella Guida per l'utente di IAM.
- Richiesta dell'autenticazione a più fattori (MFA): se hai uno scenario che richiede utenti IAM o utenti root nel tuo Account AWS, attiva 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 Configurazione dell'accesso alle API protetto con MFA nella Guida per l'utente di 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, è necessario 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 concedere le autorizzazioni minime della console agli utenti che effettuano chiamate solo alla AWS CLI o all'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 allegate alla relativa identità utente. La policy include le autorizzazioni per completare questa azione sulla console o a livello di programmazione utilizzando la AWS CLIo l'API 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`.

Per ulteriore assistenza con l'accesso, contatta l'amministratore AWS. 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 AWSconsentono di trasmettere un ruolo esistente a tale servizio, invece di creare un nuovo ruolo di servizio o un ruolo collegato ai servizi. 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`.

Per ulteriore assistenza con l'accesso, contatta l'amministratore AWS. 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 servizi che supportano policy basate su risorse o liste di controllo accessi (ACL), utilizza tali policy 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 informazioni su come garantire l'accesso alle risorse negli Account AWSche possiedi, consulta Fornire l'accesso a un utente IAM in un altro Account AWS in tuo possesso nella Guida per l'utente IAM.
- Per informazioni su come fornire l'accesso alle risorse ad Account AWS di terze parti, consulta Fornire l'accesso agli 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 di IAM.
- Per informazioni sulle differenze tra l'utilizzo di ruoli e policy basate su risorse per l'accesso multi-account, consulta Differenza tra i ruoli IAM e le policy basate su risorse nella Guida per l'utente IAM.

Convalida della conformità

Per sapere se il Servizio AWSè coperto da programmi di conformità specifici, consulta i Servizi AWScoperti dal programma di conformità e scegli il programma di conformità desiderato. Per informazioni generali, consulta Programmi per la conformità di AWS.

È possibile scaricare i report di audit di terze parti utilizzando AWS Artifact. Per ulteriori informazioni, consulta Download di report in AWS Artifact.

La responsabilità di conformità durante l'utilizzo dei Servizi AWSè determinata dalla riservatezza dei dati, dagli obiettivi di conformità dell'azienda e dalle normative vigenti. Per semplificare il rispetto della conformità, AWSmette a disposizione le seguenti risorse:

- Guide Quick Start per la sicurezza e conformità: queste guide all'implementazione illustrano considerazioni relative all'architettura e forniscono la procedura per l'implementazione di ambienti di base su AWSincentrati sulla sicurezza e sulla conformità.
- Architetture per la sicurezza e la conformità HIPAA su Amazon Web Services: questo whitepaper descrive come le aziende possono utilizzare AWSper creare applicazioni conformi alla normativa HIPAA.

> **Note**
>
> Non tutti i Servizi AWSsono conformi ai requisiti HIPAA. Per ulteriori informazioni, consulta la sezione Riferimenti sui servizi conformi ai requisiti HIPAA.

- Risorse per la conformità AWS: una raccolta di cartelle di lavoro e guide suddivise per settore e area geografica.
- AWSGuide 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 le regole nella Guida per gli sviluppatori di AWS Config: il servizio AWS Configvaluta il livello di conformità delle configurazioni delle risorse con pratiche interne, linee guida e regolamenti.
- AWS Security Hub: questo Servizio AWSfornisce una visione completa dello stato di sicurezza all'interno di AWS. La Centrale di sicurezza utilizza i controlli di sicurezza per valutare le risorse AWSe 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.
- AWS Audit Manager: questo Servizio AWSaiuta a verificare continuamente l'utilizzo di AWSper semplificare la gestione dei rischi e della conformità alle normative e agli standard di settore.

Resilienza nell'AWS

L'infrastruttura globale di AWSè basata su Regioni e zone di disponibilità AWS. AWS forniscono più zone di disponibilità fisicamente separate e isolate che sono connesse tramite reti altamente ridondanti, a bassa latenza e velocità effettiva elevata. 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 sulle regioni AWS e sulle zone di disponibilità, consulta Infrastruttura globale di AWS.

Sicurezza dell'infrastruttura in FreerTOS

AWSi servizi gestiti sono protetti dalle procedure di sicurezza di rete AWS globali descritte nel white paper Amazon Web Services: Overview of Security Processes.

Utilizza le chiamate API pubblicate di AWS per accedere ai servizi AWS tramite 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 cifratura 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.

Guida alla migrazione del repository Github di Amazon-FreeRTOS

Se disponi di un progetto FreeRTOS esistente basato sull'ormai obsoleto repository amazon-freertos, segui questi passaggi:

1. Resta aggiornato con il pubblico. Consulta.Librerie FreerTOS LTSpagina per gli aggiornamenti, oppure iscriviti alDa RT a. GitHub archivio per ricevere le ultime patch LTS con correzioni di bug critici e di sicurezza. Puoi scaricare o clonare le ultime patch FreerTOS LTS richieste direttamente dall'utente GitHub archivi.
2. Prendi in considerazione la possibilità di rifattorizzare l'implementazione dell'interfaccia di trasporto di rete per ottimizzare la piattaforma hardware. Le API astratte comeprese sicureeAPI Wifinon sono richiesti dalla versione più recenteCore MQTTbiblioteca. VediInterfaccia di trasportoper ulteriori dettagli.

Appendice

La tabella seguente fornisce consigli per tutti i progetti demo, le librerie legacy e le API astratte all'interno del repository Amazon-FreeRTOS.

Librerie e demo migrate

Nome	Type (Tipo)	Raccomandazioni	
CoreHTTP	demo e libreria	Clona o scarica la libreria CoreHTTP con ilCoreHTTPrepository (sottomodulo se si usa git) nelOrganizzazione FreerTOS su Github. Le demo CoreHTTP si trovano indistribuzione FreerTOS primaria. Per ulteriori informazioni, consulta.Pagina CoreHTTP.	
CoreMQTT	demo e libreria	Clona o scarica la libreria CoreMQTT conCoreMQTTrepository (sottomodulo se si usa git) nelOrganizzazione	

Nome	Type (Tipo)	Raccomandazioni	
		FreerTOS su Github. Le demo di CoreMQTT si trovano indistribuzione FreerTOS primaria. Per ulteriori informazioni, consulta.Pagina CoreMQTT.	
Agente CoreMQTT	demo e libreria	Clona o scarica la libreria CoreMQTT-Agent direttamente dalAgente CoreMQTTrepository (sottomodulo se si usa git) nelOrganizzazione FreerTOS su Github. Le demo di CoreMQTT-Agent si trovano indemo di CoreMQTT-Agent-Demosdeposito. Per ulteriori informazioni, consulta.pagina CoreMQTT-Agent.	
device_defender_per_aws	demo e libreria	LaAWS IoTLa libreria Device Defender si trova nel suo repository inAWS GitHub organizzazione. Clonalo o scaricalo (sottomodulo se usi git) direttamente dalAWS IoTDevice Defenderdeposito. IlAWS IoTLe demo di Device Defender sono disponibili indistribuzione FreerTOS primaria. Per ulteriori informazioni, consulta.AWS IoTPagina Device Defender.	

Nome	Type (Tipo)	Raccomandazioni	
device_shadow_per_aws	demo e libreria	LaAWS IoTLa libreria Device Shadow si trova nel suo repository nelAWS GitHub organizzazione. Clonalo o scaricalo (sottomodulo se usi git) direttamente dalAWS IoTDevice Shadow) archivio. IlAWS IoTLe demo di Device Shadow sono disponibili indistribuzione FreerTOS primaria. Per ulteriori informazioni, consulta.AWS IoTPagina Device Shadow.	
offerte di lavoro per AWS	demo e libreria	LaAWS IoTLa libreria Jobs si trova nel suo archivio inAWS GitHub organizzazione. Clonalo o scaricalo (sottomodulo se usi git) direttamente dalAWS IoTLavorideposito. IlAWS IoTLe demo di Jobs sono disponibili indistribuzione FreerTOS primaria. Per ulteriori informazioni, consulta. AWS IoTPagina delle offerte di lavoro.	
OTA	demo e libreria	LaAWS IoTLo stack è aggiornato con ilAWS GitHub organizzazione. Clonalo o scaricalo (sottomodulo se usi git) direttamente dalAWS IoTOTAdeposito. IlAWS IoTLe demo OTA sono disponibili indistribuzione FreerTOS primaria. Per ulteriori informazioni, consulta.AWS IoTPagina OTA.	

Nome	Type (Tipo)	Raccomandazioni	
CLI e FreerTOS_plus_CLI	demo e libreria	Lo stack è aggiornato con WinSim. Fare riferimento alInterfaccia a riga di comandopagina per maggiori dettagli. Le integrazioni di riferimento FreerTOS IoT in primo piano suNXP i.MX RT1060eSTM32U5piattaforme, forniscono anche esempi CLI su hardware reale.	
logging	macro	Esistono implementazioni della macro di registrazione per piattaforme hardware specifiche utilizzate da alcune librerie FreerTOS. Fate riferimento alpagina di registrazioneper come implementare la macro di registrazione. Fare riferimento auno dei riferimenti IoT con FreerTOSper esempio in esecuzione su hardware reale.	
greengrass_connettività	manifestazione	[Migrazione in corso] Questo progetto dimostrativo presupponeva che la connettività cloud fosse disponibile prima della connessione a unAWS IoTDispositivo Greengrass. È in fase di sviluppo un nuovo progetto che dimostra la capacità di autenticazione e scoperta locali. Aspettatevi che il nuovo progetto dimostrativo venga pubblicato a breve nelOrganizzazione FreerTOS su Github.	

Librerie e demo obsolete

Nome	Type (Tipo)	Raccomandazioni
BLU	demo e librerie	La libreria FreerTOS BLE implement a il protocollo MQTT proprietario e supporta la pubblicazione e la sottoscrizione di argomenti MQTT tramite Bluetooth Low Energy (BLE) tramite un dispositivo proxy come un telefono cellulare. Non è più obbligatorio. Usa il tuo stack BLE o un'opzione di terze parti comeNimBLEper ottimizzare al meglio il tuo progetto.
dev_mode_ key_provisioning	demo	Le integrazioni di riferimento FreerTOS IoT in primo piano suNXP i.MX RT1060,STM32U5, oppureESP32-C3le piattaforme forniscono esempi di provisioning cruciale utilizzando una CLI.
posix	astrazione e demo	Non consigliato per l'uso.
wifi_provisioning	example	Questo esempio ha dimostrato come effettuare il provisioning WiFi credenziali su un dispositivo che utilizza la libreria BLE Amazon-FreeRTOS. Fai riferimento al riferimento FreerTOS Featured IoT sulPiattaforma ESP32C3per un esempio di WiFi fornitura tramite BLE.
API astratte precedenti	code	Si tratta di API create per fornire un'interfaccia astratta per vari stack

Nome	Type (Tipo)	Raccomandazioni	
		software di terze parti, moduli di connettività e piattaforme MCU di diversi fornitori. Ad esempio, esistono interfacce per WiFi astrazione, socket sicuri e così via. Sono supportati nel repository Amazon-FreeRTOS e si trovano nella cartella `/libraries/abstractions/` . Queste API non sono necessarie quando si utilizza ilLibrerie FreerTOS LTS.	

Le librerie e le demo nella tabella precedente non riceveranno patch di sicurezza o correzioni di bug.

Librerie di terze parti

Quando le demo in Amazon-FreeRTOS utilizzano librerie di terze parti, ti consigliamo di sottomodularle direttamente dai loro repository di terze parti.

- CMOCK: clonalo (sottomodulo se usi git) direttamente dalCmockdeposito.
- jsmn: non consigliato e non è aggiornato.
- labbro: clonalo (sottomodulo se usi git) direttamente dallwip-tcpipdeposito.
- lwip_osal: fai riferimento alle integrazioni di riferimento in primo piano di FreerTOS sui.MX RT1060oSTM32U5per sapere come implementare lwip_osal sulla tua piattaforma hardware/scheda.
- mbedtls: clonalo (sottomodulo se usi git) direttamente dalMBed-TLSdeposito. La configurazione e le utilità di mbedtls possono essere riutilizzate; in questo caso creane una copia locale.
- pizze 11: clonalo (sottomodulo se usi git) direttamente da uno deicore PKCS 11biblioteca oIMMAGINI DI OASI 11deposito.
- minuscolo: clonalo (sottomodulo se usi git) direttamente daltinycbordeposito.
- minycrypt: ti consigliamo di utilizzare gli acceleratori di crittografia della tua piattaforma MCU, se disponibili. Se vuoi continuare a usare tinycrypt, clonalo (sottomodulo se usi git) direttamente daltinycryptdeposito.
- tracealyzer_recorder: clonalo (sottomodulo se usi git) direttamente da Percepioregistratore di traccedeposito.

- unità: clonalo (sottomodulo se usi git) direttamente dalThrowTheSwitch/Unitydeposito.
- win_pcap: win_pcap non viene più mantenuto. Si consiglia di utilizzare invece libslirp, libpcap (posix) o npcap.

Test di portabilità e test di integrazione

Tutti i test effettuati nell'ambito del`/tests`le cartelle necessarie per convalidare l'integrazione delle librerie FreerTOS sono state migrate alTest di integrazione delle librerie ERTOS gratuitideposito. Questi possono essere usati per testare l'implementazione PAL e l'integrazione delle librerie. Gli stessi test vengono utilizzati daAWS IoTDevice Tester (IDT) perAWSProgramma di qualificazione dei dispositivi per FreerTOS.

Documentazione archiviata di FreeRTOS

Archivio della guida utente di FreeRTOS

Queste versioni precedenti della Guida per l'utente di FreeRTOS sono disponibili per l'uso con le release di FreeRTOS LTS (supporto a lungo termine).

- Guida per l'utente di FreeRTOS per la versione 202210.00
- Guida utente di FreeRTOS per la versione 202012.00 di FreeRTOS

Contenuti precedenti della Guida per l'utente di FreeRTOS

Questo contenuto è obsoleto ma fornito qui come riferimento.

Vedi Nozioni di base su FreeRTOS i link ai contenuti recenti.

Nosu FreeRTOS

> **Important**
>
> Questa pagina fa riferimento al repository Amazon-FreeRTOS che è obsoleto. Consigliamo di Nozioni su, quando si crea un nuovo progetto. Se hai già un progetto FreeRTOS esistente basato sull'ormai obsoleto repository Amazon-FreeRTOS, consulta ilGuida alla migrazione del repository Github di Amazon-FreeRTOS.

Questo tutorial introduttivo a FreeRTOS mostra come scaricare e configurare FreeRTOS su una macchina host e quindi compilare ed eseguire una semplice applicazione demo su una scheda microcontrollore qualificata.

Nel tutorial si presume una certa familiarità con AWS IoT e con la console 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 di CMake con FreeRTOS
- Distribuzione delle chiavi in modalità sviluppatore
- Guide alle operazioni di base specifiche per la scheda
- Fasi successive con FreeRTOS

Applicazione demo FreeRTOS

> **Important**
>
> Questa pagina fa riferimento al repository Amazon-FreeRTOS che è obsoleto. Consigliamo di Nozioni su, quando si crea un nuovo progetto. Se hai già un progetto FreeRTOS esistente basato sull'ormai obsoleto repository Amazon-FreeRTOS, consulta ilGuida alla migrazione del repository Github di Amazon-FreeRTOS.

L'applicazione demo in questo tutorial è la demo dell'agente CoreMQTT definita nel*freertos*/`demos/coreMQTT_Agent/mqtt_agent_task.c` file. Utilizza illibreria CoreMQTT per connettersi alAWS Cloud e quindi pubblicare periodicamente messaggi su un argomento MQTT ospitato dal brokerAWS IoT MQTT.

Può essere eseguita una sola applicazione demo FreeRTOS alla volta. Quando si crea un progetto demo di FreeRTOS, la prima demo abilitata nel file di*freertos*/`vendors/`*vendor*/`boards/`*board*`/aws_demos/config_files/aws_demo_config.h` intestazione è l'applicazione che viene eseguita. Per questo tutorial, non è necessario abilitare o disabilitare le demo. La demo dell'agente CoreMQTT è abilitata per impostazione predefinita.

Per ulteriori informazioni sulle applicazioni demo incluse in FreeRTOS, consultaDemo FreeRTOS FreeRS.

Fase iniziale

> **Important**
>
> Questa pagina fa riferimento al repository Amazon-FreeRTOS che è obsoleto. Ti consigliamo di iniziare da qui quando crei un nuovo progetto. Se disponi già di un progetto FreeRTOS

esistente basato sull'ormai obsoleto repository Amazon-FreerTOS, consulta il. Guida alla migrazione del repository Github di Amazon-FreeRTOS

Per iniziare a utilizzare FreerTOS AWS IoT con, devi avere AWS un account, un utente con autorizzazioni di 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 utilizzi Espressif ESP32- DevKit C, ESP-WROVER-KIT o ESP32-WROOM-32SE, salta questi passaggi e vai a. Guida introduttiva all'Espressif ESP32- DevKit C e all'ESP-WROVER-KIT
- Se intendi utilizzare Nordic nRF52840-DK, salta questi passaggi e vai a Nozioni di base su Nordic nRF52840-DK.

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

AWSConfigurazione dell'account e delle autorizzazioni

Registrarsi per creare un Account AWS

Se non disponi di un Account AWS, completa la procedura seguente per crearne uno.

Per registrarsi a un Account AWS

1. Apri la pagina all'indirizzo https://portal.aws.amazon.com/billing/signup.
2. Segui le istruzioni online.

   Nel corso della procedura di registrazione riceverai una telefonata, durante la quale sarà necessario inserire un codice di verifica attraverso la tastiera del telefono.

Durante la registrazione di un Account AWS, viene creato un Utente root dell'account AWS. L'utente root dispone dell'accesso a tutte le risorse e tutti i Servizi AWSnell'account. Come best practice di sicurezza, assegna l'accesso amministrativo a un utente amministrativo e utilizza solo l'utente root per eseguire attività che richiedono l'accesso di un utente root.

Al termine del processo di registrazione, riceverai un'e-mail di conferma da AWS. È possibile visualizzare l'attività corrente dell'account e gestire l'account in qualsiasi momento accedendo all'indirizzo https://aws.amazon.com/ e selezionando Il mio account.

Creazione di un utente amministratore

Dopo aver effettuato la registrazione di un Account AWS, proteggi Utente root dell'account AWS, abilita AWS IAM Identity Center e crea un utente amministratore in modo da non utilizzare l'utente root per le attività quotidiane.

Protezione dell'Utente root dell'account AWS

1. Accedi alla AWS Management Console come proprietario dell'account scegliendo Utente root e immettendo l'indirizzo email del Account AWS. Nella pagina successiva, inserisci la password.

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

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

Creazione di un utente amministratore

1. Abilita IAM Identity Center

   Per istruzioni, consulta Abilitazione di AWS IAM Identity Center nella Guida per l'utente di AWS IAM Identity Center.
2. In Centro identità AWS IAM, assegna l'accesso amministrativo a un utente amministrativo.

   Per un tutorial sull'utilizzo di IAM Identity Center directory come origine di identità, consulta Configure user access with the default IAM Identity Center directory nella Guida per l'utente di AWS IAM Identity Center.

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 Accedere al portale di accesso AWS nella Guida per l'utente Accedi ad AWS.

Per fornire l'accesso, aggiungi autorizzazioni ai tuoi 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 Creating a role for a third-party identity provider (federation) (Creazione di un ruolo per un provider di identità di terze parti [federazione]) nella Guida per l'utente di IAM.
- Utenti IAM:
  - Crea un ruolo che l'utente possa assumere. Per istruzioni, consulta la pagina Creating a role for an IAM user (Creazione di un ruolo per un utente IAM) nella Guida per l'utente di 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 di IAM.

Registrazione della scheda MCU con AWS IoT

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

Una policy AWS IoT

La policy AWS IoT concede al tuo dispositivo le autorizzazioni di accesso alle risorse AWS IoT. Viene archiviata nel cloud AWS.

Un oggetto AWS IoT

Un oggetto AWS IoT ti permette di gestire i tuoi dispositivi su AWS IoT. Viene archiviata nel cloud AWS.

Una chiave privata e un certificato X.509

La chiave privata e il certificato consentono al tuo dispositivo di effettuare l'autenticazione con AWS IoT.

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

Per creare una policy AWS IoT

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 l'AWS CLI, seguire le istruzioni nella Guida per l'utente di AWS Command Line Interface. Dopo aver installato AWS CLI, aprire la finestra del prompt dei comandi e immettere il comando seguente:

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

   L'output dovrebbe essere simile al seguente:

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

   Nell'esempio, la regione è `us-west-2`.

   > **Note**
   >
   > Ti consigliamo di utilizzare gli endpoint ATS come mostrato nell'esempio.

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 regione e l'ID account AWS.

```
{
    "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 per connettersi al broker di messaggi AWS IoT con qualsiasi ID client.

`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 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 oggetti AWS IoT, scegliere Crea un oggetto singolo.
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. Scegliere Attach a policy (Collega una policy) per collegare una policy al certificato che concede al dispositivo l'accesso alle operazioni AWS IoT.
10. Scegliere la policy appena creata e poi Register thing (Registra l'oggetto).

Dopo aver registrato la scheda con AWS IoT, si può passare a Scaricare FreerTOS.

Scaricare FreerTOS

Puoi scaricare FreerTOS dal repository FreerTOS. GitHub

Dopo aver scaricato FreerTOS, puoi continuare a farlo. Configurazione delle demo di 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 l'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 riquadro di navigazione a sinistra scegliere Impostazioni.

   L'endpoint viene visualizzato nell'AWS IoTendpoint Device data. 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).

   Assegnare al dispositivo un nome oggetto AWS IoT. 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` *`Il tipo di sicurezza della rete Wi-Fi`*

  I tipi di sicurezza validi sono:

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

Per formattare le credenziali AWS IoT

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 un oggetto AWS IoT, è possibile recuperare i certificati del dispositivo dalla console AWS IoT, ma non 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 pagina fa riferimento al repository Amazon-FreeRTOS che è obsoleto. Consigliamo di iniziare da qui quando crei un nuovo progetto. Se hai già un progetto FreeRTOS esistente basato sull'ormai obsoleto repository Amazon-FreeRTOS, consulta ilGuida alla migrazione del repository Github di Amazon-FreeRTOS.

Gli argomenti seguenti possono essere utili per risolvere i problemi che possono verificarsi quando si inizia 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 Nosu FreeRTOS guida relativa alla tua scheda.

Suggerimenti generali sulla risoluzione dei problemi iniziali

Non viene visualizzato alcun messaggio nellaAWS IoT console dopo l'esecuzione del 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 mostrati 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	Value (Valore)
BAUD	115200
Dati	8 bit
Parità	nessuno
Stop	1 bit

Impostazione del terminale	Value (Valore)
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 di CMake con FreeRTOS

> **Important**
>
> Questa pagina fa riferimento al repository Amazon-FreeRTOS che è obsoleto. Ti consigliamo di iniziare da qui quando crei un nuovo progetto. Se disponi già di un progetto FreeRTOS esistente basato sull'ormai obsoleto repository Amazon-FreeRTOS, consulta ilGuida alla migrazione del repository Github di Amazon-FreeRTOS.

È possibile utilizzare CMake per generare file di build del progetto dal codice sorgente dell'applicazione FreeRTOS e per creare ed eseguire il codice sorgente.

Puoi anche usare un IDE per modificare, eseguire il debug, compilare, flashare ed eseguire il codice su dispositivi qualificati per FreeRTOS. Ogni guida alle operazioni di base specifiche della scheda include istruzioni per configurare l'IDE per una determinata piattaforma. Se preferisci non utilizzare

un'IDE, puoi usare altri strumenti di modifica e debug del codice di terze parti per lo sviluppo e il debug del codice e quindi utilizzare CMake per compilare ed eseguire le applicazioni.

CMake è supportato dalle seguenti schede:

- EspressifDevKit ESP32-C
- Espressif ESP-WROVER-KIT
- Kit di connessione IoT Infineon XMC4800
- Marvell MW320 AWS IoT Starter Kit
- Marvell MW322 AWS IoT Starter Kit
- Pacchetto Microchip Curiosity PIC32MZEF
- Kit di sviluppo Nordic NRF52840 DK
- STMicroelectronicsSTM32L4 Discovery Kit IoT Node
- Texas Instruments CC3220SF-LAUNCHXL
- Simulatore Microsoft Windows

Consulta gli argomenti seguenti per ulteriori informazioni sull'uso di CMake con FreeRTOS.

Argomenti

Prerequisiti

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

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

  Il sottosistema Windows per Linux (WSL) non è supportato. Utilizza CMake nativo su computer Windows.
- Deve essere stato installato CMake versione 3.13 o successive.

  Puoi scaricare la distribuzione binaria di CMake da CMake.org.

> **Note**
>
> Se scarichi la distribuzione binaria di CMake, assicurati di aggiungere l'eseguibile CMake alla variabile di ambiente PATH prima di utilizzare CMake dalla riga di comando.

Puoi anche scaricare e installare CMake utilizzando un programma di gestione dei pacchetti, ad esempio homebrew su macOS e scoop o chocolatey su Windows.

> **Note**
>
> Le versioni del pacchetto CMake fornite nei gestori di pacchetti per molte distribuzioni Linux sono out-of-date. Se il programma di gestione dei pacchetti della distribuzione non include la versione più recente di CMake, puoi provare a usare programmi di gestione dei pacchetti alternativi, ad esempio `linuxbrew` o `nix`.

- Devi disporre di un sistema di compilazione nativo compatibile.

  CMake può riguardare molti sistemi di compilazione nativi, incluso 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

È possibile 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, è possibile

richiamare lo strumento a riga di comando CMake per compilare il progetto all'interno di VS Code. Per ulteriori informazioni sull'uso di CMake per creare applicazioni FreeRTOS, consultaCostruire 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

Per impostazione predefinita, CMake definisce il sistema operativo host come sistema di destinazione. Per utilizzarlo per la compilazione incrociata, CMake richiede un file toolchain che specifica il compilatore da utilizzare. In FreeRTOS, forniamo i file della toolchain predefiniti in*freertos*`/tools/cmake/toolchains`. Il modo in cui fornire questo file a CMake varia a seconda se utilizzi l'interfaccia della riga di comando o la GUI di CMake. Per ulteriori dettagli, segui le istruzioni Generazione dei file di compilazione (strumento a riga di comando di CMake) di seguito. Per ulteriori informazioni sulla compilazione incrociata in CMake, consulta CrossCompilingil wiki ufficiale di CMake.

Per compilare un progetto basato su CMake

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

   È possibile utilizzare lo strumento a riga di comando di CMake o la GUI CMake per generare i file di compilazione per il sistema di compilazione nativo.

   Per informazioni sulla generazione di file di build FreeRTOS, vedereGenerazione dei file di compilazione (strumento a riga di comando di CMake) eGenerazione dei file di compilazione (GUI CMake).
2. Richiamare il sistema di compilazione nativo per rendere eseguibile il progetto.

Per informazioni sulla creazione di file di build per FreeRTOS, vedereCompilazione di FreeRTOS dai file di build generati.

Generazione dei file di compilazione (strumento a riga di comando di CMake)

Puoi usare lo strumento da riga di comando CMake (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 l'ubicazione 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, la scheda è CC3220 Launchpad e il compilatore è GCC per ARM, puoi avviare il comando seguente per compilare i file di origine 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 il sistema operativo Windows, devi specificare il sistema di compilazione nativo perché CMake utilizza Visual Studio per impostazione predefinita. Ad 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`. Ad esempio:

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

Se il file di toolchain non utilizza percorsi assoluti per il compilatore, e non hai aggiunto il compilatore alla variabile di ambiente `PATH`, CMake potrebbe non essere in grado di individuarlo. Per essere certo che CMake individui il file di toolchain, puoi utilizzare l'opzione `AFR_TOOLCHAIN_PATH`. Questa opzione esegue una ricerca nel percorso di directory della toolchain specificato e nella sottocartella della toolchain in `bin`. Ad 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
```

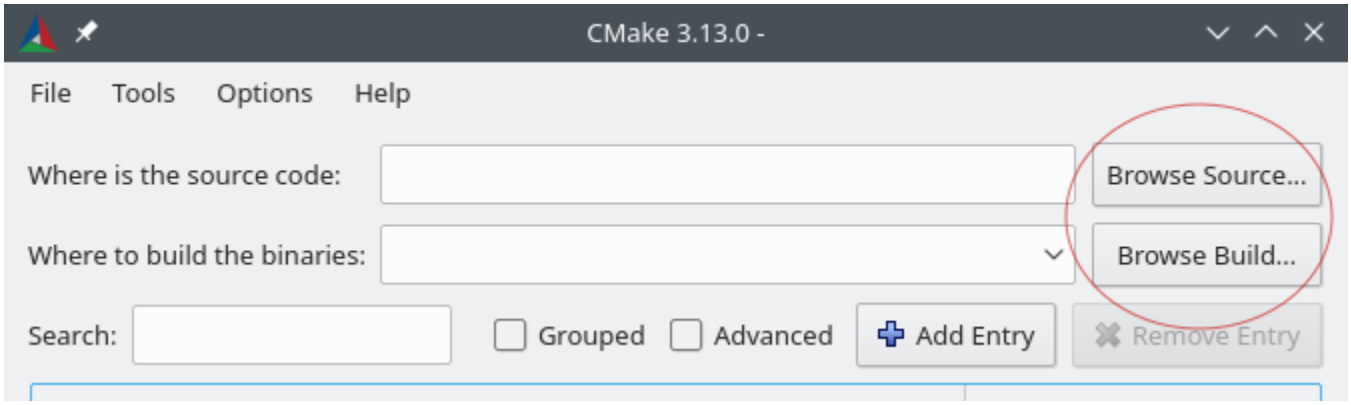
Puoi anche impostare `CMAKE_BUILD_TYPE` su `release` per aggiungere flag di ottimizzazione alle opzioni di compilazione.

Generazione dei file di compilazione (GUI CMake)

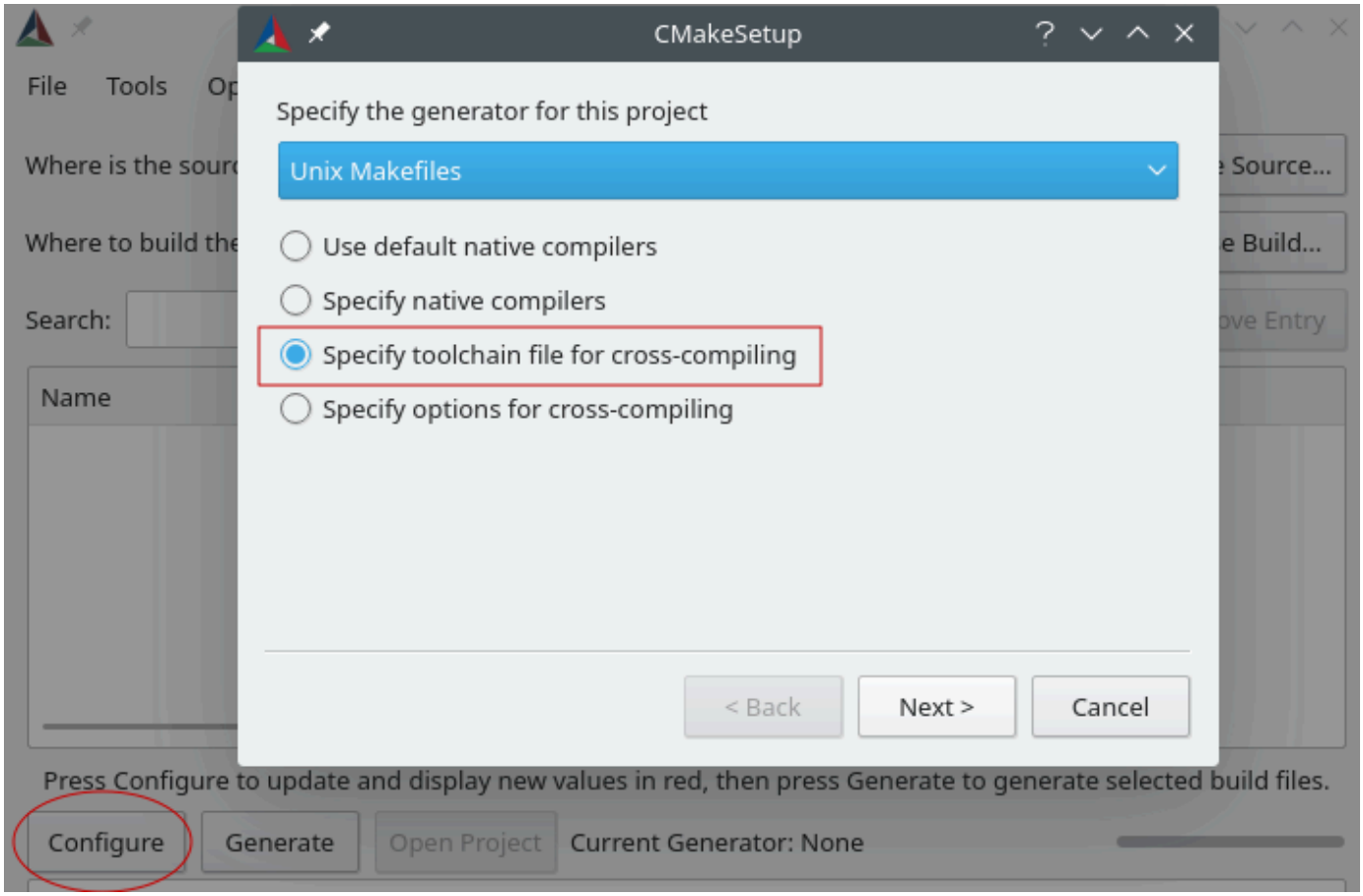
È possibile utilizzare la GUI di CMake per generare file di build FreeRTOS.

Per generare i file di compilazione 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, eliminare la cache CMake scegliendo Delete Cache (Elimina cache) dal menu File.



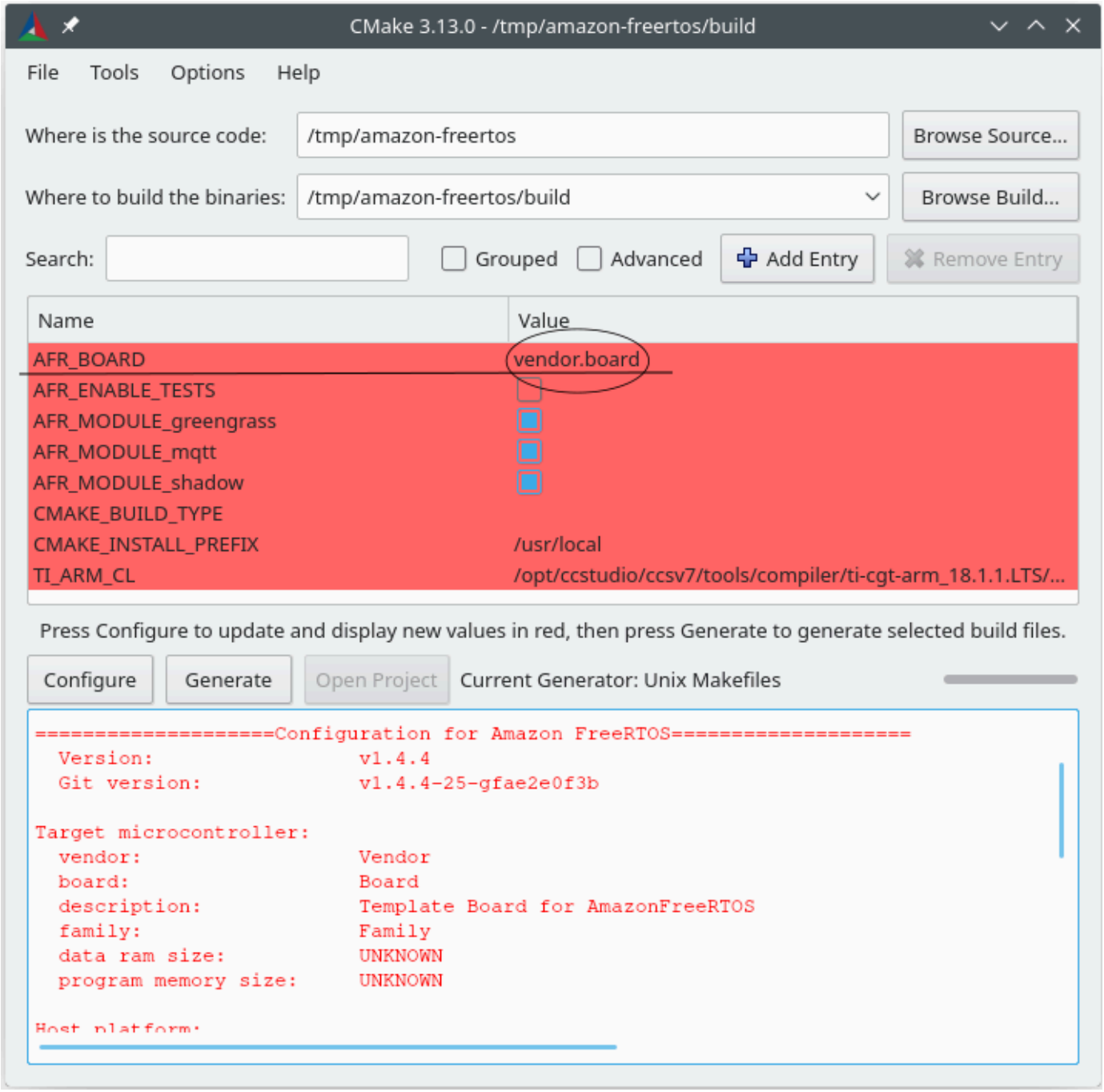
4. Scegliere Specify toolchain file for cross-compiling (Specifica file di toolchain per la compilazione incrociata), quindi selezionare Next (Successivo).
5. Scegliere il file di toolchain (ad esempio, *freertos*`/tools/cmake/toolchains/arm-ti.cmake`), quindi selezionare Fine.

   La configurazione predefinita per FreeRTOS è la scheda modello, che non fornisce alcun target di livello portatile. 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 variabile `AFR_TOOLCHAIN_PATH` nella GUI per indicare a CMake dove è 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. Scegliere Generate (Genera). CMake genera i file di sistema di compilazione (ad esempio, makefile o file ninja) e questi file vengono visualizzati nella directory di compilazione specificata nella prima fase. Segui le istruzioni nella sezione successiva per generare l'immagine binaria.

Compilazione di FreeRTOS dai file di build generati

Compilazione con un sistema di compilazione nativo

Puoi creare FreeRTOS con un sistema di compilazione 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
```

Compilazione di con CMake

Puoi anche usare lo strumento da riga di comando CMake per creare FreeRTOS. CMake fornisce un livello di astrazione per chiamare sistemi di compilazione nativi. Ad esempio:

```
cmake --build build_dir
```

Di seguito sono riportati alcuni utilizzi comuni della modalità di compilazione dello strumento a riga di comando di CMake:

```
# Take advantage of CPU cores.
cmake --build build_dir --parallel 8
```

```
# Build specific targets.
cmake --build build_dir --target afr_kernel
```

```
# Clean first, then build.
cmake --build build_dir --clean-first
```

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

Distribuzione delle chiavi in modalità sviluppatore

> **Important**
>
> Questa pagina fa riferimento al repository Amazon-FreeRTOS che è obsoleto. Per esempio l'output visualizzato sarà simile al testo visualizzato sarà simile al testo visualizzato sarà

simile al testo visualizzato sarà simile al testo visualizzato sarà simile Se disponi già di un progetto FreeRTOS esistente basato sull'ormai obsoleto repository Amazon-FreeRTOS, consulta ilGuida alla migrazione del repository Github di Amazon-FreeRTOS.

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

Innanzitutto, eseguire la procedura Configurazione delle demo di FreerTOS, ma ignorare l'ultima fase (ovvero, non eseguire Per formattare le credenziali AWS IoT). 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 autenticazione reciproca CoreMQTT come descritto nella guida della tua scheda inGuide 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 al dispositivo non sono stati forniti una chiave privata e un certificato client, la demo non riuscirà ad autenticarsiAWS IoT. Tuttavia, la demo di autenticazione reciproca CoreMQTT inizia eseguendo il provisioning delle chiavi in modalità sviluppatore, con conseguente creazione di una chiave privata se non ne era già presente una. L'output visualizzato sarà simile all'output visualizzato sarà simile all'output visualizzato sarà simile all'output visualizzato sarà simile all'output visualizzato sarà simile all'output visualizzato sarà simile all'output visualizzato

```
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 un CSR è stata esclusa dai progetti demo di 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
```

Dopo aver creato e registrato l'autorità di certificazione con AWS IoT, utilizzare il comando seguente per emettere un certificato client basato sul CSR del dispositivo firmato nella fase 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. Sarà inoltre necessario importare il certificato dell'autorità di certificazione (CA), operazione necessaria affinché la prima autenticazione su AWS IoT abbia esito positivo quando si utilizza 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

Importare `deviceCert.pem` nel registro AWS IoT come descritto in Uso di un certificato personale. È necessario creare un nuovo oggetto AWS IoT, allegare il certificato IN SOSPESO e una policy all'oggetto, quindi contrassegnare il certificato come ATTIVO. Tutti queste fasi possono essere eseguite manualmente nella console AWS IoT.

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

Guide alle operazioni di base specifiche per la scheda

> **Important**
>
> Questa pagina fa riferimento al repository Amazon-FreeRTOS che è obsoleto. Ti consigliamo di iniziare a utilizzare. Se disponi già di un progetto FreeRTOS esistente basato sull'ormai obsoleto repository Amazon-FreeRTOS, consulta ilGuida alla migrazione del repository Github di Amazon-FreeRTOS.

Dopo aver completato ilFase iniziale, consulta la guida della tua scheda per istruzioni specifiche su come iniziare a usare FreeRTOS:

- Nozioni di base sul kit di sviluppo Cypress CYW943907AEVAL1F
- Nozioni di base sul kit di sviluppo Cypress CYW954907AEVAL1F
- Guida introduttiva al kit Cypress CY8CKIT-064S0S2-4343W
- Nozioni di base su Infineon XMC4800 IoT Connectivity Kit
- Guida introduttiva alloAWS IoT Starter Kit MW32x
- Guida introduttiva al kit di sviluppo MediaTek MT7697hx
- Nozioni di base su Microchip Curiosity PIC32MZ EF
- Guida introduttiva a Nuvoton NuMaker -IoT-M487
- Nozioni di base su NXP LPC54018 IoT Module
- Nozioni di base su Renesas Starter Kit+ per RX65N-2 MB
- Nozioni di base su STMicroelectronics STM32L4 Discovery Kit IoT Node

- Nozioni di base su Texas Instruments CC3220SF-LAUNCHXL
- Nozioni di base su Windows Device Simulator
- Guida introduttiva al kit IoT MicroZed industriale Xilinx Avnet

> **Note**
>
> Non è necessario completareFase iniziale le seguenti guide introduttive autonome con FreeRTOS:
>
> - Nozioni di base sull'elemento di sicurezza Microchip ATECC608A con simulatore Windows
> - Guida introduttiva all'Espressif ESP32- DevKit C e all'ESP-WROVER-KIT
> - Guida introduttiva all'Espressif ESP32-WROOM-32SE
> - Guida introduttiva all'Espressif ESP32-S2
> - Nozioni di base su Infineon OPTIGA Trust X e XMC4800 IoT Connectivity Kit
> - Nozioni di base su Nordic nRF52840-DK

Nozioni di base sul kit di sviluppo Cypress CYW943907AEVAL1F

> **Important**
>
> Questa integrazione di riferimento è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Ti consigliamo di iniziare da qui quando crei un nuovo progetto. Se hai già un progetto FreeRTOS esistente basato sull'ormai obsoleto repository Amazon-FreeRTOS, consulta ilGuida alla migrazione del repository Github di Amazon-FreeRTOS.

Questo tutorial fornisce istruzioni per iniziare a usare il kit di sviluppo Cypress CYW943907AEVAL1F. Se non disponi del kit di sviluppo Cypress CYW943907AEVAL1F visita l'AWS Partner Device Catalog per acquistarne uno dal nostro partner.

> **Note**
>
> Questo tutorial ti guiderà nella creazione e nella realizzazione della demo di Mutual Authentication CoreMQTT. La porta FreeRTOS per questa scheda attualmente non supporta le demo del server e del client TCP.

Prima di iniziare, devi configurareAWS IoT e scaricare FreeRTOS per connettere il tuo dispositivo alAWS Cloud. 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 della directory di download di FreeRTOS possono causare errori di compilazione.
- Poiché il codice sorgente può contenere collegamenti simbolici, se utilizzi Windows per estrarre l'archivio, potresti dover:
  - Abilita la modalità sviluppatore o,
  - Usa una console con privilegi di amministratore.

  In questo modo, Windows può creare correttamente collegamenti simbolici quando estrae l'archivio. Altrimenti, i collegamenti simbolici verranno scritti come file normali che contengono i percorsi dei collegamenti simbolici come testo o sono vuoti. Per ulteriori informazioni, consulta la voce di blog Symlinks in Windows 10! .

  Se usi Git in Windows, devi abilitare la modalità sviluppatore oppure devi:
  - `core.symlinks`Imposta su true con il seguente comando:

    ```
    git config --global core.symlinks true
    ```
  - Usa una console con privilegi di amministratore ogni volta che usi un comando git che scrive nel sistema (ad esempiogit pull,git clone, egit submodule update --init --recursive).
- 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, usi Cypress WICED Studio SDK 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.

Monitoraggio dei messaggi MQTT in cloud

Prima di eseguire il progetto demo di FreeRTOS, puoi configurare il client MQTT nellaAWS IoT console per monitorare i messaggi che il tuo dispositivo invia alAWS Cloud.

Per effettuare la sottoscrizione all'argomento MQTT con il client AWS IoT

1. Accedi alla console AWS IoT.
2. Nel pannello di navigazione, scegli Test, quindi scegli MQTT test client 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

Dopo aver impostato una connessione seriale alla scheda, è possibile creare il progetto demo FreeRTOS, eseguire il flashing della demo sulla 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.

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 relativi a Getting Started with FreeRTOS, consultaNozioni di base sulla risoluzione dei problemi.

Nozioni di base sul kit di sviluppo Cypress CYW954907AEVAL1F

> **Important**
>
> Questa integrazione di riferimento è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Ti consigliamo di iniziare da qui quando crei un nuovo progetto. Se hai già un progetto FreeRTOS esistente basato sull'ormai obsoleto repository Amazon-FreeRTOS, consulta ilGuida alla migrazione del repository Github di Amazon-FreeRTOS.

Questo tutorial fornisce istruzioni per iniziare a usare il kit di sviluppo Cypress CYW954907AEVAL1F. Se non disponi del kit di sviluppo Cypress CYW954907AEVAL1F, visita l'AWS Partner Device Catalog per acquistarne uno dal nostro partner.

> **Note**
>
> Questo tutorial ti guiderà nella creazione e nella creazione della demo di Mutual Authentication di CoreMQTT. La porta FreeRTOS per questa scheda attualmente non supporta le demo del server e del client TCP.

Prima di iniziare, devi configurareAWS IoT e scaricare FreeRTOS per connettere il tuo dispositivo alAWS Cloud. Per istruzioni, consulta Fase iniziale. In questo tutorial, il percorso della directory di download di FreeRTOS viene chiamato*`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 della directory di download di FreeRTOS possono causare errori di compilazione.
> - Poiché il codice sorgente può contenere collegamenti simbolici, se utilizzi Windows per estrarre l'archivio, potresti dover:

- Abilita la modalità sviluppatore o,
- Usa una console con privilegi di amministratore.

In questo modo, Windows può creare correttamente collegamenti simbolici quando estrae l'archivio. Altrimenti, i collegamenti simbolici verranno scritti come file normali che contengono i percorsi dei collegamenti simbolici come testo o sono vuoti. Per ulteriori informazioni, consulta la voce di blog Symlinks in Windows 10! .

Se usi Git in Windows, devi abilitare la modalità sviluppatore oppure devi:

- `core.symlinks`Imposta su true con il seguente comando:

```
git config --global core.symlinks true
```

- Usa una console con privilegi di amministratore ogni volta che usi un comando git che scrive nel sistema (ad esempiogit pull,git clone, egit submodule update --init --recursive).

- 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, usi Cypress WICED Studio SDK 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.

Monitoraggio dei messaggi MQTT in cloud

Prima di eseguire il progetto demo di FreeRTOS, puoi configurare il client MQTT nellaAWS IoT console per monitorare i messaggi che il tuo dispositivo invia alAWS Cloud.

Per effettuare la sottoscrizione all'argomento MQTT con il client AWS IoT

1. Accedi alla console AWS IoT.
2. Nel pannello di navigazione, scegli Test, quindi scegli MQTT test client 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

Dopo aver impostato una connessione seriale alla scheda, è possibile creare il progetto demo FreeRTOS, eseguire il flashing della demo sulla 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.

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 relativi a Getting Started with FreeRTOS, consultaNozioni di base sulla risoluzione dei problemi.

Guida introduttiva al kit Cypress CY8CKIT-064S0S2-4343W

> **Important**
>
> Questa integrazione di riferimento è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Ti consigliamoinizia da quidurante la creazione di un nuovo progetto. Se hai già un progetto FreeRTOS esistente basato sull'ormai obsoleto repository Amazon-FreeRTOS, consulta ilGuida alla migrazione del repository Github di Amazon-FreeRTOS.

Questo tutorial fornisce istruzioni per iniziare a usareCY8CKIT-064S0S2-4343Wkit. Se non disponi già dell'account, ne richiederemo la creazione durante la registrazione. Puoi anche utilizzare quel link per accedere alla guida per l'utente del kit.

Nozioni di base

Prima di iniziare, ne richiederemo la configurazioneAWS IoTe FreerTOS per connettere il tuo dispositivo alAWSNuvola. Per istruzioni, consulta Fase iniziale. Dopo aver completato i prerequisiti, avrai un pacchetto FreerTOS conAWS IoT Corecredenziali.

> **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 con un flusso di build CMake o Make. Puoi usare ModusToolbox per il tuo Make build flow. È possibile utilizzare l'IDE Eclipse fornito con ModusToolbox o un IDE partner come IAR EW-Arm, Arm MDK o Microsoft Visual Studio Code. Eclipse IDE è compatibile con i sistemi operativi Windows, macOS e Linux.

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

Strumenti di aggiornamento per ModusToolbox 2.1 o precedente

Se si dispone di un account ModusToolbox 2.1 Eclipse IDE per programmare questo kit, è necessario aggiornare gli strumenti OpenOCD e Firmware-Loader.

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

- 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 allaPagina di rilascio di Cypress OpenOCD.

2. Scarica il file di archivio per il tuo sistema operativo (Windows/Mac/Linux).
3. Elimina i file esistenti in*ModusToolbox*`/tools_2.x/openocd`.
4. Sostituisci i file in*ModusToolbox*`/tools_2.x/openocd`con i contenuti estratti dell'archivio che è stata scaricata nel 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 allaPagina di rilascio di Cypress Firmware-Loader.
2. Scarica il file di archivio per il tuo sistema operativo (Windows/Mac/Linux).
3. Elimina i file esistenti in*ModusToolbox*`/tools_2.x/fw-loader`.
4. Sostituisci i file in*ModusToolbox*`/tools_2.x/fw-loader`con i contenuti estratti dell'archivio che è stata scaricata nel passaggio precedente.

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

Configurazione dell'hardware

Segui questi passaggi per configurare l'hardware del kit.

1. Esegui il provisioning del kit

   Segui ilGuida alla fornitura per il kit CY8CKIT-064S0S2-4343Wistruzioni per fornire in modo sicuro il kit perAWS IoT.

   Questo kit richiede CySecureTools 3.1.0 o versione successiva.
2. Configura 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 utilizzandoGestione dei dispositivisottoporti(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 la proceduraGuida alla fornitura per il kit CY8CKIT-064S0S2-4343W.
2. Crea la demo di FreerTOS.

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

   b. Dal menu File scegliere Import (Importa).

      EspandiGenerale, scegliProgetto esistente in Workspace, e poi scegliAvanti.

   c. NelDirectory principale, immettere*freertos*`/projects/cypress/CY8CKIT-064S0S2-4343W/mtb/aws_demos`e quindi seleziona il nome del progetto`aws_demos`. Per impostazione predefinita, ne richiederemo la selezione.

   d. ScegliFinireper importare il progetto nel tuo spazio di lavoro.

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

      - DalPannello rapido, selezionaCrea un'applicazione aws_demos.
      - ScegliProgettoe scegliCostruisci tutto.

      Assicurati che il progetto venga compilato senza errori.

3. Monitoraggio dei messaggi MQTT nel cloud

   Prima di eseguire la demo, è possibile configurare il client MQTT nelAWS IoTconsole per monitorare i messaggi che il dispositivo invia alAWSNuvola. Per abbonarsi all'argomento MQTT conAWS IoTClient MQTT, segui questi passaggi.

a. Accedi alla console AWS IoT.

b. Nel riquadro di navigazione, scegliTest, quindi scegliClient di test MQTTper aprire il client MQTT.

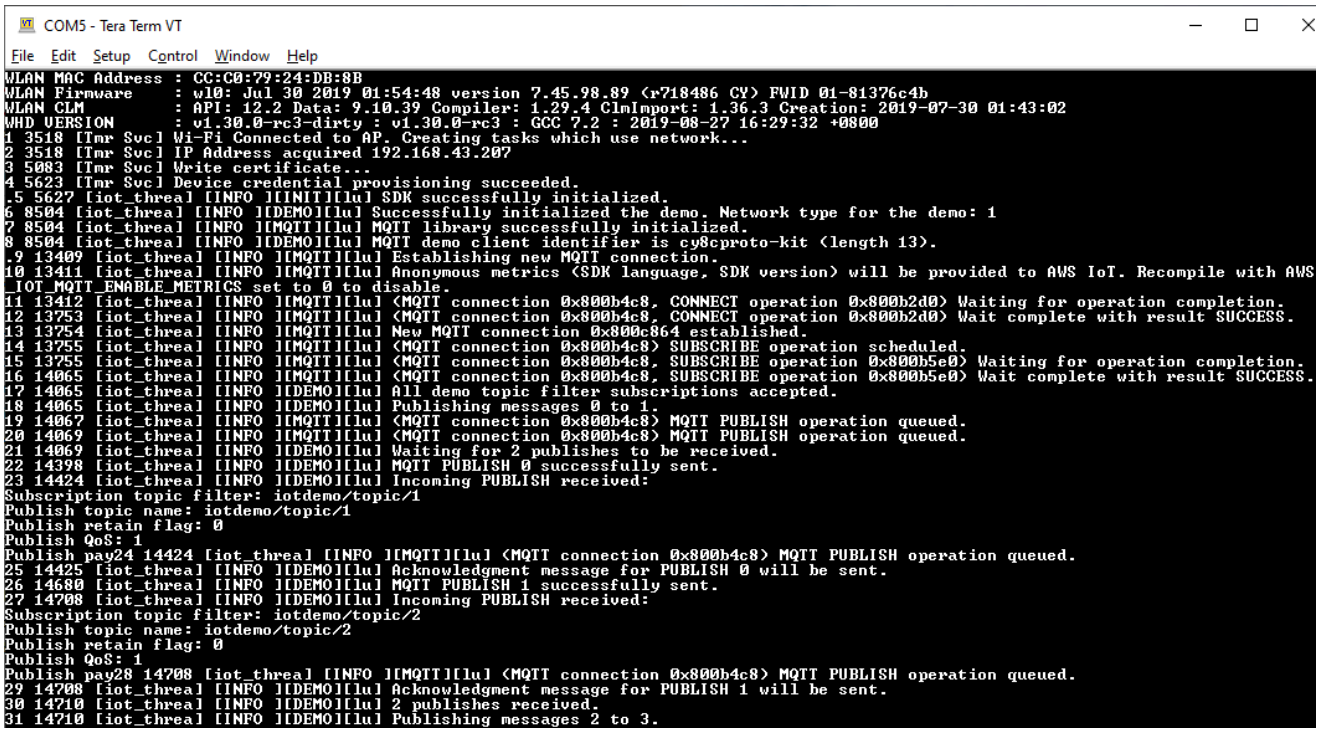
c. PerArgomento dell'account, inserisci***your-thing-name*****/example/topic**, e poi scegliIscriviti all'argomento.

4. Esegui il progetto demo FreerTOS

a. Seleziona il progettoaws_demosnell'area di lavoro.

b. DalPannello rapido, selezionaProgramma aws_demos (KitProg3). 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 si iscrive a 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 collegandosi alAWSserver che utilizza 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.

5. Usare CMake con FreerTOS

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

a. Utilizzate il seguente comando per generare i file di build. Specificate la scheda di destinazione con`-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`-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 `arm-none-eabi-gcc` non è incluso nel percorso della shell, è inoltre necessario impostare la variabile CMake `AFR_TOOLCHAIN_PATH`.

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 il`cm0.hex`e`cm4.hex`file generati con*`build_dir`*utilizzando Cypress Programmer.

Esecuzione di altre demo

Le seguenti applicazioni demo sono state testate e verificate per funzionare con la versione corrente. Puoi trovare queste demo sotto*`freertos`*`/demos`rubrica. Per informazioni su come eseguire queste demo, consultaDemo FreeRTOS FreeRS.

- Demo Bluetooth Low Energy
- Demo degli aggiornamenti via etere
- Dimostrazione del client Secure Sockets Echo
- AWS IoTDemo di Device Shadow

Debug

Il KitProg3 del kit supportano il debug tramite il protocollo SWD.

- Per eseguire il debug dell'applicazione FreerTOS, selezionaprogetto aws_demosnell'area di lavoro e quindi selezionaaws_demos Debug (KitProg3)dalPannello rapido.

Aggiornamenti OTA

Gli MCU PSoC 64 hanno superato tutti i test di qualificazione FreerTOS richiesti. Tuttavia, l'opzionale over-the-airFunzionalità (OTA) implementata nel PSoC 64 Standard SecureAWSla libreria del firmware è ancora in attesa di valutazione. La funzionalità OTA così come implementata attualmente supera tutti i test di qualificazione OTA tranneaws_ota_test_case_rollback_if_unable_to_connect_after_update.py.

Quando un'immagine OTA convalidata con successo viene applicata a un dispositivo utilizzando PSOC64 Standard Secure —AWSNon si dispone di un account, ne richiederemo la comunicazioneAWS IoT Core, il dispositivo non può tornare automaticamente alla buona immagine originale nota. Ciò potrebbe rendere il dispositivo non raggiungibile daAWS IoT Coreper ulteriori aggiornamenti. Questa funzionalità è ancora in fase di sviluppo da parte del team di Cypress.

Per ulteriori informazioni, consulta la paginaAggiornamenti OTA conAWSe il kit CY8CKIT-064S0S2-4343W. Se avete ulteriori domande o avete bisogno di assistenza tecnica, contattate ilComunità di sviluppatori di Cypress.

Nozioni di base sull'elemento di sicurezza Microchip ATECC608A con simulatore Windows

> **Important**
>
> Questa integrazione di riferimento è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Ti consigliamo di iniziare da qui quando crei un nuovo progetto. Se disponi già di un progetto FreeRTOS esistente basato sull'ormai obsoleto repository Amazon-FreerTOS, consulta il. Guida alla migrazione del repository Github di Amazon-FreeRTOS

In questo tutorial vengono fornite istruzioni per iniziare a utilizzare l'elemento di sicurezza Microchip ATECC608A con Simulatore Windows

È necessario il seguente hardware:

- Clickboard per l'elemento di sicurezza Microchip ATECC608A
- SAMD21 XPlained Pro
- Adattatore mikroBUS Xplained Pro

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 viene chiamato 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 ATECC608A

Prima di poter interagire con il dispositivo Microchip ATECC608A, è necessario innanzitutto programmare il SAMD21.

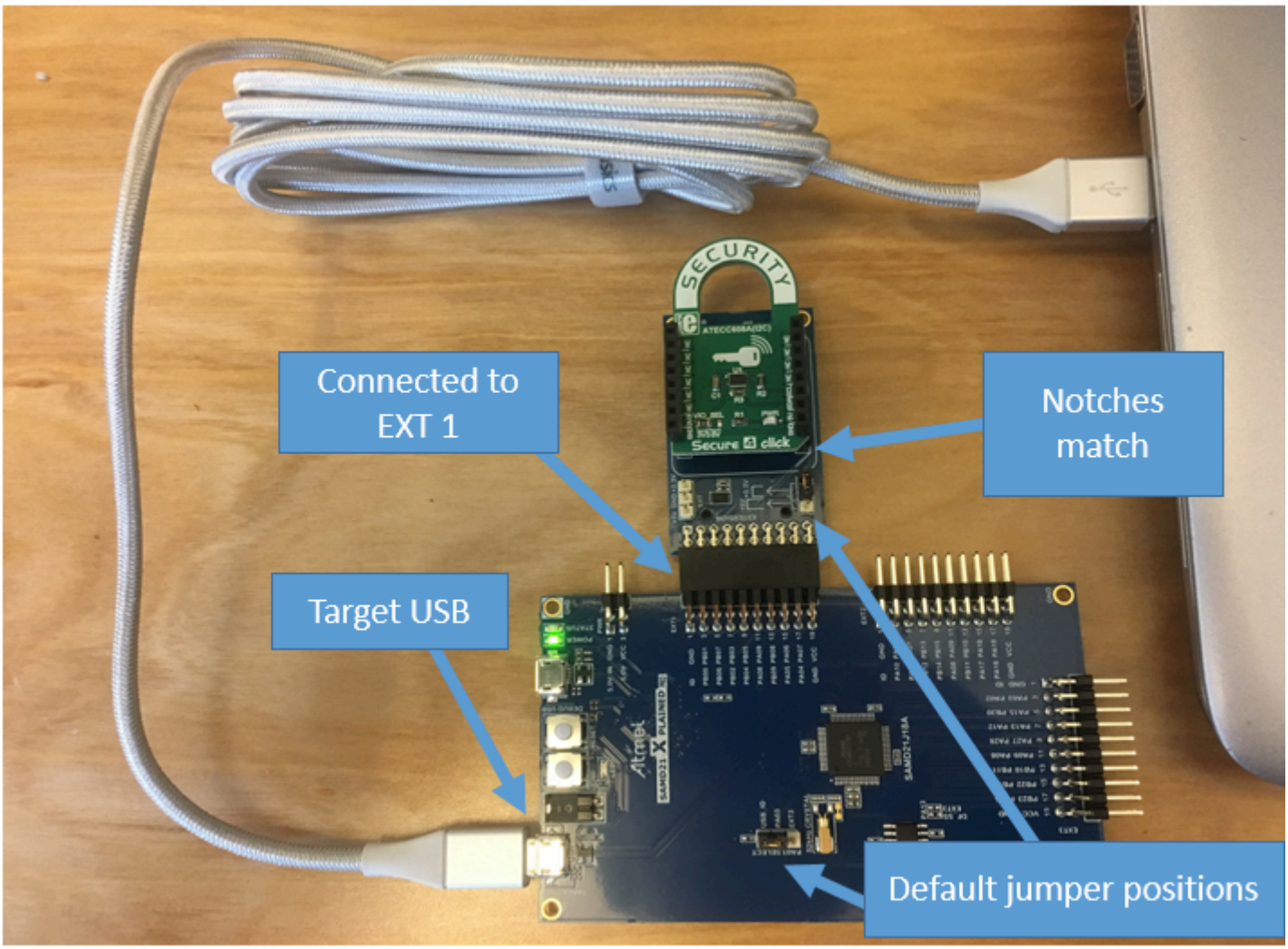
Per configurare la scheda SAMD21 XPlained Pro

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 sul 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. Collegare l'adattatore mikroBUS XPlained Pro alla scheda SAMD21 nella posizione EXT1.
3. Collegare la scheda ATECC608A Secure 4 Click all'adattatore mikroBUSX XPlained Pro. 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

Registrarsi per creare un Account AWS

Se non disponi di un Account AWS, completa la procedura seguente per crearne uno.

Per registrarsi a un Account AWS

1. Apri la pagina all'indirizzo https://portal.aws.amazon.com/billing/signup.
2. Segui le istruzioni online.

   Nel corso della procedura di registrazione riceverai una telefonata, durante la quale sarà necessario inserire un codice di verifica attraverso la tastiera del telefono.

   Durante la registrazione di un Account AWS, viene creato un Utente root dell'account AWS. L'utente root dispone dell'accesso a tutte le risorse e tutti i Servizi AWSnell'account. Come best

practice di sicurezza, assegna l'accesso amministrativo a un utente amministrativo e utilizza solo l'utente root per eseguire attività che richiedono l'accesso di un utente root.

Al termine del processo di registrazione, riceverai un'e-mail di conferma da AWS. È possibile visualizzare l'attività corrente dell'account e gestire l'account in qualsiasi momento accedendo all'indirizzo https://aws.amazon.com/ e selezionando Il mio account.

Creazione di un utente amministratore

Dopo aver effettuato la registrazione di un Account AWS, proteggi Utente root dell'account AWS, abilita AWS IAM Identity Center e crea un utente amministratore in modo da non utilizzare l'utente root per le attività quotidiane.

Protezione dell'Utente root dell'account AWS

1. Accedi alla AWS Management Console come proprietario dell'account scegliendo Utente root e immettendo l'indirizzo email del Account AWS. Nella pagina successiva, inserisci la password.

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

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

Creazione di un utente amministratore

1. Abilita IAM Identity Center

   Per istruzioni, consulta Abilitazione di AWS IAM Identity Center nella Guida per l'utente di AWS IAM Identity Center.
2. In Centro identità AWS IAM, assegna l'accesso amministrativo a un utente amministrativo.

   Per un tutorial sull'utilizzo di IAM Identity Center directory come origine di identità, consulta Configure user access with the default IAM Identity Center directory nella Guida per l'utente di AWS IAM Identity Center.

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 Accedere al portale di accesso AWS nella Guida per l'utente Accedi ad AWS.

Per fornire l'accesso, aggiungi autorizzazioni ai tuoi 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 Creating a role for a third-party identity provider (federation) (Creazione di un ruolo per un provider di identità di terze parti [federazione]) nella Guida per l'utente di IAM.
- Utenti IAM:
  - Crea un ruolo che l'utente possa assumere. Per istruzioni, consulta la pagina Creating a role for an IAM user (Creazione di un ruolo per un utente IAM) nella Guida per l'utente di 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 di IAM.

Configurazione

1. Scarica il repository FreerTOS dal repository FreerTOS. GitHub

   Per scaricare FreerTOS da: GitHub

   1. Vai al repository FreerTOS GitHub .
   2. Selezionare Clone or download (Clona o scarica).
   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.
> - Poiché il codice sorgente può contenere collegamenti simbolici, se utilizzi Windows per estrarre l'archivio, potresti dover:
>   - Attivare la modalità sviluppatore o,
>   - Utilizza una console con privilegi elevati di amministratore.
>
>   In questo modo, Windows può creare correttamente collegamenti simbolici quando estrae l'archivio. In caso contrario, i collegamenti simbolici verranno scritti come normali file che contengono i percorsi dei collegamenti simbolici sotto forma di testo o sono vuoti. Per ulteriori informazioni, consultate il post di blog Symlinks in Windows 10! .
>
>   Se usi Git in Windows, devi abilitare la modalità sviluppatore oppure devi:
>   - `core.symlinks`Imposta su true con il seguente comando:
>
>     ```
>     git config -\-global core.symlinks true
>     ```
>   - Usa una console con privilegi di amministratore ogni volta che usi un comando git che scrive sul sistema (ad esempio, git pullgit clone, egit submodule update -\-init -\-recursive).

4. Dalla directory *`freertos`*, selezionare la ramificazione da utilizzare.

2. Configurazione dell'ambiente di sviluppo.

   a. Installare la versione più recente di WinPCap.

   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 ATECC608A viene inizializzato una sola volta che viene bloccata sul dispositivo alla prima esecuzione di 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. Effettuare il provisioning del Microchip ATECC608A.

   Microchip ha diversi strumenti di scripting per facilitare la configurazione delle parti ATECC608A. 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. Generare le chiavi sul Microchip ATECC608A ed esportare 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 di 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 all'ESP-WROVER-KIT

> **Important**
>
> Questa integrazione di riferimento è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Ti consigliamo di iniziare da qui quando crei un nuovo progetto. Se disponi già di un progetto FreeRTOS esistente basato sull'ormai obsoleto repository Amazon-FreerTOS, consulta il. Guida alla migrazione del repository Github di Amazon-FreeRTOS

> **Note**
>
> Per scoprire come integrare le librerie e le demo modulari FreerTOS all'interno del tuo progetto Espressif IDF, consulta la nostra integrazione di riferimento in primo piano per la piattaforma ESP32-C3.

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

- ESP32-WROOM-32 C DevKit
- ESP32-SOLO-1
- KIT ESP32-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-KIT v4.1 sul sito Web di Espressif.

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

Panoramica

Questo tutorial descrive le seguenti procedure:

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.

Prerequisiti

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

Registrarsi per creare un Account AWS

Se non disponi di un Account AWS, completa la procedura seguente per crearne uno.

Per registrarsi a un Account AWS

1. Apri la pagina all'indirizzo https://portal.aws.amazon.com/billing/signup.
2. Segui le istruzioni online.

   Nel corso della procedura di registrazione riceverai una telefonata, durante la quale sarà necessario inserire un codice di verifica attraverso la tastiera del telefono.

   Durante la registrazione di un Account AWS, viene creato un Utente root dell'account AWS. L'utente root dispone dell'accesso a tutte le risorse e tutti i Servizi AWSnell'account. Come best practice di sicurezza, assegna l'accesso amministrativo a un utente amministrativo e utilizza solo l'utente root per eseguire attività che richiedono l'accesso di un utente root.

Al termine del processo di registrazione, riceverai un'e-mail di conferma da AWS. È possibile visualizzare l'attività corrente dell'account e gestire l'account in qualsiasi momento accedendo all'indirizzo https://aws.amazon.com/ e selezionando Il mio account.

Creazione di un utente amministratore

Dopo aver effettuato la registrazione di un Account AWS, proteggi Utente root dell'account AWS, abilita AWS IAM Identity Center e crea un utente amministratore in modo da non utilizzare l'utente root per le attività quotidiane.

Protezione dell'Utente root dell'account AWS

1. Accedi alla AWS Management Console come proprietario dell'account scegliendo Utente root e immettendo l'indirizzo email del Account AWS. Nella pagina successiva, inserisci la password.

Per informazioni sull'accesso utilizzando un utente root, consulta la pagina Accesso come utente root della Guida per l'utente di Accedi ad AWS.

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

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

Creazione di un utente amministratore

1. Abilita IAM Identity Center

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

2. In Centro identità AWS IAM, assegna l'accesso amministrativo a un utente amministrativo.

   Per un tutorial sull'utilizzo di IAM Identity Center directory come origine di identità, consulta Configure user access with the default IAM Identity Center directory nella Guida per l'utente di AWS IAM Identity Center.

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 Accedere al portale di accesso AWS nella Guida per l'utente Accedi ad AWS.

Per fornire l'accesso, aggiungi autorizzazioni ai tuoi 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 Creating a role for a third-party identity provider (federation) (Creazione di un ruolo per un provider di identità di terze parti [federazione]) nella Guida per l'utente di IAM.

- Utenti IAM:
  - Crea un ruolo che l'utente possa assumere. Per istruzioni, consulta la pagina Creating a role for an IAM user (Creazione di un ruolo per un utente IAM) nella Guida per l'utente di 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 di IAM.

Inizia a usare

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

1. Configura l'hardware Espressif.

   - Per informazioni sulla configurazione dell'hardware della scheda di sviluppo ESP32- DevKit C, consultare la Guida introduttiva a ESP32- DevKit C V4.
   - Per informazioni sulla configurazione dell'hardware della scheda di sviluppo ESP-WROVER-KIT, consultare la Guida introduttiva di ESP-WROVER-KIT V4.1.

   > **Important**
   >
   > Quando raggiungi la sezione Guida introduttiva delle guide Espressif, fermati e torna alle istruzioni in questa pagina.

2. Scarica Amazon GitHubFreerTOS da. (Per istruzioni, consulta il file README.md).
3. Configura il tuo ambiente di sviluppo.

   Per comunicare con la scheda, è necessario installare una toolchain. Espressif fornisce l'ESP-IDF per sviluppare software per le proprie schede. Poiché l'ESP-IDF ha una propria versione del kernel FreeRTOS integrata come componente, Amazon FreerTOS include una versione personalizzata di ESP-IDF v4.2 con il kernel FreerTOS rimosso. Questo risolve i problemi relativi

ai file duplicati durante la compilazione. Per utilizzare la versione personalizzata di ESP-IDF v4.2 inclusa in Amazon FreerTOS, segui le istruzioni riportate di seguito per il sistema operativo della tua macchina host.

Windows

1. Scarica l'Universal Online Installer di ESP-IDF per Windows.
2. Esegui l'Universal Online Installer.
3. Quando arrivi alla fase Scarica o usa ESP-IDF, seleziona Usa una directory ESP-IDF esistente e imposta Scegli la directory ESP-IDF esistente su. *freertos*/vendors/espressif/esp-idf
4. Completa l'installazione.

macOS

1. Segui le istruzioni contenute nella Configurazione standard dei prerequisiti della Toolchain (ESP-IDF v4.2) per macOS.

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

2. Aprire una finestra a riga di comando.
3. 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
   ```

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

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

Linux

1. Segui le istruzioni contenute nella 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.

2. Aprire una finestra a riga di comando.
3. 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
   ```

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

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

4. Stabilisci una connessione seriale.

   a. Per stabilire una connessione seriale tra la macchina host e l'ESP32- DevKit C, è necessario installare i driver CP210x USB to UART Bridge VCP. È possibile scaricare i driver da Silicon Labs.

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

   b. Segui i passaggi per stabilire una connessione seriale con ESP32.

   c. Dopo aver stabilito una connessione seriale, annotare la porta seriale per la connessione della scheda. Ne hai bisogno per eseguire il flashing della demo.

Configura le applicazioni demo FreerTOS

Per questo tutorial, il file di configurazione di FreerTOS si trova in. *freertos*`/vendors/espressif/boards/`*board-name*`/aws_demos/config_files/FreeRTOSConfig.h` (Ad esempio, se `AFR_BOARD espressif.esp32_devkitc` viene scelto, il file di configurazione

si trova in*freertos*/vendors/espressif/boards/esp32/aws_demos/config_files/FreeRTOSConfig.h.)

1. Se utilizzi macOS o Linux, apri un prompt del terminale. Se utilizzi Windows, apri l'app «ESP-IDF 4.x CMD» (se hai incluso questa opzione quando hai installato la toolchain ESP-IDF), altrimenti l'app «Command Prompt».
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 necessaria l'interfaccia CLI (AWSCommand Line Interface) per eseguire AWS IoT i comandi. Se utilizzi Windows, usa il `easy_install awscli` comando per installare la AWS CLI nell'app «Command» o «ESP-IDF 4.x CMD».

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

   ```
   aws configure
   ```

   e configura la AWS CLI con l'ID della chiave di AWS accesso, la chiave di accesso segreta e la regione predefinitaAWS. Per ulteriori informazioni, consulta Configurazione di AWS CLI.
5. Usa il seguente comando per installare l'AWSSDK per Python (boto3):

   - Su Windows, nell'app «Command» o «ESP-IDF 4.x CMD», esegui

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

     > **Note**
     >
     > Consulta la documentazione di Boto3 per i dettagli.

   - 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 si desidera assegnare all'elemento AWS IoT che rappresenta la 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` (Sicurezza WPA2)

6. Esegui lo script di configurazione.

   a. Se utilizzi macOS o Linux, apri un prompt del terminale. Se utilizzi Windows, apri l'app «ESP-IDF 4.x CMD» o «Command».

   b. Vai alla directory ed esegui *`freertos`*`/tools/aws_config_quick_start`

```
python SetupAWS.py setup
```

Lo script svolge le seguenti funzioni:

- Crea un oggetto, un certificato e una policy IoT.
- Allega la policy IoT al certificato e il certificato all'AWS IoToggetto.
- Compila il `aws_clientcredential.h` file con l'AWS IoTendpoint, 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`

Monitoraggio dei messaggi MQTT in cloud

Prima di eseguire il progetto demo FreerTOS, puoi configurare il client MQTT nella console per monitorare AWS IoT i messaggi che il tuo dispositivo invia al Cloud. AWS

Per effettuare la sottoscrizione all'argomento MQTT con il client AWS IoT

1. Passare alla console AWS IoT.
2. Nel pannello di navigazione, scegli Test, quindi scegli MQTT Test Client.
3. In Argomento sottoscrizione, digitare *`your-thing-name`*`/example/topic`, quindi scegliere Effettua sottoscrizione all'argomento.

Quando il progetto demo viene eseguito correttamente sul dispositivo, viene visualizzato «Hello World!» inviato più volte all'argomento a cui ti sei iscritto.

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

Puoi usare l'utilità IDF di Espressif (`idf.py`) per creare il progetto e eseguire 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
> ```

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

1. Vai alla radice della tua directory 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 (app «Comando»)

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

   Windows (app «ESP-IDF 4.x CMD»)

   (Questa operazione è già stata eseguita quando hai aperto l'app.)

   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
> ```

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. Compilare il progetto.

   Con Unix Makefiles

   ```
   make -C ./YOUR_BUILD_DIRECTORY -j8
   ```

Con Ninja

```
ninja -C ./YOUR_BUILD_DIRECTORY -j8
```

5. 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
   ```

Esecuzione delle demo Bluetooth Low-Energy

Libreria Bluetooth Low EnergyFreerTOS supporta la connettività.

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. Seguire le istruzioni in SDK mobili 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.

Per istruzioni su come eseguire la demo MQTT tramite Bluetooth Low Energy sulla scheda, vedere. MQTT su Bluetooth Low Energy

Per istruzioni su come eseguire la demo del provisioning Wi-Fi sulla scheda, vedere. Provisioning Wi-Fi

Usare FreerTOS nel proprio progetto CMake per ESP32

Se vuoi utilizzare FreerTOS nel tuo progetto CMake, 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.

```
cmake_minimum_required(VERSION 3.13)

project(freertos_examples)

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

```
# Add FreeRTOS as a subdirectory. AFR_BOARD tells which board to target.
set(AFR_BOARD espressif.esp32_devkitc CACHE INTERNAL "")
add_subdirectory(freertos)

# Link against the mqtt library so that we can use it. Dependencies are transitively
# linked.
target_link_libraries(my_app PRIVATE AFR::core_mqtt)
```

Per creare il progetto, eseguire i seguenti comandi CMake. Assicurarsi che il compilatore ESP32 si trovi nella variabile di ambiente PATH.

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

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

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

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

Utilizzo di componenti di FreerTOS

Dopo aver eseguito CMake, è possibile trovare tutti i componenti disponibili nell'output di riepilogo. Dovrebbe assomigliare all'esempio seguente.

```
====================Configuration for FreeRTOS====================
  Version:                 202107.00
  Git version:             202107.00-g79ad6defb

Target microcontroller:
  vendor:                  Espressif
  board:                   ESP32-DevKitC
  description:             Development board produced by Espressif that comes in two
                           variants either with ESP-WROOM-32 or ESP32-WROVER module
  family:                  ESP32
  data ram size:           520KB
  program memory size:     4MB

Host platform:
  OS:                      Linux-4.15.0-66-generic
  Toolchain:               xtensa-esp32
  Toolchain path:          /opt/xtensa-esp32-elf
```

```
  CMake generator:          Ninja

FreeRTOS modules:
  Modules to build:         backoff_algorithm, common, common_io, core_http,
                            core_http_demo_dependencies, core_json, core_mqtt,
                            core_mqtt_agent, core_mqtt_agent_demo_dependencies,
                            core_mqtt_demo_dependencies, crypto, defender, dev_mode_key_
                            provisioning, device_defender, device_defender_demo_
                            dependencies, device_shadow,
 device_shadow_demo_dependencies,
                            freertos_cli_plus_uart, freertos_plus_cli, greengrass,
                            http_demo_helpers, https, jobs, jobs_demo_dependencies,
                            kernel, logging, mqtt, mqtt_agent_interface, mqtt_demo_
                            helpers, mqtt_subscription_manager, ota, ota_demo_
                            dependencies, ota_demo_version, pkcs11, pkcs11_helpers,
                            pkcs11_implementation, pkcs11_utils, platform,
 secure_sockets,
                            serializer, shadow, tls, transport_interface_secure_sockets,
                            wifi
  Enabled by user:          common_io, core_http_demo_dependencies, core_json,
                            core_mqtt_agent_demo_dependencies, core_mqtt_demo_
                            dependencies, defender, device_defender,
 device_defender_demo_
                            dependencies, device_shadow,
 device_shadow_demo_dependencies,
                            freertos_cli_plus_uart, freertos_plus_cli, greengrass,
 https,
                            jobs, jobs_demo_dependencies, logging,
 ota_demo_dependencies,
                            pkcs11, pkcs11_helpers, pkcs11_implementation, pkcs11_utils,
                            platform, secure_sockets, shadow, wifi
  Enabled by dependency:    backoff_algorithm, common, core_http, core_mqtt,
                            core_mqtt_agent, crypto, demo_base,
 dev_mode_key_provisioning,
                            freertos, http_demo_helpers, kernel, mqtt, mqtt_agent_
                            interface, mqtt_demo_helpers, mqtt_subscription_manager,
 ota,
                            ota_demo_version, pkcs11_mbedtls, serializer, tls,
                            transport_interface_secure_sockets, utils
  3rdparty dependencies:    jsmn, mbedtls, pkcs11, tinycbor
  Available demos:          demo_cli_uart, demo_core_http, demo_core_mqtt,
 demo_core_mqtt_
                            agent, demo_device_defender, demo_device_shadow,
                            demo_greengrass_connectivity, demo_jobs, demo_ota_core_http,
```

```
                            demo_ota_core_mqtt, demo_tcp
  Available tests:
================================================================================
```

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::core_mqttAFR::ota`, e così via.

Aggiungi componenti personalizzati utilizzando ESP-IDF

È possibile aggiungere altri componenti durante l'utilizzo di ESP-IDF. Ad esempio, supponendo che tu voglia aggiungere un componente denominato `example_component` e il tuo progetto sia simile a questo:

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

Di seguito è riportato un esempio del `CMakeLists.txt` file per il componente.

```
add_library(example_component src/example_component.c)
target_include_directories(example_component PUBLIC include)
```

Quindi, nel `CMakeLists.txt` file di primo livello, aggiungete il componente inserendo la riga seguente subito dopo`add_subdirectory(freertos)`.

```
add_subdirectory(component/example_component)
```

Quindi, modificate `target_link_libraries` per includere il componente.

```
target_link_libraries(my_app PRIVATE AFR::core_mqtt PRIVATE example_component)
```

Questo componente è ora automaticamente collegato al codice dell'applicazione per impostazione predefinita. Ora puoi includere i suoi file di intestazione e richiamare 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 *freertos*/vendors/espressif/boards/esp32/aws_demos/config_files/ e *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 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 il tuo file sdkconfig.default, puoi impostare la variabile CMake 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 utilizza il file predefinito che si trova in. *freertos*/vendors/espressif/boards/esp32/aws_demos/sdkconfig.defaults

Per maggiori informazioni, consulta la sezione Configurazione del progetto nell'Espressif API Reference e, se riscontri problemi dopo la compilazione con successo, consulta la sezione sulle opzioni obsolete e le loro sostituzioni in quella pagina.

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.

```
cmake_minimum_required(VERSION 3.13)

project(freertos_examples)

set(IDF_PROJECT_EXECUTABLE my_app)
set(IDF_EXECUTABLE_SRCS "src/main.c")

# Tell IDF build to link against this target.
set(IDF_PROJECT_EXECUTABLE my_app)

# Add some extra components. IDF_EXTRA_COMPONENT_DIRS is a variable used by ESP-IDF
# to collect extra components.
get_filename_component(
    EXTRA_COMPONENT_DIRS
    "components/example_component" ABSOLUTE
)
list(APPEND IDF_EXTRA_COMPONENT_DIRS ${EXTRA_COMPONENT_DIRS})

# Override the configurations for FreeRTOS.
include_directories(BEFORE freertos-configs)

# Add FreeRTOS as a subdirectory. AFR_BOARD tells which board to target.
set(AFR_BOARD espressif.esp32_devkitc CACHE INTERNAL "")
add_subdirectory(freertos)

# Link against the mqtt library so that we can use it. Dependencies are transitively
# linked.
target_link_libraries(my_app PRIVATE AFR::core_mqtt)
```

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'utilità di monitoraggio fornita da ESP-IDF (e richiamata utilizzando make monitor) ti aiuta a 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 per la comunicazione con gdb senza richiedere alcun hardware JTAG speciale. Per ulteriori 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. Per ridurre il tempo di compilazione, puoi usare l'`-j4`interruttore sul comando make.

```
make flash monitor -j4
```

- Se il dispositivo presenta problemi nella connessione a AWS IoT, aprire il file `aws_clientcredential.h` e verificare che le variabili di configurazione siano definite correttamente. `clientcredentialMQTT_BROKER_ENDPOINT[]` dovrebbe essere simile a `1234567890123-ats.iot.us-east-1.amazonaws.com`.
- Se stai seguendo la procedura descritta in Usare FreerTOS nel proprio progetto CMake per 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 file `CMakeLists.txt` (sotto la directory principale) usando la funzione CMake standard `target_link_libraries`.

- ESP-IDF v4.2 supporta l'uso di xtensa\ -esp32\ -elf\ -gcc 8\ .2\ .0\. catena di strumenti. 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, consulta. Nozioni di base sulla risoluzione dei problemi

Debug

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

Questa sezione mostra come eseguire il debug dell'hardware Espressif utilizzando ESP-IDF v4.2. È necessario un cavo da JTAG a USB. Utilizziamo un cavo da USB a MPSSE (ad esempio, FTDI C232HM-DDHSL-0).

DevKitConfigurazione ESP-C JTAG

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

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

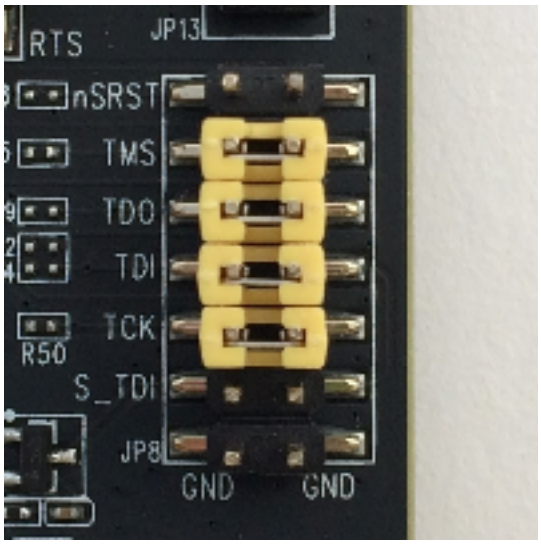
Configurazione di JTAG per ESP-WROVER-KIT

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


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


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 sull'ESP-WROVER-KIT, posiziona i jumper 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-C ed DevKit ESP-WROVER-KIT (ESP-IDF v4.2). 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 Universal Serial Bus, fate clic con il pulsante destro del mouse sul dispositivo C232HM-DDHSL-0, quindi scegliete Proprietà.

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

   Per visualizzare le proprietà del dispositivo, nella finestra delle proprietà, scegli la scheda Dettagli. Se il dispositivo non è nell'elenco, installa il driver Windows per FTDI C232HM-DDHSL-0.
3. Nella scheda Details (Dettagli), selezionare Property (Proprietà), quindi Hardware IDs (ID hardware). 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.

   Verificare che questi ID corrispondano a quelli presenti in `projects/espressif/esp32/make/aws_demos/esp32_devkitj_v1.cfg`. Gli ID 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, scegli C232HM-DDHSL-0.

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

e. Per l'elenco 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. Usa il seguente comando per scaricare il driver della porta seriale.

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

7. 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
```

8. 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)
```

9. 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 gli ID affinché corrispondano agli ID dall'output `system_profiler` della fase precedente.
10. 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.

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

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

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

```
idf.py flash monitor
```

13. 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` `aplugdev`.

```
   $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 porta seriale dell'ESP32 (UART) e viene utilizzata per caricare il codice nella memoria flash dell'ESP32.

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 tua 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'Espressif ESP32-WROOM-32SE

> **Important**
>
> Questa integrazione di riferimento è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Ti consigliamo di iniziare da qui quando crei un nuovo progetto. Se disponi già di un progetto FreeRTOS esistente basato sull'ormai obsoleto repository Amazon-FreerTOS, consulta il. Guida alla migrazione del repository Github di Amazon-FreeRTOS

Note

- Per scoprire come integrare le librerie e le demo modulari FreerTOS all'interno del tuo progetto Espressif IDF, consulta la nostra integrazione di riferimento in primo piano per la piattaforma ESP32-C3.
- Attualmente, la porta FreerTOS per ESP32-WROOM-32SE non supporta la funzionalità di multiprocessing simmetrico (SMP).

Questo tutorial mostra come iniziare con Espressif ESP32-WROOM-32SE. Per acquistarne uno dal nostro partner sul catalogo Partner Device, consulta ESP32-WROOM-32SE. AWS

Panoramica

Questo tutorial descrive le seguenti procedure:

1. Connessione della scheda a un computer host.
2. Installazione del software sul computer host per lo sviluppo e il debug di applicazioni integrate per la scheda con microcontrollore.
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.
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

Registrarsi per creare un Account AWS

Se non disponi di un Account AWS, completa la procedura seguente per crearne uno.

Per registrarsi a un Account AWS

1. Apri la pagina all'indirizzo https://portal.aws.amazon.com/billing/signup.
2. Segui le istruzioni online.

Nel corso della procedura di registrazione riceverai una telefonata, durante la quale sarà necessario inserire un codice di verifica attraverso la tastiera del telefono.

Durante la registrazione di un Account AWS, viene creato un Utente root dell'account AWS. L'utente root dispone dell'accesso a tutte le risorse e tutti i Servizi AWSnell'account. Come best practice di sicurezza, assegna l'accesso amministrativo a un utente amministrativo e utilizza solo l'utente root per eseguire attività che richiedono l'accesso di un utente root.

Al termine del processo di registrazione, riceverai un'e-mail di conferma da AWS. È possibile visualizzare l'attività corrente dell'account e gestire l'account in qualsiasi momento accedendo all'indirizzo https://aws.amazon.com/ e selezionando Il mio account.

Creazione di un utente amministratore

Dopo aver effettuato la registrazione di un Account AWS, proteggi Utente root dell'account AWS, abilita AWS IAM Identity Center e crea un utente amministratore in modo da non utilizzare l'utente root per le attività quotidiane.

Protezione dell'Utente root dell'account AWS

1. Accedi alla AWS Management Console come proprietario dell'account scegliendo Utente root e immettendo l'indirizzo email del Account AWS. Nella pagina successiva, inserisci la password.

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

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

Creazione di un utente amministratore

1. Abilita IAM Identity Center

   Per istruzioni, consulta Abilitazione di AWS IAM Identity Center nella Guida per l'utente di AWS IAM Identity Center.
2. In Centro identità AWS IAM, assegna l'accesso amministrativo a un utente amministrativo.

Per un tutorial sull'utilizzo di IAM Identity Center directory come origine di identità, consulta Configure user access with the default IAM Identity Center directory nella Guida per l'utente di AWS IAM Identity Center.

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 Accedere al portale di accesso AWS nella Guida per l'utente Accedi ad AWS.

Per fornire l'accesso, aggiungi autorizzazioni ai tuoi 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 Creating a role for a third-party identity provider (federation) (Creazione di un ruolo per un provider di identità di terze parti [federazione]) nella Guida per l'utente di IAM.
- Utenti IAM:
  - Crea un ruolo che l'utente possa assumere. Per istruzioni, consulta la pagina Creating a role for an IAM user (Creazione di un ruolo per un utente IAM) nella Guida per l'utente di 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 di IAM.

Inizia a usare

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

1. Configura l'hardware Espressif.

   Per informazioni sulla configurazione dell'hardware della scheda di sviluppo ESP32-WROOM-32SE, consultare la Guida introduttiva a ESP32-C V4. DevKit

   > **Important**
   >
   > Una volta raggiunta la sezione Step by Step della guida, procedi fino al completamento del Passaggio 4 (Impostazione delle variabili di ambiente). Interrompi dopo aver completato il Passaggio 4 e segui i passaggi rimanenti qui.

2. Scarica Amazon GitHubFreerTOS da. (Per istruzioni, consulta il file README.md).
3. Configura il tuo ambiente di sviluppo.

   Per comunicare con la scheda, è necessario installare una toolchain. Espressif fornisce l'ESP-IDF per sviluppare software per le proprie schede. Poiché l'ESP-IDF ha una propria versione del kernel FreeRTOS integrata come componente, Amazon FreerTOS include una versione personalizzata di ESP-IDF v4.2 con il kernel FreerTOS rimosso. Questo risolve i problemi relativi ai file duplicati durante la compilazione. Per utilizzare la versione personalizzata di ESP-IDF v4.2 inclusa in Amazon FreerTOS, segui le istruzioni riportate di seguito per il sistema operativo della tua macchina host.

   Windows

   1. Scarica l'Universal Online Installer di ESP-IDF per Windows.
   2. Esegui l'Universal Online Installer.
   3. Quando arrivi alla fase Scarica o usa ESP-IDF, seleziona Usa una directory ESP-IDF esistente e imposta Scegli la directory ESP-IDF esistente su. *`freertos`*`/vendors/espressif/esp-idf`
   4. Completa l'installazione.

   macOS

   1. Segui le istruzioni contenute nella Configurazione standard dei prerequisiti della Toolchain (ESP-IDF v4.2) per macOS.

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

2. Aprire una finestra a riga di comando.
3. 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
```

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

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

Linux

1. Segui le istruzioni contenute nella 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.

2. Aprire una finestra a riga di comando.
3. 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
```

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

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

4. Stabilisci una connessione seriale.

   a. Per stabilire una connessione seriale tra il computer host ed ESP32-WROOM-32SE, occorre installare i driver CP210x USB to UART Bridge VCP. È possibile scaricare i driver da Silicon Labs.

   b. Segui i passaggi per stabilire una connessione seriale con ESP32.

   c. Dopo aver stabilito una connessione seriale, annotare la porta seriale per la connessione della scheda. Ne hai bisogno per eseguire il flashing della demo.

Configura le applicazioni demo FreerTOS

Per questo tutorial, il file di configurazione di FreerTOS si trova in. *freertos*`/vendors/espressif/boards/`*board-name*`/aws_demos/config_files/FreeRTOSConfig.h` (Ad esempio, se `AFR_BOARD espressif.esp32_devkitc` viene scelto, il file di configurazione si trova in*freertos*`/vendors/espressif/boards/esp32/aws_demos/config_files/FreeRTOSConfig.h`.)

> **Important**
>
> Il dispositivo ATECC608A 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 ed esegui i seguenti passaggi.
2. Microchip ha diversi strumenti di scripting per facilitare la configurazione delle parti ATECC608A. Passare alla directory *freertos*`/vendors/microchip/example_trust_chain_tool` e aprire il file `README.md`.
3. Per effettuare il provisioning del dispositivo, segui le istruzioni contenute nel `README.md` file. Queste fasi includono quanto segue:

   1. Creare e registrare un'autorità di certificazione con AWS.

2. Generare le chiavi su ATECC608A ed esportare la chiave pubblica e il numero di serie del dispositivo.
3. Generare un certificato per il dispositivo e registrare il certificato con AWS.

4. Caricare il certificato CA e il certificato del dispositivo sul dispositivo seguendo le istruzioni in Distribuzione delle chiavi in modalità sviluppatore.

Monitoraggio dei messaggi MQTT sul cloud AWS

Prima di eseguire il progetto demo FreerTOS, puoi configurare il client MQTT nella console per monitorare AWS IoT i messaggi che il tuo dispositivo invia al Cloud. AWS

Per effettuare la sottoscrizione all'argomento MQTT con il client AWS IoT

1. Accedi alla console AWS IoT.
2. Nel pannello di navigazione, scegli Test, quindi scegli MQTT Test Client.
3. Nell'argomento Abbonamento, inserisci *your-thing-name*`/example/topic` e quindi scegli Sottoscrivi all'argomento.

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

Puoi usare l'utilità IDF di Espressif (`idf.py`) per generare i file di build, creare il binario dell'applicazione e eseguire il flashing dei binari sul tuo dispositivo.

> **Note**
>
> Alcune configurazioni potrebbero richiedere l'utilizzo dell'opzione port "`-p port-name`" con `idf.py` per specificare la porta corretta, come nell'esempio seguente.
>
> ```
> idf.py -p /dev/cu.usbserial-00101301B flash
> ```

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

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 (app «Comando»)

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

Windows (app «ESP-IDF 4.x CMD»)

(Questa operazione è già stata eseguita quando hai aperto l'app.)

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
>   ```
>
> - 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 `idf.py` script 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
   ```

Informazioni aggiuntive

Per ulteriori informazioni sull'utilizzo e la risoluzione dei problemi delle schede Espressif ESP32, consulta i seguenti argomenti:

- Usare FreerTOS nel proprio progetto CMake per ESP32

- Risoluzione dei problemi
- Debug

Guida introduttiva all'Espressif ESP32-S2

> **Important**
>
> Questa integrazione di riferimento è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Ti consigliamo di iniziare da qui quando crei un nuovo progetto. Se hai già un progetto FreeRTOS esistente basato sull'ormai obsoleto repository Amazon-FreerTOS, consulta il. Guida alla migrazione del repository Github di Amazon-FreeRTOS

> **Note**
>
> Per scoprire come integrare le librerie e le demo modulari FreerTOS all'interno del tuo progetto Espressif IDF, consulta la nostra integrazione di riferimento in primo piano per la piattaforma ESP32-C3.

Questo tutorial mostra come iniziare a usare il SoC Espressif ESP32-S2 e le schede di sviluppo ESP32-S2-Saola-1.

Panoramica

Questo tutorial descrive le seguenti procedure:

1. Connessione della scheda a un computer host.
2. Installazione del software sul computer host per lo sviluppo e il debug di applicazioni integrate per la scheda con microcontrollore.
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.
5. Monitoraggio ed esecuzione del 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

Registrarsi per creare un Account AWS

Se non disponi di un Account AWS, completa la procedura seguente per crearne uno.

Per registrarsi a un Account AWS

1. Apri la pagina all'indirizzo https://portal.aws.amazon.com/billing/signup.
2. Segui le istruzioni online.

   Nel corso della procedura di registrazione riceverai una telefonata, durante la quale sarà necessario inserire un codice di verifica attraverso la tastiera del telefono.

   Durante la registrazione di un Account AWS, viene creato un Utente root dell'account AWS. L'utente root dispone dell'accesso a tutte le risorse e tutti i Servizi AWSnell'account. Come best practice di sicurezza, assegna l'accesso amministrativo a un utente amministrativo e utilizza solo l'utente root per eseguire attività che richiedono l'accesso di un utente root.

Al termine del processo di registrazione, riceverai un'e-mail di conferma da AWS. È possibile visualizzare l'attività corrente dell'account e gestire l'account in qualsiasi momento accedendo all'indirizzo https://aws.amazon.com/ e selezionando Il mio account.

Creazione di un utente amministratore

Dopo aver effettuato la registrazione di un Account AWS, proteggi Utente root dell'account AWS, abilita AWS IAM Identity Center e crea un utente amministratore in modo da non utilizzare l'utente root per le attività quotidiane.

Protezione dell'Utente root dell'account AWS

1. Accedi alla AWS Management Console come proprietario dell'account scegliendo Utente root e immettendo l'indirizzo email del Account AWS. Nella pagina successiva, inserisci la password.

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

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

Creazione di un utente amministratore

1. Abilita IAM Identity Center

   Per istruzioni, consulta Abilitazione di AWS IAM Identity Center nella Guida per l'utente di AWS IAM Identity Center.
2. In Centro identità AWS IAM, assegna l'accesso amministrativo a un utente amministrativo.

   Per un tutorial sull'utilizzo di IAM Identity Center directory come origine di identità, consulta Configure user access with the default IAM Identity Center directory nella Guida per l'utente di AWS IAM Identity Center.

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 Accedere al portale di accesso AWS nella Guida per l'utente Accedi ad AWS.

Per fornire l'accesso, aggiungi autorizzazioni ai tuoi 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 Creating a role for a third-party identity provider (federation) (Creazione di un ruolo per un provider di identità di terze parti [federazione]) nella Guida per l'utente di IAM.
- Utenti IAM:
  - Crea un ruolo che l'utente possa assumere. Per istruzioni, consulta la pagina Creating a role for an IAM user (Creazione di un ruolo per un utente IAM) nella Guida per l'utente di 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 di IAM.

Inizia a usare

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

1. Configura l'hardware Espressif.

   Per informazioni sulla configurazione dell'hardware della scheda di sviluppo ESP32-S2, consulta la Guida introduttiva ESP32-S2-Saola-1.

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

2. Scarica Amazon GitHubFreerTOS da. (Per istruzioni, consulta il file README.md).
3. Configura il tuo ambiente di sviluppo.

   Per comunicare con la scheda, è necessario installare una toolchain. Espressif fornisce l'ESP-IDF per sviluppare software per le proprie schede. Poiché l'ESP-IDF ha una propria versione del kernel FreeRTOS integrata come componente, Amazon FreerTOS include una versione personalizzata di ESP-IDF v4.2 con il kernel FreerTOS rimosso. Questo risolve i problemi relativi ai file duplicati durante la compilazione. Per utilizzare la versione personalizzata di ESP-IDF v4.2 inclusa in Amazon FreerTOS, segui le istruzioni riportate di seguito per il sistema operativo della tua macchina host.

   Windows

   1. Scarica l'Universal Online Installer di ESP-IDF per Windows.
   2. Esegui l'Universal Online Installer.

3. Quando arrivi alla fase Scarica o usa ESP-IDF, seleziona Usa una directory ESP-IDF esistente e imposta Scegli la directory ESP-IDF esistente su. *freertos*/vendors/espressif/esp-idf
4. Completa l'installazione.

macOS

1. Segui le istruzioni nella configurazione standard dei prerequisiti della Toolchain (ESP-IDF v4.2) per macOS.

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

2. Aprire una finestra a riga di comando.
3. 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
```

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

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

Linux

1. Segui le istruzioni contenute nella 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.

2. Aprire una finestra a riga di comando.

3. 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
```

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

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

4. Stabilisci una connessione seriale.

   a. Per stabilire una connessione seriale tra la macchina host e l'ESP32- DevKit C, installa i driver CP210x USB to UART Bridge VCP. È possibile scaricare i driver da Silicon Labs.

   b. Segui i passaggi per stabilire una connessione seriale con ESP32.

   c. Dopo aver stabilito una connessione seriale, annotare la porta seriale per la connessione della scheda. È necessario per eseguire il flashing della demo.

Configura le applicazioni demo FreerTOS

Per questo tutorial, il file di configurazione di FreerTOS si trova in. *freertos*/vendors/espressif/boards/*board-name*/aws_demos/config_files/FreeRTOSConfig.h (Ad esempio, se AFR_BOARD espressif.esp32_devkitc viene scelto, il file di configurazione si trova in*freertos*/vendors/espressif/boards/esp32/aws_demos/config_files/FreeRTOSConfig.h.)

1. Se utilizzi macOS o Linux, apri un prompt del terminale. Se utilizzi Windows, apri l'app «ESP-IDF 4.x CMD» (se hai incluso questa opzione quando hai installato la toolchain ESP-IDF), altrimenti l'app «Command Prompt».
2. Per verificare che Python3 sia installato, esegui quanto segue:

```
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 necessaria l'interfaccia CLI (AWSCommand Line Interface) per eseguire AWS IoT i comandi. Se utilizzi Windows, usa il `easy_install awscli` comando per installare la AWS CLI nell'app «Command» o «ESP-IDF 4.x CMD».

   Se utilizzi macOS o Linux, consulta Installazione della CLI AWS.

4. Esecuzione

   ```
   aws configure
   ```

   e configura la AWS CLI con l'ID della chiave di AWS accesso, la chiave di accesso segreta e la regione predefinitaAWS. Per ulteriori informazioni, consulta Configurazione di AWS CLI.

5. Usa il seguente comando per installare l'AWSSDK per Python (boto3):

   - Su Windows, nell'app «Command» o «ESP-IDF 4.x CMD», 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 eseguire lo script di configurazione

1. 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 si desidera assegnare all'elemento AWS IoT che rappresenta la 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` (Sicurezza WPA2)

2. Se utilizzi macOS o Linux, apri un prompt del terminale. Se utilizzi Windows, apri l'app «ESP-IDF 4.x CMD» o «Command».
3. Vai alla directory ed esegui *`freertos`*`/tools/aws_config_quick_start`

```
python SetupAWS.py setup
```

Lo script svolge le seguenti funzioni:

- Crea qualsiasi AWS IoT cosa, certificato e criterio.
- Allega la AWS IoT policy al certificato e il certificato all'AWS IoToggetto.
- Compila il `aws_clientcredential.h` file con l'AWS IoTendpoint, 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 meritoSetupAWS.py, vedere README.md nella directory. *freertos*/tools/aws_config_quick_start

Monitoraggio dei messaggi MQTT sul cloud AWS

Prima di eseguire il progetto demo FreerTOS, puoi configurare il client MQTT nella console per monitorare AWS IoT i messaggi che il tuo dispositivo invia al Cloud. AWS

Per effettuare la sottoscrizione all'argomento MQTT con il client AWS IoT

1. Accedi alla console AWS IoT.
2. Nel pannello di navigazione, scegli Test, quindi scegli MQTT Test Client.
3. In Argomento sottoscrizione, digitare *your-thing-name*/example/topic, quindi scegliere Effettua sottoscrizione all'argomento.

Quando il progetto demo viene eseguito correttamente sul dispositivo, viene visualizzato «Hello World!» inviato più volte all'argomento a cui ti sei iscritto.

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 radice della tua directory 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 (app «Comando»)

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

   Windows (app «ESP-IDF 4.x CMD»)

   (Questa operazione è già stata eseguita quando hai aperto l'app.)

   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=esp32s2_saola_1 -DCOMPILER=xtensa-esp32s2 build
   ```

   Dovresti vedere un output come questo nell'esempio seguente.

   ```
   Executing action: all (aliases: build)
      Running cmake in directory /path/to/hello_world/build
      Executing "cmake -G Ninja -DPYTHON_DEPS_CHECKED=1 -DESP_PLATFORM=1
    -DVENDOR=espressif -DBOARD=esp32s2_saola_1 -DCOMPILER=xtensa-esp32s2 -
   DCCACHE_ENABLE=0 /path/to/hello_world"...
      -- The C compiler identification is GNU 8.4.0
      -- The CXX compiler identification is GNU 8.4.0
      -- The ASM compiler identification is GNU

      ... (more lines of build system output)

      [1628/1628] Generating binary image from built executable
      esptool.py v3.0
   ```

```
  Generated /path/to/hello_world/build/aws_demos.bin

  Project build complete. To flash, run this command:
  esptool.py -p (PORT) -b 460800 --before default_reset --after hard_reset --chip
esp32s2  write_flash --flash_mode dio --flash_size detect --flash_freq 80m 0x1000
build/bootloader/bootloader.bin 0x8000 build/partition_table/partition-table.bin
0x16000 build/ota_data_initial.bin 0x20000 build/aws_demos.bin
  or run 'idf.py -p (PORT) flash'
```

Se non ci sono errori, la build genera 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
>   ```
>
> - 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 `idf.py` script 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=esp32s2_saola_1 -DCOMPILER=xtensa-esp32s2 -S . -
   B ./YOUR_BUILD_DIRECTORY -DAFR_ENABLE_ALL_MODULES=1 -DAFR_ENABLE_TESTS=0
   ```

   - Con Ninja

   ```
   cmake -DVENDOR=espressif -DBOARD=esp32s2_saola_1 -DCOMPILER=xtensa-esp32s2 -S . -
   B ./YOUR_BUILD_DIRECTORY -DAFR_ENABLE_ALL_MODULES=1 -DAFR_ENABLE_TESTS=0 -GNinja
   ```

4. Compilare il progetto.

   - Con Unix Makefiles

   ```
   make -C ./YOUR_BUILD_DIRECTORY -j8
   ```

   - Con Ninja

   ```
   ninja -C ./YOUR_BUILD_DIRECTORY -j8
   ```

5. 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
```

Informazioni aggiuntive

Per ulteriori informazioni sull'utilizzo e la risoluzione dei problemi delle schede Espressif ESP32, consulta i seguenti argomenti:

- Usare FreerTOS nel proprio progetto CMake per ESP32
- Risoluzione dei problemi
- Debug

Nozioni di base su Infineon XMC4800 IoT Connectivity Kit

> **Important**
>
> Questa integrazione di riferimento è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Ti consigliamo di iniziare da qui quando crei un nuovo progetto. Se disponi già di un progetto FreeRTOS esistente basato sull'ormai obsoleto repository Amazon-FreerTOS, consulta il. Guida alla migrazione del repository Github di Amazon-FreeRTOS

Questo tutorial fornisce istruzioni per iniziare a utilizzare il kit di connessione Infineon XMC4800 IoT. Se non disponi del kit di connettività IoT Infineon XMC4800, visita AWS il Partner Device Catalog per acquistarne uno dal nostro partner.

Se desideri aprire una connessione seriale con la scheda per visualizzare le informazioni di accesso e di debug, devi avere a disposizione un convertitore USB/seriale 3,3V, oltre a XMC4800 IoT Connectivity Kit. CP2104 è un comune convertitore USB/seriale ampiamente disponibile su schede come Adafruit CP2104 Friend.

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.
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 l'XMC4800. 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 della scheda XMC4800 Relax EtherCAT, sono necessari i driver inclusi nel pacchetto J-Link Software and Documentation. 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 dimostrativa XMC4800 utilizza una connessione seriale UART su connettori P0.0 e P0.1, che vengono etichettati nella serigrafia della scheda XMC4800 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 XMC4800 e il modulo Wi-Fi Click richiedono un voltaggio da 3,3V. 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

Prima di eseguire la demo di FreerTOS, puoi configurare il client MQTT nella console per monitorare AWS IoT i messaggi che il tuo dispositivo invia al Cloud. AWS

Per effettuare la sottoscrizione all'argomento MQTT con il client AWS IoT

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

Quando il progetto demo viene eseguito correttamente sul tuo dispositivo, vedi «Hello World!» inviato più volte all'argomento a cui ti sei iscritto.

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 Import DAVE Projects (Importa progetti DAVE), scegliere Select Root Directory (Seleziona directory principale), scegliere Browse (Sfoglia) e quindi scegliere il progetto demo XMC4800.

   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. Utilizzare un cavo USB per collegare XMC4800 IoT Connectivity Kit 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 console AWS IoT, il client MQTT dei passaggi 4-5 dovrebbe essere visibile nei 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 utilizzare CMake 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'uso di CMake su Windows con MingW come sistema di compilazione nativo. Per ulteriori informazioni sull'uso di CMake con altri sistemi operativi e altre opzioni, consulta Utilizzo di 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. Installare CMake e MingW.

   Per istruzioni, consultare CMake Prerequisiti.

3. Creare una cartella per contenere i file della 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. Passare alla directory di compilazione (*`build-folder`*) e utilizzare il comando seguente per compilare 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 di compilazione (*`build-folder`*), utilizzare i comandi seguenti per creare uno script di flashing:

```
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.

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

Nozioni di base su Infineon OPTIGA Trust X e XMC4800 IoT Connectivity Kit

> **Important**
>
> Questa integrazione di riferimento è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Consigliamo di iniziare da qui quando crei un nuovo progetto. Se hai già un progetto FreeRTOS esistente basato sull'ormai obsoleto repository Amazon-FreeRTOS, consulta ilGuida alla migrazione del repository Github di Amazon-FreeRTOS.

Questo tutorial fornisce istruzioni per iniziare a utilizzare il kit di connessione Infineon OPTIGA Trust X Secure Element e XMC4800 IoT. Rispetto al tutorial Nozioni di base su Infineon XMC4800 IoT Connectivity Kit, questa guida illustra come fornire credenziali sicure utilizzando Infineon OPTIGA Trust X Secure Element.

È necessario il seguente hardware:

1. Host MCU - Kit di connettività IoT Infineon XMC4800, visita il catalogo dei dispositivi deiAWS partner per acquistarne uno dal nostro partner.
2. Pacchetto di estensione di sicurezza:
   - Elemento di sicurezza - Infineon OPTIGA Trust X.

     Visita il catalogo dei dispositivi per iAWS partner 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 in aggiunta al kit di connettività IoT XMC4800. Il convertitore USB/seriale JBtek EL-PN-47310126 funziona con questa demo. Sono inoltre necessari tre cavi male-to-male jumper (per la ricezione (RX), la trasmissione (TX) e la messa a terra (GND)) per collegare il cavo seriale alla scheda adattatore Infineon MyIo T.

Prima di iniziare, devi configurareAWS IoT e scaricare FreeRTOS per connettere il tuo dispositivo alAWS Cloud. Per istruzioni, consulta Opzione 2: Generazione di chiavi private integrate. In questo tutorial, il percorso della directory di download di FreeRTOS viene chiamato*`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 un'applicazione demo FreeRTOS in 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 l'XMC4800. 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 della scheda XMC4800 IoT del kit di connessione, sono necessari i driver inclusi nel pacchetto J-Link Software and Documentation. 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 XMC4800 e il modulo Wi-Fi Click richiedono un voltaggio da 3,3V. 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

Prima di eseguire il progetto demo di FreeRTOS, puoi configurare il client MQTT nellaAWS IoT console per monitorare i messaggi che il tuo dispositivo invia alAWS Cloud.

Per effettuare la sottoscrizione all'argomento MQTT con il client AWS IoT

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

Quando il progetto demo viene eseguito correttamente sul tuo dispositivo, viene visualizzato «Hello World!» inviato più volte all'argomento a cui ti sei iscritto.

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 Import DAVE Projects (Importa progetti DAVE), scegliere Select Root Directory (Seleziona directory principale), scegliere Browse (Sfoglia) e quindi scegliere il progetto demo XMC4800.

   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 Finish (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. Al termine di tutti i passaggi, andare alla console AWS IoT. 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'uso di CMake su Windows con MingW come sistema di compilazione nativo. Per ulteriori informazioni sull'uso di CMake con altri sistemi operativi e altre opzioni, consulta Utilizzo di 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 usare CMake 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. Installare CMake e MingW.

   Per istruzioni, consultare CMake Prerequisiti.

3. Creare una cartella per contenere i file della build generati (*`build-folder`*).

4. Cambia le directory nella cartella 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. Passare alla directory di compilazione (*`build-folder`*) e utilizzare il comando seguente per compilare 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 di compilazione (*build-folder*), utilizzare i comandi seguenti per creare uno script di flashing:

```
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. Al termine di tutti i passaggi, andare alla console AWS IoT. 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 alloAWS IoT Starter Kit MW32x

> **Important**
>
> Questa integrazione di riferimento è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Consigliamo di iniziare da qui quando si crea un nuovo progetto. Se hai già un progetto FreeRTOS esistente basato sull'ormai obsoleto repository Amazon-FreeRTOS, consulta ilGuida alla migrazione del repository Github di Amazon-FreeRTOS.

LoAWS IoT Starter Kit è un kit di sviluppo basato sull'88MW320/88MW322, l'ultimo microcontrollore integrato Cortex M4 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 AWSPartner Device Catalog

per acquistarne uno dal nostro partner. I moduli 88MW320/88MW322 sono inoltre certificati FCC e disponibili per la personalizzazione e la vendita in serie.

Questa guida introduttiva mostra come compilare in modo incrociato l'applicazione con 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. I seguenti strumenti esterni necessari per creare l'SDK:

- Piattaforma host Ubuntu 16.04
- Toolchain ARM versione 4_9_2015q3
- Eclipse 4.9.0 IDE

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'uso di versioni precedenti della toolchain non è consigliato. Il kit di sviluppo è preinstallato con il firmware di progetto Wireless Microcontroller Demo.

Argomenti

Configurazione dell'hardware

Collega la scheda MW32x al tuo laptop utilizzando un cavo da mini-USB a USB. Collega il cavo mini-USB all'unico connettore mini-USB presente sulla scheda. Non è necessario cambiare il maglione.

Se la scheda è collegata a un computer portatile o desktop, non è necessario un alimentatore esterno.

Questa connessione USB fornisce quanto segue:

- Accesso alla scheda da console. Una porta tty/com virtuale è registrata con 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

Ai fini dello sviluppo, il requisito minimo è la toolchain ARM e gli strumenti inclusi nell'SDK. Nelle seguenti sezioni sono fornite maggiori informazioni sulla configurazione della toolchain ARM.

Catena degli strumenti GNU

L'SDK supporta ufficialmente la toolchain del compilatore GCC. La toolchain cross-compilator 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 per impostazione predefinita. I Makefile presuppongono che i binari della toolchain del compilatore GNU siano disponibili nel PATH dell'utente e possano essere richiamati dai Makefile. I Makefile presuppongono anche che i nomi dei file dei binari della toolchain GNU abbiano il prefisso con`arm-none-eabi-`.

La toolchain GCC può essere utilizzata con GDB per il debug con OpenOCD (in bundle con l'SDK). Questo fornisce il software che si interfaccia con JTAG.

Consigliamo la versione 4_9_2015q3 della gcc-arm-embedded toolchain.

Procedura di configurazione Linux Toolchain

Seguire questa procedura per configurare la toolchain GCC in Linux.

1. Scarica il file tarball della toolchain disponibile su GNU Arm Embedded Toolchain 4.9-2015-q3-update. Il file è`gcc-arm-none-eabi-4_9-2015q3-20150921-linux.tar.bz2`.
2. Copiare il file in una cartella a scelta. Assicurati che non ci siano spazi nel nome della cartella.
3. Utilizzare il comando seguente per decomprimere il file.

   ```
   tar -vxf filename
   ```

4. Aggiungi il percorso della toolchain installata al PATH di 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 distribuzioni più recenti di Ubuntu potrebbero avere una versione Debian del GCC Cross Compiler. In tal caso, è necessario rimuovere il Cross Compiler nativo e seguire la procedura di configurazione precedente.

Lavorare con un host di sviluppo Linux

È possibile utilizzare qualsiasi distribuzione desktop Linux moderna come Ubuntu o Fedora. Consigliamo, tuttavia, 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 appena configurata. Lo script tenta di rilevare auto il tipo di macchina e installare il software appropriato, tra cui librerie C, libreria USB, libreria FTDI, ncurses, python e latex. In questa sezione, il nome della directory generica*amzsdk_bundle-x.y.z* indica la directory principale dell'AWSSDK. Il nome effettivo della directory potrebbe essere diverso. È necessario avere privilegi root.

- Accedere alla*amzsdk_bundle-x.y.z/* directory ed eseguire questo comando.

```
./lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/bin/installpkgs.sh
```

Evitare il sudo

In questa guida, l'`flashprog`operazione utilizza lo`flashprog.py` script per flashare la 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 far flashare la 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 IDE Eclipse fluida.

Configurazione della console seriale

Inserisci il cavo USB nello slot USB dell'host Linux. Questo 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
```

Verificare che siano stati creati due dispositivi ttyUSB. La seconda ttyUSB è la console seriale. Nell'esempio riportato sopra, a questo 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 setup.

```
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 in silicio FTDI. 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 uartX fisica dell'MCU.

Installazione di OpenOCD

OpenOCD è un software che fornisce debug, programmazione interna al sistema e test boundary-scan per dispositivi di destinazione integrati.

La disponibilità di OpenOCD versione 0.9 è obbligatoria. È inoltre richiesto per la funzionalità Eclipse. Se sul tuo host Linux è stata installata una versione precedente, ad esempio la versione 0.7, rimuovi quel 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 comando seguente per installare libusb-1.0.

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

2. Scarica 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. Configurare openocd con il comando seguente.

```
./configure --enable-ftdi --enable-jlink
```

5. Esegui l'utilità make per compilare opencd.

```
make install
```

Configurazione di Eclipse

> **Note**
>
> In questa sezione si presuppone che l'utente abbia completato la procedura diEvitare il sudo

Eclipse è l'IDE preferito per lo sviluppo e il debug delle applicazioni. Fornisce un IDE ricco e intuitivo con supporto di debug integrato, incluso il debug sensibile ai thread. Questa sezione descrive la configurazione comune di Eclipse per tutti gli host di sviluppo supportati.

Per configurare Eclipse

1. Scarica e installa Java Run Time Environment (JRE).

   Eclipse richiede l'installazione di JRE. Ti consigliamo 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. Scarica e installa «Eclipse IDE per sviluppatori C/C++» da http://www.eclipse.org. Eclipse 4.9.0 o versioni successive è supportata. L'installazione richiede solo l'estrazione dell'archivio scaricato. Esegui l'eseguibile Eclipse specifico della piattaforma per avviare l'applicazione.

Crea ed esegui il progetto demo FreeRTOS

Ci sono due modi per eseguire il progetto demo di FreeRTOS:

- Utilizzare la riga di comando.
- Usa l'IDE Eclipse.

Questo argomento tratta entrambe le opzioni.

Approvvigionamento

- A seconda che si utilizzi l'applicazione di test o demo, impostate i dati di provisioning in uno dei seguenti file:
  - `./tests/common/include/aws_clientcredential.h`
  - `./demos/common/include/aws_clientcredential.h`

  Ad esempio:

  ```
  #define clientcredentialWIFI_SSID "Wi-Fi SSID"
  #define clientcredentialWIFI_PASSWORD "Wi-Fi password"
  #define clientcredentialWIFI_SECURITY "Wi-Fi security"
  ```

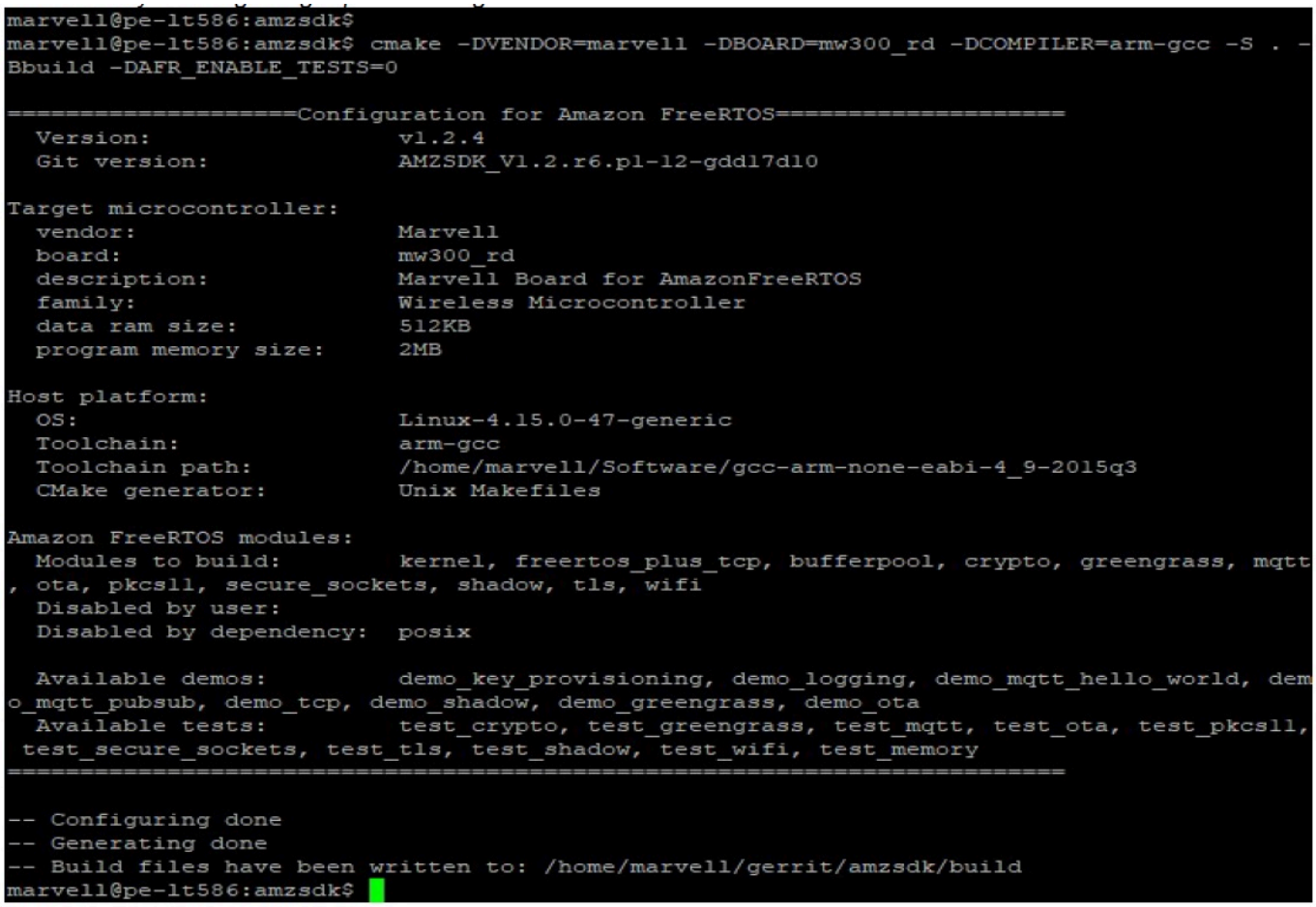
  > **Note**
  >
  > Puoi inserire i seguenti valori di sicurezza Wi-Fi:
  >
  > - `eWiFiSecurityOpen`
  > - `eWiFiSecurityWEP`
  > - `eWiFiSecurityWPA`
  > - `eWiFiSecurityWPA2`
  >
  > L'SSID e la password devono essere racchiusi tra virgolette doppie.

Crea 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
```

   Assicurarsi di ottenere lo stesso output visualizzato nell'esempio seguente.



2. Passare alla directory di build.

```
cd build
```

3. Esegui l'utilità make per creare l'applicazione.

```
make all -j4
```

Assicurarsi di ottenere lo stesso output mostrato nella seguente

4. Utilizzare il seguente comando 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`aws_demos project` e`aws_tests 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 inviare l'immagine al microcontrollore.

   a. Prima di eseguire il flashing dell'immagine del firmware, prepara il flash della scheda di sviluppo con i componenti comuni Layout e Boot2. Utilizzare il seguente comando.

   ```
   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: l'utilità flashprog viene prima istruita a scrivere un layout sul 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 boot-loader 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. Assicurati 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

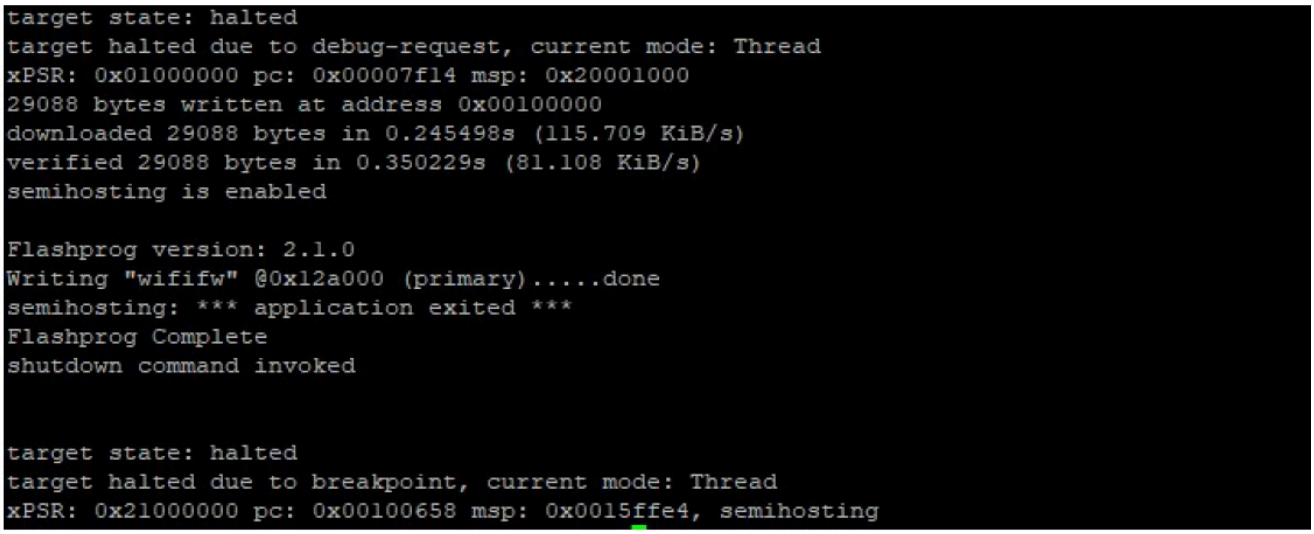
Flashprog version: 2.1.0
Erasing primary flash...done
Writing new flash layout...done
Writing "boot2" @0x0 (primary)...done
semihosting: *** application exited ***
Flashprog Complete
shutdown command invoked


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 la sua funzionalità e il chipset Wi-Fi ha il proprio firmware che deve essere presente anche nel flash. Si utilizza l'`flashprog.py`utilità per eseguire il flashing del firmware Wi-Fi nello stesso modo in cui è stato eseguito il flashing del boot-loader Boot2 e del firmware MCU. Usa 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
```

Assicurati che l'output del comando sia simile alla figura seguente.



c. Usa i seguenti comandi per eseguire il flashing del firmware 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 Dovresti vedere i log dell'app demo.

e. Per eseguire l'app di test, esegui il flashing del`aws_tests.bin` 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 visualizzato dovrebbe essere simile a quello mostrato nella 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 aver 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----
```

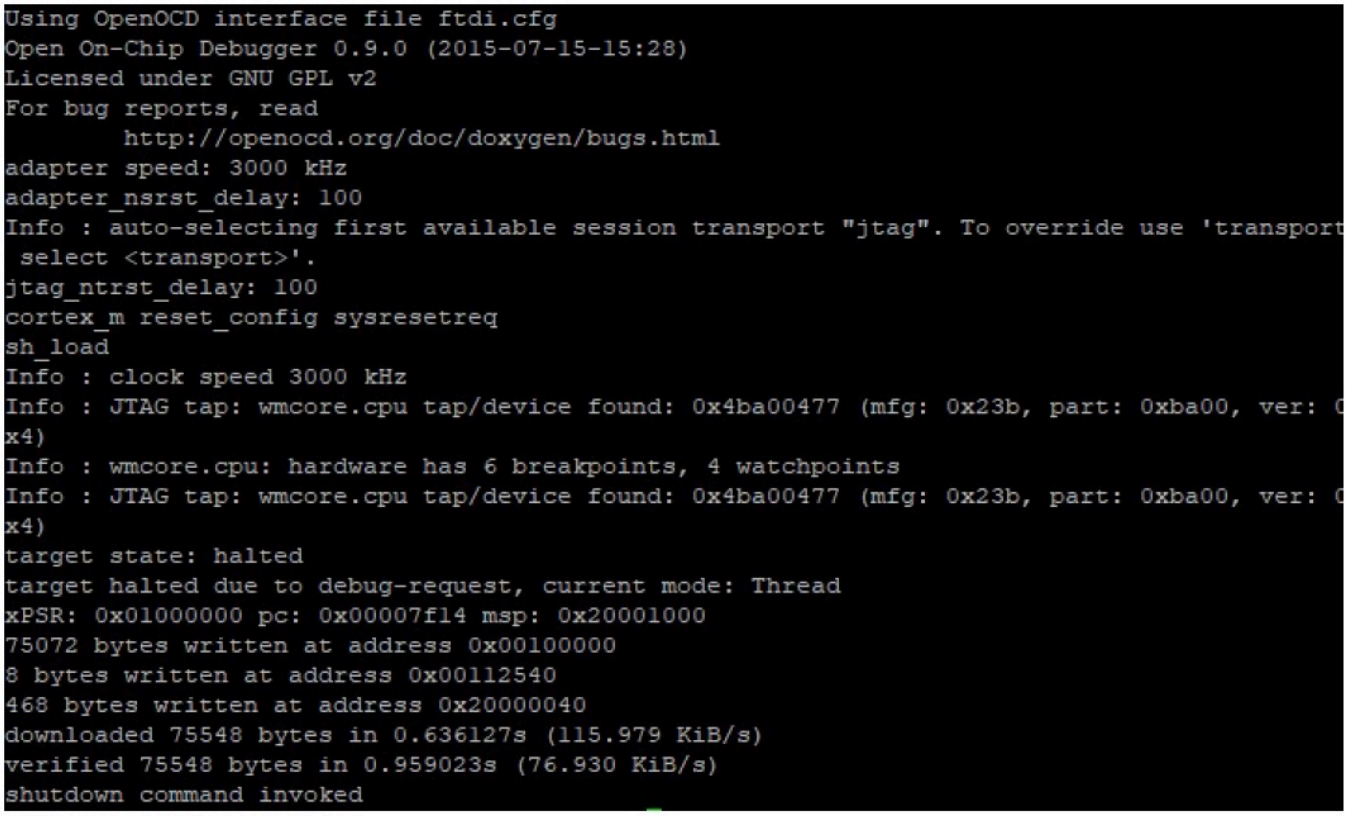
7. (Facoltativo) Come metodo alternativo per testare l'immagine, utilizzate l'utilità flashprog per copiare l'immagine del microcontrollore dall'host direttamente nella RAM del microcontrollore. L'immagine non viene copiata nel flash, quindi andrà persa dopo il riavvio del microcontrollore.

   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 mostrato nella figura seguente.



Quando l'esecuzione del comando è completa, dovresti vedere i registri dell'app demo.

Crea ed esegui la demo di FreeRTOS usando l'IDE Eclipse

1. Prima di configurare un'area di lavoro Eclipse, è necessario eseguire il`cmake` comando.

   Esegui il seguente comando per lavorare con il progetto`aws_demos` Eclipse.

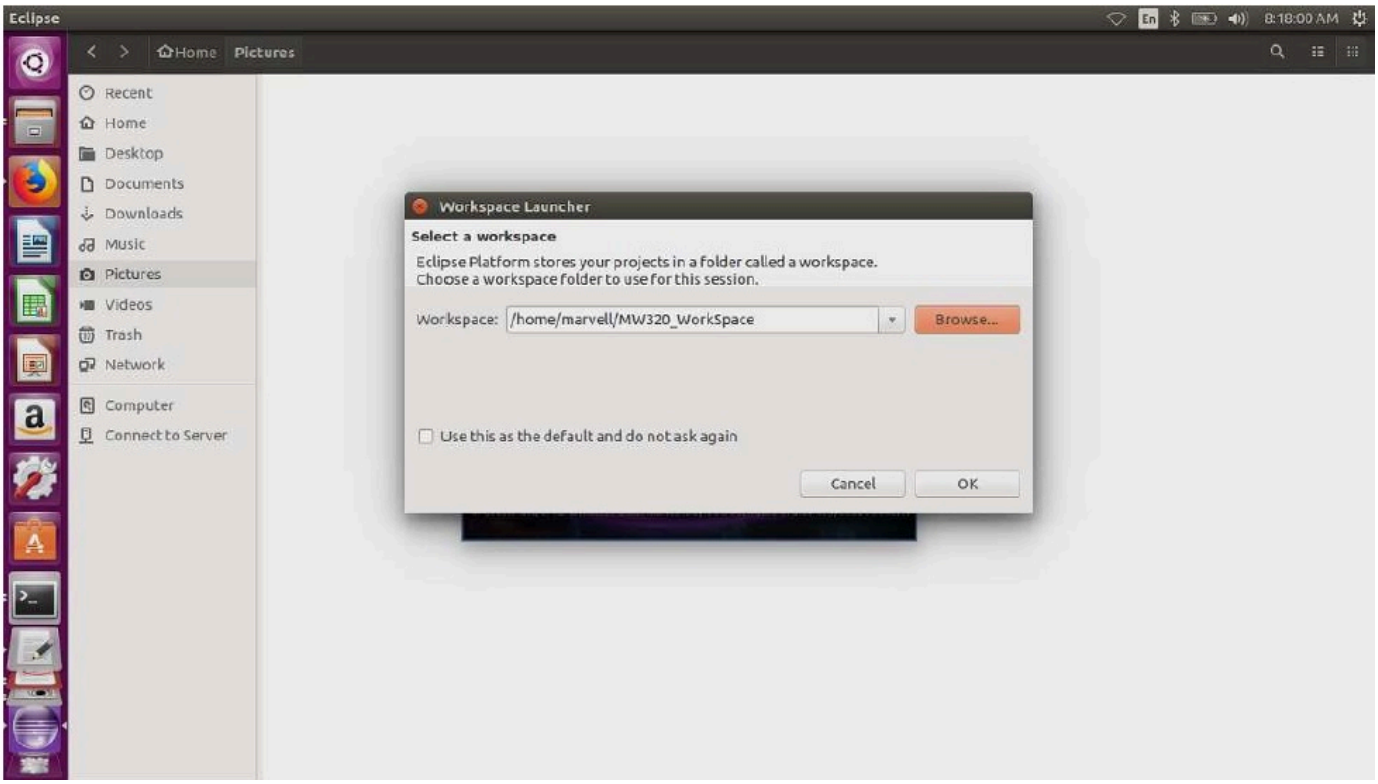
   ```
   cmake -DVENDOR=marvell -DBOARD=mw320 -DCOMPILER=arm-gcc -S .  -Bbuild -
   DAFR_ENABLE_TESTS=0
   ```

   Esegui il seguente comando per lavorare con il progetto`aws_tests` Eclipse.

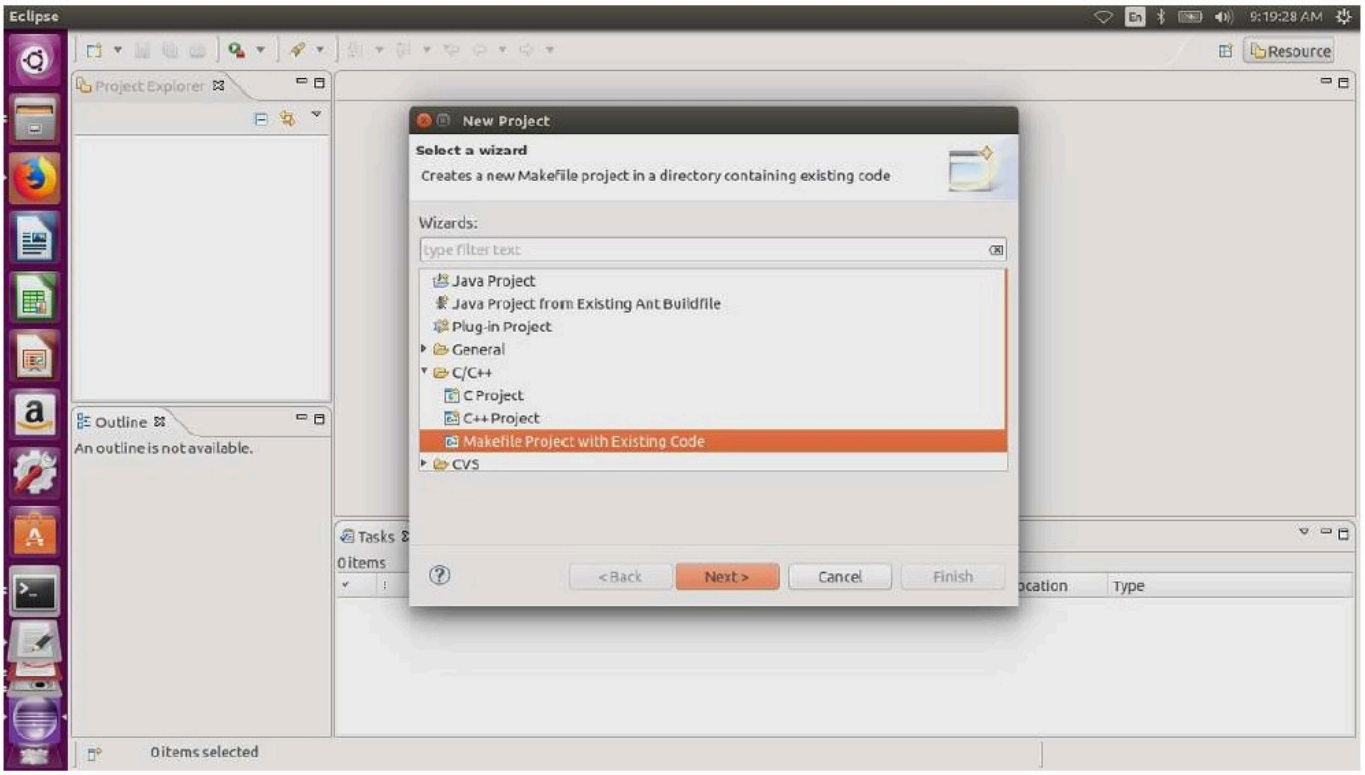
   ```
   cmake -DVENDOR=marvell -DBOARD=mw320 -DCOMPILER=arm-gcc -S .  -Bbuild -
   DAFR_ENABLE_TESTS=1
   ```

> **Tip**
>
> Esegui il`cmake` comando ogni volta che passi da`aws_demos` progetto a`aws_tests` progetto.

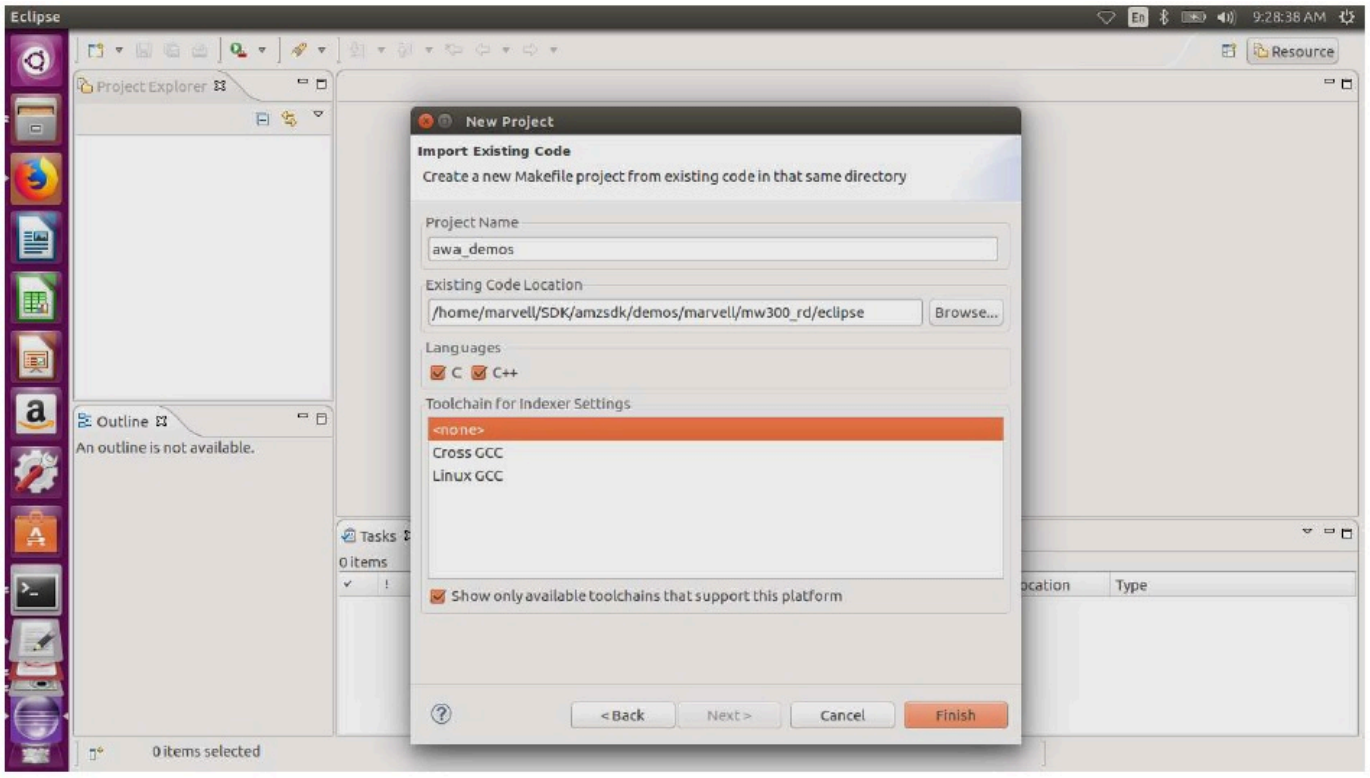
2. Apri Eclipse e, quando richiesto, scegli il tuo spazio di lavoro Eclipse come mostrato nella figura seguente.



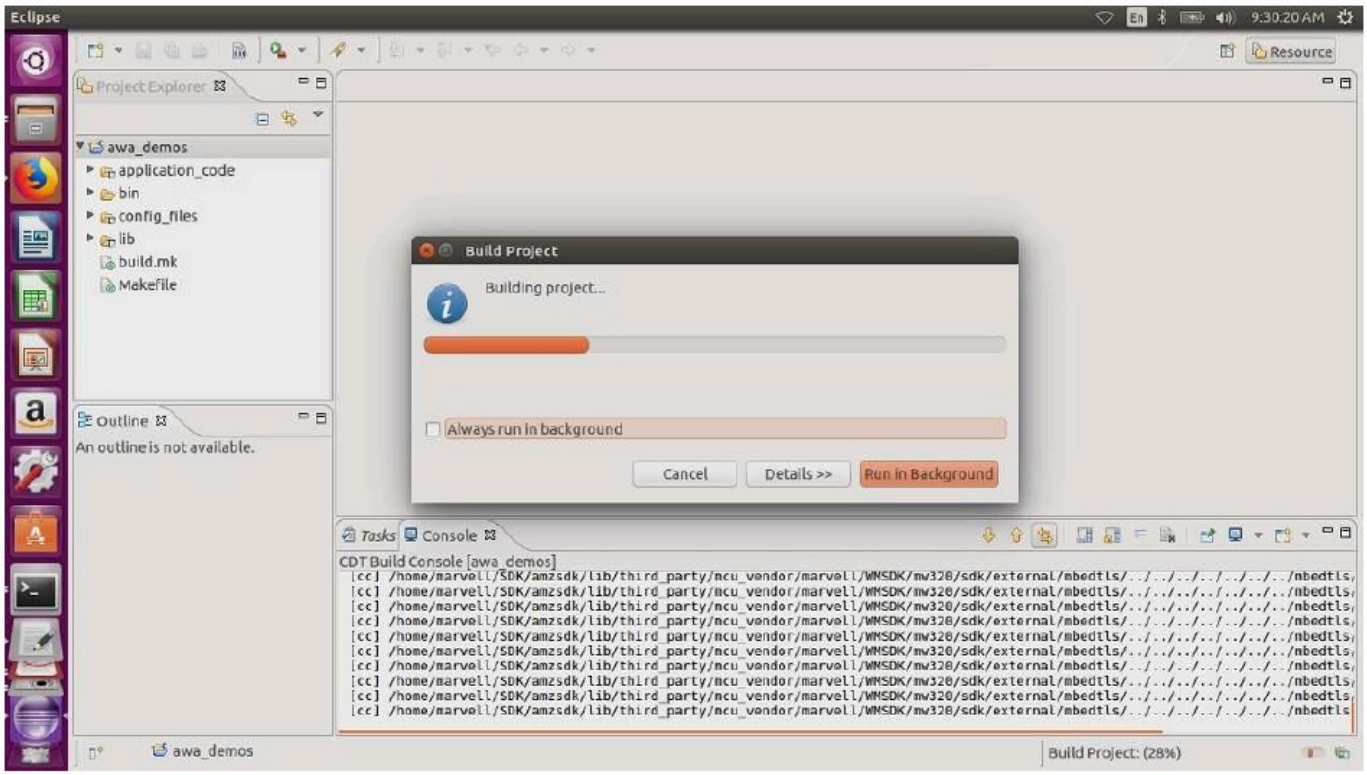
3. Scegli l'opzione per creare un progetto Makefile: con codice esistente come mostrato nella figura seguente.



4. Scegliete Sfoglia, specificate la directory del codice esistente e quindi scegliete Fine.



5. Nel riquadro di navigazione, seleziona aws_demos nell'esplorazione del progetto. Fai clic con il pulsante destro del mouse su aws_demos per aprire il menu, quindi scegli Build.



Se la compilazione ha esito positivo, genera il`build/cmake/vendors/marvell/mw300_rd/aws_demos.bin` file.

6. Usa gli strumenti della riga di comando per eseguire il flashing del file di layout (`layout.txt`), del binario Boot2 (`boot2.bin`), del firmware MCU (`aws_demos.bin`) e del firmware Wi-Fi.

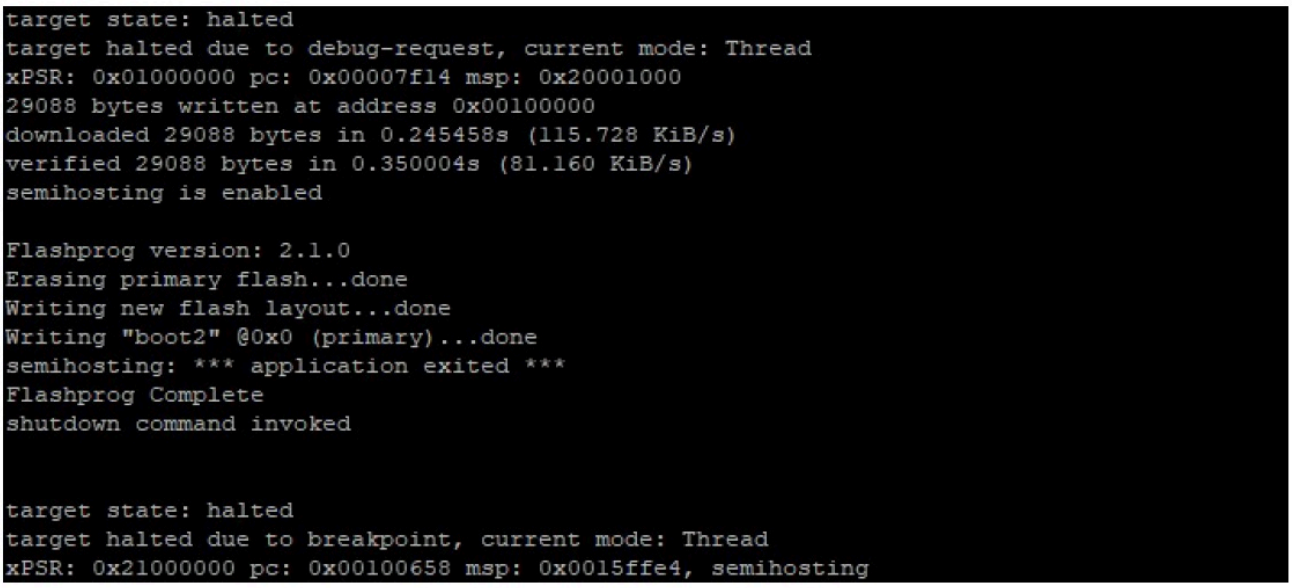
   a. Prima di eseguire il flashing dell'immagine del firmware, prepara il flash della scheda di sviluppo con i componenti comuni, Layout e Boot2. Utilizzare il seguente comando.

   ```
   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: l'utilità flashprog viene prima istruita a scrivere un layout sul 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 boot-loader 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. Assicurati di ottenere lo stesso risultato mostrato nella figura seguente.



b. Il firmware utilizza il chipset Wi-Fi per la sua funzionalità e il chipset Wi-Fi ha il proprio firmware che deve essere presente anche nel flash. Si utilizza l'`flashprog.py`utilità per eseguire il flashing del firmware Wi-Fi nello stesso modo in cui è stato eseguito il flashing del boot-loader boot2 e del firmware MCU. Usa 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
```

Assicurati 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. Usa i seguenti comandi per eseguire il flashing del firmware MCU.

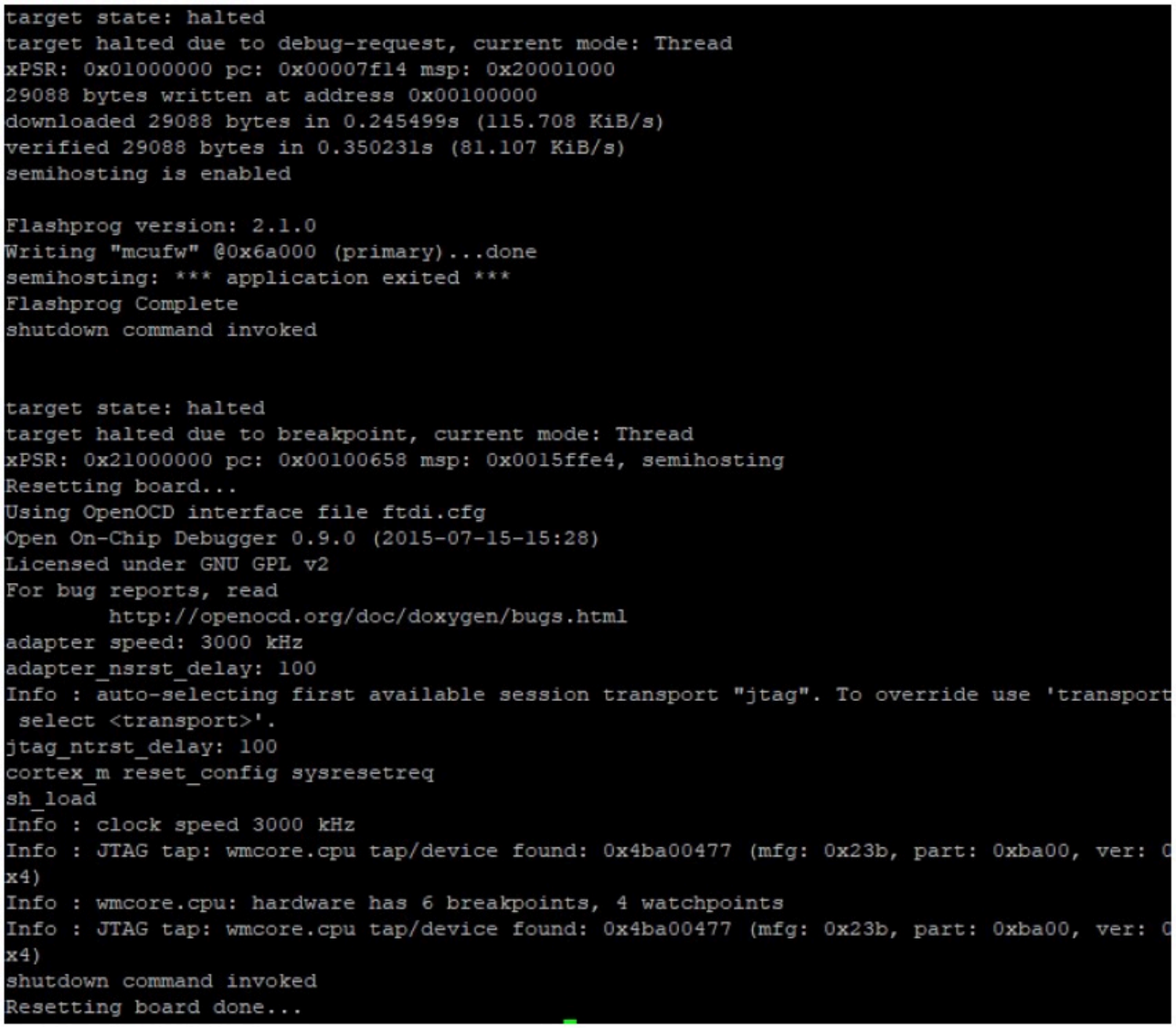
```
cd amzsdk_bundle-x.y.z
./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py --mcufw build/
cmake/vendors/marvell/mw300_rd/aws_demos.bin -r
```

d. Reimpostazione della scheda Dovresti vedere i log dell'app demo.

e. Per eseguire l'app di test, esegui il flashing del`aws_tests.bin` binario che si trova nella stessa directory.

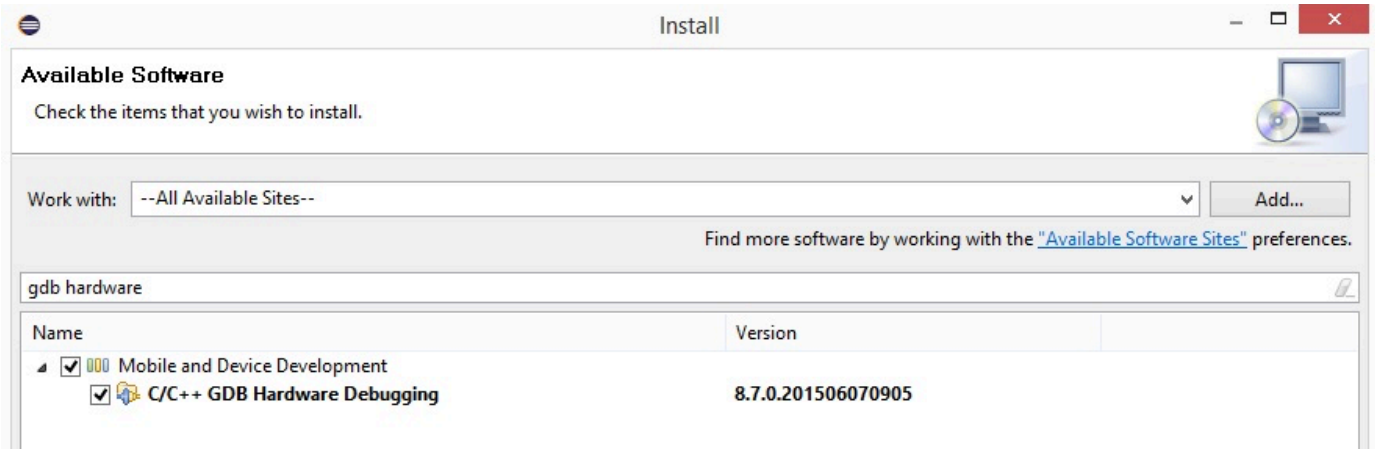
```
cd amzsdk_bundle-x.y.z
./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py --mcufw build/
cmake/vendors/marvell/mw300_rd/aws_tests.bin -r
```

L'output del comando visualizzato dovrebbe essere simile a quello mostrato nella 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 tue credenziali di rete. Vedi «Approvvigionamento» in[Crea ed esegui il progetto demo FreeRTOS](#).

Abilitazione di log aggiuntivi

- Abilita i registri specifici della scheda.

  Abilita le chiamate allawmstdio_init(UART0_ID, 0) funzioneprvMiscInitialization nelmain.c file per test o demo.

- Abilitazione dei log Wi-Fi

  Abilita la macroCONFIG_WLCMGR_DEBUG nel*freertos*/vendors/marvell/WMSDK/mw320/sdk/src/incl/autoconf.h file.

Usare GDB

Ti consigliamo di utilizzare i file digdb comandoarm-none-eabi-gdb and inclusi nel pacchetto SDK. Passa alla directory .

```
cd freertos/lib/third_party/mcu_vendor/marvell/WMSDK/mw320
```

Esegui il seguente comando (su una singola 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 MT7697hx

> **Important**
>
> Questa integrazione di riferimento è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Ti consigliamo di iniziare da qui quando crei un nuovo progetto. Se hai già un progetto FreeRTOS esistente basato sull'ormai obsoleto repository Amazon-FreerTOS, consulta il. Guida alla migrazione del repository Github di Amazon-FreeRTOS

Questo tutorial fornisce istruzioni per iniziare a usare il kit di sviluppo MT7697hx. MediaTek Se non disponi del kit di sviluppo MediaTek MT7697Hx, 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.
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, collegate il computer alla porta USB del kit di sviluppo MediaTek MT7697hx.

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 (serie MT76x7). MediaTek

Stabilire una connessione seriale

Connect la scheda al computer host con un cavo USB. In Gestione dispositivi di Windows viene visualizzata una porta COM. Per il debug, è possibile aprire una sessione sulla porta con uno strumento di utilità terminale come o. HyperTerminal TeraTerm

Monitoraggio dei messaggi MQTT in cloud

Prima di eseguire il progetto demo FreerTOS, puoi configurare il client MQTT nella console per monitorare AWS IoT i messaggi che il tuo dispositivo invia al Cloud. AWS

Per effettuare la sottoscrizione all'argomento MQTT con il client AWS IoT

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

Quando il progetto demo viene eseguito correttamente sul tuo dispositivo, vedi «Hello World!» inviato più volte all'argomento a cui ti sei iscritto.

Crea ed esegui il progetto demo FreerTOS con Keil MDK

Per creare il progetto dimostrativo 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 il kit di sviluppo MediaTek MT7697Hx 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 Target '**`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 kit di sviluppo MediaTek MT7697Hx.

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). Verifica che siano selezionati IRAM1 e 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 il kit di sviluppo MediaTek MT7697Hx 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).
7. Dal menu, scegli Flash, quindi Download (Scarica).

   µVision invia una notifica al completamento del download.
8. Premi il pulsante RESET sul tuo MediaTek kit di sviluppo MT7697Hx.
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`.

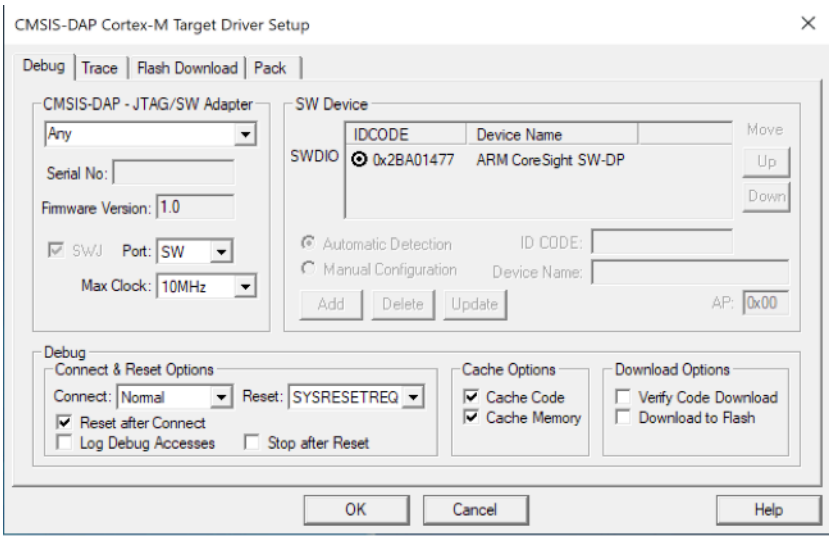
12. Imposta i punti di interruzione nel codice origine del progetto ed esegui il codice.

Risoluzione dei problemi relativi alle configurazioni del debugger dell'IDE

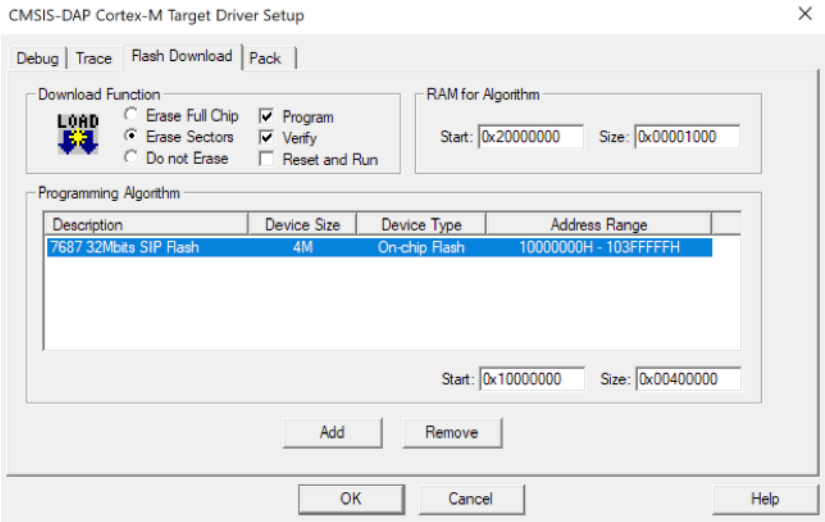
In caso di problemi durante il debug di un'applicazione, le impostazioni del debugger potrebbero essere errate.

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

Nozioni di base su Microchip Curiosity PIC32MZ EF

Important

Questa integrazione di riferimento è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Consigliamo di iniziare da qui quando crei un nuovo progetto. Se hai già un progetto FreeRTOS esistente basato sull'ormai obsoleto repository Amazon-FreeRTOS, consulta ilGuida alla migrazione del repository Github di Amazon-FreeRTOS.

Note

In accordo con Microchip, stiamo rimuovendo il Curiosity PIC32MZEF (DM320104) dal ramo principale del repository FreeRTOS Reference Integration e non lo includeremo più nelle nuove versioni. Microchip ha emesso un avviso ufficiale secondo cui il PIC32MZEF (DM320104) non è più consigliato per i nuovi progetti. È ancora possibile accedere ai progetti e al codice sorgente PIC32MZEF tramite i tag della versione precedente. Microchip consiglia ai clienti di utilizzare la scheda di sviluppo Curiosity PIC32MZ-EF-2.0 (DM320209) per nuovi progetti. La piattaforma PIC32mzv1 è ancora disponibile nella versione 202012.00 del repository FreeRTOS Reference Integration. Tuttavia, la piattaforma non è più supportata dalla versione 202107.00 del FreeRTOS Reference.

Questo tutorial fornisce istruzioni per iniziare a usare Microchip Curiosity PIC32MZ EF. Se non disponi del pacchetto Microchip Curiosity PIC32MZ EF, visita il catalogo dei dispositiviAWS partner per acquistarne uno dal nostro partner.

Il bundle include le seguenti voci:

- Scheda di sviluppo Curiosity PIC32MZ EF
- MikroElectronika Scheda USB UART Click
- MikroElectronika WiFi Bacheca con 7 clic
- Scheda figlia PIC32 LAN8720 PHY

È inoltre necessario disporre delle seguenti voci per il debug:

- Debugger MPLAB Snap In-Circuit
- (Facoltativo) Kit cavo di programmazione PICkit 3

Prima di iniziare, devi configurareAWS IoT e scaricare FreeRTOS per connettere il tuo dispositivo alAWS Cloud. 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 della directory di download di FreeRTOS possono causare errori di compilazione.
- Poiché il codice sorgente può contenere collegamenti simbolici, se utilizzi Windows per estrarre l'archivio, potresti dover:
  - Abilita la modalità sviluppatore o,
  - Usa una console con privilegi di amministratore.

  In questo modo, Windows può creare correttamente collegamenti simbolici quando estrae l'archivio. Altrimenti, i collegamenti simbolici verranno scritti come file normali che

contengono i percorsi dei collegamenti simbolici come testo o sono vuoti. Per ulteriori informazioni, consulta il post del blog Symlinks in Windows 10! .

Se usi Git in Windows, devi abilitare la modalità sviluppatore oppure devi:

- `core.symlinks`Imposta su true con il seguente comando:

```
git config --global core.symlinks true
```

- Usa una console con privilegi di amministratore ogni volta che usi un comando git che scrive nel sistema (ad esempiogit pull,git clone, egit submodule update --init --recursive).

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 Microchip Curiosity PIC32MZ EF

1. Collega la click board MikroElectronika USB UART al connettore MicroBus 1 sul Microchip Curiosity PIC32MZ EF.
2. Collegare la scheda figlia PIC32 LAN8720 PHY al collettore J18 su Microchip Curiosity PIC32MZ EF.
3. Collega la scheda MikroElectronika USB UART click 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 scheda MikroElectronika Wi-Fi 7 click al connettore MicroBus 2 sul Microchip Curiosity PIC32MZ EF. Consultare Configurazione delle demo di FreerTOS.

- Per utilizzare Ethernet per collegare la scheda Microchip Curiosity PIC32MZ EF a Internet, collegare la scheda figlia PIC32 LAN8720 PH all'intestazione J18 del Microchip Curiosity PIC32MZ EF. Collegare un'estremità di un cavo Ethernet alla scheda figlia LAN8720 PHY. Collegare l'altra estremità al router o a un'altra porta Internet. È inoltre necessario definire la macro del preprocessore`PIC32_USE_ETHERNET`.

5. Se non lo si è già fatto, saldare il connettore ad angolo al collettore ICSP in Microchip Curiosity PIC32MZ EF.
6. Collegare un'estremità del cavo ICSP dal kit del cavo di programmazione PICkit 3 a Microchip Curiosity PIC32MZ EF.

   Se non si dispone del kit del cavo di programmazione PICkit 3, è possibile allora utilizzare i ponticelli dei cavi M-F Dupont per effettuare 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.

   Verificare che il cablaggio al Pin 1 su Microchip Curiosity PIC32MZ EF, indicato da un cerchio bianco, si allinei con il Pin 1 sul debugger snap MPLAB.

   Per ulteriori informazioni su MPLAB Snap Debugger, vedere la scheda informativa del debugger MPLAB Snap In-Circuit.

Configurazione dell'hardware Microchip Curiosity PIC32MZ EF utilizzando PICkit On Board (PKOB)

Consigliamo di seguire la procedura di configurazione descritta nella sezione precedente. Tuttavia, puoi valutare ed eseguire le demo di FreeRTOS con il debug di base utilizzando il programmatore/debugger integrato PicKit On Board (PKOB) seguendo questi passaggi.

1. Collega la click board MikroElectronika USB UART al connettore MicroBus 1 sul Microchip Curiosity PIC32MZ EF.
2. Per connettere la scheda a Internet, effettuare una delle seguenti operazioni:

   - Per utilizzare il Wi-Fi, collega la scheda MikroElectronika Wi-Fi 7 click al connettore MicroBus 2 sul Microchip Curiosity PIC32MZ EF. Attenersi alla procedura "Per configurare il Wi-Fi" in Configurazione delle demo di FreerTOS.

- Per utilizzare Ethernet per collegare la scheda Microchip Curiosity PIC32MZ EF a Internet, collegare la scheda figlia PIC32 LAN8720 PH all'intestazione J18 del Microchip Curiosity PIC32MZ EF. Collegare un'estremità di un cavo Ethernet alla scheda figlia LAN8720 PHY. Collegare l'altra estremità al router o a un'altra porta Internet. È inoltre necessario definire la macro del preprocessore`PIC32_USE_ETHERNET`.

3. Collegare la porta USB micro-B denominata "USB DEBUG" sulla scheda Microchip Curiosity PIC32MZ EF al computer utilizzando un cavo da USB tipo A a USB micro-B.
4. Collega la scheda MikroElectronika USB UART click 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 compilare il progetto, è necessario utilizzare le versioni degli strumenti MPLAB compatibili con Harmony v2, ad esempio la versione 2.10 di MPLAB XC32 Compiler 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**
   >
   > FreeRTOSAWS Reference Integrations v202007.00 è attualmente supportato solo su MPLab v5.35. Le versioni precedenti di FreeRTOSAWS Reference Integrations sono supportate su MPLabV5.40.

   download MPLABv5.35

   - Ambiente di sviluppo integrato MPLAB X per Windows
   - Ambiente di sviluppo integrato MPLAB X per macOS
   - Ambiente di sviluppo integrato MPLAB X per Linux

Download MPLab più recenti (MPLab v5.40)

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

3. Installare MPLAB XC32 Compiler:

- MPLAB XC32/32++ Compiler per Windows
- MPLAB XC32/32++ Compiler per macOS
- MPLAB XC32/32++ Compiler 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

Monitoraggio dei messaggi MQTT nel cloud

Prima di eseguire il progetto demo di FreeRTOS, puoi configurare il client MQTT nellaAWS IoT console per monitorare i messaggi che il tuo dispositivo invia alAWS Cloud.

Per effettuare la sottoscrizione all'argomento MQTT con il client AWS IoT

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

Quando il progetto demo viene eseguito correttamente sul tuo dispositivo, viene visualizzato «Hello World!» inviato più volte all'argomento a cui ti sei iscritto.

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 connettersi tramite l'IDE MPLAB

1. Nell'IDE MPLAB, fai cli con il pulsante destro del mouse sul progetto e scegli 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 utilizzare CMake 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. Utilizza il comando seguente per generare i 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 binari Hexmate e toolchain, come i`C:\Program Files\Microchip\xc32\v2.40\bin` percorsi`C:\Program Files (x86)\Microchip\MPLABX\v5.35\mplab_platform\bin` and.

3. Cambia le directory nella directory di build (*build-directory*), quindi eseguimake da quella directory.

Per ulteriori informazioni, consulta Utilizzo di CMake con FreeRTOS.

Per utilizzare Ethernet per la connessione, è necessario definire la macro del preprocessore`PIC32_USE_ETHERNET`.

Risoluzione dei problemi

Per informazioni sulla risoluzione dei problemi, consulta Nozioni di base sulla risoluzione dei problemi.

Nozioni di base su Nordic nRF52840-DK

> **Important**
>
> Questa integrazione di riferimento è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Consigliamo di iniziare da qui quando si crea un nuovo progetto. Se hai già un

progetto FreeRTOS esistente basato sull'ormai obsoleto repository Amazon-FreeRTOS, consulta ilGuida alla migrazione del repository Github di Amazon-FreeRTOS.

Questo tutorial fornisce istruzioni per iniziare a usare Nordic nRF52840-DK. Se non disponi del dispositivo Nordic nRF52840-DK, visita il catalogo dei dispositiviAWS partner per acquistarne uno dal nostro partner.

Prima di iniziare, è necessario ConfigurazioneAWS 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

Collega il computer host alla porta USB etichettata J2, situata direttamente sopra il supporto per batteria a bottone sulla scheda Nordic nRF52840.

Per ulteriori informazioni sulla configurazione di Nordic nRF52840-DK, consulta il documento nRF52840 Development Kit User Guide.

Configurazione dell'ambiente di sviluppo

Scaricare e installare Segger Embedded Studio

FreeRTOS supporta Segger Embedded Studio come ambiente di sviluppo per il Nordic nRF52840-DK.

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 SDK mobili 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 (Strumenti), Terminal Emulator (Emulatore di terminale), Connect ***porta*** (115200,N,8,1) (Connetti porta (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.

Scaricare e configurare FreeRTOS

Dopo aver configurato l'hardware e l'ambiente, puoi scaricare FreeRTOS.

Scarica FreeRTOS

Per scaricare FreeRTOS per Nordic nRF52840-DK, vai alla GitHub pagina 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 della directory di download di FreeRTOS possono causare errori di compilazione.
> - Poiché il codice sorgente può contenere collegamenti simbolici, se utilizzi Windows per estrarre l'archivio, potresti dover:
>   - Abilita la modalità sviluppatore o,

- Usa una console con privilegi di amministratore.

In questo modo, Windows può creare correttamente collegamenti simbolici quando estrae l'archivio. Altrimenti, i collegamenti simbolici verranno scritti come file normali che contengono i percorsi dei collegamenti simbolici come testo o sono vuoti. Per ulteriori informazioni, consulta il post del blog Symlinks in Windows 10! .

Se usi Git in Windows, devi abilitare la modalità sviluppatore oppure devi:

- `core.symlinks`Imposta su true con il seguente comando:

```
git config --global core.symlinks true
```

- Usa una console con privilegi di amministratore ogni volta che usi un comando git che scrive nel sistema (ad esempiogit pull,git clone, egit submodule update --init --recursive).

Configurare il progetto

Per abilitare la demo, è necessario configurare il progetto con cui lavorareAWS IoT. Per configurare il progetto per l'utilizzo con AWS IoT, il dispositivo deve essere registrato come oggetto AWS IoT. È necessario aver registrato il dispositivo in fase di ConfigurazioneAWS IoT e Amazon Cognito per FreeRTOS Bluetooth Low Energy.

Per configurare l'endpoint AWS IoT

1. Accedi alla console AWS IoT.
2. Nel pannello di navigazione scegli Settings (Impostazioni).

   L'AWS IoTendpoint viene visualizzato nella casella di testo Device data 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). Prendere nota del nome dell'oggetto AWS IoT per il dispositivo.
4. Con l'endpoint AWS IoT e il nome dell'oggetto AWS IoT a disposizione, aprire *freertos*`/demos/include/aws_clientcredential.h` nell'IDE e specificare i valori per le seguenti costanti `#define`:

   - `clientcredentialMQTT_BROKER_ENDPOINT` *L'endpoint AWS IoT*

- clientcredentialIOT_THING_NAME *Il nome dell'oggetto AWS IoT della scheda*

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. Aprivendors/nordic/boards/nrf52840-dk/aws_demos/config_files/aws_demo_config.h e definisci unoCONFIG_OTA_MQTT_BLE_TRANSPORT_DEMO_ENABLED oCONFIG_OTA_HTTP_BLE_TRANSPORT_DEMO_ENABLED come in questo esempio.

```
/* To run a particular demo you need to define one of these.
 * Only one demo can be configured at a time
 *
 *          CONFIG_BLE_GATT_SERVER_DEMO_ENABLED
 *          CONFIG_MQTT_BLE_TRANSPORT_DEMO_ENABLED
 *          CONFIG_SHADOW_BLE_TRANSPORT_DEMO_ENABLED
 *          CONFIG_OTA_MQTT_BLE_TRANSPORT_DEMO_ENABLED
 *          CONFIG_OTA_HTTP_BLE_TRANSPORT_DEMO_ENABLED
 *          CONFIG_POSIX_DEMO_ENABLED
 *
 *  These defines are used in iot_demo_runner.h for demo selection */

#define CONFIG_OTA_MQTT_BLE_TRANSPORT_DEMO_ENABLED
```

3. 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 UUID specifici del fornitore. Aumenta questo numero di 1 quando aggiungi un nuovo UUID specifico del fornitore.

   - NRF_SDH_BLE_GATTS_ATTR_TAB_SIZE

Dimensione della tabella degli attributi in byte. La dimensione deve essere un multiplo di 4. Questo valore indica la quantità di memoria impostata dedicata alla tabella degli attributi (inclusa la dimensione caratteristica), quindi varierà da progetto a progetto. Se superi la dimensione della tabella degli attributi, riceverai un errore NRF_ERROR_NO_MEM. Se si modifica NRF_SDH_BLE_GATTS_ATTR_TAB_SIZE in genere è necessario riconfigurare anche le impostazioni della RAM.

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. Richiedi una licenza gratuita, quindi, durante la configurazione, scegli Attiva la tua 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 dell'applicazione demo MQTT over 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 relativi a Getting Started with FreeRTOS, consultaNozioni di base sulla risoluzione dei problemi.

Guida introduttiva a Nuvoton NuMaker -IoT-M487

> **Important**
>
> Questa integrazione di riferimento è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Consigliamo di iniziare da qui quando crei un nuovo progetto. Se disponi già di un progetto FreeRTOS esistente basato sull'ormai obsoleto repository Amazon-FreeRTOS, consulta ilGuida alla migrazione del repository Github di Amazon-FreeRTOS.

Questo tutorial fornisce istruzioni per iniziare a usare la scheda di sviluppo Nuvoton NuMaker -IoT-M487. La serie di microcontrollori include moduli Ethernet e Wi-Fi RJ45 integrati. Se non disponi di Nuvoton NuMaker -IoT-M487, visita il catalogo dei dispositiviAWS partner per acquistarne uno dal nostro partner.

Prima di iniziare, è necessario configurareAWS IoT il software FreeRTOS per connettere la scheda di sviluppo alAWS Cloud. Per istruzioni, consulta Fase iniziale. In questo tutorial, il percorso della directory di download di FreeRTOS viene chiamato*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 un'applicazione demo FreeRTOS in 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 i MCU della serie Nuvoton Cortex-M4. Keil MDK è supportato solo su Windows.

Per installare lo strumento di sviluppo per NuMaker -IoT-M487

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 dello strumento di sviluppo Nuvoton.

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).

È possibile utilizzare il client MQTT nella console AWS IoT per monitorare i messaggi inviati dal dispositivo al cloud AWS.

Per effettuare la sottoscrizione all'argomento MQTT con il client AWS IoT

1. Accedi alla console AWS IoT.
2. Nel riquadro di navigazione, scegli Test, quindi scegli MQTT test client 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 di FreeRTOS di

1. Collegare la scheda Numaker-IoT-M487 al computer 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)).

   Nel client MQTT nella console AWS IoT, dovrebbero essere visibili i messaggi MQTT inviati dal dispositivo.

Utilizzo di CMake con FreeRTOS

Puoi anche usare CMake per creare ed eseguire le applicazioni demo o le applicazioni demo di FreeRTOS 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 di 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 nellaAWS IoT console per monitorare i messaggi che il tuo dispositivo invia alAWS Cloud.

Per effettuare la sottoscrizione all'argomento MQTT con il client AWS IoT

1. Accedi alla console AWS IoT.
2. Nel riquadro di navigazione, scegli Test, quindi scegli MQTT test client 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 tuo computer host, apri il prompt dei comandi e vai alla cartella *freertos*.
2. Creare una cartella per i file compilati generati. Questa cartella verrà definita *BUILD_FOLDER*.
3. Genera i file della build per la dimostrazione Wi-Fi o Ethernet.

   - Per Wi-Fi:

Vai alla directory che contiene i file di origine per il progetto demo di 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:

  Vai alla directory che contiene i file di origine per il progetto demo di 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
```

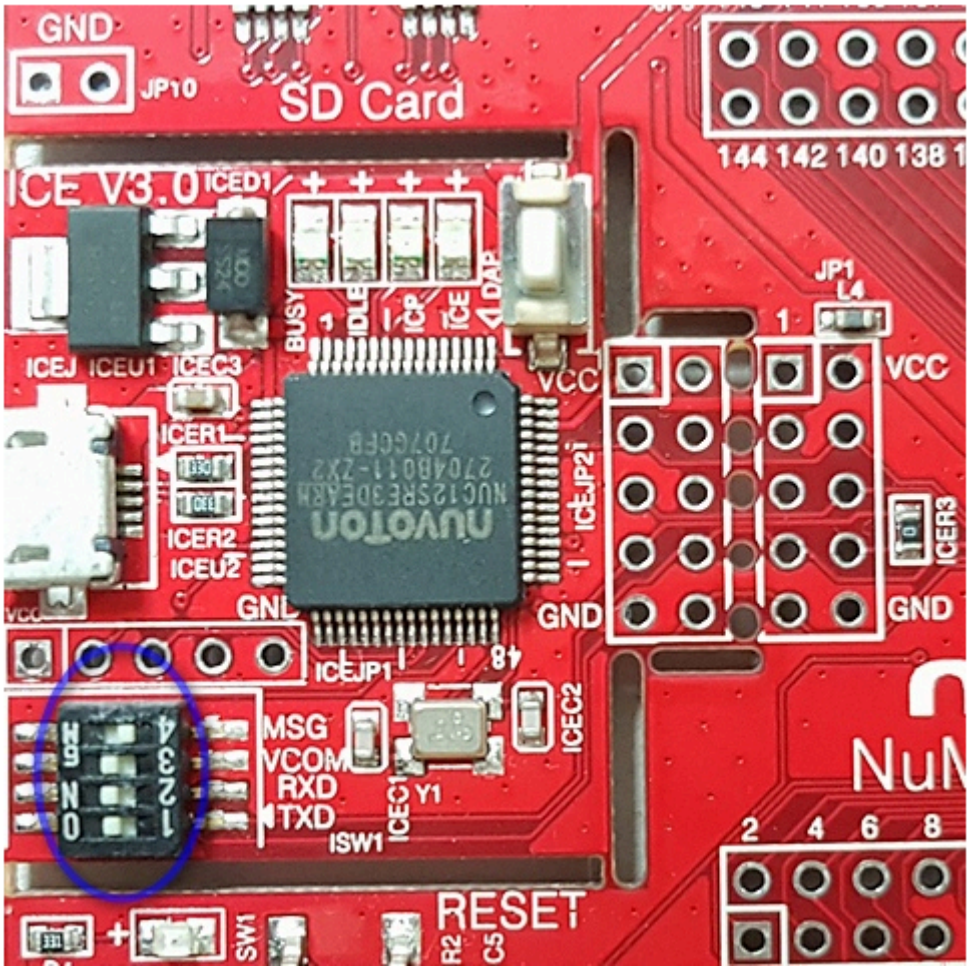
   A questo punto, il file binario `aws_demos.bin` deve trovarsi nella cartella *BUILD_FOLDER*`/vendors/Nuvoton/boards/numaker_iot_m487_wifi`.

5. Per configurare la scheda per la modalità lampeggiante, assicurati che lo switch MSG (No.4 di ISW1 su ICE) sia attivato. 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. Seguire le istruzioni riportate in Installazione di un emulatore di terminale:
7. Esegui il progetto demo copiando il file binario generato sul dispositivo.

   Se hai effettuato la sottoscrizione all'argomento MQTT con il client AWS IoT MQTT, dovresti visualizzare i messaggi MQTT inviati dal dispositivo nella AWS IoT console.

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 dispositivo a Keil MDK (IDE) tramite Nu-Link, assicurati che lo switch MSG (No.4 di ISW1 su ICE) sia OFF, come mostrato.



In caso di problemi durante la configurazione dell'ambiente di sviluppo o la connessione alla scheda, contatta Nuvoton.

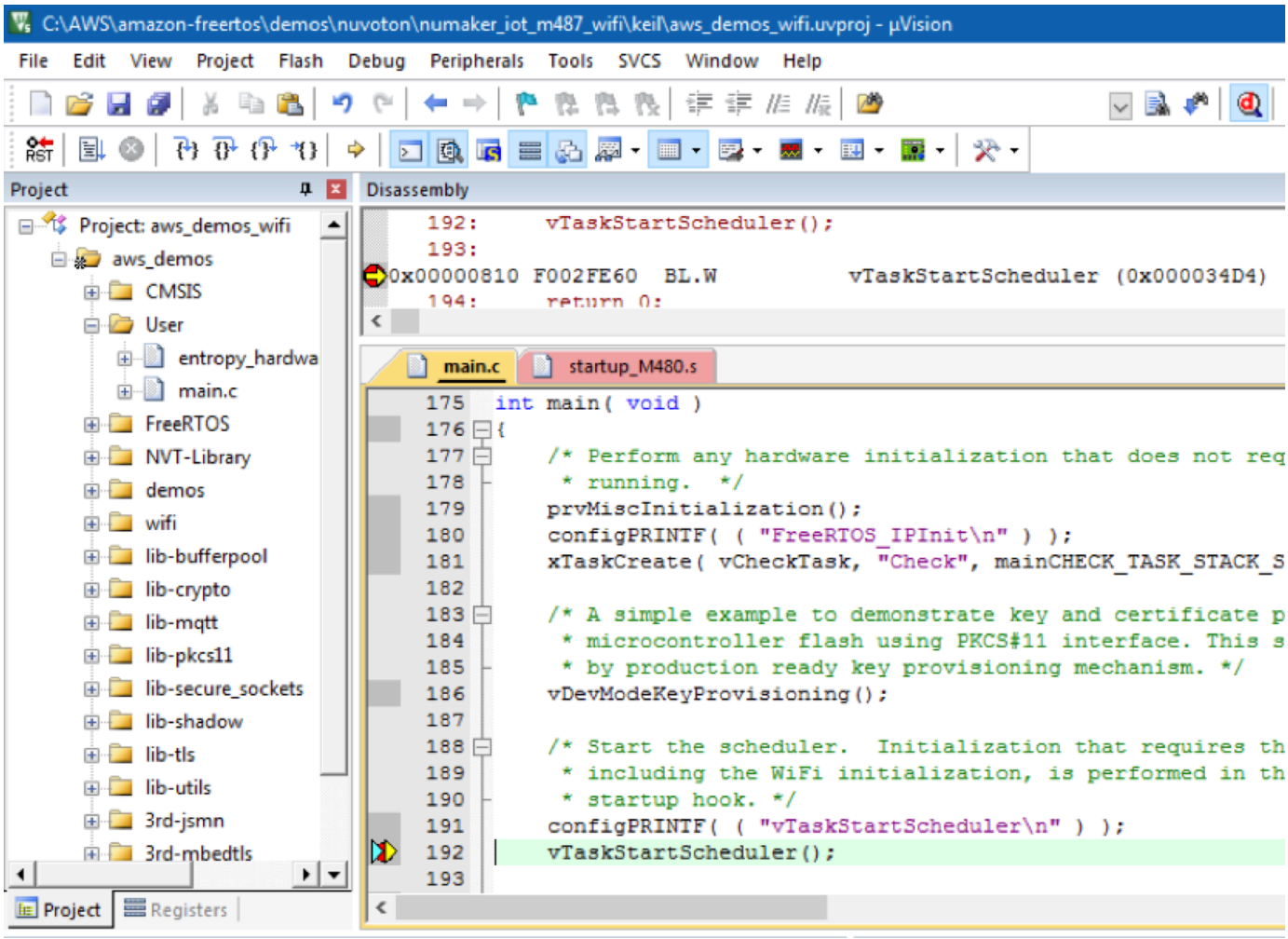
Eseguire il debug dei progetti FreeRTOS in Keil µVision

Per avviare una sessione di debug in Keil µVision

1. Aprire Keil µVision.
2. Segui i passaggi per creare il progetto demo di FreeRTOS inCrea 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:



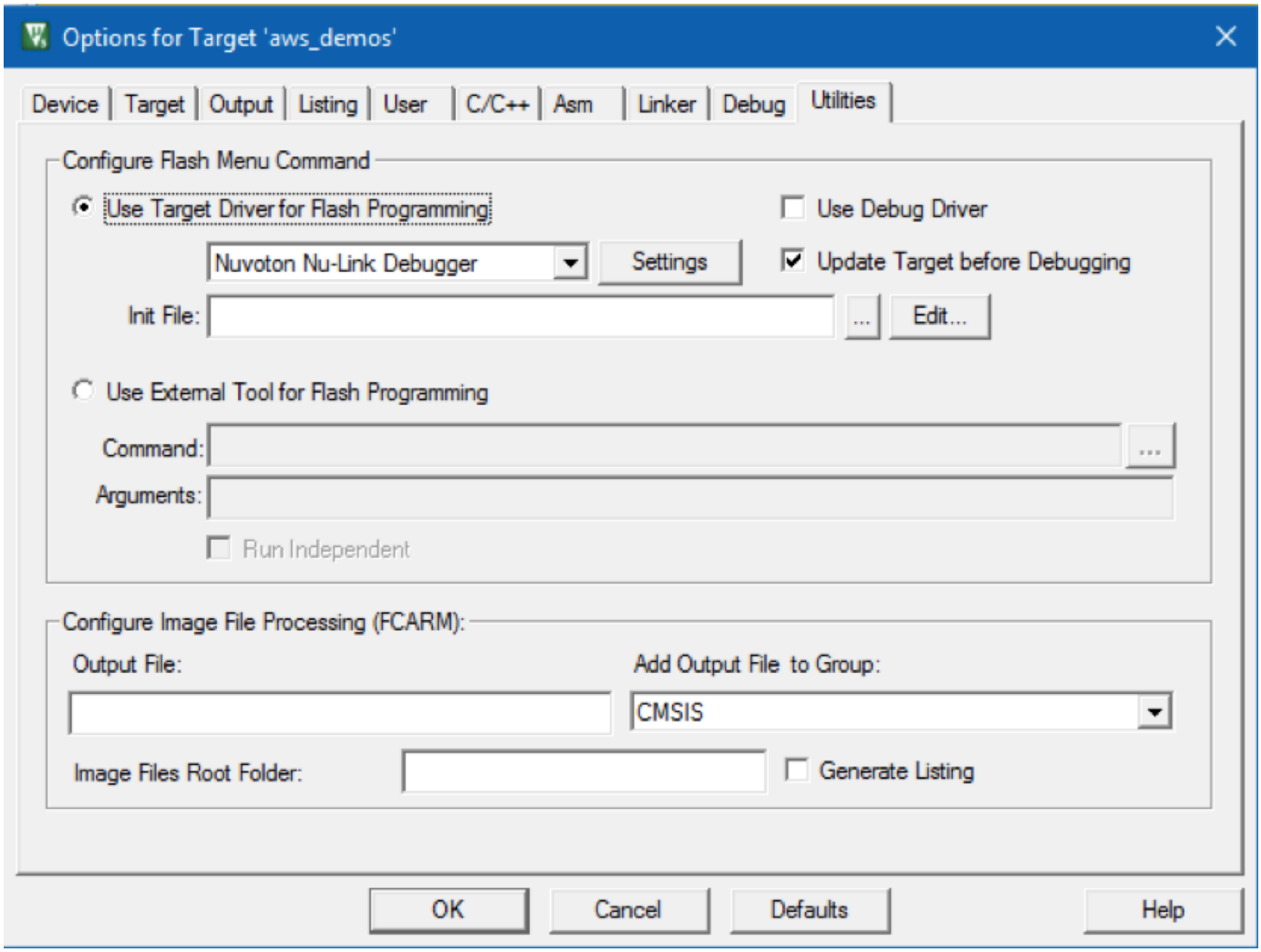
Risoluzione dei problemi delle impostazioni di debug di µVision

Se si verificano problemi durante il debug di un'applicazione, controllare che le impostazioni di debug siano impostate correttamente in Keil µVision.

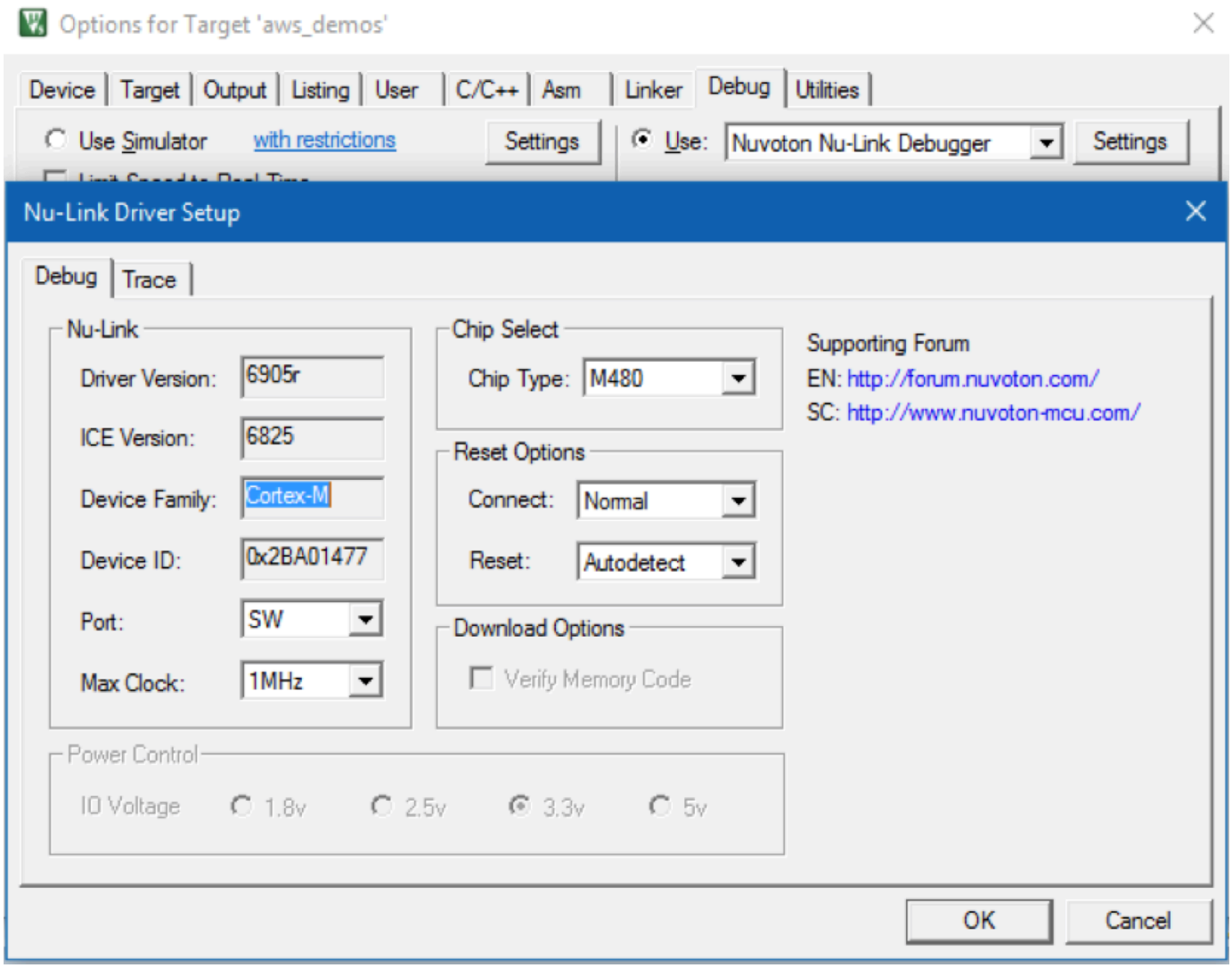
Per verificare che le impostazioni di debug di µVision siano corrette

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.

Nozioni di base su NXP LPC54018 IoT Module

> **Important**
>
> Questa integrazione di riferimento è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Ti consigliamo di iniziare da qui quando crei un nuovo progetto. Se hai già un progetto FreeRTOS esistente basato sull'ormai obsoleto repository Amazon-FreerTOS, consulta il. Guida alla migrazione del repository Github di Amazon-FreeRTOS

Questo tutorial fornisce istruzioni per iniziare a usare NXP LPC54018 IoT Module. Se non disponi di un modulo IoT NXP LPC54018, visita AWS il Partner Device Catalog per acquistarne uno dal nostro partner. Utilizzare un cavo USB per collegare NXP LPC54018 IoT Module 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 LPC54018

- Collegare il computer alla porta USB su NXP LPC54018.

Per configurare il debugger JTAG

È necessario un debugger JTAG per avviare ed eseguire il debug del codice in esecuzione sulla scheda NXP LPC54018. FreerTOS è stato testato utilizzando un modulo IoT OM40006. Per ulteriori informazioni sui debugger supportati, consulta il manuale per l'utente per il modulo IoT NXP LPC54018 disponibile nella pagina prodotto del modulo IoT OM40007 LPC54018 .

1. Se utilizzi un debugger di modulo IoT OM40006, usa un convertitore per collegare il connettore 20 pin dal debugger al connettore 10 pin sul modulo IoT NXP.
2. Collega l'NXP LPC54018 e il debugger di modulo IoT OM40006 alle porte USB sul computer utilizzando cavi da mini-USB a USB.

Configurazione dell'ambiente di sviluppo

FreerTOS supporta due IDE per il modulo IoT NXP LPC54018: IAR Embedded Workbench e MCUXpresso.

Prima di iniziare, installa uno di questi IDE.

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 installare MCUXpresso da NXP

1. Scaricare ed eseguire i programma di installazione di MCUXpresso da NXP.

   > **Note**
   >
   > Sono supportate le versioni 10.3.x e successive.

2. Passare a SDK MCUXpresso e scegliere Build your SDK (Crea 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 (Schede), scegliere LPC54018-IoT-Module.

6. Verificare i dettagli sull'hardware e quindi scegliere Build MCUXepresso SDK (Crea SDK MCUXepresso).
7. L'SDK per Windows che utilizza l'IDE MCUXpresso è già stato creato. 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. Avviare l'IDE MCUXpresso e scegliere la scheda Installed SDK (SDK installati).
9. Trascinare il file di archivio dell'SDK scaricato nella finestra (SDK installati).

In caso di problemi durante l'installazione, consulta Supporto NXP o Risorse per gli sviluppatori NXP.

Monitoraggio dei messaggi MQTT in cloud

Prima di eseguire il progetto demo FreerTOS, puoi configurare il client MQTT nella console per monitorare AWS IoT i messaggi che il tuo dispositivo invia al Cloud. AWS

Per effettuare la sottoscrizione all'argomento MQTT con il client AWS IoT

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

Quando il progetto demo viene eseguito correttamente sul tuo dispositivo, vedi «Hello World!» inviato più volte all'argomento a cui ti sei iscritto.

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 FreerTOS nell'IDE McUXpresso

1. Aprire MCUXpresso e dal menu File scegliere Open Projects From File System (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. (Questa operazione è obbligatoria a causa di un problema con il debugger MCUXpresso per NXP54018-IoT-Module).

4. Quando il debugger si arresta sul punto di interruzione in `main()`, dal menu Debug, scegliere Go (Vai).

Risoluzione dei problemi

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

Nozioni di base su Renesas Starter Kit+ per RX65N-2 MB

> **Important**
>
> Questa integrazione di riferimento è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Consigliamo di iniziare da qui quando crei un nuovo progetto. Se hai già un progetto FreeRTOS esistente basato sull'ormai obsoleto repository Amazon-FreeRTOS, consulta ilGuida alla migrazione del repository Github di Amazon-FreeRTOS.

Questo tutorial fornisce istruzioni per iniziare a usare Renesas Starter Kit+ per RX65N-2 MB. Se non disponi di Renesas RSK+ per RX65N-2 MB, visita il catalogo dei dispositivi deiAWS partner e acquistane uno dai nostri partner.

Prima di iniziare, devi configurareAWS IoT e scaricare FreeRTOS per connettere il tuo dispositivo alAWS Cloud. Per istruzioni, consulta Fase iniziale. In questo tutorial, il percorso della directory di download di FreeRTOS viene chiamato*`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

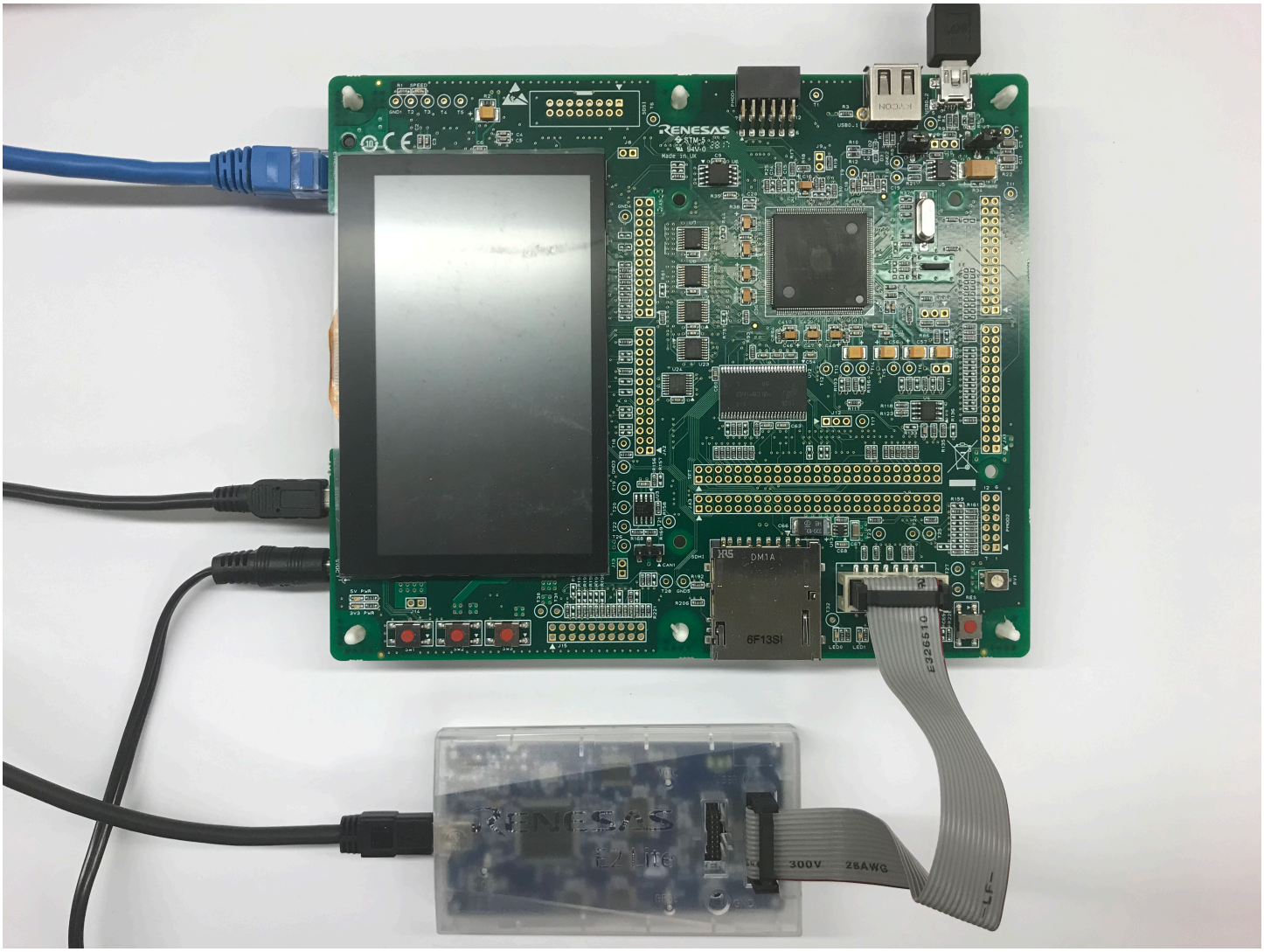
Per configurare RSK+ for RX65N-2MB

1. Collegare l'adattatore di alimentazione positive+5V al connettore PWR su RSK+ for RX65N-2 MB.
2. Collegare il computer alla porta USB2.0 FS su RSK+ for USB2-RX65N.
3. Collegare il computer alla porta USB-porta seriale su RSK+forRX65N-2 MB.
4. Collegare un router o una porta Ethernet connessa a Internet alla porta Ethernet su RSK +forRX65N-2MB.

Per configurare il modulo E2 Lite Debugger

1. Utilizzare il cavo a nastro da 14 pin per collegare il modulo E2 Lite Debugger alla porta 'E1/E2 Lite' su RSK+for RX65N-2 MB.
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. Quando il debugger è connesso al computer host e a RSK+for RX65N-2 MB, viene avviata l'installazione dei driver di E2 Lite Debugger.

   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+ per RX65N-2 MB, usa l'IDE da 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 quindi 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, consultare 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 Package del 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 e2studio

1. Avviare e2studio 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 e2studio, 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 e2studio 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 usare Code Generator, scegli OK.
6. Viene visualizzata la finestra Restart e 2 studio. Scegli OK.
7. e2studio 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 della freccia Vai al banco di lavoro di 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.

Monitoraggio dei messaggi MQTT in cloud

Prima di eseguire il progetto demo di FreeRTOS, puoi configurare il client MQTT nellaAWS IoT console per monitorare i messaggi che il tuo dispositivo invia alAWS Cloud.

Per effettuare la sottoscrizione all'argomento MQTT con il client AWS IoT

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

Quando il progetto demo viene eseguito correttamente sul tuo dispositivo, viene visualizzato «Hello World!» inviato più volte all'argomento a cui ti sei iscritto.

Esegui il progetto FreeRTOS

Per eseguire il progetto in e^{2}studio

1. Confermare di aver connesso il modulo E2 Lite Debugger a RSK+for RX65N-2 MB
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.

Per gli ultimi progetti rilasciati da Renesas, consulta il`renesas-rx` fork dela`mazon-freertos` repository su GitHub.

Risoluzione dei problemi

Per informazioni generali sulla risoluzione dei problemi relativi a Getting Started with FreeRTOS, consultaNozioni di base sulla risoluzione dei problemi.

Nozioni di base su STMicroelectronics STM32L4 Discovery Kit IoT Node

> **Important**
>
> Questa integrazione di riferimento è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Ti consigliamo di iniziare da qui quando crei un nuovo progetto. Se hai già un progetto FreeRTOS esistente basato sull'ormai obsoleto repository Amazon-FreerTOS, consulta il. Guida alla migrazione del repository Github di Amazon-FreeRTOS

Questo tutorial fornisce istruzioni per iniziare a usare STMicroelectronics STM32L4 Discovery Kit IoT Node. Se non disponi già del nodo IoT STMicroelectronics STM32L4 Discovery Kit, visita AWS il 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, consulta STM32L4 Discovery kit IoT node, low-power wireless, Bluetooth Low Energy, NFC, SubGHz, 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 configurareAWS IoT, scaricare FreerTOS e Wi-Fi 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. Andare su OpenSTM32.org.
2. Effettuare la registrazione sulla pagina Web OpenSTM32. È necessario effettuare l'accesso per scaricare System Workbench.
3. Andare sul programma di installazione di System STM32 Workbench per scaricare e installare System Workbench.

In caso di problemi durante l'installazione, consulta le domande frequenti sul sito Web di System Workbench.

Crea ed esegui il progetto demo FreerTOS

Importa la demo di FreerTOS nell'ambiente di lavoro del sistema STM32

1. Aprire STM32 System Workbench e immettere un nome per un nuovo workspace.
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. Per importare il progetto in STM32 System Workbench, scegliere Finish (Termina).
6. Dal menu Project (Progetto), scegliere Build All (Crea tutti). Confermare che il progetto è stato compilato senza errori.

Monitoraggio dei messaggi MQTT in cloud

Prima di eseguire il progetto demo FreerTOS, puoi configurare il client MQTT nella console per monitorare AWS IoT i messaggi che il tuo dispositivo invia al Cloud. AWS

Per effettuare la sottoscrizione all'argomento MQTT con il client AWS IoT

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

Quando il progetto demo viene eseguito correttamente sul tuo dispositivo, vedi «Hello World!» inviato più volte all'argomento a cui ti sei iscritto.

Esegui il progetto demo FreerTOS

1. Utilizzare un cavo USB per collegare STM32L4 Discovery Kit IoT Node di STMicroelectronics al computer. (Consulta la documentazione del produttore fornita con la scheda per conoscere la porta USB corretta da utilizzare.)
2. Da Project Explorer, fare clic con il pulsante destro del mouse su `aws_demos`, scegliere Debug As (Debug come) e quindi scegliere Ac6 STM32 C/C++ Application.

   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 Run (Esegui), scegliere Debug Configurations (Configurazioni di debug).
   2. Scegliere aws_demos Debug. (Potrebbe essere necessario espandere Ac6 STM32 Debugging (Debugging di Ac6 STM32).)
   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).

Usare CMake con FreerTOS

Se preferisci non utilizzare un IDE per lo sviluppo di FreerTOS, puoi in alternativa utilizzare CMake per creare ed eseguire le applicazioni demo o le applicazioni che hai sviluppato utilizzando editor di codice e strumenti di debug di terze parti.

Creare innanzitutto una cartella per i file compilati 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 `arm-none-eabi-gcc` non è incluso nel percorso della shell, è inoltre necessario impostare la variabile CMake `AFR_TOOLCHAIN_PATH`. Per esempio:

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

Per ulteriori informazioni sull'utilizzo di CMake con FreerTOS, vedere. Utilizzo di CMake con FreeRTOS

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, consulta STM32L4 Discovery kit IoT node, low-power wireless, Bluetooth Low Energy, NFC, SubGHz, 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

Nozioni di base su Texas Instruments CC3220SF-LAUNCHXL

> **Important**
>
> Questa integrazione di riferimento è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Ti consigliamo di iniziare da qui quando crei un nuovo progetto. Se disponi già di un progetto FreeRTOS esistente basato sull'ormai obsoleto repository Amazon-FreerTOS, consulta il. Guida alla migrazione del repository Github di Amazon-FreeRTOS

Questo tutorial fornisce istruzioni per iniziare a usare Texas Instruments CC3220SF-LAUNCHXL. Se non disponi del kit di sviluppo CC3220SF-LAUNCHXL di Texas Instruments (TI), visita il Partner Device Catalog per acquistarne uno dal nostro 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. 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 supporta due IDE per il kit di sviluppo TI CC3220SF-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 SimpleLink le famiglie di prodotti da installare, scegli MCU wireless Wi-Fi CC32xx.
5. Nella pagina successiva accettare le impostazioni predefinite per il debug delle sonde e quindi scegliere Finish (Termina).

Se si verificano problemi durante l'installazione di Code Composer Studio, consulta Supporto per gli strumenti di sviluppo di TI, Domande frequenti su Code Composer Studio e Risoluzione dei problemi di 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 l'SDK CC3220 SimpleLink

Installa l'SDK CC3220. SimpleLink L'SDK SimpleLink Wi-Fi CC3220 contiene i driver per l'MCU programmabile CC3220SF, 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 standalone utilizzato per programmare la memoria flash on-chip sui microcontrollori TI. Uniflash dispone di un'interfaccia utente grafica, a riga di comando e di scripting.

Installazione del Service Pack più recente

1. Sul tuo TI CC3220SF-LAUNCHXL posizionare il jumper SOP intermedio set di spine (posizione = 1) e reimposta la scheda.
2. Avviare Uniflash. Se la scheda CC3220SF viene visualizzata in Dispositivi rilevati, scegli Avvia. LaunchPad Se la scheda non viene rilevata, selezionare CC3220SF-LAUNCHXL dall'elenco

delle schede in New Configuration (Nuova configurazione), quindi selezionare Start Image Creator (Avvia Image Creator).

3. Scegliere Nuovo progetto.
4. Nella pagina Start new project (Inizia nuovo progetto) immettere un nome per il progetto. Per Device Type (Tipo dispositivo), scegliere CC3220SF. 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 CC3220SF. SimpleLink 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. Verificare che il collegamento Sense On Power (SOP) su Texas Instruments CC3220SF-LAUNCHXL sia nella posizione 0. Per ulteriori informazioni, consulta la Guida per l'utente del processore di SimpleLink rete Wi-Fi CC3x20, CC3x3x.
2. Utilizzare un cavo USB per collegare Texas Instruments CC3220SF-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).

Monitoraggio dei messaggi MQTT in cloud

Prima di eseguire il progetto demo FreerTOS, puoi configurare il client MQTT nella console per monitorare AWS IoT i messaggi che il tuo dispositivo invia al Cloud. AWS

Per effettuare la sottoscrizione all'argomento MQTT con il client AWS IoT

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

Quando il progetto demo viene eseguito correttamente sul tuo dispositivo, vedi «Hello World!» inviato più volte all'argomento a cui ti sei iscritto.

Crea ed esegui un progetto dimostrativo 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. Verificare che il collegamento Sense On Power (SOP) su Texas Instruments CC3220SF-LAUNCHXL sia nella posizione 0. Per ulteriori informazioni, consulta la Guida per l'utente del processore di SimpleLink rete Wi-Fi CC3x20, CC3x3x.
2. Utilizzare un cavo USB per collegare Texas Instruments CC3220SF-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 il messaggio «Avviso: inizializzazione EnergyTrace non riuscita», se è 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).

Usare CMake con FreerTOS

Se preferisci non utilizzare un IDE per lo sviluppo di FreerTOS, puoi in alternativa utilizzare CMake 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. Creare una cartella per contenere i file della 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 di CMake con FreeRTOS.

Risoluzione dei problemi

Se non vengono visualizzati i messaggi nel client MQTT della console AWS IoT, potrebbe essere necessario configurare le impostazioni di debug della 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

Questo tutorial fornisce istruzioni per iniziare a usare FreeRTOS Windows Device Simulator.

Prima di iniziare, devi configurareAWS IoT e scaricare FreeRTOS per connettere il tuo dispositivo alAWS Cloud. Per istruzioni, consulta Fase iniziale. In questo tutorial, il percorso della directory di download di FreeRTOS viene chiamato*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 sulle istanze Amazon EC2 Windows.

Configurazione dell'ambiente di sviluppo

1. Installazione della versione più recente di Npcap. Seleziona la "modalitàWinPcap compatibile con l'API» durante l'installazione.
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 basato su CMake per creare i tuoi progetti FreeRTOS, installa l'ultima versione di CMake. FreeRTOS richiede la versione 3.13 o successiva di CMake.

Monitoraggio dei messaggi MQTT in cloud

Prima di eseguire il progetto demo di FreeRTOS, puoi configurare il client MQTT nellaAWS IoT console per monitorare i messaggi che il tuo dispositivo invia alAWS Cloud.

Per effettuare la sottoscrizione all'argomento MQTT con il client AWS IoT

1. Accedi alla console AWS IoT.
2. Nel riquadro di navigazione, scegli Test, quindi scegli MQTT test client per aprire il client MQTT.

3. In Argomento sottoscrizione, digitare ***your-thing-name*/example/topic**, quindi scegliere Effettua sottoscrizione all'argomento.

Quando il progetto demo viene eseguito correttamente sul tuo dispositivo, viene visualizzato «Hello World!» inviato più volte all'argomento a cui ti sei iscritto.

Crea ed esegui il progetto demo FreeRTOS

Puoi usare Visual Studio o CMake per 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). Scegli una versione di Windows SDK presente sul tuo computer (il valore iniziale nel menu a discesa va bene), quindi scegli 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. Alla prima esecuzione, è necessario selezionare un'interfaccia di rete.

Creazione ed esecuzione del progetto demo FreeRTOS con CMake

Ti consigliamo di utilizzare la GUI di CMake anziché lo strumento da riga di comando CMake per creare il progetto demo per Windows Simulator.

Dopo aver installato CMake aprire la relativa interfaccia utente grafica. In Windows, l'interfaccia utente grafica è disponibile nel menu Start in CMake, CMake (cmake-gui).

1. Imposta la directory del codice sorgente di FreeRTOS.

   Nella GUI, impostate la directory del codice sorgente di FreeRTOS (*`freertos`*) per Dov'è il codice sorgente.

   Impostare *`freertos`*`/build` per Where to build the binaries (Posizione di creazione file binari).

2. Configurare il progetto CMake.

   Nell'interfaccia utente grafica CMake, scegliere Add Entry (Aggiungi voce), quindi nella finestra Add Cache Entry (Aggiungi voce cache), impostare i seguenti valori:

   Nome

   AFR_BOARD

   Type (Tipo)

   STRING

   Value (Valore)

   pc.windows

   Descrizione

   (Opzionale)

3. Scegliere Configure (Configura). Se CMake richiede di creare la directory, scegliere Yes (Sì), quindi selezionare un generatore in Specify the generator for this project (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. Mantieni invariate le altre opzioni del generatore e scegli Fine.

4. Generare e aprire il progetto CMake.

   Dopo aver configurato il progetto, l'interfaccia utente grafica CMake mostra tutte le opzioni disponibili per il progetto generato. Ai fini di questo tutorial, è possibile lasciare le opzioni ai valori predefiniti.

   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 su`aws_demos` progetto e scegli Imposta come StartUp progetto. In questo modo è possibile creare ed eseguire il progetto. Alla prima esecuzione, è necessario selezionare un'interfaccia di rete.

Per ulteriori informazioni sull'utilizzo di CMake con FreeRTOS, consultaUtilizzo di 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 conta 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 precedente, il numero da utilizzare è1.

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.

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 sull'utilizzo di FreeRTOS, consultaNozioni di base sulla risoluzione dei problemi.

Guida introduttiva al kit IoT MicroZed industriale Xilinx Avnet

> **Important**
>
> Questa integrazione di riferimento è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Consigliamo di iniziare da qui quando crei un nuovo progetto. Se disponi già di un progetto FreeRTOS esistente basato sull'ormai obsoleto repository Amazon-FreeRTOS, consulta ilGuida alla migrazione del repository Github di Amazon-FreeRTOS.

Questo tutorial fornisce istruzioni per iniziare a usare il kit IoT MicroZed industriale Xilinx Avnet. Se non disponi del kit IoT MicroZed industriale Xilinx Avnet, visita il catalogo dei dispositiviAWS partner per acquistarne uno dal nostro partner.

Prima di iniziare, devi configurareAWS IoT e scaricare FreeRTOS per connettere il tuo dispositivo alAWS Cloud. Per istruzioni, consulta Fase iniziale. In questo tutorial, il percorso della directory di download di FreeRTOS viene chiamato*`freertos`*.

Panoramica

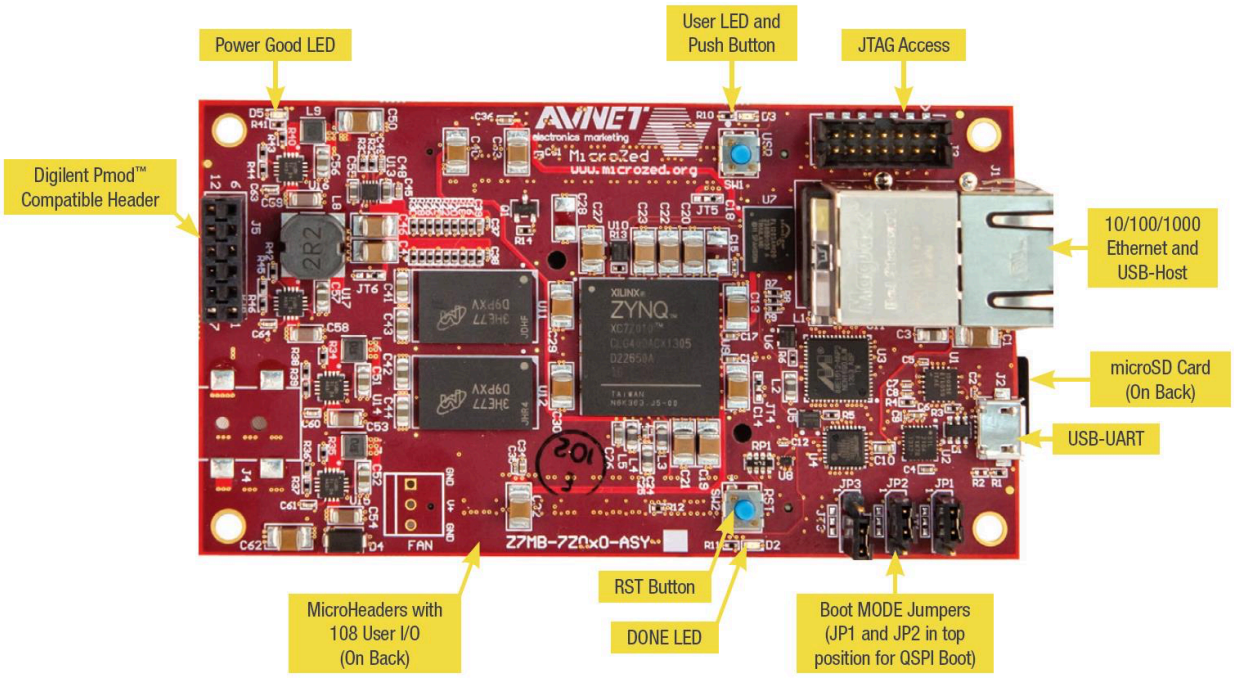
Questo tutorial contiene le istruzioni per i seguenti passaggi iniziali:

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.

Configurare l' MicroZed hardware

Il seguente diagramma può essere utile per configurare l' MicroZed hardware:



Per configurare la MicroZed scheda

1. Collega il tuo computer alla porta USB-UART della MicroZed scheda.
2. Collega il tuo computer alla porta di accesso JTAG sulla MicroZed scheda.
3. Collega un router o una porta Ethernet connessa a Internet alla porta Ethernet e USB-host della MicroZed scheda.

Configurazione dell'ambiente di sviluppo

Per configurare le configurazioni FreeRTOS per il MicroZed 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 WebInstall Client.
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 quindi 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 MicroZed la connessione USB-UART, installa manualmente i driver VCP del bridge USB-to-UART CP210x. Per le istruzioni, consulta il documento Silicon Labs CP210x USB-to-UART Installation Guide.

Per ulteriori informazioni su XSDK, consulta la pagina Getting Started with Xilinx SDK sul sito Web Xilink.

Monitoraggio dei messaggi MQTT in cloud

Prima di eseguire il progetto demo di FreeRTOS, puoi configurare il client MQTT nellaAWS IoT console per monitorare i messaggi che il tuo dispositivo invia alAWS Cloud.

Per effettuare la sottoscrizione all'argomento MQTT con il client AWS IoT

1. Accedi alla console AWS IoT.

2. Nel pannello di navigazione, scegli Test, quindi scegli MQTT test client 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

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).

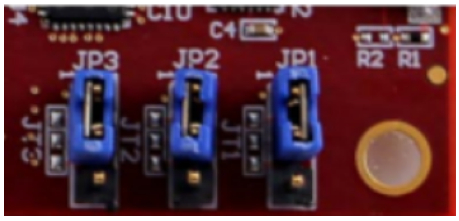
Genera l'immagine di avvio per il progetto demo di FreeRTOS

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 Add (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

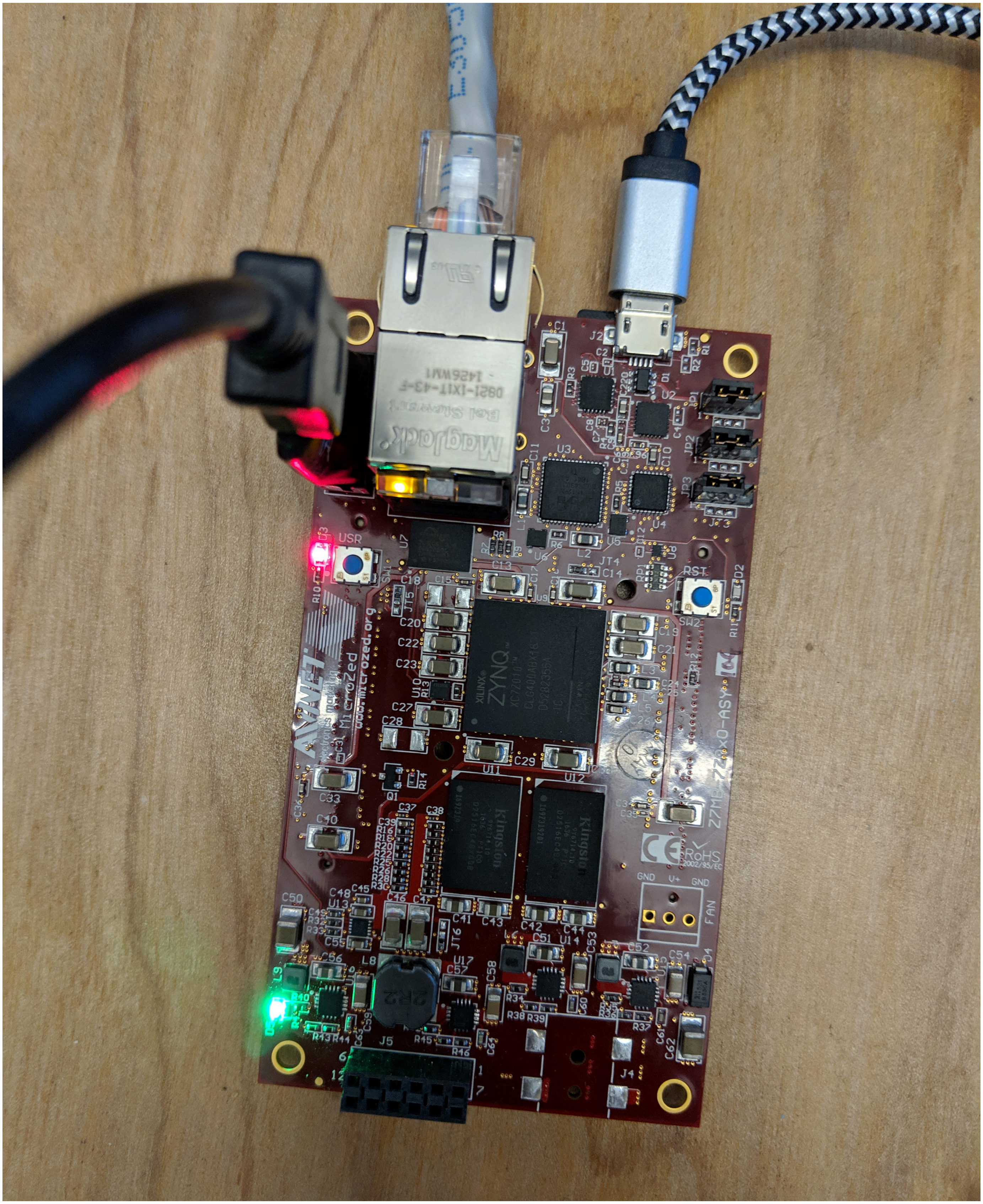
1. Imposta i jumper della modalità di avvio della tua MicroZed scheda sulla modalità di avvio JTAG.



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
GND V+ GND
FAN
J5
J4

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 di FreeRTOS e Esegui il progetto demo FreeRTOS.

Esegui il progetto demo FreeRTOS

Per eseguire il progetto demo di FreeRTOS, puoi avviare la MicroZed scheda da una scheda microSD o da una memoria flash QSPI.

Quando configuri la MicroZed scheda per l'esecuzione del progetto demo di FreeRTOS, fai riferimento al diagramma inConfigurare l' MicroZed hardware. Verifica di aver collegato la MicroZed scheda al computer.

Avvia il progetto FreeRTOS da una scheda microSD

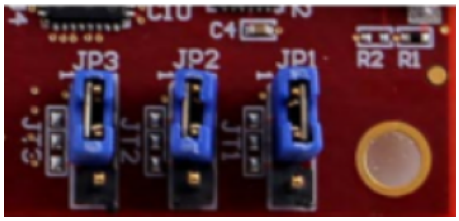
Formatta la scheda microSD fornita con il kit IoT MicroZed industriale Xilinx.

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 tua 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 Connessione, scegli il 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.

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 relativi a Getting Started with FreeRTOS, consultaNozioni di base sulla risoluzione dei problemi.

Fasi successive con FreeRTOS

> **Important**
>
> Questa pagina fa riferimento al repository Amazon-FreeRTOS che è obsoleto. Consigliamo di Nozioni su, quando si crea un nuovo progetto. Se hai già un progetto FreeRTOS esistente basato sull'ormai obsoleto repository Amazon-FreeRTOS, consulta ilGuida alla migrazione del repository Github di Amazon-FreeRTOS.

Dopo aver creato, eseguito il flashing ed eseguito il progetto demo FreeRTOS per la tua scheda, puoi visitare il sito web FreeRTOS.org per saperne di più sulla creazione di un nuovo progetto FreeRTOS. Esistono anche demo per molte librerie FreeRTOS che mostrano come eseguire attività importanti, interagire conAWS IoT i servizi e programmare funzionalità specifiche della scheda (come i modem cellulari). Per ulteriori informazioni, consulta pagina della libreria FreeRTOS.

Il sito web FreeRTOS.org contiene anche informazioni approfondite sul kernel FreeRTOS e concetti fondamentali del sistema operativo in tempo reale. Per ulteriori informazioni, consulta le pagine FreeRTOS Kernel Developer Docs e FreeRTOS Kernel Secondary Docs.

Aggiornamenti via etere di FreeRTOS

> **Note**
>
> Vedi AWS IoTOver-the-air (OTA) sul sito Web di FreeRTOS per informazioni recenti sull'esecuzione degli aggiornamenti O ver-the-air (OTA).

Gli aggiornamenti O ver-the-air (OTA) ti consentono di distribuire gli aggiornamenti del firmware su uno o più dispositivi della tua flotta. 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 per le offerte di AWS IoT lavoro 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 si utilizza l'applicazione demo FreeRTOS OTA Update, è necessario integrarla AWS IoTLibreria via etere (OTA) nella propria applicazione per ottenere la funzionalità di aggiornamento del firmware.

over-the-airGli aggiornamenti di 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 categorizzare le tue risorse AWS IoT in modi diversi (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

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 si sta usando Code Signing for AWS IoT, Concessione dell'accesso alla firma del codice per AWS IoT.
- Scarica FreeRTOS con la libreria OTA.

Crea un bucket Amazon S3 per archiviare l'aggiornamento

I file di aggiornamento OTA sono archiviati in bucket Amazon S3.

Se si utilizza Code Signing for AWS IoT, il comando utilizzato per la creazione di un processo di firma del codice utilizza 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 GUID in modo che non vengano sovrascritti i file originali.

Come creare un bucket Amazon S3.

1. Accedi alla console Amazon S3 all'indirizzo https://console.aws.amazon.com/s3/.
2. Seleziona Create bucket (Crea bucket).
3. Inserisci un nome per il bucket.
4. Nelle impostazioni del bucket per Blocca accesso pubblico, mantieni selezionata l'opzione Blocca tutti gli accessi pubblici per accettare le autorizzazioni predefinite.
5. In Controllo delle versioni del bucket, seleziona Abilita per mantenere tutte le versioni nello stesso bucket.
6. Seleziona Create bucket (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

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 a 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.

> **Important**
>
> Per risolvere il confuso problema di sicurezza del vice, è necessario seguire le istruzioni contenute nella AWS IoT Coreguida.

Per aggiungere autorizzazioni per gli aggiornamenti OTA al ruolo del servizio OTA

1. Nella casella di ricerca della pagina della console IAM, inserisci il nome del tuo ruolo, quindi sceglilo dall'elenco.
2. Scegli Collega policy.
3. Nella casella di ricerca, inserisci "AmazonFreeRTOSOTAupdate», seleziona AmazonFreeRTOSOTAupdate dall'elenco delle politiche filtrate, quindi scegli Allega politica per allegare la politica al tuo ruolo di servizio.

Per aggiungere le autorizzazioni IAM richieste al tuo ruolo di servizio OTA

1. Nella casella di ricerca della pagina della console IAM, inserisci il nome del tuo ruolo, quindi sceglilo 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"
      }
    ]
}
```

Ricordare di sostituire *your_account_id* con l'ID dell'account AWS e *your_role_name* con il nome del ruolo del servizio OTA.

5. Scegli Review policy (Esamina policy).
6. Immettere un nome per la policy e scegliere Create policy (Crea policy).

> **Note**
>
> La seguente procedura non è richiesta se il nome del bucket Amazon S3 inizia con «afr-ota». In caso contrario, la policy `AmazonFreeRTOSOTAUpdate` gestita da AWS include già le autorizzazioni necessarie.

Per aggiungere le autorizzazioni Amazon S3 richieste al tuo ruolo di servizio OTA

1. Nella casella di ricerca della pagina della console IAM, inserisci il nome del tuo ruolo, quindi sceglilo 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 tuo ruolo di servizio OTA l'autorizzazione a leggere gli oggetti Amazon S3. Ricordare di sostituire *example-bucket* con il nome del bucket.

5. Scegli Review policy (Esamina policy).
6. Immettere un nome per la policy e scegliere Create policy (Crea policy).

Creazione di una policy utente OTA

È necessario concedere all'utente 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.
- Accedere 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.
- Accedere ai processi AWS IoT.
- Accedi a IAM.
- Accedere a Code Signing for AWS IoT. Consultare Concessione dell'accesso alla firma del codice per AWS IoT.
- Elenca le piattaforme hardware FreeRTOS.
- Etichetta e rimuovi i tag dalle AWS IoT risorse.

Per concedere al tuo utente le autorizzazioni richieste, consulta le politiche IAM. Vedi anche Autorizzazione di utenti e servizi cloud a utilizzare AWS IoT Jobs.

Per fornire l'accesso, aggiungi autorizzazioni ai tuoi 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 Creating a role for a third-party identity provider (federation) (Creazione di un ruolo per un provider di identità di terze parti [federazione]) nella Guida per l'utente di IAM.

- Utenti IAM:
  - Crea un ruolo che l'utente possa assumere. Per istruzioni, consulta la pagina Creating a role for an IAM user (Creazione di un ruolo per un utente IAM) nella Guida per l'utente di 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 di IAM.

Creazione di un certificato di firma del codice

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 FreeRTOS.

Argomenti

- Creazione di un certificato di firma del codice per Texas Instruments CC3220SF-LAUNCHXL
- 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 CC3220SF-LAUNCHXL

> **Important**
>
> Questa integrazione di riferimento è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Ti consigliamo di iniziare da qui quando crei un nuovo progetto. Se disponi già di un progetto FreeRTOS basato sull'ormai obsoleto repository Amazon-FreeRTOS, consulta il. Guida alla migrazione del repository Github di Amazon-FreeRTOS

Il kit di sviluppo del microcontrollore wireless SimpleLink Wi-Fi CC3220SF 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 in codice autofirmato.

Utilizzare l'AWS Command Line Interface per importare in AWS Certificate Manager il certificato di firma del codice, la chiave privata e la catena di certificati. Per ulteriori informazioni, vedere Installazione del AWS CLI file nella Guida per l'AWS Command Line Interfaceutente.

Scarica e installa l'ultima versione di SimpleLinkCC3220 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 dell'SDK SimpleLink CC3220 sono in formato DER. Per creare un certificato di firma in codice autofirmato, è necessario convertirlo in formato PEM.

Segui queste fasi per creare un certificato di firma del codice che sia collegato alla gerarchia di certificati dei prodotti Texas Instruments e soddisfi i criteri AWS Certificate Manager e di Code Signing for 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`. Sostituire *`test_signer@amazon.com`* con il proprio indirizzo e-mail.

```
[ 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. Importare 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**
>
> Questa fase presuppone che si intenda utilizzare Code Signing for AWS IoT per firmare le immagini del firmware. Anche se si consiglia di utilizzare Code Signing for AWS IoT, è possibile firmare le immagini del firmware anche manualmente.

Creazione di un certificato di firma del codice per Espressif ESP32

> **Important**
>
> Questa integrazione di riferimento è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Ti consigliamo di iniziare da qui quando crei un nuovo progetto. Se disponi già di un progetto FreeRTOS basato sull'ormai obsoleto repository Amazon-FreeRTOS, consulta il. Guida alla migrazione del repository Github di Amazon-FreeRTOS

Le schede Espressif ESP32 supportano un certificato SHA-256 autofirmato con firma del codice 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.
> Utilizzare l'AWS Command Line Interface per importare in AWS Certificate Manager il certificato di firma del codice, la chiave privata e la catena di certificati. Per informazioni sull'installazione di AWS CLI, consulta Installazione di AWS CLI.

1. Nella directory di lavoro, utilizzare il testo seguente per creare un file denominato `cert_config.txt`. Sostituire *`test_signer@amazon.com`* con il proprio indirizzo e-mail:

```
[ 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. Importare 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**
>
> Questa fase presuppone che si intenda utilizzare Code Signing for AWS IoT per firmare le immagini del firmware. Anche se si consiglia di utilizzare Code Signing for AWS IoT, è possibile firmare le immagini del firmware anche manualmente.

Creazione di un certificato di firma del codice per Nordic nrf52840-dk

> **Important**
>
> Questa integrazione di riferimento è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Ti consigliamo di iniziare da qui quando crei un nuovo progetto. Se disponi già di un progetto FreeRTOS basato sull'ormai obsoleto repository Amazon-FreeRTOS, consulta il. Guida alla migrazione del repository Github di Amazon-FreeRTOS

La scheda Nordic nrf52840-dk supporta un certificato SHA256 autofirmato con firma del codice 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.
> Utilizzare l'AWS Command Line Interface per importare in AWS Certificate Manager il certificato di firma del codice, la chiave privata e la catena di certificati. Per informazioni sull'installazione di AWS CLI, consulta Installazione di AWS CLI.

1. Nella directory di lavoro, utilizzare il testo seguente per creare un file denominato `cert_config.txt`. Sostituire *`test_signer@amazon.com`* con il proprio indirizzo e-mail:

```
[ 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. Importare 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**
>
> Questa fase presuppone che si intenda utilizzare Code Signing for AWS IoT per firmare le immagini del firmware. Anche se si consiglia di utilizzare Code Signing for AWS IoT, è possibile firmare le immagini del firmware anche manualmente.

Creazione di un certificato di firma del codice per il simulatore Windows FreeRTOS

Il simulatore Windows FreeRTOS richiede un certificato di firma del codice con una chiave ECDSA P-256 e un 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.
> Utilizzare l'AWS Command Line Interface per importare in AWS Certificate Manager il certificato di firma del codice, la chiave privata e la catena di certificati. Per informazioni sull'installazione di AWS CLI, consulta Installazione di AWS CLI.

1. Nella directory di lavoro, utilizzare il testo seguente per creare un file denominato `cert_config.txt`. Sostituire *`test_signer@amazon.com`* con il proprio indirizzo e-mail:

```
[ 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. Importare 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**
>
> Questa fase presuppone che si intenda utilizzare Code Signing for AWS IoT per firmare le immagini del firmware. Anche se si consiglia di utilizzare Code Signing for AWS IoT, è possibile firmare le immagini del firmware anche manualmente.

Creazione di un certificato di firma del codice per l'hardware personalizzato

Utilizzando un set di strumenti appropriato, è possibile creare una chiave privata e del certificato autofirmato per l'hardware.

Utilizzare l'AWS Command Line Interface per importare in AWS Certificate Manager il certificato di firma del codice, la chiave privata e la catena di certificati. Per informazioni sull'installazione di AWS CLI, consulta 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 che i certificati utilizzino algoritmi e dimensioni delle chiavi specifici. Per ulteriori informazioni, consulta la sezione Prerequisiti per l'importazione di certificati. Per ulteriori informazioni su ACM, vedere Importazione di certificati in. AWS Certificate Manager

Devi copiare, incollare e formattare il contenuto del tuo certificato di firma del codice nel `vendors/vendor/boards/board/aws_demos/config_files/ota_demo_config.h` file che fa parte del codice FreeRTOS che scaricherai in seguito.

Concessione dell'accesso alla firma del codice per AWS IoT

Per fornire l'accesso, aggiungi autorizzazioni ai tuoi 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 Creating a role for a third-party identity provider (federation) (Creazione di un ruolo per un provider di identità di terze parti [federazione]) nella Guida per l'utente di IAM.

- Utenti IAM:
  - Crea un ruolo che l'utente possa assumere. Per istruzioni, consulta la pagina Creating a role for an IAM user (Creazione di un ruolo per un utente IAM) nella Guida per l'utente di 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 di IAM.

Scarica FreeRTOS con la libreria OTA

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.

⚠ Important

- In questo argomento, si fa riferimento al percorso della directory di download di FreeRTOS 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 percorsi lunghi delle directory di download di FreeRTOS possono causare errori di compilazione.
- Poiché il codice sorgente può contenere collegamenti simbolici, se utilizzi Windows per estrarre l'archivio, potresti dover:
  - Abilita la modalità sviluppatore o,
  - Usa una console con un livello elevato di amministratore.

  In questo modo, Windows può creare correttamente collegamenti simbolici quando estrae l'archivio. Altrimenti, i collegamenti simbolici verranno scritti come normali file che contengono i percorsi dei collegamenti simbolici come testo o sono vuoti. Per ulteriori informazioni, consulta il post di blog Symlinks in Windows 10! .

  Se usi Git in Windows, devi abilitare la modalità sviluppatore oppure devi:

  - `core.symlinks`Imposta su true con il seguente comando:

    ```
    git config --global core.symlinks true
    ```

  - Usa una console con il livello di amministratore ogni volta che usi un comando git che scrive nel sistema (ad esempio, git pullgit clone, egit submodule update --init --recursive).

Prerequisiti per gli aggiornamenti OTA mediante MQTT

Questa sezione descrive i requisiti generali per l'utilizzo di MQTT per eseguire over-the-air (aggiornamenti OTA).

Requisiti minimi

- Il firmware del dispositivo deve includere le librerie FreeRTOS necessarie (CoreMQTT Agent, aggiornamento OTA e 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 degli aggiornamenti del firmware dai dispositivi ai dispositivi. AWS IoT Se specifichi entrambi i protocolli quando crei 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 prevede l'utilizzo del protocollo MQTT. `ota_config.h`

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 utilizzando MQTT 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/
${iot:Connection.Thing.ThingName}/streams/*",
                "arn:partition:iot:region:account:topicfilter/$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 autorizzazioni `iot:Connect` consentono al dispositivo di connettersi a AWS IoT tramite MQTT.
- 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 autorizzazioni `iot:Publish` e `iot:Subscribe` sugli argomenti dei flussi OTA AWS IoT (`.../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 autorizzazione `iot:Receive` consentono a AWS IoT Core di pubblicare messaggi su tali argomenti per il 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

Questa sezione descrive i requisiti generali per l'utilizzo di HTTP per eseguire aggiornamenti over-the-air (OTA). A partire dalla versione 201912.00, FreeRTOS OTA può utilizzare il protocollo HTTP o MQTT per trasferire le immagini degli aggiornamenti del firmware dai dispositivi ai dispositivi. AWS IoT

> **Note**
>
> - Sebbene il protocollo HTTP possa essere utilizzato per trasferire l'immagine del firmware, la libreria CoreMQTT Agent è comunque necessaria perché altre interazioni AWS IoT Core utilizzano la libreria CoreMQTT Agent, tra cui l'invio o la ricezione di notifiche di esecuzione del lavoro, documenti di lavoro e aggiornamenti dello stato dell'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 Agent, HTTP, OTA Agent e le relative dipendenze).
- È necessaria la versione 201912.00 o successiva di FreeRTOS 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\ota_config.h`.

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 dell'heap per tutte le piattaforme supportate, calcolati utilizzando l'API `xPortGetFreeHeapSize` FreeRTOS. È necessario assicurarsi che ci sia RAM sufficiente per utilizzare la libreria OTA.

Texas Instruments CC3220SF-LAUNCHXL

- 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 Curiosity PIC32MZEF

- Operazioni di controllo (MQTT): 65 KB
- Operazioni dei dati (HTTP): 43 KB

Espressif ESP32

- Operazioni di controllo (MQTT): 65 KB
- Operazioni dei dati (HTTP): 45 KB

> **Note**
>
> BLE su 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 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 autorizzazioni `iot:Receive` consentono a AWS IoT Core di pubblicare messaggi su tali argomenti sul dispositivo connesso corrente. 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

Questa sezione contiene un tutorial per aggiornare il 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.

Per creare un aggiornamento OTA, puoi utilizzare la console AWS IoT o AWS CLI. La console è il modo più semplice per iniziare a utilizzare il sistema OTA perché gran parte del lavoro viene eseguito automaticamente. AWS CLIÈ utile quando si automatizzano i processi di aggiornamento OTA, si lavora con un gran numero di dispositivi o si utilizzano dispositivi che non sono qualificati per FreeRTOS. Per ulteriori informazioni sui dispositivi idonei per FreeRTOS, consulta il sito Web di 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 stai firmando manualmente il firmware, firma e carica l'immagine del firmware firmata nel tuo bucket Amazon S3. Se utilizzi Code Signing perAWS IoT, carica l'immagine del firmware non firmata su 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

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 utilizzi FreeRTOS, salta questo passaggio. È invece necessario copiare l'implementazione dell'agente OTA sui dispositivi.

Argomenti

Installare la versione iniziale del firmware sulla scheda Texas Instruments CC3220SF-LAUNCHXL

> **Important**
>
> Questa integrazione di riferimento è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Ti consigliamo di iniziare da qui quando crei un nuovo progetto. Se disponi già di un progetto FreeRTOS basato sull'ormai obsoleto repository Amazon-FreeRTOS, consulta il. Guida alla migrazione del repository Github di Amazon-FreeRTOS

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 FreeRTOS su Texas Instruments CC3220SF-LAUNCHXL.

1. Sul Texas Instruments CC3220SF-LAUNCHXL posizionare il jumper SOP nel set di pin intermedio (posizione = 1) e reimpostare la scheda.
2. Scaricare e installare lo strumento Uniflash di TI.
3. Avviare Uniflash. Dall'elenco delle configurazioni, scegliere CC3220SF-LAUNCHXL, quindi selezionare Start Image Creator (Avvia Image Creator).
4. Scegliere New project (Nuovo progetto).
5. Nella pagina Start new project (Inizia nuovo progetto) immettere un nome per il progetto. Per Device Type (Tipo dispositivo), scegliere CC3220SF. Per Device Mode (Modalità dispositivo), scegliere Develop (Sviluppo). Scegliere Create project (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`. Scegliere Open (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 simplelink_cc32xx_sdk_ version /tools/cc32xx_tools/certificate-playground/ 20160911.lst, quindi scegli Apri. certcatalogPlayGround***
19. ***In File sorgente della firma, scegli Sfoglia, scegli simplelink_cc32xx_sdk_ version /tools/cc32xx_tools/certificate-playground/ 20160911.lst.signed_3220.bin, quindi scegli Apri. 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
```

Installare la versione iniziale del firmware sulla scheda Espressif ESP32

> **Important**
>
> Questa integrazione di riferimento è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Ti consigliamo di iniziare da qui quando crei un nuovo progetto. Se disponi già di un progetto FreeRTOS basato sull'ormai obsoleto repository Amazon-FreeRTOS, consulta il. Guida alla migrazione del repository Github di Amazon-FreeRTOS

Questa guida è stata redatta presupponendo che siano già stati eseguiti i passaggi indicati in Guida introduttiva a Espressif DevKit ESP32-C e ai prerequisiti di aggiornamento ESP-WROVER-KIT e Over-the-Air. Prima di tentare un aggiornamento OTA, potresti voler eseguire il progetto demo 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. Apri*freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h, commenta #define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED e definisci CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED oCONFIG_OTA_HTTP_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 vendors/*vendor*/boards/*board*/aws_demos/config_files/ota_demo_config.h. Deve essere formattato nel modo seguente:

```
#define otapalconfigCODE_SIGNING_CERTIFICATE  \
"-----BEGIN CERTIFICATE-----\n" \
"...base64 data...\n" \
"-----END CERTIFICATE-----\n";
```

3. Con la demo dell'aggiornamento OTA selezionata, seguire le stesse fasi descritte nella pagina delle nozioni di base su ESP32 per compilare ed eseguire il flashing dell'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. La scheda ESP32 ora rimane in ascolto di 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.

Installare la versione iniziale del firmware sulla scheda Nordic nRF52840 DK

> **Important**
>
> Questa integrazione di riferimento è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Ti consigliamo di iniziare da qui quando crei un nuovo progetto. Se disponi già di un progetto FreeRTOS basato sull'ormai obsoleto repository Amazon-FreeRTOS, consulta il. Guida alla migrazione del repository Github di Amazon-FreeRTOS

Questa guida è stata scritta presupponendo che siano state già eseguite le fasi descritte nella pagina Nozioni di base su Nordic nRF52840-DK e nella pagina Prerequisiti degli aggiornamenti Over-the-Air. Prima di tentare un aggiornamento OTA, potresti voler eseguire il progetto demo MQTT descritto in Guida introduttiva a FreeRTOS per assicurarti che la tua scheda e la tua 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_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED` con `CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED` o `CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED`.
3. Con la demo dell'aggiornamento OTA selezionata, seguire le stesse fasi descritte nella pagina Nozioni di base su Nordic nRF52840-DK 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

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

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 FreeRS]

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

L'agente OTA incluso in FreeRTOS verifica la versione di qualsiasi aggiornamento e la 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. Individua il file `/vendors/vendor/boards/board/aws_demos/config_files/ota_demo_config.h` e incrementa il valore di APP_VERSION_BUILD.
3. Per pianificare un aggiornamento a una piattaforma Renesas rx65n con un tipo di file diverso da 0 (file non firmware), è necessario firmare il file con lo strumento Renesas Secure Flash Programmer, altrimenti non verrà eseguito il controllo della firma sul dispositivo. Lo strumento crea un pacchetto di file firmato con l'estensione `.rsu` che è un tipo di file proprietario per Renesas. Lo strumento può essere trovato su Github. È possibile utilizzare il seguente comando di esempio per generare l'immagine:

   ```
   "Renesas Secure Flash Programmer.exe" CUI Update "RX65N(ROM 2MB)/Secure
    Bootloader=256KB" "sig-sha256-ecdsa" 1 "file_name" "output_file_name.rsu"
   ```

4. Ricreare il progetto.

Devi copiare l'aggiornamento del firmware nel bucket Amazon S3 che hai 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 CC3220SF-LAUNCHXL: `vendors/ti/boards/cc3220_launchpad/aws_demos/ccs/debug/aws_demos.bin`
- Espressif ESP32: `vendors/espressif/boards/esp32/aws_demos/make/build/aws_demos.bin`

Creazione di un aggiornamento OTA (console AWS IoT)

1. Nel riquadro di navigazione della AWS IoT console, in Gestisci, seleziona Azioni remote, quindi scegli Lavori.
2. Scegli Create job (Crea processo).
3. In Tipo di lavoro seleziona Crea lavoro di aggiornamento OTA FreeRTOS, quindi scegli Avanti.
4. In Proprietà del lavoro, inserisci un nome del lavoro e (facoltativamente) inserisci una descrizione del lavoro, quindi scegli Avanti.
5. È possibile distribuire un aggiornamento OTA in un singolo dispositivo o in un gruppo di dispositivi. In Dispositivi da aggiornare, scegli uno o più elementi o gruppi di cose dal menu a discesa.
6. In Seleziona il protocollo per il trasferimento dei file, seleziona HTTP o MQTT oppure seleziona entrambi per consentire a ciascun dispositivo di determinare il protocollo da utilizzare.
7. In Firma e scegli il tuo file, seleziona Firma un nuovo file per me.
8. Nella sezione Profilo di firma del codice, scegli Crea nuovo profilo.
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 intende utilizzare una piattaforma non qualificata e si utilizza un pacchetto di crittografia diverso da ECDSA P-256 SHA-256 per la firma, è possibile utilizzare Code Signing for AWS IoT o firmare autonomamente l'aggiornamento del firmware. Per ulteriori informazioni, consulta Aggiungere una firma digitale all'aggiornamento del firmware.

    b. In Certificato di firma del codice, scegli Seleziona un certificato esistente e quindi seleziona un certificato precedentemente importato oppure scegli Importa un nuovo certificato di firma del codice, scegli i tuoi file e seleziona 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. Per la maggior parte dei dispositivi puoi lasciare questo campo vuoto. Per il simulatore di Windows e per i dispositivi che collocano il certificato in una posizione di file specifica, inserisci qui il nome del percorso.

> **Important**
>
> In Texas Instruments CC3220SF-LAUNCHXL, non anteporre un carattere barra (**/**) 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`.

d. Selezionare Crea.

10. In File seleziona Seleziona un file esistente, quindi scegli Sfoglia S3. 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 Curiosity PIC32MZEF di Microchip producono due immagini binarie con i nomi predefiniti `mplab.production.bin` e `mplab.production.ota.bin`. Utilizzare il secondo file quando si carica un'immagine per l'aggiornamento OTA.

11. In Percorso del file sul dispositivo, inserisci il nome del percorso completo della posizione sul dispositivo in cui il job OTA copierà l'immagine del firmware. Questo percorso varia in base alla piattaforma.

> **Important**
>
> In Texas Instruments CC3220SF-LAUNCHXL, a causa delle limitazioni di sicurezza, il nome del percorso dell'immagine del firmware deve essere `/sys/mcuflashimg.bin`.

12. Apri Tipo di file e inserisci un valore intero compreso tra 0 e 255. Il tipo di file immesso verrà aggiunto al documento di lavoro consegnato all'MCU. Lo sviluppatore del firmware/software dell'MCU ha la piena proprietà su cosa fare con questo valore. Gli scenari possibili includono un

MCU con 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 Ruolo IAM, scegli un ruolo in base alle istruzioni inCreazione di un ruolo del servizio per gli aggiornamenti OTA.
14. Seleziona Successivo.
15. Immetti un ID e una descrizione per il processo di aggiornamento OTA.
16. 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)).
17. Scegli le configurazioni facoltative appropriate per il processo (rollout esecuzioni processi, interruzione processo, timeout esecuzioni processi e tag).
18. Seleziona 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. Per la maggior parte dei dispositivi puoi lasciare questo campo vuoto. Per il simulatore di Windows e per i dispositivi che collocano il certificato in una posizione di file specifica, inserisci qui il nome del percorso.

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 console AWS IoT con lo stato IN PROGRESS (IN CORSO).

> **Note**
>
> - La console AWS IoT non aggiorna automaticamente lo stato dei processi. 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 la AWS CLI

Quando si utilizza AWS CLI per creare un aggiornamento OTA, è sufficiente:

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 utilizzi AWS IoT per eseguire gli aggiornamenti OTA, puoi utilizzare Code Signing for AWS CLI o firmare tu stesso l'aggiornamento del firmware. Per un elenco degli algoritmi di firma crittografica e hashing supportati da Code Signing for, vedere. AWS IoT SigningConfigurationOverrides Se desideri utilizzare un algoritmo crittografico non supportato da Code Signing forAWS 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 forAWS IoT, puoi utilizzare uno degli AWSSDK o degli strumenti della riga di comando. Per ulteriori informazioni sulla firma del codice perAWS IoT, vedere 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, scegliendo Mostra.

`destination`

La destinazione sul dispositivo su 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 si importa 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 firmato 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 si crea un aggiornamento OTA AWS IoT, è anche possibile creare un flusso contenente l'aggiornamento del firmware con firma. 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 intendi utilizzare Code Signing for AWS IoT, il nome file è un GUID generato da Code Signing for AWS IoT.

Usa il create-stream AWS CLI comando per creare uno stream.

```
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 dove si `my-job-id` trova l'ID del processo 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	Type (Tipo)	Descrizione
`otaUpdateId`	string (max:128 min:1)	ID dell'aggiornamento OTA da creare.
`description`	string (max:2028)	Descrizione dell'aggi ornamento OTA.

Nome	Type (Tipo)	Descrizione
`targets`	list	Dispositivi destinati alla ricezione di aggiornamenti OTA.
`protocols`	list	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.

Nome	Type (Tipo)	Descrizione
`targetSelection`	string	Specifica se l'esecuzione dell'aggiornamento continuerà (CONTINUOUS) o se l'aggiornamento verrà completato o dopo che tutti gli oggetti specificati come target avranno completato l'aggiornamento (SNAPSHOT). Se è continuo, l'aggiornamento può anche essere eseguito in un oggetto quando viene rilevata una modifica in un target. Ad esempio, un aggiornamento verrà eseguito in un oggetto quando l'oggetto viene aggiunto a un gruppo target, anche dopo che l'aggiornamento è stato completato da tutti gli oggetti originariamente nel gruppo. Valori validi: CONTINUOUS | SNAPSHOT. Enumerazione: CONTINUOUS | SNAPSHOT
`awsJobExecutionsRolloutConfig`		Configurazione per l'implementazione degli aggiornamenti OTA.
`maximumPerMinute`	integer (max:1000 min:1)	Numero massimo di esecuzioni del processo di aggiornamento OTA avviate al minuto.

Nome	Type (Tipo)	Descrizione
`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.
`rateIncreaseCriteria`		I criteri per avviare l'aumento della velocità di rollout per un processo. AWS IoT supporta al massimo 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 URL prefirmati.

Nome	Type (Tipo)	Descrizione
`expiresInSec`	Long	Periodo di validità (in secondi) degli URL prefirmati. I valori validi sono compresi tra 60 e 3600 e il valore predefinito è 1800 secondi. Gli URL prefirmati vengono generati quando viene ricevuta una richiesta per il documento di processo.
`awsJobAbortConfig`		I criteri che determinano quando e come si verifica un'interruzione del lavoro.
`abortCriteriaList`	list	L'elenco dei criteri che determinano quando e come interrompere il processo.
`failureType`	string	Il tipo di errori di esecuzione del processo che possono causare l'interruzione di un processo. enum: FAILED | REJECTED | TIMED_OUT | ALL
`action`	string	Il tipo di azione lavorativa da intraprendere per avviare l'interruzione del lavoro. enum: CANCEL

Nome	Type (Tipo)	Descrizione
`minNumberOfExecutedThings`	integer (min:1)	Il numero minimo di elementi che devono ricevere notifiche di esecuzione del processo prima che il processo possa essere interrotto.
`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` .

Nome	Type (Tipo)	Descrizione
`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.
`files`	list	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.

Nome	Type (Tipo)	Descrizione
`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
`codeSigning`		Metodo di firma del codice del file.
`awsSignerJobId`	string	L'ID di chi AWSSignerJob è stato creato per firmare il file.
`startSigningJobPar ameter`		Descrive il processo di firma del codice.
`signingProfilePara meter`		Descrive il profilo di firma del codice.
`certificateArn`	string	ARN del certificato.

Nome	Type (Tipo)	Descrizione
`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.
`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.

Nome	Type (Tipo)	Descrizione
`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	Elenco di coppie nome/attr ibuto.
`roleArn`	string (max:2048 min:20)	Il ruolo IAM che AWS IoT consente l'accesso ad Amazon S3, ai AWS IoT job e alle risorse di AWS 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`	list	Metadati utilizzabili per la gestione degli aggiornamenti.
`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 CLIcampi di output

Nome	Type (Tipo)	Descrizione
`otaUpdateId`	string (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. Enumerazione: CREATE_PENDING | CREATE_IN_PROGRESS | CREATE_COMPLETE | CREATE_FAILED

Di seguito è riportato un esempio di file JSON passato al comando create-ota-update 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 job ID esistente per la firma del codice.

```
[
  {
    "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

Puoi usare 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 Descrivere l'esecuzione dei processi.

Eliminazione di dati correlati al sistema OTA

Al momento, non è possibile utilizzare la console AWS IoT per eliminare flussi o gli aggiornamenti OTA. È possibile utilizzare AWS CLI per eliminare i flussi, gli aggiornamenti OTA e i processi AWS IoT 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 console AWS IoT per creare un flusso per suddividere il firmware in blocchi in modo che possa essere inviato tramite MQTT. Puoi eliminare questo flusso con il delete-stream AWS CLI comando, come mostrato 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 processo AWS IoT per eseguire l'aggiornamento.
- Esecuzione di un processo AWS IoT per ogni dispositivo che viene aggiornato.

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 processo AWS IoT 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 un AWS IoT job 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 processo e le relative esecuzioni di processi associati.

```
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
```

Come con il delete-job AWS CLI comando, è possibile passare il `--force` parametro a per delete-job-execution forzare l'eliminazione dell'esecuzione di un processo. 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

Il servizio over-the-air (OTA) Update Manager consente di:

- Creare un aggiornamento OTA e le risorse utilizzate, inclusi un'attività AWS IoT, un flusso AWS IoT 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 Offerte di lavoro della Guida per gli AWS IoT sviluppatori per una descrizione del 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 AWS IoT consente l'accesso ad Amazon S3, ai AWS IoT job e alle risorse di AWS Code Signing per creare un processo di aggiornamento OTA.
- Elenco facoltativo di coppie nome-valore definite dall'utente.

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 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 puoi 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 di FreeRTOS Agent per regolare la dimensione dei blocchi che invii e ricevi.

Quando crei un aggiornamento OTA, il servizio OTA Manager crea un'attività AWS IoT per notificare ai dispositivi che è disponibile un aggiornamento. L'agente OTA di 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

L'agente over-the-air (OTA) è progettato per semplificare la quantità di codice da scrivere per aggiungere funzionalità di aggiornamento OTA al prodotto. Tale onere di integrazione consiste

principalmente nell'inizializzazione dell'agente OTA e nella creazione di una funzione di callback personalizzata per rispondere ai messaggi degli eventi dell'agente OTA. Durante l'inizializzazione del sistema operativo, le interfacce MQTT, HTTP (se si utilizza HTTP per il download dei file) e le interfacce di implementazione specifica della piattaforma (PAL) vengono passate all'agente OTA. I buffer possono anche essere inizializzati e passati all'agente 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 dimestichezza con la configurazione del tuo AWS account con AWS IoT elementi, credenziali, certificati di firma del codice, dispositivi di provisioning e processi di aggiornamento OTA, vedi Prerequisiti di FreeRTOS.

Gestione delle connessioni

L'agente OTA utilizza il protocollo MQTT per tutte le operazioni di comunicazione di controllo che coinvolgono i servizi AWS IoT, 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, OTA Agent utilizza la stessa connessione per le operazioni di controllo e per il download di file.

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 predefinito dell'applicazione OTA 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 quindi tenta di ristabilire la connessione MQTT. I tentativi di riconnessione MQTT vengono ritardati di un tempo che viene aumentato esponenzialmente fino a un valore massimo e viene inoltre aggiunto un jitter. Se la connessione viene ristabilita, l'agente OTA continua le sue operazioni.

Per un esempio funzionante che usa il broker MQTT AWS IoT, consulta il codice della demo OTA nella directory `demos/ota`.

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.

```
static BaseType_t prvRunOTADemo( void )
{
    /* Status indicating a successful demo or not. */
    BaseType_t xStatus = pdFAIL;

    /* OTA library return status. */
    OtaErr_t xOtaError = OtaErrUninitialized;

    /* OTA event message used for sending event to OTA Agent.*/
    OtaEventMsg_t xEventMsg = { 0 };

    /* OTA interface context required for library interface functions.*/
    OtaInterfaces_t xOtaInterfaces;

    /* OTA library packet statistics per job.*/
    OtaAgentStatistics_t xOtaStatistics = { 0 };

    /* OTA Agent state returned from calling OTA_GetState.*/
    OtaState_t xOtaState = OtaAgentStateStopped;

    /* Set OTA Library interfaces.*/
    prvSetOtaInterfaces( &xOtaInterfaces );

    /************************** Init OTA Library. **************************/

    if( ( xOtaError = OTA_Init( &xOtaBuffer,
                                &xOtaInterfaces,
                                ( const uint8_t * ) ( democonfigCLIENT_IDENTIFIER ),
                                prvOtaAppCallback ) ) != OtaErrNone )
    {
        LogError( ( "Failed to initialize OTA Agent, exiting = %u.",
                    xOtaError ) );
    }
    else
    {
        xStatus = pdPASS;
    }
```

```
    /************************ Create OTA Agent Task. ************************/

    if( xStatus == pdPASS )
    {
        xStatus = xTaskCreate( prvOTAAgentTask,
                               "OTA Agent Task",
                               otaexampleAGENT_TASK_STACK_SIZE,
                               NULL,
                               otaexampleAGENT_TASK_PRIORITY,
                               NULL );

        if( xStatus != pdPASS )
        {
            LogError( ( "Failed to create OTA agent task:" ) );
        }
    }

    /***************************** Start OTA ******************************/

    if( xStatus == pdPASS )
    {
        /* Send start event to OTA Agent.*/
        xEventMsg.eventId = OtaAgentEventStart;
        OTA_SignalEvent( &xEventMsg );
    }

    /******************** Loop and display OTA statistics ********************/

    if( xStatus == pdPASS )
    {
        while( ( xOtaState = OTA_GetState() ) != OtaAgentStateStopped )
        {
            /* Get OTA statistics for currently executing job. */
            if( xOtaState != OtaAgentStateSuspended )
            {
                OTA_GetStatistics( &xOtaStatistics );

                LogInfo( ( " Received: %u   Queued: %u   Processed: %u   Dropped: %u",
                           xOtaStatistics.otaPacketsReceived,
                           xOtaStatistics.otaPacketsQueued,
                           xOtaStatistics.otaPacketsProcessed,
                           xOtaStatistics.otaPacketsDropped ) );
            }
```

```
            vTaskDelay( pdMS_TO_TICKS( otaexampleEXAMPLE_TASK_DELAY_MS ) );
        }
    }

    return xStatus;
}
```

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 nuovamente con ritardo esponenziale e jitter.
- In caso di riconnessione, riprendi le operazioni dell'agente OTA.
- Se l'agente si ferma, ritarda di un secondo, quindi prova a riconnetterti.

Utilizzo del callback dell'applicazione per gli eventi OTA Agent

L'esempio precedente utilizzato `prvOtaAppCallback` come gestore di callback per gli eventi OTA Agent. (Vedi il quarto parametro della chiamata `OTA_Init` API). Se si desidera implementare una gestione personalizzata degli eventi di completamento, è necessario modificare la gestione predefinita nella demo/applicazione OTA. Durante il processo OTA, l'agente OTA può inviare una delle seguenti enumerazioni di eventi al gestore di callback. Sarà lo sviluppatore dell'applicazione a decidere come e quando gestire tali eventi.

```
/**
 * @ingroup ota_enum_types
 * @brief OTA Job callback events.
 *
 * After an OTA update image is received and authenticated, the agent calls the user
 * callback (set with the @ref OTA_Init API) with the value OtaJobEventActivate to
 * signal that the device must be rebooted to activate the new image. When the device
 * boots, if the OTA job status is in self test mode, the agent calls the user callback
 * with the value OtaJobEventStartTest, signaling that any additional self tests
 * should be performed.
```

```
 *
 * If the OTA receive fails for any reason, the agent calls the user callback with
 * the value OtaJobEventFail instead to allow the user to log the failure and take
 * any action deemed appropriate by the user code.
 *
 * See the OtaImageState_t type for more information.
 */
typedef enum OtaJobEvent
{
    OtaJobEventActivate = 0,       /*!< @brief OTA receive is authenticated and ready
 to activate. */
    OtaJobEventFail = 1,           /*!< @brief OTA receive failed. Unable to use this
 update. */
    OtaJobEventStartTest = 2,      /*!< @brief OTA job is now in self test, perform
 user tests. */
    OtaJobEventProcessed = 3,      /*!< @brief OTA event queued by OTA_SignalEvent is
 processed. */
    OtaJobEventSelfTestFailed = 4, /*!< @brief OTA self-test failed for current job. */
    OtaJobEventParseCustomJob = 5, /*!< @brief OTA event for parsing custom job
 document. */
    OtaJobEventReceivedJob = 6,    /*!< @brief OTA event when a new valid AFT-OTA job
 is received. */
    OtaJobEventUpdateComplete = 7, /*!< @brief OTA event when the update is completed.
 */
    OtaLastJobEvent = OtaJobEventStartTest
} OtaJobEvent_t;
```

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:

`OtaJobEventActivate`

Quando il gestore di callback riceve questo evento, puoi reimpostare immediatamente il dispositivo o pianificare una chiamata per reimpostare il dispositivo in un secondo momento. In questo modo è possibile rinviare la fase si reimpostazione del dispositivo e di self-test, se necessario.

`OtaJobEventFail`

Quando il gestore di callback riceve questo evento, l'aggiornamento non è riuscito. 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.

`OtaJobEventStartTest`

La fase di autotest ha lo scopo di consentire al firmware appena aggiornato di eseguirsi e testarsi prima di determinare se funziona correttamente e impegnarsi a essere l'ultima immagine permanente dell'applicazione. Quando viene ricevuto e autenticato un nuovo aggiornamento e il dispositivo è stato ripristinato, l'agente OTA invia l'evento `OtaJobEventStartTest` 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( OtaImageStateAccepted )`.

`OtaJobEventProcessed`

L'evento OTA messo in coda da `OTA_SignalEvent` viene elaborato, quindi è possibile eseguire operazioni di pulizia come la liberazione dei buffer OTA.

`OtaJobEventSelfTestFailed`

L'autotest OTA non è riuscito per il lavoro corrente. La gestione predefinita per questo evento consiste nello spegnere l'agente OTA e riavviarlo in modo che il dispositivo torni all'immagine precedente.

`OtaJobEventUpdateComplete`

L'evento di notifica per il completamento dell'aggiornamento del processo OTA.

Sicurezza OTA

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 degli aggiornamenti OTA passa attraverso il gateway di dispositivo AWS IoT e utilizza meccanismi di

sicurezza AWS IoT. 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 Version 4. Per creare una distribuzione, è necessario disporre delle autorizzazioni per invocare le API `CreateDeployment`, `CreateJob` e `CreateStream`. 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.

Code Signing for AWS IoT

La console AWS IoT utilizza Code Signing for AWS IoT per firmare automaticamente l'immagine del firmware per i dispositivi supportati 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 una nota autorità di certificazione commerciale (CA).

I certificati con 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 Guida per gli sviluppatori di Code Signing for AWS IoT e Riferimento alle API di Code Signing for AWS IoT.

> **Note**
>
> Puoi scaricare l'SDK Code Signing for AWS IoT da Strumenti per Amazon Web Services.

Risoluzione dei problemi di OTA

Le sezioni seguenti contengono informazioni che possono essere utili per risolvere i problemi con gli aggiornamenti OTA.

Argomenti

Configura CloudWatch i registri per gli aggiornamenti OTA

Il servizio OTA Update supporta la registrazione con AmazonCloudWatch. 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 accesso ai CloudWatch registri. Gli utenti con accesso ai CloudWatch log 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 Update (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 processo AWS IoT.
- Viene completato un processo AWS IoT.
- 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 Gruppi di log, 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`: un processo AWS IoT è stato creato.
- `CreateStream`: un flusso è stato creato.
- `GetStream`: è stata inviata una richiesta per uno stream 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 consegna dei file AWS IoT basata su MQTT.

awsJobId

L'ID processo AWS IoT che ha generato il log.

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:

- Success (Riuscito)
- Errore

streamId

L'ID del flusso AWS IoT 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 processo AWS IoT:

```
{
    "timestamp": "2018-07-23 22:59:45.363",
    "logLevel": "DEBUG",
    "accountId": "123456789012",
    "status": "Success",
    "actionType": "CreateAWSJob",
    "otaUpdateId": "08957b03-eea3-448a-87fe-743e6891ca3a",
    "awsJobId": "08957b03-eea3-448a-87fe-743e6891ca3a",
    "details": "Create AWS Job The request status is SUCCESS."
}
```

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 registro 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."
}
```

Registrazione delle chiamate API OTA AWS IoT con AWS CloudTrail

FreeRTOS è integrato con CloudTrail un servizio che acquisisce le chiamate API AWS IoT OTA e consegna i file di registro a un bucket Amazon S3 specificato dall'utente. CloudTrailacquisisce le chiamate API dal codice alle API AWS IoT OTA. Le informazioni raccolte da CloudTrail consentono di individuare la richiesta effettuata ad AWS IoT OTA, l'indirizzo IP di origine da cui è stata eseguita la richiesta, l'autore della richiesta, il momento in cui è stata eseguita e così via.

Per ulteriori informazioni su CloudTrail, incluse le modalità di configurazione e abilitazione, consulta la Guida per l'utente di AWS CloudTrail.

Informazioni su FreeRTOS in CloudTrail

Quando la CloudTrail registrazione è abilitata nel tuo AWS account, le chiamate API effettuate alle azioni AWS IoT OTA vengono tracciate nei file di CloudTrail registro in cui vengono scritte con altri

record di AWS servizio. CloudTrail determina quando creare e compilare un nuovo file in base a un periodo di tempo e alle dimensioni del file.

Le seguenti operazioni di AWS IoT relative al piano di controllo OTA vengono registrate da CloudTrail.

- CreateStream
- DescribeStream
- ListStreams
- UpdateStream
- DeleteStream
- CreateOTAUpdate
- GetOTAUpdate
- ListOTAUpdates
- DeleteOTAUpdate

> **Note**
>
> Le operazioni di AWS IoT relative al piano dei dati OTA (lato dispositivo) non vengono registrate da CloudTrail. Usa CloudWatch 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 servizio AWS.

Per ulteriori informazioni, vedere l'elemento CloudTrail userIdentity. AWS IoT Le azioni OTA sono documentate nell'AWS IoTOTA API Reference.

Puoi archiviare i tuoi file di registro 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 registro sono crittografati con la crittografia lato server (SSE) di Amazon S3.

Se desideri ricevere una notifica quando i file di registro vengono consegnati, puoi configurare CloudTrail la pubblicazione di notifiche Amazon SNS. Per ulteriori informazioni, consulta Configurazione delle notifiche Amazon SNS per. CloudTrail

Puoi anche aggregare file di registro AWS IoT OTA da più AWS regioni e più AWS account in un unico bucket Amazon S3.

Per ulteriori informazioni, vedere 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

I file di log di CloudTrail possono contenere una o più voci di log. 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 log di CloudTrail relativa a una chiamata all'operazione `CreateOTAUpdate`.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "EXAMPLE",
        "arn": "arn:aws:iam::your_aws_account:user/your_user_id",
        "accountId": "your_aws_account",
        "accessKeyId": "your_access_key_id",
        "userName": "your_username",
        "sessionContext": {
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2018-08-23T17:27:08Z"
            }
        },
        "invokedBy": "apigateway.amazonaws.com"
    },
    "eventTime": "2018-08-23T17:27:19Z",
```

```
    "eventSource": "iot.amazonaws.com",
    "eventName": "CreateOTAUpdate",
    "awsRegion": "your_aws_region",
    "sourceIPAddress": "apigateway.amazonaws.com",
    "userAgent": "apigateway.amazonaws.com",
    "requestParameters": {
        "targets": [
            "arn:aws:iot:your_aws_region:your_aws_account:thing/Thing_CMH"
        ],
        "roleArn": "arn:aws:iam::your_aws_account:role/Role_FreeRTOSJob",
        "files": [
            {
                "fileName": "/sys/mcuflashimg.bin",
                "fileSource": {
                    "fileId": 0,
                    "streamId": "your_stream_id"
                },
                "codeSigning": {
                    "awsSignerJobId": "your_signer_job_id"
                }
            }
        ],
        "targetSelection": "SNAPSHOT",
        "otaUpdateId": "FreeRTOSJob_CMH-23-1535045232806-92"
    },
    "responseElements": {
        "otaUpdateArn": "arn:aws:iot:your_aws_region:your_aws_account:otaupdate/
FreeRTOSJob_CMH-23-1535045232806-92",
        "otaUpdateStatus": "CREATE_PENDING",
        "otaUpdateId": "FreeRTOSJob_CMH-23-1535045232806-92"
    },
    "requestID": "c9649630-a6f9-11e8-8f9c-e1cf2d0c9d8e",
    "eventID": "ce9bf4d9-5770-4cee-acf4-0e5649b845c0",
    "eventType": "AwsApiCall",
    "recipientAccountId": "recipient_aws_account"
}
```

Ottieni i dettagli degli errori di CreateOtaUpdate utilizzando AWS CLI

Se il processo di creazione di un processo di aggiornamento OTA fallisce, è possibile intraprendere delle azioni per risolvere il problema. Quando crei un processo di aggiornamento OTA, il servizio di gestione OTA crea un processo IoT e lo pianifica per i dispositivi di destinazione, inoltre questo processo crea o utilizza 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 AWS IoT processo. In questa sezione di 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
   ```

   Dove *ota_update_job_001* è l'ID che hai fornito all'aggiornamento OTA al momento della creazione.
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.

Ottenere i codici di errore OTA con la AWS CLI

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
```

Dove *`jobID`* è la stringa completa dell'ID del processo di cui vogliamo ottenere lo stato (era associato al processo di aggiornamento OTA quando è stato creato) ed *`ThingName`*è il nome dell'AWS IoToggetto in cui è registrato il dispositivo 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 "`k0TA_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

Per eseguire OTA su più dispositivi (oggetti) utilizzando la stessa immagine del firmware, implementare 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.

Risoluzione dei problemi degli aggiornamenti OTA con Texas Instruments CC3220SF Launchpad

La piattaforma software CC3220SF Launchpad fornisce un meccanismo di rilevamento di manomissione. 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 il restore-to-factory processo per ripristinare l'immagine di fabbrica. È consigliabile programmare il comportamento desiderato aggiornando il gestore eventi asincroni in `network_if.c`.

Librerie

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 conAWS IoT i servizi. FreeRTOS include librerie che consentono di:

- Connettere in modo sicuro i dispositivi al cloud AWS IoT usando MQTT e copie shadow del dispositivo.
- Individuare e connettersi ai componenti principali AWS IoT Greengrass.
- Gestire connessioni Wi-Fi.
- Ascolta ed elabora Aggiornamenti via etere di FreeRTOS.

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

Le seguenti librerie di porting sono incluse nelle configurazioni di FreeRTOS disponibili per il download sulla console FreeRTOS. Queste librerie sono dipendenti dalla piattaforma. I loro contenuti variano in base alla piattaforma hardware. Per informazioni sulla portabilità di queste librerie su un dispositivo, consulta la Guida alla portabilità di FreeRTOS.

Librerie

Libreria	Documentazione di riferimento delle API	Descrizione
Bluetooth Low Energy	Documentazione di riferimento dell'API Bluetooth Low Energy	Utilizzando la libreria FreeRTOS Bluetooth Low Energy, il microcontrollore può comunicare con il brokerAWS IoT MQTT tramite un dispositivo gateway. Per ulteriori informazioni, consulta Libreria Bluetooth Low Energy.
Aggiornamenti over-the-air	AWS IoTOver-the-air aggiorna il riferimento dell'API	La libreria di aggiornamenti FreeRTOSAWS IoT Over-the-air (OTA) consente di gestire le notifiche di aggiornamento, scaricare gli aggiornamenti ed eseguire la verifica crittografica degli aggiornamenti del firmware sul dispositivo FreeRTOS. Per ulteriori informazioni, consulta AWS IoTLibreria via etere (OTA).
FreeRTOS+POSIX	Documentazione di riferimento all'API FreeRTOS+POSIX	È possibile utilizzare la libreria FreeRTOS+POSIX per trasferire applicazioni conformi a POSIX nell'ecosistema FreeRTOS. Per ulteriori informazioni, consulta FreeRTOS+POSIX.

Libreria	Documentazione di riferimento delle API	Descrizione
Secure Sockets	Riferimento all'API Secure Sockets	Per ulteriori informazioni, consulta Libreria Secure Sockets.
FreeRTOS+TCP	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.
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 microcontrollore. Per ulteriori informazioni, consulta la Libreria Wi-Fi.
Core PKCS 11		La libreria CorePKCS11 è un'implementazione di riferimento del Public Key Cryptography Standard #11, per supportare il provisioning e l'autenticazione del client TLS. Per ulteriori informazioni, consulta la Libreria del lettore ivS.
TLS		Per ulteriori informazioni, consulta Transport Layer Security.
I/O comuni	Riferimento comune all'API I/O	Per ulteriori informazioni, consulta I/O comuni.

Libreria	Documentazione di riferimento delle API	Descrizione
Interfaccia cellulare	Documentazione di riferimento	La libreria Cellular Interface espone le funzionalità di alcuni modem cellulari popolari tramite un'API uniforme. Per ulteriori informazioni, consulta la Libreria di interfaccia cellulare.

Librerie

Puoi opzionalmente includere le seguenti librerie di applicazioni autonome nella configurazione di FreeRTOS per interagire conAWS IoT i servizi sul cloud.

> **Note**
>
> Alcune librerie di applicazioni hanno le stesse API delle librerie diAWS IoT Device SDK for Embedded C. Per queste librerie, consulta il AWS IoTDevice SDK C API Reference. Per ulteriori informazioni sulAWS IoT Device SDK for Embedded C, consultaAWS IoT Device SDK for Embedded C.

Librerie

Libreria	Documentazione di riferimento delle API	Descrizione
AWS IoT Device Defender	Riferimento all'API SDK di Device Defender C	LaAWS IoT Device Defender libreria FreeRTOS connette il tuo dispositivo FreeRTOS aAWS IoT Device Defender. Per ulteriori informazioni, consulta Libreria AWS IoT Device Defender.
AWS IoT Greengrass	Riferimento alle API Greengrass	LaAWS IoT Greengrass libreria FreeRTOS connette il tuo dispositi

Libreria	Documentazione di riferimento delle API	Descrizione
		vo FreeRTOS aAWS IoT Greengras s. Per ulteriori informazioni, consulta Libreria Discovery AWS IoT Greengrass.
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.x) C	La libreria CoreMQTT fornisce un client per il tuo dispositivo MQTT è il protocollo utilizzato dai dispositivi per interagire con AWS IoT. Per ulteriori informazioni sulla versione 3.0.0 della libreria CoreMQTT, vederelibreria CoreMQTT.
Agente Core	Riferimento all'API della libreria Agent CoreMQTT	La libreria CoreMQTT Agent è un'API di alto livello che aggiunge la sicurezza dei thread alla libreria CoreMQTT. Consente di creare un'attività di agente MQTT dedicata che gestisce una connessione MQTT in background e non richiede l'intervento di altre attività. La libreria fornisce thread safe equivalenti alle API di CoreMQTT, quindi può essere utilizzata in ambienti multi-thr ead. Per ulteriori informazioni sulla libreria CoreLibreria del componente CoreMQTT

Libreria	Documentazione di riferimento delle API	Descrizione
AWS IoT Device Shadow	Documentazione di riferimento delle API SDK C di Device Shadow	La libreriaAWS IoT Device Shadow consente al tuo dispositivo FreeRTOS di interagire con le ombreAWS IoT del dispositivo. Per ulteriori informazioni, consulta Libreria AWS IoT Device Shadow.

Configurazione delle librerie FreeRTOS

Le impostazioni di configurazione per FreeRTOS eAWS 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 di FreeRTOS.

Per ulteriori informazioni sulle opzioni di configurazione globali supportate, consulta Documentazione di riferimento sui file di configurazione globali.

Libreria delle librerie Backoff

> **Note**
>
> Il contenuto di questa pagina potrebbe non essere up-to-date. Si prega di fare riferimento alPagina della libreria Freertos.orgper l'ultimo aggiornamento.

Introduzione

LaAlgoritmo BackOfflibrary è una libreria di utilità 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 unbackoff esponenziale con jitteralgoritmo.

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 aISO C90eMISRA 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 sottoLicenza open source MIT.

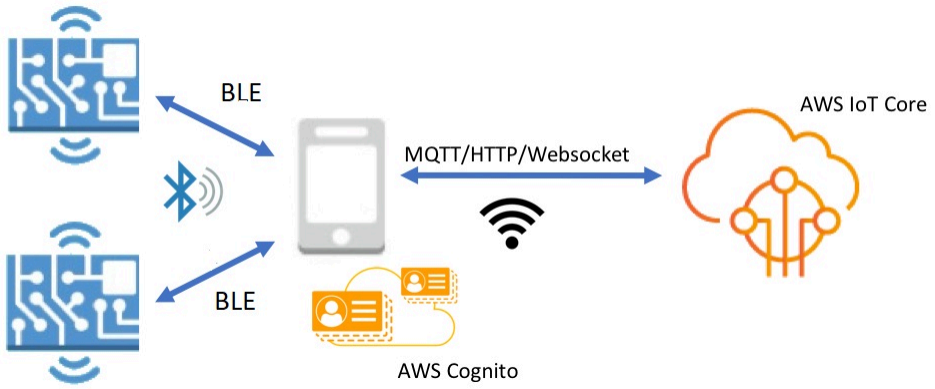
Dimensione del codice di BackOffAlgorithm (esempio generato con GCC per ARM Cortex-M)		
File	Con ottimizzazione -O1	Con ottimizzazione -Os
backoff_algorithm.c	0,1 K	0,1 K
Stime totali	0,1 K	0,1 K

Libreria Bluetooth Low Energy

> **Important**
>
> Questa libreria è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Consigliamo di iniziare da qui quando si crea un nuovo progetto. Se hai già un progetto FreeRTOS esistente basato sull'ormai obsoleto repository Amazon-FreeRTOS, consulta ilGuida alla migrazione del repository Github di Amazon-FreeRTOS.

Panoramica

FreeRTOS supporta la pubblicazione e la sottoscrizione di argomenti MQTT (Message Queuing Telemetry Transport) tramite Bluetooth Low Energy tramite un dispositivo proxy, come un telefono cellulare. Con la libreria FreeRTOS Bluetooth Low Energy (BLE), il tuo microcontrollore può comunicare in modo sicuro con il brokerAWS IoT MQTT.

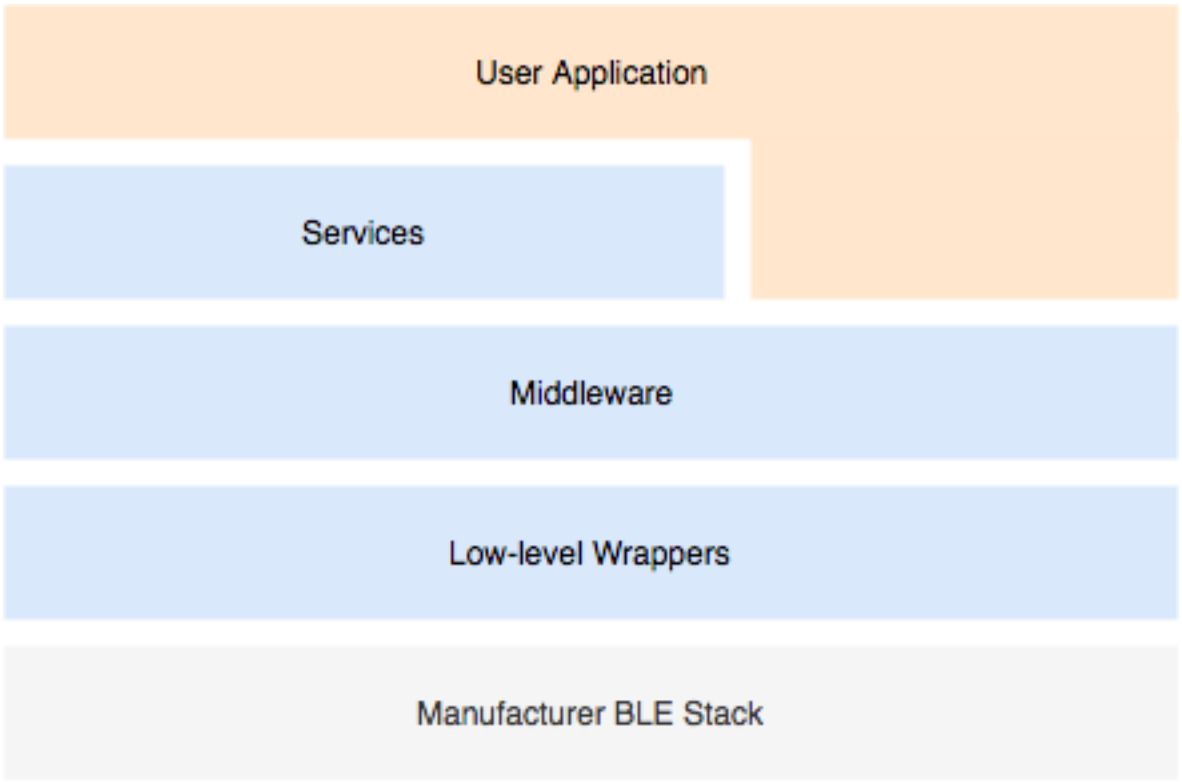


Utilizzando gli SDK mobili per dispositivi Bluetooth FreeRTOS, puoi scrivere applicazioni mobili native che comunicano con le applicazioni integrate sul tuo microcontrollore tramite BLE. Per ulteriori informazioni sugli SDK per dispositivi mobili, consulta SDK mobili per dispositivi Bluetooth FreeRTOS.

La libreria FreeRTOS BLE include servizi per la configurazione di reti Wi-Fi, il trasferimento di grandi quantità di dati e la fornitura di astrazioni di rete tramite BLE. La libreria FreeRTOS BLE include anche middleware e API di livello inferiore per un controllo più diretto sullo stack BLE.

Architettura

La libreria BLE di FreeRTOS è composta da tre livelli: servizi, middleware e wrapper di basso livello.



Servizi

Il livello di servizi FreeRTOS BLE è composto da quattro servizi Generic Attribute (GATT) che sfruttano le API middleware:

- Informazioni sul dispositivo
- Provisioning Wi-Fi
- Astrazione di rete
- trasferimento di oggetti

Informazioni sul dispositivo

Il servizio di informazioni sul dispositivo raccoglie dettagli sul microcontrollore, tra cui:

- La versione di FreeRTOS utilizzata dal tuo dispositivo.
- L'endpoint AWS IoT dell'account per il quale il dispositivo è registrato.
- 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 Large Object Transfer invia e riceve dati da un client. Altri servizi, come il provisioning Wi-Fi e l'astrazione della rete, utilizzano il servizio Large Object Transfer per inviare e ricevere dati. Puoi anche utilizzare l'API Large Object Transfer per interagire direttamente con il servizio.

MQTT over BLE

MQTT over BLE contiene il profilo GATT per la creazione di un servizio proxy MQTT su BLE. Il servizio proxy MQTT consente a un client MQTT di comunicare con il brokerAWS MQTT tramite un dispositivo gateway. Ad esempio, puoi utilizzare il servizio proxy per connettere un dispositivo che esegue FreeRTOS aAWS MQTT tramite un'app per smartphone. Il dispositivo BLE è il server GATT ed espone i servizi e le caratteristiche del dispositivo gateway. Il server GATT utilizza questi servizi e caratteristiche esposti per eseguire operazioni MQTT con il cloud per quel dispositivo. Per ulteriori informazioni, consulta Appendice A: profilo MQTT su BLE GATT .

Middleware

Il middleware FreeRTOS Bluetooth Low Energy è un'astrazione delle API di livello inferiore. Le API del middleware costituiscono un'interfaccia più intuitiva allo stack Bluetooth Low Energy.

Utilizzando le API del middleware, puoi registrare diversi callback, su più livelli, in 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

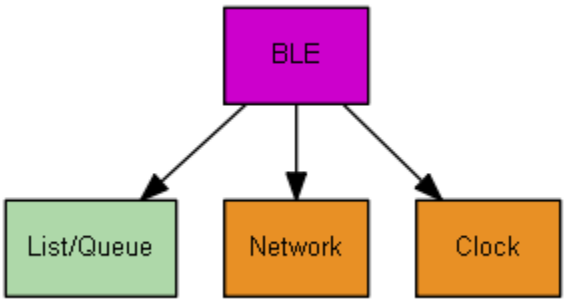
I wrapper Bluetooth Low Energy FreeRTOS di basso livello sono un'astrazione dello stack Bluetooth Low Energy del produttore. I wrapper di basso livello offrono un set comune di API per il controllo diretto dell'hardware. Le API di basso livello ottimizzano l'uso della RAM, ma presentano limiti di funzionalità.

Utilizza le API del servizio Bluetooth Low Energy per interagire con i servizi Bluetooth Low Energy. Le API del servizio richiedono un maggior numero di risorse rispetto alle API di basso livello.

Dipendenze e requisiti

La libreria Bluetooth Low Energy è caratterizzata dalle seguenti dipendenze dirette:

- Libreria di contenitori lineari
- 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 account AWS 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 un pool di utenti Amazon Cognito. A ogni Amazon Cognito Identity deve essere associata la politica 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 su Bluetooth Low Energy devono fornire un file di`iot_ble_config.h` 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:

- Le API di basso livello utilizzano meno RAM, ma offrono 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 laeBTpropertySecureConnectionOnly proprietà su TRUE, che posiziona 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**
>
> Questa fase di inizializzazione non è necessaria se si eseguono leAWS demo. L'inizializzazione delle demo viene gestita dal network manager, disponibile in *freertos*`/demos/network_manager`.

API di basso livello

Se non desideri utilizzare i servizi GATT Bluetooth Low Energy di FreeRTOS, puoi bypassare il middleware e interagire direttamente con le API di basso livello per risparmiare risorse.

Per inizializzare le API di basso livello

1. È necessario inizializzare qualsiasi driver hardware Bluetooth Low Energy prima di chiamare le API. L'inizializzazione del driver non fa parte delle API Bluetooth Low Energy di basso livello.
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 le API, è necessario abilitare Bluetooth Low Energy.

```
const BTInterface_t * pxIface = BTGetBluetoothInterface();
xStatus = pxIface->pxEnable( 0 );
```

3. Il manager Bluetooth contiene le API comuni a Bluetooth Low Energy e Bluetooth classico. 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 ] );
```

Documentazione di riferimento dell'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. Creare gli UUID 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 trasferire FreeRTOS su una nuova destinazione, è necessario implementare alcune API per il servizio Wi-Fi Provisioning e la funzionalità Bluetooth Low Energy.

API Bluetooth Low Energy

Per utilizzare il middleware FreeRTOS Bluetooth Low Energy, è necessario implementare alcune API.

API comuni tra GAP per Bluetooth classico 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`

API specifiche di 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 sulla portabilità della libreria FreeRTOS Bluetooth Low Energy sulla tua piattaforma, consulta Porting the Bluetooth Low Energy Library nella FreeRTOS Porting Guide.

SDK mobili per dispositivi Bluetooth FreeRTOS

> **Important**
>
> Questa libreria è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Consigliamo di iniziare da qui quando si crea un nuovo progetto. Se hai già un progetto FreeRTOS esistente basato sull'ormai obsoleto repository Amazon-FreeRTOS, consulta ilGuida alla migrazione del repository Github di Amazon-FreeRTOS.

Puoi utilizzare gli SDK mobili per dispositivi Bluetooth FreeRTOS per creare applicazioni mobili che interagiscono con il tuo microcontrollore tramite Bluetooth Low Energy. Gli SDK mobili possono anche comunicare conAWS i servizi, utilizzando Amazon Cognito per l'autenticazione degli utenti.

SDK Android per dispositivi Bluetooth FreeRTOS

Usa l'SDK Android per i 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 i 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 utilizzarlosudo 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.

Appendice A: profilo MQTT su BLE GATT

Dettagli del servizio GATT

MQTT over BLE utilizza un'istanza del servizio GATT di trasferimento dati per inviare messaggi MQTT Concise Binary Object Representation (CBOR) tra il dispositivo FreeRTOS e il dispositivo proxy. Il servizio di trasferimento dati espone determinate caratteristiche che aiutano a inviare

e ricevere dati grezzi tramite il protocollo BLE GATT. Gestisce inoltre la frammentazione e l'assemblaggio di carichi utili superiori alle dimensioni dell'unità di trasferimento massima (MTU) BLE.

UUID di servizio

`A9D7-166A-D72E-40A9-A002-4804-4CC3-FF00`

Istanze di servizio

Viene creata un'istanza del servizio GATT per ogni sessione MQTT con il broker. Ogni servizio ha un UUID univoco (due byte) che ne identifica il tipo. Ogni singola istanza è differenziata dall'ID dell'istanza.

Ogni servizio viene istanziato come servizio primario su ogni dispositivo server BLE. È possibile creare più istanze del servizio su un determinato dispositivo. Il tipo di servizio proxy MQTT dispone di un UUID univoco.

Caratteristiche

Formato del contenuto caratteristico: CBOR

Dimensione massima del valore caratteristico: 512 byte

Caratteristica	Requisito	Proprietà obbligatorie	Proprietà opzionali	Autorizzazioni di sicurezza	Breve descrizione	UUID
Controllo	M	Scrittura	Nessuno	La scrittura richiede la crittografia	Utilizzato per avviare e interrompere il proxy MQTT.	`A9D7-166A-D72E-40A9-A002-4804-4CC3-FF01`
Messaggio TX	M	Lettura, notifica	Nessuno	La lettura richiede la crittografia	Utilizzato per inviare una notifica contenente un	`A9D7-166A-D72E-40A9-A002-48`

Caratteristica	Requisito	Proprietà obbligatorie	Proprietà opzionali	Autorizzazioni di sicurezza	Breve descrizione	UUID
					messaggio a un broker tramite un proxy.	04-4CC3-FF02
Messaggio RX	M	Leggi, scrivi senza risposta	Nessuno	La lettura e la scrittura richiedono la crittografia	Utilizzato per ricevere un messaggio da un broker tramite un proxy.	A9D7-166A-D72E-40A9-A002-4804-4CC3-FF03
TXLargeMessage	M	Lettura, notifica	Nessuno	La lettura richiede la crittografia	Utilizzato per inviare un messaggio di grandi dimensioni (Messaggio > BLE MTU Size) a un broker tramite un proxy.	A9D7-166A-D72E-40A9-A002-4804-4CC3-FF04

Caratteristica	Requisito	Proprietà obbligatorie	Proprietà opzionali	Autorizzazioni di sicurezza	Breve descrizione	UUID
RXLargeMessage	M	Leggi, scrivi senza risposta	Nessuno	La lettura e la scrittura richiedono la crittografia	Utilizzato per ricevere messaggi di grandi dimensioni (Messaggio > BLE MTU Size) da un broker tramite un proxy.	A9D7-166A-D72E-40A9-A002-4804-4CC3-FF05

Requisiti della procedura GATT

Leggi i valori caratteristici	Obbligatorio
Leggi valori caratteristici lunghi	Obbligatorio
Scrivere valori caratteristici	Obbligatorio
Scrivi valori caratteristici lunghi	Obbligatorio
Leggi Descrittori caratteristici	Obbligatorio
Descrittori delle caratteristiche	Obbligatorio
Notifiche	Obbligatorio
Indicazioni	Obbligatorio

Tipi di messaggi

Vengono scambiati i seguenti tipi di messaggi.

Tipo di messaggio	Message	Mappa con queste coppie chiave-valore
0x01	CONNECT	• Chiave = «w», valore = Numero intero di tipo 0, tipo di messaggio (1) • Chiave = «d», valore = Tipo 3, stringa di testo, identificatore client per la sessione • Chiave = «a», valore = Tipo 3, stringa di testo, endpoint del broker per la sessione • Chiave = «c», Valore = Tipo di valore semplice Vero/Falso
0x02	CONNACK	• Chiave = «w», valore = Numero intero di tipo 0, tipo di messaggio (2) • Chiave = «s», valore = numero intero di tipo 0, codice di stato
0x03	PUBLISH	• Chiave = «w», valore = Numero intero di tipo 0, tipo di messaggio (3) • Chiave = «u», valore = Tipo 3, stringa di testo, argomento da pubblicare

Tipo di messaggio	Message	Mappa con queste coppie chiave-valore
		• Chiave = «n», valore = Tipo 0, numero intero, QoS per la pubblicazione • Chiave = «i», valore = tipo 0, numero intero, identific atore del messaggio, solo per le pubblicazioni QoS 1 • Chiave = «k», valore = tipo 2, stringa di byte, payload per la pubblicazione
0x04	PANCA	• Inviato solo per messaggi QoS 1. • Chiave = «w», valore = Numero intero di tipo 0, tipo di messaggio (4) • Chiave = «i», valore = Tipo 0, numero intero, identific atore del messaggio

Tipo di messaggio	Message	Mappa con queste coppie chiave-valore
0x08	SUBSCRIBE	• Chiave = «w», valore = Numero intero di tipo 0, tipo di messaggio (8) • Chiave = «v», valore = Tipo 4, matrice di stringhe di testo, argomenti per la sottoscrizione • Chiave = «o», valore = Tipo 4, matrice di numeri interi, QoS per l'abbonamento • Chiave = «i», valore = Tipo 0, numero intero, identificatore del messaggio
0x09	SUBACK	• Chiave = «w», valore = Numero intero di tipo 0, tipo di messaggio (9) • Chiave = «i», valore = Tipo 0, numero intero, identificatore del messaggio • Chiave = «s», valore = Tipo 0, numero intero, codice di stato per l'abbonamento

Tipo di messaggio	Message	Mappa con queste coppie chiave-valore
0X0A	UNSUBSCRIBE	• Chiave = «w», valore = Numero intero di tipo 0, tipo di messaggio (10) • Chiave = «v», valore = Tipo 4, Matrice di stringhe di testo, argomenti per la cancellazione dell'iscr izione • Chiave = «i», valore = Tipo 0, numero intero, identific atore del messaggio
0x0B	UNSUBACK	• Chiave = «w», valore = Numero intero di tipo 0, tipo di messaggio (11) • Chiave = «i», valore = Tipo 0, numero intero, identific atore del messaggio • Chiave = «s», valore = Tipo 0, numero intero, codice di stato per UnSubscription
0X0C	PINGREQ	• Chiave = «w», valore = Numero intero di tipo 0, tipo di messaggio (12)
0x0D	PINGRESP	• Chiave = «w», valore = Numero intero di tipo 0, tipo di messaggio (13)
0x0E	DISCONNETTERSI	• Chiave = «w», valore = Numero intero di tipo 0, tipo di messaggio (14)

Caratteristiche di trasferimento di carichi utili di grandi dimensioni

TXLargeMessage

TXLargeMessage viene utilizzato dal dispositivo per inviare un payload di grandi dimensioni superiore alla dimensione MTU negoziata per la connessione BLE.

- Il dispositivo invia i primi byte MTU del payload come notifica tramite la caratteristica.
- Il proxy invia una richiesta di lettura su questa caratteristica per i byte rimanenti.
- Il dispositivo invia fino alla dimensione MTU o ai byte rimanenti del payload, a seconda di quale dei due valori sia inferiore. Ogni volta, aumenta l'offset letto in base alla dimensione del payload inviato.
- Il proxy continuerà a leggere la caratteristica fino a ottenere un payload di lunghezza zero o un carico utile inferiore alla dimensione MTU.
- Se il dispositivo non riceve una richiesta di lettura entro un timeout specificato, il trasferimento fallisce e il proxy e il gateway rilasciano il buffer.
- Se il proxy non riceve una risposta di lettura entro un timeout specificato, il trasferimento fallisce e il proxy rilascia il buffer.

RXLargeMessage

RXLargeMessage viene utilizzato dal dispositivo per ricevere un carico utile di grandi dimensioni superiore alla dimensione MTU negoziata per la connessione BLE.

- Il proxy scrive i messaggi, fino alla dimensione MTU, uno per uno, utilizzando la funzione write with response in base a questa caratteristica.
- Il dispositivo memorizza il messaggio nel buffer finché non riceve una richiesta di scrittura di lunghezza zero o inferiore alla dimensione MTU.
- Se il dispositivo non riceve una richiesta di scrittura entro un timeout specificato, il trasferimento fallisce e il dispositivo rilascia il buffer.
- Se il proxy non riceve una risposta di scrittura entro un timeout specificato, il trasferimento fallisce e il proxy rilascia il buffer.

Libreria di interfaccia cellulare

> **Note**
>
> Il contenuto di questa pagina potrebbe non esserlo up-to-date. Consulta la pagina della libreria di FreeRTOS.org per l'ultimo aggiornamento.

Introduzione

La libreria Cellular Interface implementa una semplice API unificata che nasconde la complessità dei comandi AT specifici del modem cellulare ed espone un'interfaccia simile a un socket ai programmatori C.

La maggior parte dei modem cellulari implementa più o meno i comandi AT definiti dallo standard 3GPP TS v27.007. Questo progetto fornisce un'implementazione di tali comandi AT standard in un componente comune riutilizzabile. Le tre librerie di interfaccia cellulare di questo progetto sfruttano tutte quel codice comune. La libreria per ogni modem implementa solo i comandi AT specifici del fornitore, quindi espone l'API completa della libreria Cellular Interface.

Il componente comune che implementa lo standard 3GPP TS v27.007 è stato scritto in conformità con i seguenti criteri di qualità del codice:

- I punteggi GNU Complexity non sono superiori a 8
- MISRA C: standard di codifica 2012. Eventuali deviazioni dallo standard sono documentate nei commenti al codice sorgente contrassegnati da «copertura».

Dipendenze e requisiti

Non esiste una dipendenza diretta per la libreria di interfaccia cellulare. Tuttavia, Ethernet, Wi-Fi e cellulare non possono coesistere nello stack di rete FreeRTOS. Gli sviluppatori devono scegliere una delle interfacce di rete da integrare con la libreria Secure Sockets.

Portabilità

Per informazioni sulla portabilità della libreria Cellular Interface sulla tua piattaforma, vedi Porting the Cellular Interface Library nella FreeRTOS Porting Guide.

Utilizzo della memoria

Dimensione del codice della libreria di interfaccia cellulare (esempio generato con GCC per ARM Cortex-M)		
File	Con ottimizzazione -O1	Con ottimizzazione -Os
cellulare_3gpp_api.c	6,3 K	5,7 K
cellulare_3gpp_urc_handler.c	0,9 K	0,8 K
cellulare_at_core.c	1,4 K	1,2 K
cellulare_common_api.c	0,5 K	0,5 K
cellulare_common.c	1,6 K	1,4 K
cellular_pkthandler.c	1,4 K	1,2 K
cellulare_pktio.c	1,8 K	1,6 K
Stime totali	13,9K	12,4K

Nozioni di base

Scarica il codice di origine

Il codice sorgente può essere scaricato come parte delle librerie FreeRTOS o da solo.

Per clonare la libreria da Github utilizzando HTTPS:

```
git clone https://github.com/FreeRTOS/FreeRTOS-Cellular-Interface.git
```

Utilizzo di SSH:

```
git clone git@github.com:FreeRTOS/FreeRTOS-Cellular-Interface.git
```

Struttura delle cartelle

Alla radice di questo repository vedrai queste cartelle:

- `source`: codice comune riutilizzabile che implementa i comandi AT standard definiti da 3GPP TS v27.007
- `doc`: documentazione
- `test`: test unitario e cbmc
- `tools`: strumenti per l'analisi statica di Coverity e cMock

Configura e crea la libreria

La libreria di interfaccia cellulare deve disporre dell'interfaccia di rete cellulare come parte di un'applicazione. Per farlo, devi disporre di determinate configurazioni. Il progetto FreeRTOS_Cellular_Interface_Windows_Simulator fornisce un esempio di come configurare la build. Ulteriori informazioni sono disponibili nei riferimenti delle API di rete cellulare.

Per ulteriori informazioni, consulta la pagina dell'interfaccia cellulare.

Integra la libreria di interfaccia cellulare con le piattaforme MCU

La libreria Cellular Interface funziona su MCU utilizzando un'interfaccia astratta, la Comm Interface, per comunicare con i modem cellulari. Un'interfaccia di comunicazione deve essere implementata anche sulla piattaforma MCU. Le implementazioni più comuni di Comm Interface comunicano tramite hardware UART, ma possono essere implementate anche su altre interfacce fisiche, come SPI. La documentazione per l'interfaccia Comm è disponibile nei riferimenti API della libreria cellulare. Sono disponibili le seguenti implementazioni di esempio dell'interfaccia Comm:

- Interfaccia di comunicazione del simulatore Windows FreeRTOS
- FreeRTOS Interfaccia di comunicazione IO UART comune
- Interfaccia di comunicazione per scheda di rilevamento STM32 L475
- Interfaccia di comunicazione della scheda Sierra Sensor Hub

I/O comuni

> **Important**
>
> Questa libreria è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Consigliamo di iniziare da qui quando si crea un nuovo progetto. Se disponi già di un progetto FreeRTOS

esistente basato sull'ormai obsoleto repository Amazon-FreeRTOS, consulta ilGuida alla migrazione del repository Github di Amazon-FreeRTOS.

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 utilizzare le stesse API per accedere a vari driver di dispositivo attraverso schede di riferimento basate su più microcontroller (MCU).

L'I/O comune di FreeRTOS funge da questo livello di astrazione hardware. Fornisce un set di API standard per l'accesso a dispositivi seriali comuni su schede di riferimento supportate. Queste API comuni comunicano e interagiscono con queste periferiche e consentono al codice di funzionare tra le 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 viene restituita 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 portabilità di.

Libreria AWS IoT Device Defender

> **Note**
>
> Il contenuto di questa pagina potrebbe non essere up-to-date. Si prega di fare riferimento alPagina della libreria Freertos.orgper l'ultimo aggiornamento.

Introduzione

Puoi utilizzare il pluginAWS IoT Device Defenderlibreria a cui inviare metriche di sicurezza dai tuoi dispositivi IoTAWS IoT Device Defender. Puoi usareAWS IoT Device Defenderper monitorare continuamente queste metriche di sicurezza dai dispositivi per rilevare eventuali deviazioni da quello che avete definito come comportamento appropriato per ciascun dispositivo. Se qualcosa non sembra giusto,AWS IoT Device Defenderinvia un avviso in modo da poter agire per risolvere il problema. Interazioni conAWS IoT Device DefenderusoMQTT, un protocollo di pubblicazione e sottoscrizione leggero. Questa libreria fornisce un'API per comporre e riconoscere le stringhe di argomenti MQTT utilizzate daAWS 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 alISO C90eMISRA C: 2012. La libreria non ha dipendenze da altre librerie oltre alla libreria C standard. Inoltre non ha alcuna dipendenza dalla piattaforma, come il threading o la sincronizzazione. Può essere utilizzato con qualsiasi libreria MQTT e qualsiasiJSONoCORNObiblioteca. La biblioteca haprovemostra un uso sicuro della memoria e nessuna allocazione di heap, rendendola adatta ai microcontrollori IoT, ma anche completamente portabile su altre piattaforme.

IlAWS IoT Device Defenderla libreria può essere utilizzata liberamente ed è distribuita sottoLicenza open source MIT.

Dimensione del codice diAWS IoTDevice 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

Libreria Discovery AWS IoT Greengrass

> **Note**
>
> Il contenuto di questa pagina potrebbe non esserlo up-to-date. Consulta la pagina della libreria FreeRTOS.org per l'ultimo aggiornamento.

Panoramica

La libreria AWS IoT GreengrassDiscovery viene utilizzata dai dispositivi microcontrollori per scoprire un core Greengrass sulla rete. Utilizzando le API Discovery AWS IoT Greengrass, il tuo dispositivo può inviare messaggi a un core Greengrass dopo avere individuato l'endpoint del core.

Dipendenze e requisiti

Per utilizzare la libreria Discovery Greengrass, è necessario creare un oggetto in AWS IoT, inclusi certificato e policy. 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`

L'endpoint AWS IoT.

`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, consultare l'articolo relativo alle nozioni di base su AWS IoT Greengrass.

Sebbene la libreria CoreMQTT non sia necessaria per la connettività di Greengrass, consigliamo vivamente di installarla. La libreria può essere utilizzata per comunicare con il core Greengrass dopo che è stato individuato.

Documentazione di riferimento dell'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 individuazione richiedendo da AWS IoT un file JSON che contiene i parametri di connettività del core 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, ad esempio per l'allocazione dinamica dei certificati, è necessario chiamare le seguenti API:

`GGD_JSONRequestStart`

Effettua una richiesta HTTP GET a AWS IoT per avviare la richiesta di individuazione per individuare un core Greengrass. Viene utilizzato `GD_SecureConnect_Send` 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

> **Note**
>
> Il contenuto di questa pagina potrebbe non essere up-to-date. Si prega di fare riferimento alPagina della libreria Freertos.orgper l'ultimo aggiornamento.

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 diHTTP/1.1di serie. Lo standard HTTP fornisce un protocollo stateless che funziona su TCP/IP e viene spesso utilizzato in sistemi di informazione ipertestuali distribuiti e collaborativi.

La libreria CoreHTTP implementa un sottoinsieme diHTTP/1.1protocollo standard. 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 aISO C90eMISRA C: 2012. Le uniche dipendenze della libreria sono la libreria C standard eversione LTS (v12.19.1) del parserda Node.js. La biblioteca haprovemostra un uso sicuro della memoria e nessuna allocazione di heap, rendendola 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 nelDemo dell'autenticazione reciproca CoreHTTP.

Questa libreria può essere utilizzata liberamente ed è distribuita sottoLicenza 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
api.c (lhttp)	2,6K	2,0K
http.c (llhttp)	0,3 K	0,3 K
llhttp.c (llhttp)	17.9	15,9
Stime totali	23,9 K	20,7 K

Libreria CoreJSON

> **Note**
>
> Il contenuto di questa pagina potrebbe non esserlo up-to-date. Consulta la pagina della libreria di FreeRTOS.org per l'ultimo aggiornamento.

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 servizioAWS IoT Device Shadow, e fa parte di molte API, 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 un uso sicuro della memoria e l'assenza di allocazione di heap, il che lo rende adatto per microcontrollori IoT, ma anche completamente portabile su altre piattaforme.

Uso delle memerie P

La libreria CoreJSON utilizza uno stack interno per tenere traccia delle strutture annidate in un documento JSON. Lo stack esiste per la durata di una singola chiamata di funzione; non è conservato. 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	2,9 K	2,4 K
Stime totali	2,9K	2,4 K

libreria CoreMQTT

> **Note**
>
> Il contenuto di questa pagina potrebbe non essere up-to-date. Si prega di fare riferimento alPagina della libreria Freertos.orgper l'ultimo aggiornamento.

Introduzione

La libreria CoreMQTT è un'implementazione client diMQTT(Message Queue Telemetry Transport) standard. Lo standard MQTT fornisce una modalità leggera di pubblicazione/sottoscrizione (oPubSub) protocollo di messaggistica 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 aMQT 3.1.1protocollo standard. 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 inCe progettato per essere conforme aISO C90eMISRA 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 espone anche un'API serializzatore/deserializzatore di basso livello. Questa API può essere utilizzata per creare una semplice applicazione IoT composta solo dal sottoinsieme richiesto di funzionalità MQTT, senza alcun altro sovraccarico. L'API serializzatore/deserializzatore può essere utilizzata insieme a qualsiasi API del 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 haproveche dimostrano un uso sicuro della memoria e l'assenza di allocazione dell'heap, il che la rende adatta ai microcontrollori IoT, ma anche completamente portabile su altre piattaforme. Può essere utilizzato liberamente ed è distribuito sottoLicenza 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	4,0K	3,4K
core_mqtt_state.c	1,7 K	1,3 K
core_mqtt_serializer.c	2,8K	2,2 K
Stime totali	8,5 K	6,9K

Libreria del componente CoreMQTT

> **Note**
>
> Il contenuto di questa pagina potrebbe non essere up-to-date. Si prega di fare riferimento alPagina della libreria Freertos.orgper l'ultimo aggiornamento.

Introduzione

La libreria CoreMQTT Agent è un'API di alto livello che aggiunge la sicurezza dei thread alibreria CoreMQTT. Consente di creare un'attività di agente MQTT dedicata che gestisce una connessione MQTT in background e non richiede l'intervento di altre attività. La libreria fornisce equivalenti thread-safe alle API di CoreMQTT, quindi può essere utilizzata in ambienti multithread.

L'agente MQTT è un'attività indipendente (o thread di esecuzione). Garantisce la sicurezza dei thread essendo l'unica attività autorizzata ad accedere all'API della libreria MQTT. Serializza l'accesso isolando tutte le chiamate API MQTT in una singola attività ed elimina la necessità di semafori o altre primitive di sincronizzazione.

La libreria utilizza una coda di messaggistica thread-safe (o altro meccanismo di comunicazione tra processi) per serializzare tutte le richieste di chiamata alle API MQTT. L'implementazione della messaggistica è disaccoppiata dalla libreria tramite un'interfaccia di messaggistica, che consente il trasferimento della libreria su altri sistemi operativi. L'interfaccia di messaggistica è composta da funzioni per inviare e ricevere puntatori alle strutture di comando dell'agente e da funzioni per allocare questi oggetti di comando, il che consente all'autore dell'applicazione di decidere la strategia di allocazione della memoria appropriata per la propria applicazione.

La libreria è scritta in C e progettata per essere conforme aISO C90eMISRA C: 2012. La libreria non ha dipendenze da altre librerie diverse dalibreria CoreMQTTe la libreria C standard. La biblioteca haproveche dimostrano un uso sicuro della memoria e l'assenza di allocazione dell'heap, quindi può essere utilizzata per i microcontrollori IoT, ma è anche completamente portabile su altre piattaforme.

Questa libreria può essere utilizzata liberamente ed è distribuita sottoLicenza open source MIT.

Dimensione del codice dell'agente CoreMQTT (esempio generato con GCC per ARM Cortex-M)		
File	Con ottimizzazione -O1	Con ottimizzazione -Os
core_mqtt_agent.c	1,7 K	1,5 K
core_mqtt_agent_command_functions.c	0,3 K	0,2 K
core_mqtt.c (CoreMQTT)	4,0K	3,4K
core_mqtt_state.c (CoreMQTT)	1,7 K	1,3 K
core_mqtt_serializer.c (CoreMQTT)	2,8K	2,2 K
Stime totali	10,5 K	8,6K

AWS IoTLibreria via etere (OTA)

> **Note**
>
> Il contenuto di questa pagina potrebbe non esserlo up-to-date. Consulta la pagina della libreria di FreeRTOS.org per l'ultimo aggiornamento.

Introduzione

La libreria di aggiornamentiAWS IoT Over-the-air (OTA) consente 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.

L'Internet of Things (IoT) estende la connettività Internet ai dispositivi integrati che tradizionalmente non erano connessi. Questi dispositivi possono essere programmati per comunicare dati utilizzabili su Internet e possono essere monitorati e controllati da remoto. Grazie ai progressi tecnologici, questi dispositivi integrati tradizionali stanno aumentando rapidamente le funzionalità Internet negli spazi consumer, industriali e aziendali.

I dispositivi IoT vengono in genere distribuiti in grandi quantità e spesso in luoghi difficili o poco pratici da accedere per un operatore umano. Immagina uno scenario in cui viene scoperta una vulnerabilità di sicurezza che può esporre i dati. In tali scenari, è importante aggiornare i dispositivi interessati con correzioni di sicurezza in modo rapido e affidabile. Senza la possibilità di eseguire aggiornamenti OTA, può anche essere difficile aggiornare i dispositivi dislocati geograficamente. Avere un tecnico che aggiorni questi dispositivi sarà costoso, dispendioso in termini di tempo e spesso poco pratico. Il tempo necessario per aggiornare questi dispositivi li espone a vulnerabilità di sicurezza per un periodo più lungo. Richiamare questi dispositivi per l'aggiornamento sarà inoltre costoso e potrebbe causare gravi interruzioni ai consumatori a causa dei tempi di inattività.

Gli aggiornamenti via etere (OTA) consentono di aggiornare il firmware del dispositivo senza un costoso richiamo o la visita di un tecnico. Questo metodo aggiunge i seguenti vantaggi:

- Sicurezza: capacità di rispondere rapidamente alle vulnerabilità di sicurezza e ai bug del software scoperti dopo l'implementazione dei dispositivi sul campo.
- Innovazione: i prodotti possono essere aggiornati frequentemente man mano che vengono sviluppate nuove funzionalità, guidando il ciclo di innovazione. Gli aggiornamenti possono avere effetto rapidamente con tempi di inattività minimi rispetto ai metodi di aggiornamento tradizionali.
- Costo: gli aggiornamenti OTA possono ridurre significativamente i costi di manutenzione rispetto ai metodi tradizionalmente utilizzati per aggiornare questi dispositivi.

La fornitura della funzionalità OTA richiede le seguenti considerazioni di progettazione:

- Comunicazione sicura: gli aggiornamenti devono utilizzare canali di comunicazione crittografati per evitare che i download vengano manomessi durante il transito.
- Ripristino: gli aggiornamenti possono fallire a causa di problemi quali la connettività di rete intermittente o la ricezione di un aggiornamento non valido. In questi scenari, il dispositivo deve essere in grado di tornare a uno stato stabile ed evitare che si blocchi.
- Verifica dell'autore: è necessario verificare che gli aggiornamenti provengano da una fonte attendibile, insieme ad altre convalide come la verifica della versione e della compatibilità.

Per ulteriori informazioni sull'impostazione di aggiornamenti OTA con FreeRTOS, consultaAggiornamenti via etere di FreeRTOS.

AWS IoTLibreria via etere (OTA)

La libreriaAWS IoT OTA consente di gestire le notifiche dei nuovi aggiornamenti disponibili, scaricarli ed eseguire la verifica crittografica degli aggiornamenti del firmware. Utilizzando la libreria client over-the-air (OTA), è possibile separare logicamente i meccanismi di aggiornamento del firmware dall'applicazione in esecuzione sul dispositivo. La libreria client over-the-air (OTA) può condividere una connessione di rete con l'applicazione, risparmiando memoria nei dispositivi con risorse limitate. Inoltre, la libreria client over-the-air (OTA) consente di definire una logica specifica dell'applicazione per testare, eseguire o ripristinare un aggiornamento del firmware. La libreria supporta diversi protocolli applicativi come Message Queuing Telemetry Transport (MQTT) e Hypertext Transfer Protocol (HTTP) e offre varie opzioni di configurazione che puoi ottimizzare in base al tipo e alle condizioni della rete.

Le API di questa libreria forniscono queste funzioni principali:

- Registrati per ricevere notifiche o sondaggi per le nuove richieste di aggiornamento disponibili.
- Ricevi, analizza e convalida la richiesta di aggiornamento.
- Scarica e verifica il file in base alle informazioni contenute nella richiesta di aggiornamento.
- Esegui un autotest prima di attivare l'aggiornamento ricevuto per assicurarti la validità funzionale dell'aggiornamento.
- Aggiorna lo stato del dispositivo.

Questa libreria utilizza iAWS servizi per gestire varie funzioni relative al cloud, come l'invio di aggiornamenti del firmware, il monitoraggio di un gran numero di dispositivi in più regioni, la riduzione del raggio d'azione delle implementazioni difettose e la verifica della sicurezza degli aggiornamenti. Questa libreria può essere utilizzata con qualsiasi libreria MQTT o HTTP.

Le demo di questa libreria mostrano over-the-air aggiornamenti completi utilizzando la libreria e iAWS servizi CoreMQTT su un dispositivo FreeRTOS.

Funzionalità

Di seguito è riportata l'interfaccia completa dell'agente OTA:

OTA_Init

Inizializza il motore OTA avviando OTA Agent («OTA Task») nel sistema. Può esistere un solo agente OTA.

OTA_Shutdown

Segnala all'agente OTA di spegnere. Facoltativamente, l'agente OTA annullerà l'iscrizione a tutti gli argomenti relativi alle notifiche di lavoro MQTT, interromperà eventuali lavori OTA in corso e cancellerà tutte le risorse.

OTA_GetState

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

Sospendi tutte le operazioni dell'agente OTA.

OTA_Resume

Riprendi le operazioni dell'agente OTA.

OTA_SignalEvent

Segnala un evento all'attività dell'agente OTA.

`OTA_EventProcessingTask`

Ciclo di elaborazione degli eventi dell'agente OTA.

`OTA_GetStatistics`

Ottieni le statistiche dei pacchetti di messaggi OTA che includono il numero di pacchetti ricevuti, messi in coda, elaborati e scartati.

`OTA_Err_strerror`

Conversione da codice di errore a stringa per errori OTA.

`OTA_JobParse_strerror`

Converte un codice di errore OTA Job Parsing in una stringa.

`OTA_PalStatus_strerror`

Conversione da codice di stato a stringa per lo stato OTA PAL.

`OTA_OsStatus_strerror`

Conversione da codice di stato a stringa per lo stato del sistema operativo OTA.

Documentazione di riferimento dell'API

Per ulteriori informazioni, vedere AWS IoTOver-the-air Update: Functions.

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 all'agenteAWS IoT CoreMQTT. Per ulteriori informazioni, consulta Libreria del componente CoreMQTT.
2. Inizializza l'agente OTA chiamando`OTA_Init`, inclusi i buffer, le interfacce ota richieste, il nome dell'oggetto e il callback dell'applicazione. Il callback implementa la logica specifica dell'applicazione che viene eseguita dopo il completamento di un processo di aggiornamento OTA.
3. Quando l'aggiornamento OTA è completo, FreeRTOS chiama il callback di completamento del lavoro con uno dei seguenti eventi:`accepted`,`rejected`, `oself test`.

4. 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.
5. 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.

Utilizzo della memoria

Dimensione del codiceAWS IoT OTA (esempio generato con GCC per ARM Cortex-M)		
File	Con ottimizzazione -O1	Con ottimizzazione -Os
ota.c	8,3 K	7,5 K
ota_interface.c	0,1 K	0,1 K
ota_base64.c	0,6 K	0,6 K
ota_mqtt.c	2,4 K	2,2 K
ota_cbor.c	0,8 K	0,6 K
ota_http.c	0,3 K	0,3 K
Stime totali	12,5 K	11,3 K

Libreria del lettore ivS

> **Note**
>
> Il contenuto di questa pagina potrebbe non esserloup-to-date. Si prega di fare riferimento alla pagina della libreria FreeRTOS.org per l'ultimo aggiornamento.

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 utilizzato 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 AWS riferimento di 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» mai la chiave.

La libreria CorePKCS11 contiene un'implementazione fittizia basata su software dell'interfaccia PKCS #11 (API) che utilizza la funzionalità crittografica fornita da Mbed TLS. L'utilizzo di un software simulato consente uno sviluppo rapido e una maggiore flessibilità, ma è prevedibile che sostituiate la simulazione 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 libreria fittizia basata esclusivamente sul software CorePKCS11 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 il criptoprocessore nei dispositivi di produzione.

È 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 incorporati. Vedi 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 inoltre effettuate dal nostro workflow dedicato agli sviluppatori che fornisce una tantum 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 una descrizione dettagliata delle funzioni, consulta 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 istruzioni di seguito descrivono come creare un oggetto AWS IoT in base a un certificato client P-256.

Assicurati di utilizzare le seguenti versioni (o più recenti) 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 Interfaceutente.

Per creare AWS IoT qualcosa basato su un certificato client P-256

1. Creare un oggetto AWS IoT

   ```
   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. Inviare la richiesta di registrazione dei certificati 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 desideri eseguire la demo di FreeRTOS per la connettività e la scoperta 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 trasferimento della libreria CorePKCS11 sulla tua piattaforma, vedi Porting the CorePKCS11 Library nella FreeRTOS Porting Guide.

Uso delle librerie

Dimensione del codice di CorePKCS11 (esempio generato con GCC per ARM Cortex-M)		
File	Con ottimizzazione -O1	Con -Os Optimization
core_pkcs11.c	0,8 K	0,8 K
core_pki_utils.c	0,5 K	0,3 K
core_pkcs11_mbedtls.c	8.9K	7,5 K
Stime totali	10,2 K	8,6 K

Libreria Secure Sockets

> **Important**
>
> Questa libreria è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Ti consigliamo di iniziare da qui quando crei un nuovo progetto. Se hai già un progetto FreeRTOS esistente basato sull'ormai obsoleto repository Amazon-FreerTOS, consulta il. Guida alla migrazione del repository Github di Amazon-FreeRTOS

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 le API client, più un'implementazione IP leggera (LWiP) dell'API `Bind` lato server, sono supportate per FreerTOS Secure Sockets.

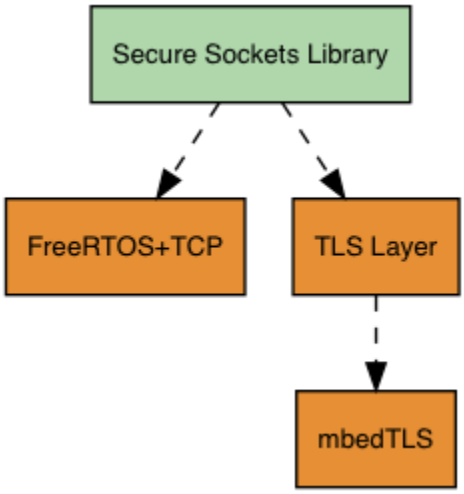
Dipendenze e requisiti

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
- API thread-safe per l'invio e la ricezione di dati
- E asy-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.

Le API del server non sono supportate dalla libreria FreerTOS Secure Sockets, ad eccezione di un'implementazione IP leggera (LWiP) dell'API lato server. `Bind` Le API client sono supportate.

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 un riferimento completo alle API, consulta Secure Sockets API Reference.

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 Libreria del lettore ivS
- Il Transport Layer Security

Per ulteriori informazioni sul porting, consulta Porting the Secure Sockets Library nella FreerTOS Porting Guide.

Libreria AWS IoT Device Shadow

> **Note**
>
> Il contenuto di questa pagina potrebbe non esserlo up-to-date. Consulta la pagina della libreria di FreeRTOS.org per l'ultimo aggiornamento.

Introduzione

È possibile utilizzare la libreriaAWS 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 servizioAWS 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'oggetto corrispondente, una rappresentazione di un dispositivo o di un'entità logica specifica sulAWS Cloud. Per ulteriori informazioni, consultare Managing dei dispositiviAWS IoT. Maggiori dettagli sulle ombre sono disponibili nella AWS IoTdocumentazione.

La libreriaAWS IoT Device Shadow non dipende 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 con la licenza open source MIT.

Dimensione del codice delAWS IoT Device Shadow (esempio generato con GCC per ARM Cortex-M)		
File	Con ottimizzazione -O1	Con ottimizzazione -Os
ombra c	1,2 K	0,9 K
Stime totali	1,2 K	0,9 K

AWS IoTLibreria di lavori

> **Note**
>
> Il contenuto di questa pagina potrebbe non essere up-to-date. Si prega di fare riferimento alPagina della libreria Freertos.orgper l'ultimo aggiornamento.

Introduzione

AWS IoTJobs è un servizio che notifica a uno o più dispositivi connessi una richiesta in sospesooccupazione. 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 della diagnostica. Per ulteriori informazioni, consulta la paginaOfferte di lavoronelAWS IoTGuida per gli sviluppatori. Interagire conAWS IoTUtilizzo del servizio JobsMQTT, un protocollo di pubblicazione e sottoscrizione leggero. Questa libreria fornisce un'API per comporre e riconoscere le stringhe di argomenti MQTT utilizzate daAWS IoTServizio Jobs.

IlAWS IoTLa libreria Jobs è scritta in C e progettata per essere conforme alISO C90eMISRA C: 2012. La libreria non ha dipendenze da altre librerie oltre alla libreria C standard. Può essere utilizzata con qualsiasi libreria MQTT e qualsiasi libreria JSON. La biblioteca haprovemostra un uso sicuro della memoria e nessuna allocazione di heap, rendendola adatta ai microcontrollori IoT, ma anche completamente portabile su altre piattaforme.

Questa libreria può essere utilizzata liberamente ed è distribuita sottoLicenza open source MIT.

Dimensione del codice diAWS IoTJob (esempio generato con GCC per ARM Cortex-M)		
File	Con ottimizzazione -O1	Con ottimizzazione -Os
jobs.c	1,9K	1,6 K
Stime totali	1,9 K	1,6 K

Transport Layer Security

> **Important**
>
> Questa libreria è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Consigliamo di iniziare da qui quando crei un nuovo progetto. Se disponi già di un progetto FreeRTOS esistente basato sull'ormai obsoleto repository Amazon-FreeRTOS, consulta ilGuida alla migrazione del repository Github di Amazon-FreeRTOS.

L'interfaccia FreeRTOS Transport Layer Security (TLS) è un sottile wrapper opzionale utilizzato per astrarre i dettagli dell'implementazione crittografica dall'interfaccia Secure Sockets Layer (SSL) sopra di essa 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 sulla portabilità dell'interfaccia TLS, consulta Porting the TLS Library nella FreeRTOS Porting Guide.

Libreria Wi-Fi

> **Important**
>
> Questa libreria è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Ti consigliamo di iniziare da qui quando crei un nuovo progetto. Se disponi già di un progetto FreeRTOS esistente basato sull'ormai obsoleto repository Amazon-FreerTOS, consulta il. Guida alla migrazione del repository Github di Amazon-FreeRTOS

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 l'autenticazione WEP, WPA, WPA2 e 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 i tipi di sicurezza WEP, WPA, WPA2 e 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 di punti di accesso nelle vicinanze

```
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 FreeRS

FreeRTOS include alcune applicazioni demo nella `demos` cartella, nella directory principale di FreeRTOS. Tutti gli esempi che possono essere eseguiti da FreeRTOS vengono visualizzati nella `common` cartella, sotto `demos`. C'è anche una cartella per ogni piattaforma qualificata per FreeRTOS sotto la `demos` cartella.

Prima di provare le applicazioni demo, si consiglia di completare il tutorial in Nosu FreeRTOS. Viene mostrato come configurare ed eseguire la demo di CoreMQTT.

Esecuzione delle demo di FreeRTOS

Negli argomenti seguenti viene illustrato come configurare ed eseguire le demo di FreeRS:

- Applicazioni demo Bluetooth Low Energy
- Bootloader demo per Microchip Curiosity PIC32MZEF
- Demo AWS IoT Device Defender
- AWS IoT GreengrassApplicazione dimostrativa V1 discovery
- AWS IoT Greengrass V2

- Demo CoreHTTP
- AWS IoTDemo della libreria Jobs
- Demo CoreMQTT
- Over-the-air aggiorna l'applicazione demo
- Demo Client Echo Secure Sockets
- Applicazione demo AWS IoT 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 CoreMQTT Agent. A seconda della configurazione selezionata quando hai scaricato FreeRTOS e da dove hai scaricato FreeRTOS, le altre funzioni di esempio del runner 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 limitazioni 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 demo è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Consigliamo di iniziare da qui quando si crea un nuovo progetto. Se disponi già di un progetto FreeRTOS

esistente basato sull'ormai obsoleto repository Amazon-FreeRTOS, consulta ilGuida alla migrazione del repository Github di Amazon-FreeRTOS.

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 le API middleware FreeRTOS Bluetooth Low Energy per creare un semplice server GATT.

> **Note**
>
> Per configurare ed eseguire le demo di FreeRTOS, segui i passaggi indicatiNosu 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 SDK Android per dispositivi Bluetooth FreeRTOS.

ConfigurazioneAWS IoT e Amazon Cognito per FreeRTOS Bluetooth Low Energy

Per connettere i tuoiAWS IoT dispositivi a MQTT, devi configurareAWS IoT Amazon Cognito.

Per configurare AWS IoT

1. Crea unAWS 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 gli SDK mobili 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. Annotare l'endpoint dell'API REST di AWS IoT.

Per ulteriori informazioni sulla configurazione, consulta la pagina relativa alle nozioni di base di AWS IoT.

Come creare un bacino d'utenza 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.

Come 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 le 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.

Come creare e allegare una policy IAM all'identità autenticata

1. Nel pannello di navigazione della console IAM seleziona Roles.
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 mano le tue informazioniAWS IoT e quelle di Amazon Cognito. È necessario che l'endpoint e l'ID effettuino l'autenticazione dell'applicazione per dispositivi mobili con il cloud AWS.

Configura il tuo ambiente FreeRTOS per Bluetooth Low Energy

Per configurare il tuo ambiente, devi scaricare FreeRTOS conLibreria 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 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 qualificato per FreeRTOS, consulta la guida relativa alla tua scheda in Guida introduttiva a FreeRTOS.

   > **Note**
   >
   > Puoi eseguire le demo su qualsiasi microcontrollore abilitato a Bluetooth Low Energy con FreeRTOS e librerie FreeRTOS Bluetooth Low Energy trasferite. Attualmente, il progettoMQTT su Bluetooth Low Energy demo di FreeRTOS è completamente portabile sui seguenti dispositivi Bluetooth Low Energy:
   >
   > - EspressifDevKit ESP32-C e ESP-WROVER-KIT
   > - Nordic nRF52840-DK

Componenti comuni

Le applicazioni demo di 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 è abilitata per Wi-Fi, il 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, aggiunge il tipo di connessione di rete al parametro `configENABLED_NETWORKS` in `vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/aws_iot_network_config.h` (dove *`vendor`* è il nome del fornitore e *`board`* è il nome della scheda che si sta 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 nell'SDK Android per i dispositivi Bluetooth FreeRTOS`amazon-freertos-ble-android-sdk/app` e nell'SDK iOS per i dispositivi Bluetooth FreeRTOS sotto`amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo`. GitHub 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 policy AWS IoT nella tua regione (se non l'hai già fatto).

> **Note**
>
> Questa politica è diversa dalla politica IAM creata per l'identità autenticata di Amazon Cognito.

   a. Aprire la console AWS IoT.
   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 tuaAWS 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. Seleziona Create (Crea).

5. Apri `amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo/AmazonFreeRTOSDemo/Amazon/AmazonConstants.swift` e ridefinisci le seguenti variabili:

   - `region`: la regione AWS.
   - `iotPolicyName`: il nome della policy AWS IoT.
   - `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`: ID bacino d'utenza Amazon Cognito.
   - `Region`: la regione AWS.

   In `CognitoUserPool` ridefinisci le seguenti variabili:

   - `PoolId`: ID bacino d'utenza Amazon Cognito.
   - `AppClientId`: l'ID del client dell'app.
   - `AppClientSecret`: il segreto del client dell'app.
   - `Region`: la regione AWS.

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 SDK Android per dispositivi Bluetooth FreeRTOS.
2. Crea una policy AWS IoT nella tua regione (se non l'hai già fatto).

   > **Note**
   >
   > Questa politica è diversa dalla politica IAM creata per l'identità autenticata di Amazon Cognito.

   a. Aprire la console AWS IoT.
   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 tuaAWS 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. Seleziona 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`: il nome della policy AWS IoT.
   - `AWS_IOT_REGION`: la regione AWS.

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`: ID bacino d'utenza Amazon Cognito.
   - `Region`: la regione AWS.

   In `CognitoUserPool` ridefinisci le seguenti variabili:

   - `PoolId`: ID bacino d'utenza Amazon Cognito.
   - `AppClientId`: l'ID del client dell'app.
   - `AppClientSecret`: il segreto del client dell'app.
   - `Region`: la regione AWS.

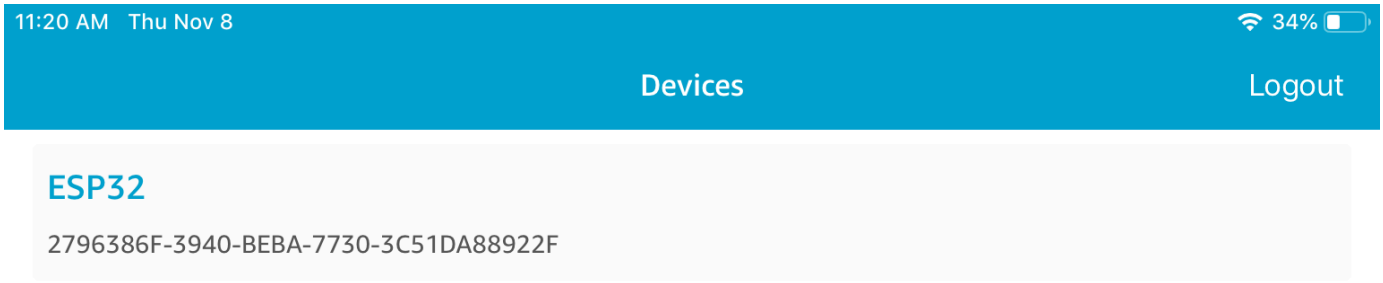
Per individuare e stabilire connessioni sicure con il microcontroller su Bluetooth Low Energy

1. Per accoppiare in modo sicuro il microcontrollore e il dispositivo mobile (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:

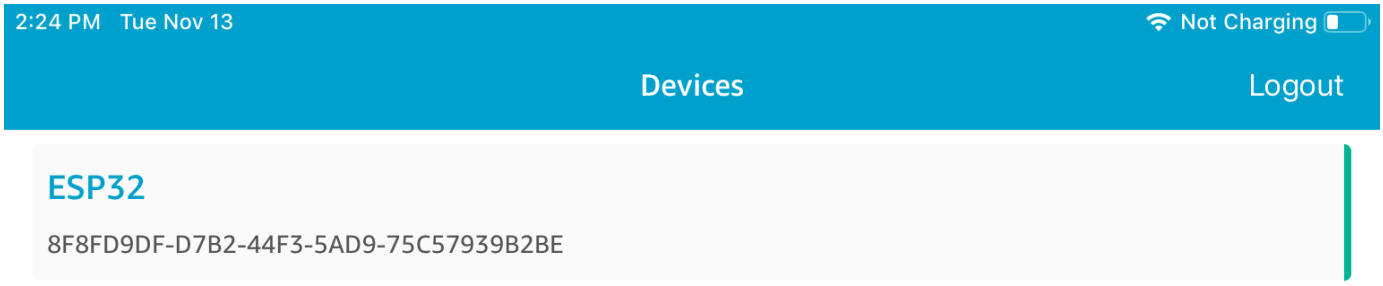
```
$ ./gradlew installDebug
```

4. Verificare che il microcontroller sia visualizzato in Devices (Dispositivi) nell'applicazione demo SDK Bluetooth Low Energy Mobile.

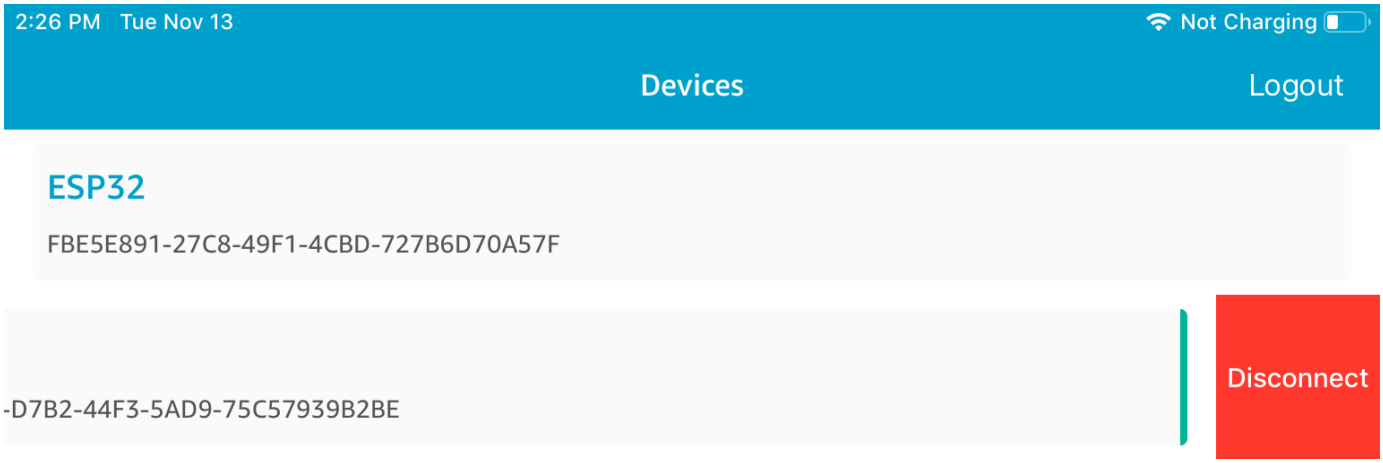


> **Note**
>
> Nell'elenco vengono visualizzati tutti i dispositivi con FreeRTOS e il servizio di informazione sui dispositivi (*freertos*`/.../device_information`) che si trovano nel raggio di copertura.

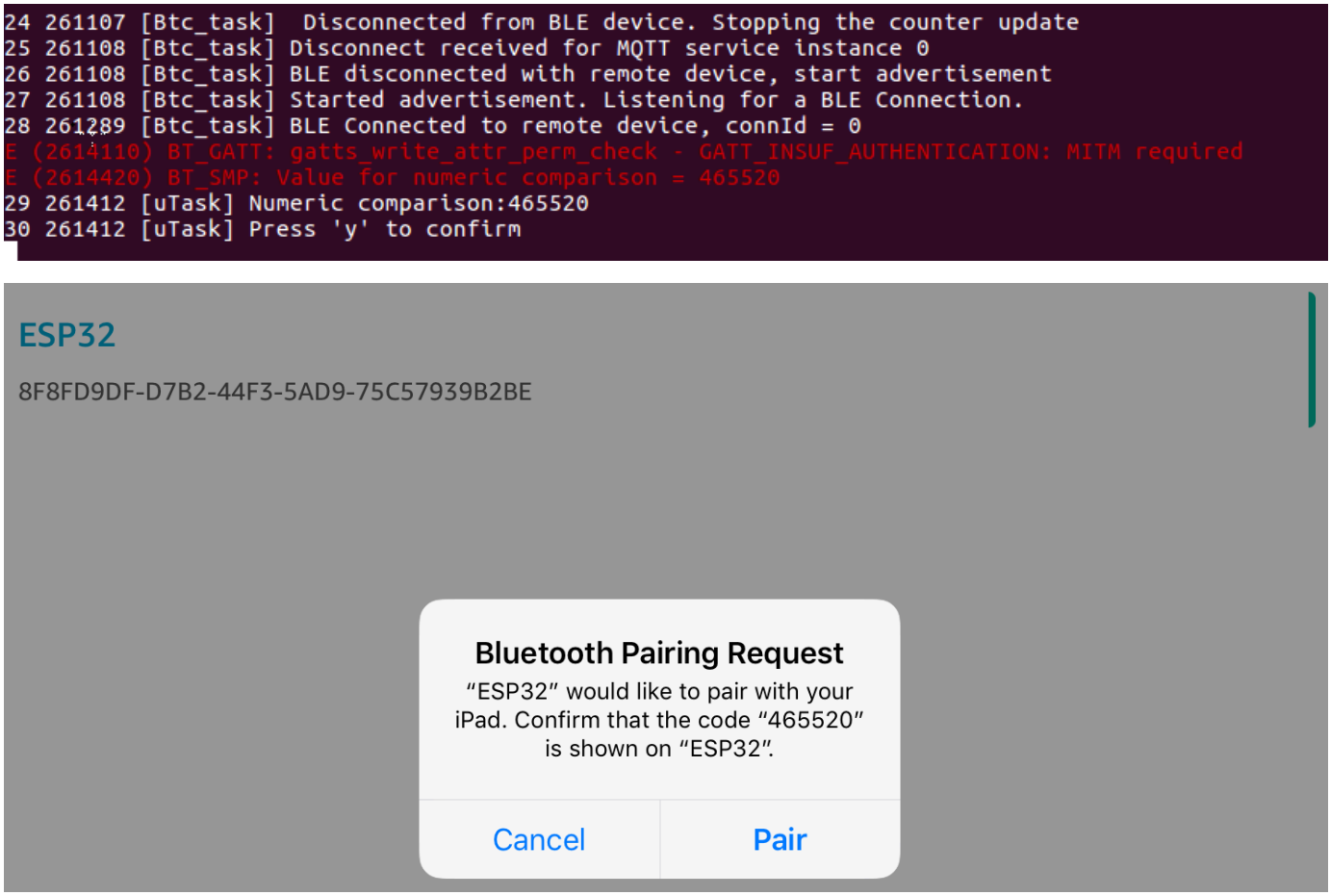
5. Scegliere il microcontroller dall'elenco di dispositivi. L'applicazione stabilisce una connessione con la scheda e una riga verde viene visualizzata accanto al dispositivo connesso.



È possibile disconnettersi dal microcontrollore trascinando la linea 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 Amazon Cognito e di aver allegato le politiche IAM alle identità autenticate.

MQTT su Bluetooth Low Energy

Nella demo MQTT over Bluetooth Low Energy, il microcontrollore pubblica messaggi nelAWS cloud tramite un proxy MQTT.

Per effettuare la sottoscrizione a un argomento MQTT demo

1. Accedere alla console AWS IoT.
2. Nel riquadro di navigazione, scegli Test, quindi scegli MQTT test client 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.

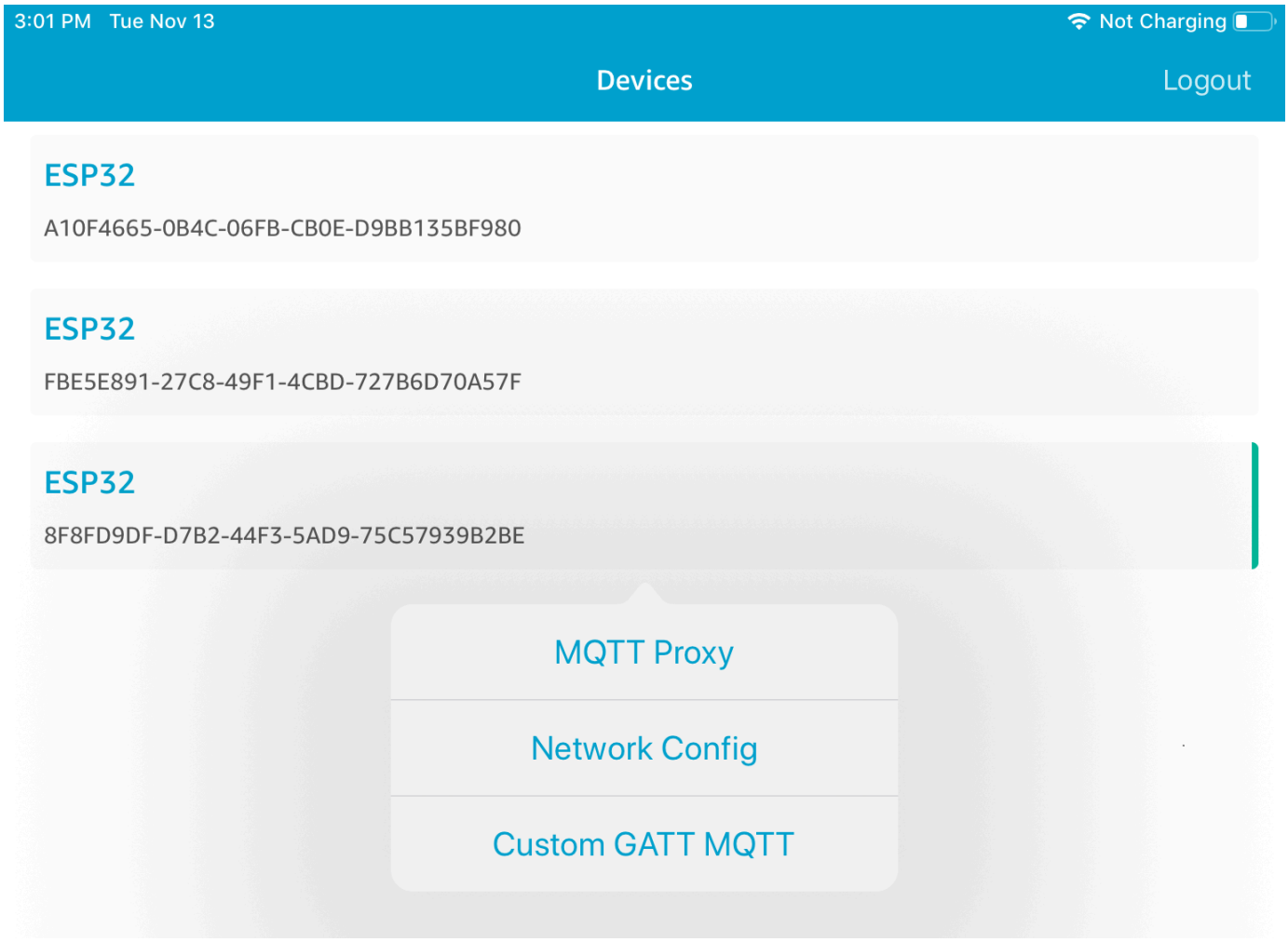
Per abilitare la demo tramite Bluetooth Low Energy

1. Aprire `vendors/vendor/boards/board/aws_demos/config_files/aws_demo_config.h` e definire `CONFIG_MQTT_BLE_TRANSPORT_DEMO_ENABLED`.
2. Apri`demos/include/aws_clientcredential.h` e configura`clientcredentialMQTT_BROKER_ENDPOINT` con l'endpoint delAWS IoT broker. Configura`clientcredentialIOT_THING_NAME` con il nome dell'oggetto per il dispositivo microcontrollore BLE. L'endpoint delAWS IoT broker può essere ottenuto dallaAWS IoT console scegliendo Impostazioni nel riquadro di navigazione a sinistra o tramite la CLI eseguendo il comando:`aws iot describe-endpoint --endpoint-type=iot:Data-ATS`.

> **Note**
>
> L'endpoint delAWS IoT broker e il nome dell'oggetto devono trovarsi entrambi nella stessa area in cui sono configurati l'identità cognita e il pool di utenti.

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 EspressifDevKit ESP32-C

Per abilitare la demo

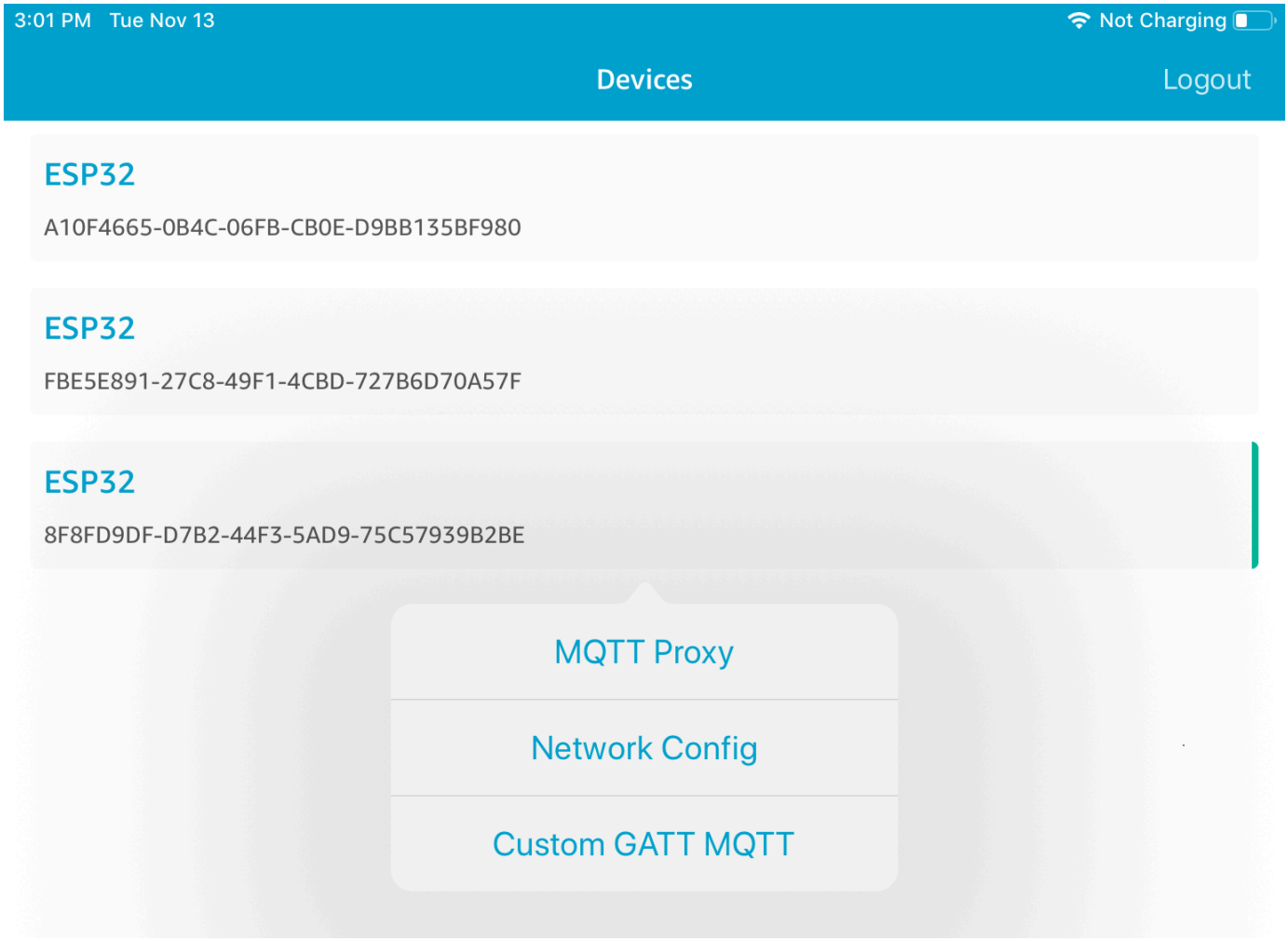
1. Abilitare il servizio di provisioning Wi-Fi. Aprire vendors/*vendor*/boards/*board*/aws_demos/config_files/iot_ble_config.h e impostare #define IOT_BLE_ENABLE_WIFI_PROVISIONING su 1 (dove *vendor* è il nome del fornitore e *board* è il nome della scheda che si sta utilizzando per eseguire le demo).

   > **Note**
   >
   > Per impostazione predefinita, il servizio di provisioning Wi-Fi è disabilitato.

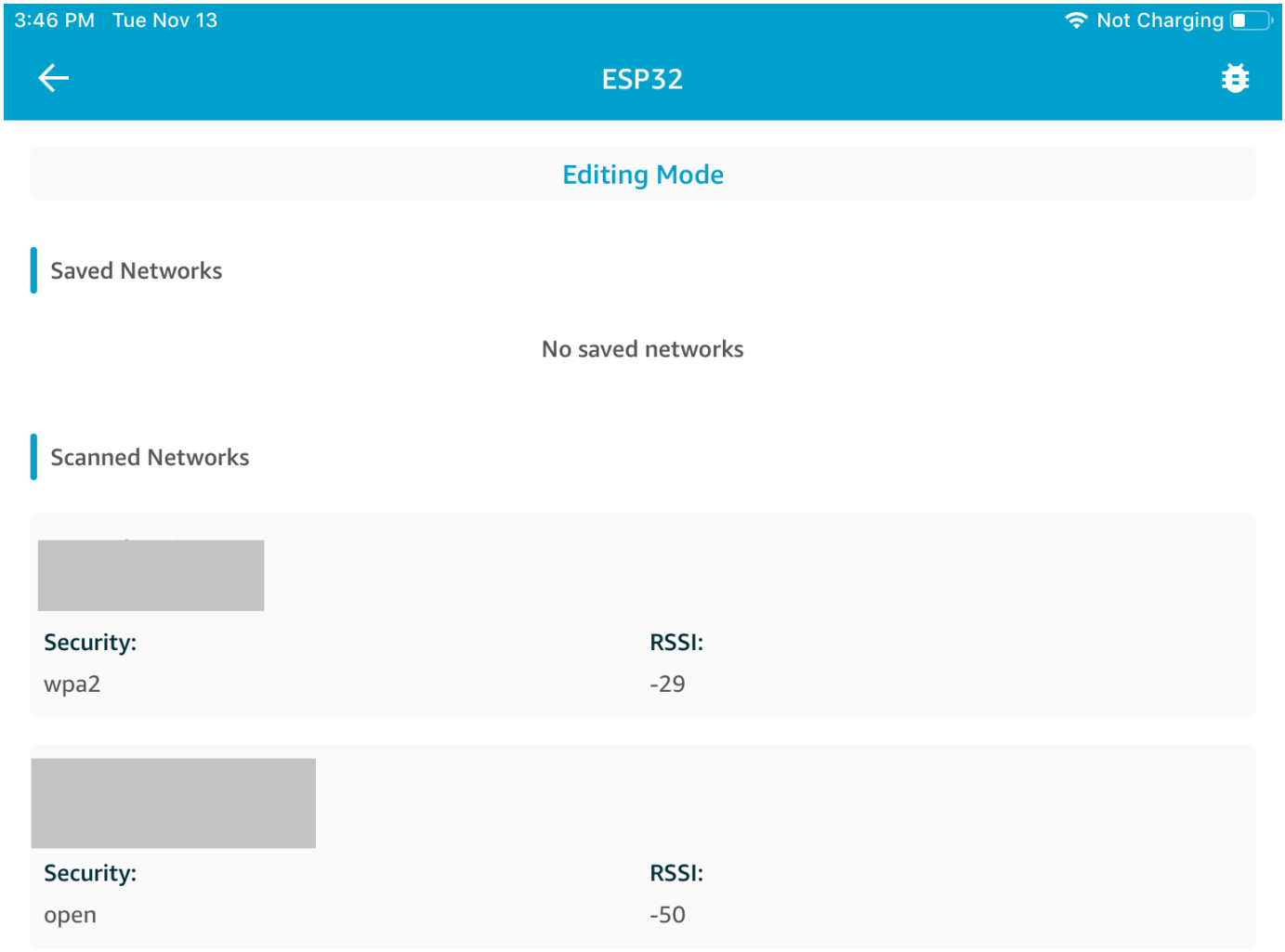
2. Configurare Network Manager per abilitare Bluetooth Low Energy e Wi-Fi.

Per eseguire la demo

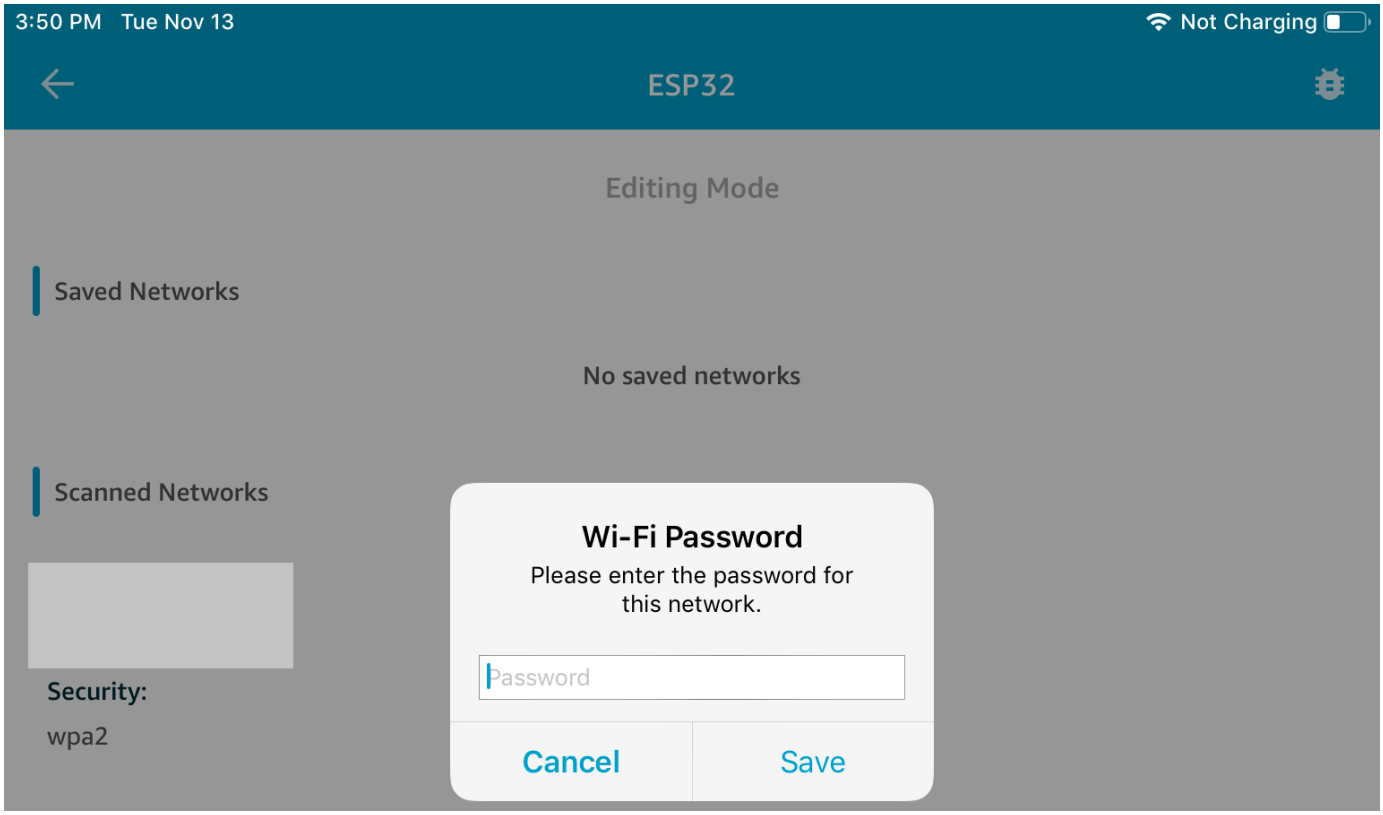
1. Compilare ed eseguire il progetto demo sul microcontroller.
2. Assicurati di aver associato il microcontrollore e il dispositivo mobile utilizzando ilApplicazione dimostrativa FreeRTOS Bluetooth Low Energy Mobile SDK.
3. Dall'elenco Devices (Dispositivi) nell'app demo per dispositivi mobili, scegliere il microcontroller, quindi selezionare Network Config (Configurazione rete) per aprire le impostazioni di configurazione della rete.



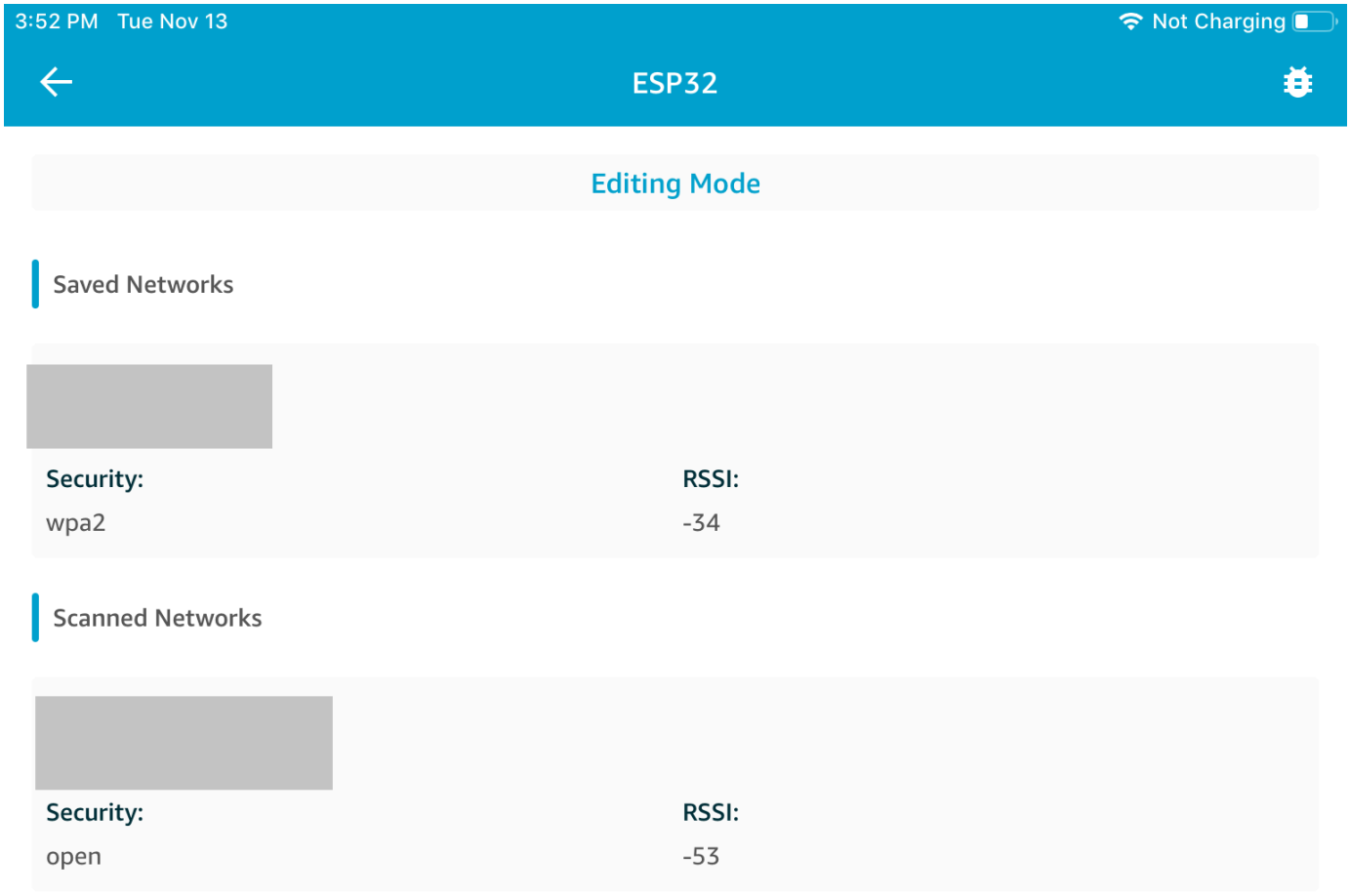
4. Dopo aver scelto Network Config (Configurazione rete) per la scheda, il microcontroller invia un elenco delle reti che si trovano vicino al dispositivo mobile. Le reti Wi-Fi disponibili vengono visualizzate in un elenco in Scanned Networks (Reti scansite).



Dall'elenco Scanned Networks (Reti scansite), scegliere la rete, quindi immettere l'SSID e la password, se richiesti.

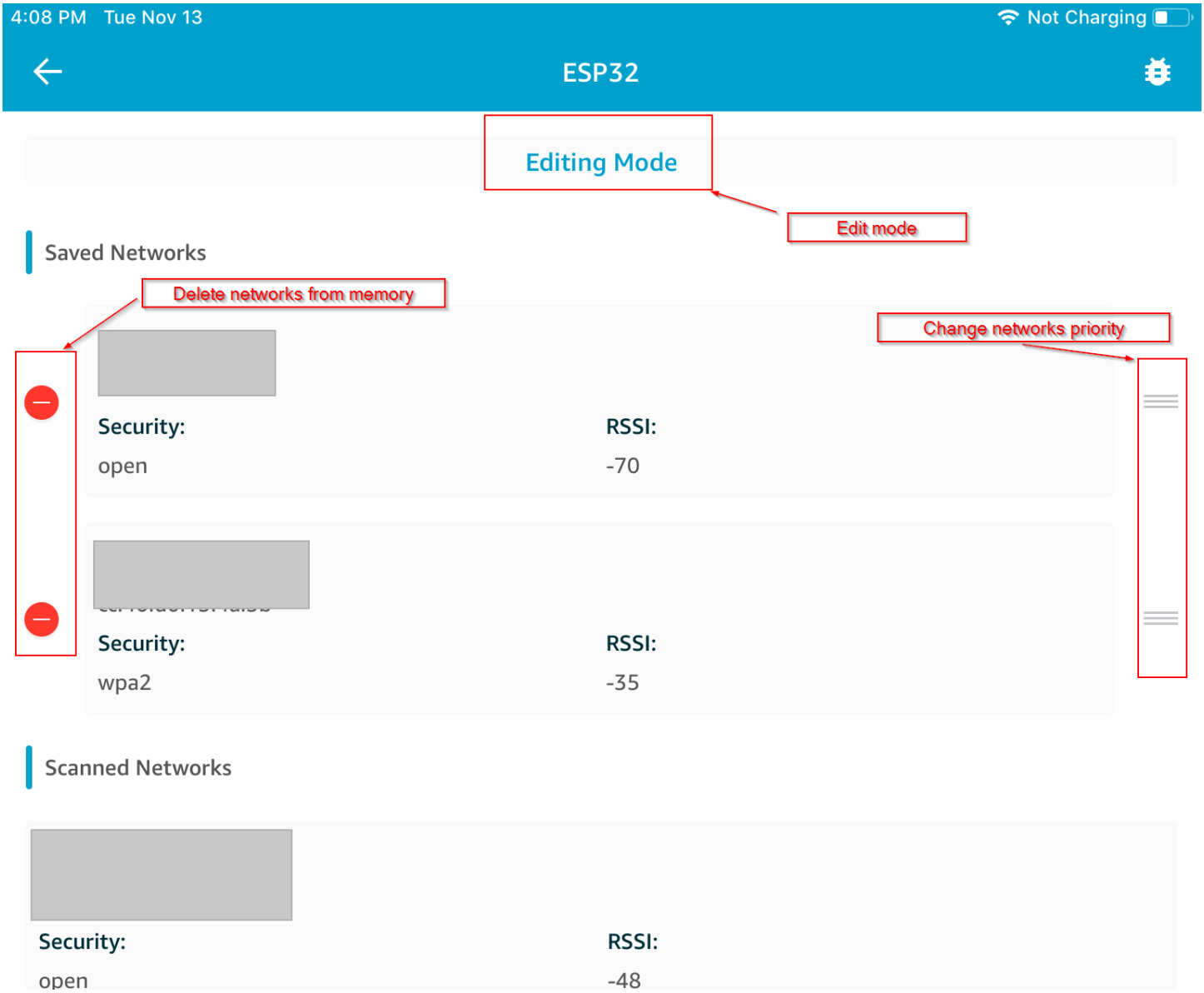


Il microcontrollore si connette alla rete e la salva. La rete viene visualizzata in Saved Networks (Reti salvate).



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 gli SDK Bluetooth Low Energy Mobile, puoi creare il tuo client GATT per un dispositivo mobile che viene connesso al server GATT sul microcontroller ed eseguito in parallelo con l'applicazione demo per dispositivi mobili.

Per abilitare la demo

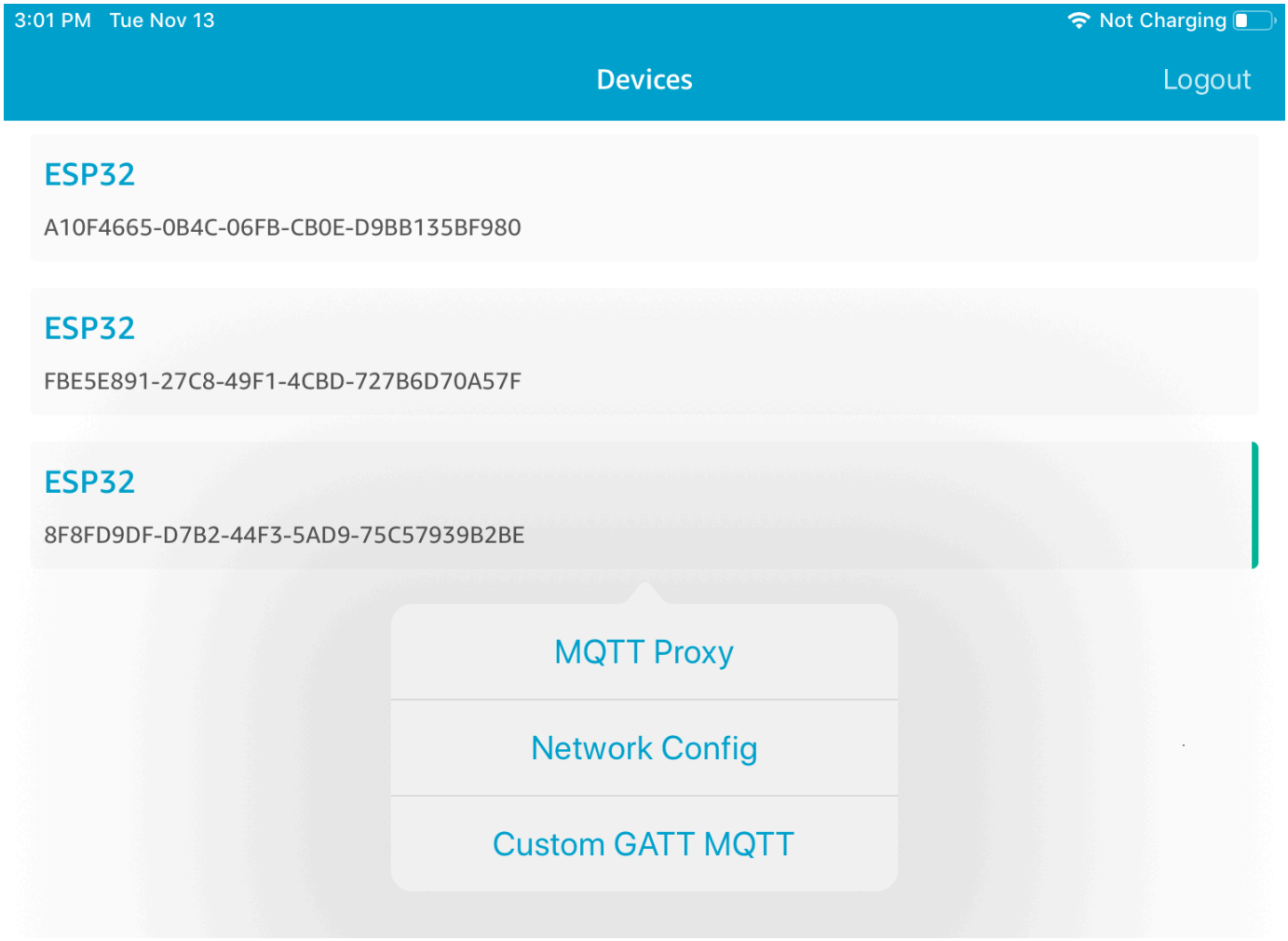
1. Abilitare la demo GATT Bluetooth Low Energy. In `vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/iot_ble_config.h` (dove *`vendor`* è il nome del fornitore e *`board`* è il nome della scheda che si sta utilizzando per eseguire le demo), aggiungere `#define IOT_BLE_ADD_CUSTOM_SERVICES ( 1 )` all'elenco di 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_CORE_MQTT_MUTUAL_AUTH_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 demo per Microchip Curiosity PIC32MZEF

> **Important**
>
> Questa demo è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Consigliamo di iniziare da qui quando si crea un nuovo progetto. Se disponi già di un progetto FreeRTOS

esistente basato sull'ormai obsoleto repository Amazon-FreeRTOS, consulta ilGuida alla migrazione del repository Github di Amazon-FreeRTOS.

> **Note**
>
> In accordo con Microchip, stiamo rimuovendo il Curiosity PIC32MZEF (DM320104) dal ramo principale del repository FreeRTOS Reference Integration e non lo includeremo più nelle nuove versioni. Microchip ha emesso un avviso ufficiale secondo cui il PIC32MZEF (DM320104) non è più consigliato per i nuovi progetti. È ancora possibile accedere ai progetti e al codice sorgente PIC32MZEF tramite i tag della versione precedente. Microchip consiglia ai clienti di utilizzare la scheda di sviluppo Curiosity PIC32MZ-EF-2.0 (DM320209) per nuovi progetti. La piattaforma PIC32mzv1 è ancora disponibile nella versione 202012.00 del repository FreeRTOS Reference Integration. Tuttavia, la piattaforma non è più supportata dalla versione 202107.00 del FreeRTOS Reference.

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.

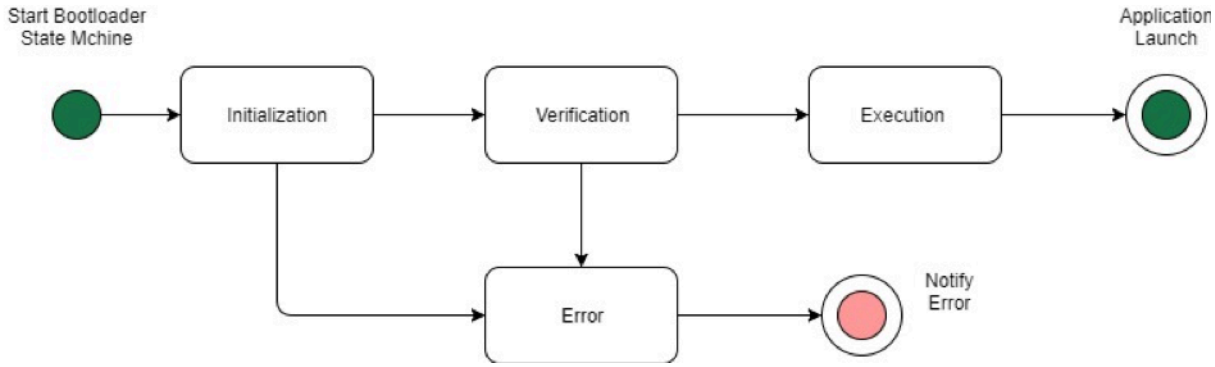
Il bootloader supporta le seguenti caratteristiche necessarie per OTA:

- 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 indicatiNosu 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

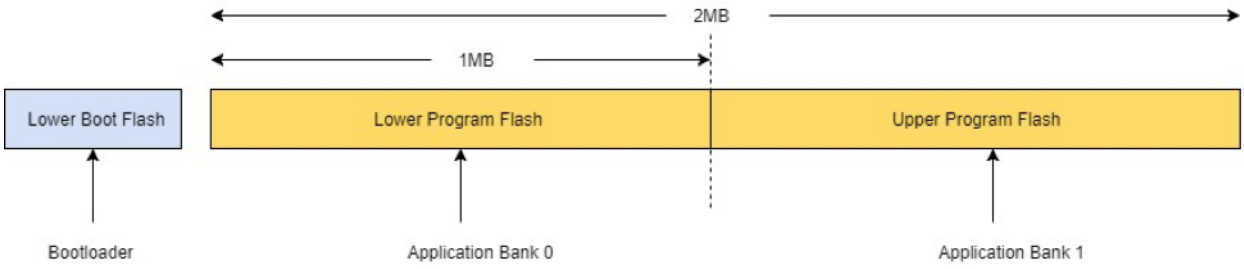
Se l'opzione di configurazione per avviare l'immagine predefinita è abilitata, il bootloader lancia l'applicazione da un indirizzo di esecuzione predefinito. Questa opzione deve essere disabilitata, ma non durante il debug.

Stato di errore del bootloader

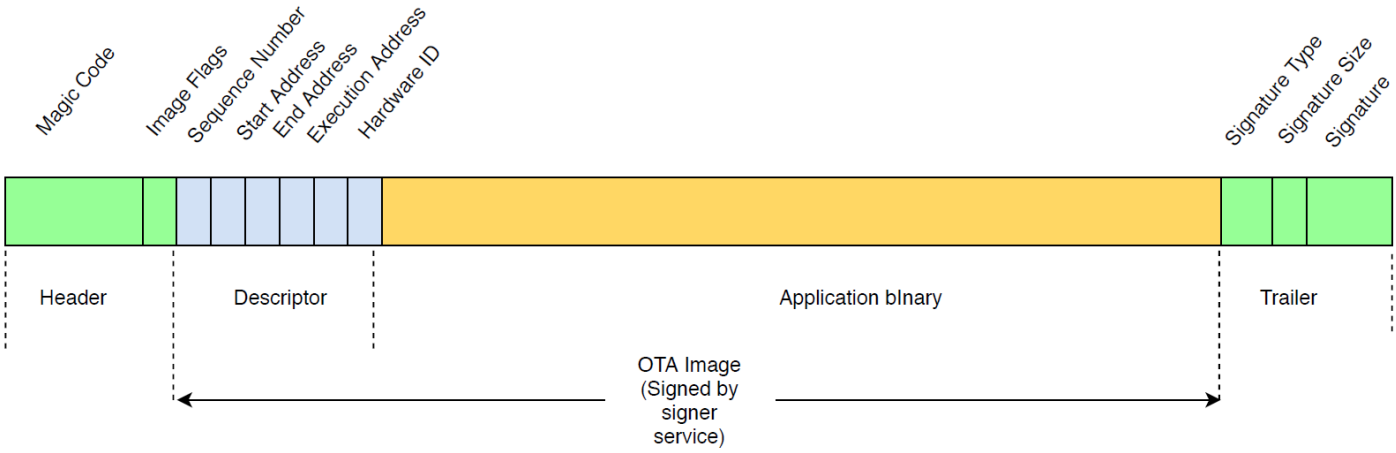
Il bootloader è in stato di errore e sul dispositivo non è presente nessuna immagine valida. Il bootloader deve inviare notifica all'utente. L'implementazione predefinita invia un messaggio di log alla console e il LED sulla scheda lampeggia velocemente in modo continuo.

Dispositivo Flash

La piattaforma Microchip Curiosity PIC32MZEF contiene un dispositivo Flash del programma interno di due megabyte (MB) diviso in due banchi di memoria. 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	Value (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
Indirizzo finale	4 byte

Campo descrittore	Dimensione (in 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*freertos*`/vendors/microchip/boards/curiosity_pic32mzef/aws_demos/mplab/` nel repository del codice sorgente di FreeRTOS.`aws_demos` 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.

Demo AWS IoT Device Defender

> **Important**
>
> Questa demo è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Consigliamo di iniziare da qui quando crei un nuovo progetto. Se disponi già di un progetto FreeRTOS esistente basato sull'ormai obsoleto repository Amazon-FreeRTOS, consulta ilGuida alla migrazione del repository Github di Amazon-FreeRTOS.

Introduzione

Questa demo mostra come utilizzare la libreriaAWS IoT Device Defender a cui connettersi AWS IoT Device Defender. La demo utilizza la libreria CoreMQTT per stabilire una connessione MQTT tramite TLS (autenticazione reciproca) al brokerAWS IoT MQTT e alla libreria CoreJSON per convalidare e analizzare le risposte ricevute dalAWS IoT Device Defender servizio. La demo mostra come creare un report in formato JSON utilizzando le metriche raccolte dal dispositivo e come inviare il rapporto costruito alAWS IoT Device Defender servizio. La demo mostra anche come registrare una funzione di callback con la libreria CoreMQTT per gestire le risposte delAWS IoT Device Defender servizio per confermare se un rapporto inviato è stato accettato o rifiutato.

> **Note**
>
> Per configurare ed eseguire le demo di FreeRTOS, segui i passaggi indicatiNosu FreeRTOS.

Funzionalità

Questa demo crea una singola attività applicativa che dimostra come raccogliere metriche, costruire un report Device Defender in formato JSON e inviarlo alAWS IoT Device Defender servizio tramite una connessione MQTT sicura al brokerAWS IoT MQTT. La demo include le metriche di rete standard e le metriche personalizzate. Per le metriche personalizzate, la demo include:

- Una metrica denominata "`task_numbers`" che è un elenco di ID di attività FreeRTOS. Il tipo di questa metrica è «elenco di numeri».
- Una metrica denominata "`stack_high_water_mark`" che rappresenta la filigrana massima dello stack per l'attività dell'applicazione demo. Il tipo di questa metrica è «numero».

Il modo in cui raccogliamo le metriche di rete dipende dallo stack TCP/IP in uso. Per le configurazioni FreeRTOS+TCP e LwIP supportate, forniamo implementazioni di raccolta di metriche che raccolgono metriche reali dal dispositivo e le inviano nelAWS IoT Device Defender rapporto. Puoi trovare le implementazioni per FreeRTOS+TCP e LWIP su GitHub.

Per le schede che utilizzano qualsiasi altro stack TCP/IP, forniamo definizioni stub delle funzioni di raccolta delle metriche che restituiscono zeri per tutte le metriche di rete. Implementa le funzioni*freertos*`/demos/device_defender_for_aws/metrics_collector/stub/metrics_collector.c` per il tuo stack di rete per inviare metriche reali. Il file è disponibile anche sul GitHubsito Web.

Per ESP32, la configurazione LWIP predefinita non utilizza il core locking e pertanto la demo utilizzerà metriche errate. 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 dell'esecuzione di una demo.

```
24 1682 [iot_thread] [INFO] MQTT connection successfully established with broker.


25 1682 [iot_thread] [INFO] A clean MQTT connection is established. Cleaning up all the stored outgoing publishes.


26 1682 [iot_thread] [INFO] Attempt to subscribe to the MQTT topic $aws/things/DemoThing/defender/metrics/json/accepted.27 1682 [
iot_thread] [INFO] SUBSCRIBE sent for topic $aws/things/DemoThing/defender/metrics/json/accepted to broker.
28 1722 [iot_thread] [INFO] Packet received. ReceivedBytes=3.
29 1722 [iot_thread] [INFO] MQTT_PACKET_TYPE_SUBACK.


30 1722 [iot_thread] [INFO] Subscribed to the topic $aws/things/DemoThing/defender/metrics/json/accepted with maximum QoS 1.
31 2322 [iot_thread] [INFO] Attempt to subscribe to the MQTT topic $aws/things/DemoThing/defender/metrics/json/rejected.32 2322 [
iot_thread] [INFO] SUBSCRIBE sent for topic $aws/things/DemoThing/defender/metrics/json/rejected to broker.
33 2382 [iot_thread] [INFO] Packet received. ReceivedBytes=3.
34 2382 [iot_thread] [INFO] MQTT_PACKET_TYPE_SUBACK.


35 2382 [iot_thread] [INFO] Subscribed to the topic $aws/things/DemoThing/defender/metrics/json/rejected with maximum QoS 1.
36 2982 [iot_thread] [INFO] the published payload:{"hed": {"rid": 1109,"v": "1.0"},"met": {"tp": {"pts": [{"pt": 33251}],"t": 1},
"up": {"pts": [],"t": 0},"ns": {"bi": 0,"bo": 0,"pi": 0,"po": 0},"tc": {"ec": {"cs": [{"lp": 33251,"rad": "44.236.152.27:888337 2
982 [iot_thread] [INFO] PUBLISH sent for topic $aws/things/DemoThing/defender/metrics/json to broker with packet ID 3.


38 3102 [iot_thread] [INFO] Packet received. ReceivedBytes=2.
39 3102 [iot_thread] [INFO] Ack packet deserialized with result: MQTTSuccess.
40 3102 [iot_thread] [INFO] State record updated. New state=MQTTPublishDone.
41 3102 [iot_thread] [INFO] PUBACK received for packet id 3.


42 3102 [iot_thread] [INFO] Cleaned up outgoing publish packet with packet id 3.


43 3102 [iot_thread] [INFO] Packet received. ReceivedBytes=141.
44 3102 [iot_thread] [INFO] De-serialized incoming PUBLISH packet: DeserializerResult=MQTTSuccess.
45 3102 [iot_thread] [INFO] State record updated. New state=MQTTPublishDone.
46 3502 [iot_thread] [INFO] UNSUBSCRIBE sent topic $aws/things/DemoThing/defender/metrics/json/accepted to broker.


47 3542 [iot_thread] [INFO] Packet received. ReceivedBytes=2.
48 3542 [iot_thread] [INFO] MQTT_PACKET_TYPE_UNSUBACK.


49 4142 [iot_thread] [INFO] UNSUBSCRIBE sent topic $aws/things/DemoThing/defender/metrics/json/rejected to broker.


50 4202 [iot_thread] [INFO] Packet received. ReceivedBytes=2.
51 4202 [iot_thread] [INFO] MQTT_PACKET_TYPE_UNSUBACK.


52 4802 [iot_thread] [INFO] Disconnected from the broker.
53 4802 [iot_thread] [INFO ][DEMO][4802] memory_metrics::freertos_heap::before::bytes::2088152

54 4802 [iot_thread] [INFO ][DEMO][4802] memory_metrics::freertos_heap::after::bytes::1985556

55 4802 [iot_thread] [INFO ][DEMO][4802] memory_metrics::demo_task_stack::before::bytes::1908

56 4802 [iot_thread] [INFO ][DEMO][4802] memory_metrics::demo_task_stack::after::bytes::1908

57 5802 [iot_thread] [INFO ][DEMO][5802] Demo completed successfully.

58 5804 [iot_thread] [INFO ][INIT][5804] SDK cleanup done.

59 5804 [iot_thread] [INFO ][DEMO][5804] -------DEMO FINISHED-------
```

Se la scheda non utilizza FreeRTOS+TCP o una configurazione LWIP supportata, l'output sarà simile al seguente.

```
24 1716 [iot_thread] [INFO] MQTT connection successfully established with broker.


25 1716 [iot_thread] [INFO] A clean MQTT connection is established. Cleaning up all the stored outgoing publishes.


26 1716 [iot_thread] [INFO] Attempt to subscribe to the MQTT topic $aws/things/DemoThing/defender/metrics/json/accepted.27 1716
 [iot_thread] [INFO] SUBSCRIBE sent for topic $aws/things/DemoThing/defender/metrics/json/accepted to broker.
28 1756 [iot_thread] [INFO] Packet received. ReceivedBytes=3.
29 1756 [iot_thread] [INFO] MQTT_PACKET_TYPE_SUBACK.


30 1756 [iot_thread] [INFO] Subscribed to the topic $aws/things/DemoThing/defender/metrics/json/accepted with maximum QoS 1.
31 2356 [iot_thread] [INFO] Attempt to subscribe to the MQTT topic $aws/things/DemoThing/defender/metrics/json/rejected.32 2356
 [iot_thread] [INFO] SUBSCRIBE sent for topic $aws/things/DemoThing/defender/metrics/json/rejected to broker.
33 2436 [iot_thread] [INFO] Packet received. ReceivedBytes=3.
34 2436 [iot_thread] [INFO] MQTT_PACKET_TYPE_SUBACK.


35 2436 [iot_thread] [INFO] Subscribed to the topic $aws/things/DemoThing/defender/metrics/json/rejected with maximum QoS 1.
36 3036 [iot_thread] [ERROR] Using stub definition of GetNetworkStats! Please implement for your network stack to get correct m
etrics.
37 3036 [iot_thread] [ERROR] Using stub definition of GetOpenTcpPorts! Please implement for your network stack to get correct m
etrics.
38 3036 [iot_thread] [ERROR] Using stub definition of GetOpenUdpPorts! Please implement for your network stack to get correct m
etrics.
39 3036 [iot_thread] [ERROR] Using stub definition of GetEstablishedConnections! Please implement for your network stack to get
 correct metrics.
40 3036 [iot_thread] [INFO] the published payload:{"hed": {"rid": 1079,"v": "1.0"},"met": {"tp": {"pts": [],"t": 0},"up": {"pts
": [],"t": 0},"ns": {"bi": 0,"bo": 0,"pi": 0,"po": 0},"tc": {"ec": {"cs": [],"t": 0}}},"cmet": {"stack_high_water_mark": [{"num
41 3036 [iot_thread] [INFO] PUBLISH sent for topic $aws/things/DemoThing/defender/metrics/json to broker with packet ID 3.


42 3196 [iot_thread] [INFO] Packet received. ReceivedBytes=2.
43 3196 [iot_thread] [INFO] Ack packet deserialized with result: MQTTSuccess.
44 3196 [iot_thread] [INFO] State record updated. New state=MQTTPublishDone.
45 3196 [iot_thread] [INFO] PUBACK received for packet id 3.


46 3196 [iot_thread] [INFO] Cleaned up outgoing publish packet with packet id 3.


47 3196 [iot_thread] [INFO] Packet received. ReceivedBytes=141.
48 3196 [iot_thread] [INFO] De-serialized incoming PUBLISH packet: DeserializerResult=MQTTSuccess.
49 3196 [iot_thread] [INFO] State record updated. New state=MQTTPublishDone.
50 3596 [iot_thread] [INFO] UNSUBSCRIBE sent topic $aws/things/DemoThing/defender/metrics/json/accepted to broker.


51 3656 [iot_thread] [INFO] Packet received. ReceivedBytes=2.
52 3656 [iot_thread] [INFO] MQTT_PACKET_TYPE_UNSUBACK.


53 4256 [iot_thread] [INFO] UNSUBSCRIBE sent topic $aws/things/DemoThing/defender/metrics/json/rejected to broker.


54 4336 [iot_thread] [INFO] Packet received. ReceivedBytes=2.
55 4336 [iot_thread] [INFO] MQTT_PACKET_TYPE_UNSUBACK.


56 4936 [iot_thread] [INFO] Disconnected from the broker.
57 4936 [iot_thread] [INFO ][DEMO][4936] memory_metrics::freertos_heap::before::bytes::2088152

58 4936 [iot_thread] [INFO ][DEMO][4936] memory_metrics::freertos_heap::after::bytes::1985556

59 4936 [iot_thread] [INFO ][DEMO][4936] memory_metrics::demo_task_stack::before::bytes::1908

60 4936 [iot_thread] [INFO ][DEMO][4936] memory_metrics::demo_task_stack::after::bytes::1908

61 5936 [iot_thread] [INFO ][DEMO][5936] Demo completed successfully.

62 5938 [iot_thread] [INFO ][INIT][5938] SDK cleanup done.

63 5938 [iot_thread] [INFO ][DEMO][5938] -------DEMO FINISHED-------
```

Il codice sorgente della demo è scaricabile nella*freertos*/demos/device_defender_for_aws/ directory o sul GitHubsito Web.

Iscrizione agliAWS IoT Device Defender argomenti

La funzione subscribeToDefenderArgomenti sottoscrive gli argomenti MQTT su cui 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 accettati da Device Defender. Utilizza la macro`DEFENDER_API_JSON_REJECTED` per costruire la stringa dell'argomento su cui verranno ricevute le risposte alle segnalazioni rifiutate da Device Defender.

Racdi parametri dell'Device Parametri

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 delAWS IoT Device Defender rapporto

La funzione generateDeviceMetricsReport genera un report 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 previstoAWS IoT Device Defender e lo scrive nel buffer fornito. Il formato del documento JSON previsto daAWS IoT Device Defender è specificato nelle metriche relative al dispositivo nella Guida per gliAWS IoT sviluppatori.

Pubblicazione delAWS IoT Device Defender rapporto

IlAWS IoT Device Defender rapporto è pubblicato sull'argomento MQTT per la pubblicazione diAWS IoT Device Defender report JSON. Il report è costruito utilizzando la macro`DEFENDER_API_JSON_PUBLISH`, come mostrato in questo frammento di codice sul GitHub sito Web.

Richiamata per la gestione delle risposte

La funzione publishCallback gestisce i messaggi di pubblicazione MQTT in entrata. Utilizza l'`Defender_MatchTopic`API dellaAWS IoT Device Defender libreria per verificare se il messaggio MQTT in entrata proviene dalAWS IoT Device Defender servizio. Se il messaggio proviene dalAWS IoT Device Defender servizio, analizza la risposta JSON ricevuta ed estrae l'ID del report nella risposta. L'ID del report viene quindi verificato in modo che corrisponda a quello inviato nel rapporto.

AWS IoT GreengrassApplicazione dimostrativa V1 discovery

> **Important**
>
> Questa demo è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Consigliamo di iniziare da qui quando si crea un nuovo progetto. Se hai già un progetto FreeRTOS esistente basato sull'ormai obsoleto repository Amazon-FreeRTOS, consulta ilGuida alla migrazione del repository Github di Amazon-FreeRTOS.

Prima di eseguire la demo diAWS IoT Greengrass Discovery per FreeRTOS, devi configurareAWSAWS IoT Greengrass, eAWS IoT. Per configurare AWS, segui le istruzioni in AWSConfigurazione dell'account e delle autorizzazioni. Per configurare AWS IoT Greengrass, è necessario creare un gruppo Greengrass e aggiungere un core Greengrass. Per ulteriori informazioni sulla configurazione di AWS IoT Greengrass, consulta le nozioni di base su AWS IoT Greengrass.

Dopo aver configurato AWS e AWS IoT Greengrass, è necessario configurare alcune ulteriori autorizzazioni per AWS IoT Greengrass.

Per configurare le autorizzazioni AWS IoT Greengrass

1. Accedi alla console IAM.
2. Dal pannello di navigazione, scegli Ruoli, quindi trova e scegli Greengrass_ServiceRole.
3. Scegli Allega politiche, seleziona AmazonS3FullAccess e AWSIoTFullAccessquindi scegli Allega politica.
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. Scegliete Greengrass_ServiceRole, quindi scegliete Salva.

Collega la tua schedaAWS IoT e configura la 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 policy 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 policy concede le autorizzazioni AWS IoT Greengrass per tutte le risorse.
5. Seleziona Create (Crea).

Per collegare la policy AWS IoT Greengrass al certificato del 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 Connect aAWS IoT Greengrass - ***Piattaforma*** anziché Connect aAWS IoT - ***Piattaforma***.

3. Configurazione delle demo di FreerTOS.

   Aprire *freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h, commentare #define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED e definire CONFIG_GREENGRASS_DISCOVERY_DEMO_ENABLED.

Dopo aver impostatoAWS IoT eAWS IoT Greengrass scaricato e configurato FreeRTOS, puoi creare, eseguire il flashing 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 Greengrass pubblica una serie di messaggi per il core Greengrass e per il client MQTT AWS IoT. Per visualizzare i messaggi nel clientAWS IoT MQTT, apri la AWS IoTconsole, scegli Test, scegli MQTT test client e quindi aggiungi un abbonamento afreertos/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'istanza Amazon EC2

Se lavori con un'istanza Amazon EC2

1. Trova il DNS pubblico (IPv4) associato alla tua istanza Amazon EC2: vai alla console Amazon EC2 e, nel pannello di navigazione a sinistra, scegli Istanze. Scegli l'istanza Amazon EC2, quindi scegli il pannello Descrizione. Cerca la voce relativa al Public DNS (IPv4) (DNS pubblico IPv4) e prendi nota.
2. Trova la voce Gruppi di sicurezza e scegli il gruppo di sicurezza collegato alla tua istanza Amazon EC2.

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	-
Tutti ICMP - IPv4	ICMP	Tutti	0.0.0.0/0	-
Tutti ICMP - IPv4	ICMP	Tutti	::0/0	-

4. Nella console AWS IoT scegli Greengrass, quindi Groups (Gruppi), e scegli il gruppo Greengrass creato in precedenza. Selezionare Settings (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 consiste nell'utilizzare il DNS pubblico (IPv4) 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 gruppo AWS IoT Greengrass. 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 del gruppo Greengrass, 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 allaAWS IoT cosa creata nellaAWS IoT console quando hai registrato la tua scheda - «TIGG1" nell'esempio qui riportato.

9. Avvia una distribuzione del gruppo AWS IoT Greengrass e assicurati che la distribuzione abbia esito positivo. Ora dovresti essere in grado di eseguire correttamente la demo di AWS IoT Greengrass.

AWS IoT Greengrass V2

Compatibilità conAWS IoT Greengrass V2 i dispositivi

AWS IoT Greengrass V2il supporto per i dispositivi client è retrocompatibile conAWS IoT Greengrass V1. È possibile connettere i dispositivi client FreeRTOS ai dispositivi core V2 senza modificare il codice dell'applicazione. Per consentire ai dispositivi client di connettersi a un dispositivo core V2, procedi come segue.

- Implementa il software Greengrass sul dispositivo principale Greengrass. Vedi Connect i dispositivi client ai dispositivi principali a cui connettere un dispositivoAWS IoT Greengrass V2.
- Per inoltrare messaggi (comprese le funzioni Lambda) tra dispositivi client, servizioAWS IoT Core cloud e componenti Greengrass, implementa e configura il componente bridge MQTT.
- Implementa il componente del rilevatore IP per rilevare automaticamente le informazioni sulla connettività o gestisci manualmente gli endpoint.
- Per ulteriori informazioni, consulta Interagire conAWS IoT i dispositivi locali.

Per maggiori dettagli, consultaAWS la documentazione sull'esecuzione AWS IoT Greengrass V1delle applicazioni suAWS IoT Greengrass V2.

Demo CoreHTTP

> **Important**
>
> Questa demo è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Consigliamo di iniziare da qui quando si crea un nuovo progetto. Se hai già un progetto FreeRTOS esistente basato sull'ormai obsoleto repository Amazon-FreeRTOS, consulta ilGuida alla migrazione del repository Github di Amazon-FreeRTOS.

Queste demo possono aiutarti a imparare a utilizzare la libreria CoreHTTP.

Argomenti

- Demo dell'autenticazione reciproca CoreHTTP
- Demo di caricamento di base CoreHTTP su Amazon S3
- Scarica la demo di CoreHTTP basic S3
- Demo multithread di base CoreHTTP

Demo dell'autenticazione reciproca CoreHTTP

> **Important**
>
> Questa demo è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Consigliamo di iniziare da qui quando si crea un nuovo progetto. Se hai già un progetto FreeRTOS esistente basato sull'ormai obsoleto repository Amazon-FreeRTOS, consulta ilGuida alla migrazione del repository Github di Amazon-FreeRTOS.

Introduzione

Il progetto demo CoreHTTP (Mutual Authentication) mostra come stabilire una connessione a un server HTTP utilizzando TLS con autenticazione reciproca tra client e 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 un flusso di lavoro di risposta alle richieste in HTTP.

> **Note**
>
> Per configurare ed eseguire le demo di FreeRTOS, segui i passaggi indicatiNosu FreeRTOS.

Funzionalità

Questa demo crea una singola attività applicativa con esempi che mostrano come completare quanto segue:

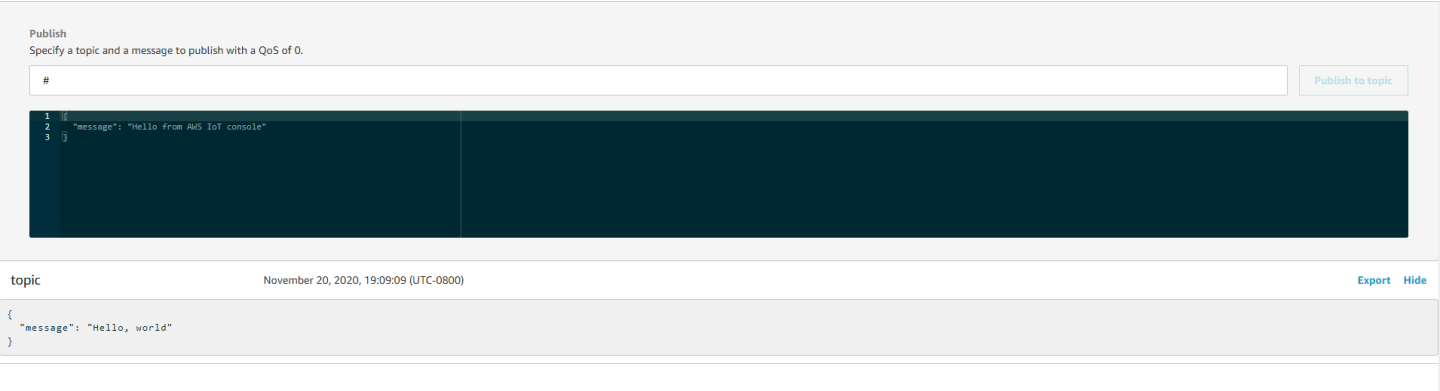
- Connect al server HTTP sull'AWS IoTendpoint.
- Inviare una richiesta POST.
- Ricevi la risposta.
- Disconnettiti dal server.

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------
```

LaAWS IoT console genera un output simile a quello della 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 serverAWS IoT HTTP

La connectToServerWithBackoffRetriesfunzione tenta di creare una connessione TLS reciprocamente autenticata al serverAWS IoT HTTP. Se la connessione fallisce, riprova dopo un timeout. Il valore di timeout aumenta in modo esponenziale fino al raggiungimento del numero massimo di tentativi o fino al raggiungimento del valore massimo di timeout. 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 non è possibile stabilire la connessione TLS al broker dopo il numero di tentativi configurato.

Invio di una richiesta HTTP e ricezione della risposta

La funzione prvSendHttpRequest mostra come inviare una richiesta POST al serverAWS IoT HTTP. Per ulteriori informazioni su come effettuare una richiesta all'API REST inAWS IoT, vedi Protocolli di comunicazione dei dispositivi - HTTPS. La risposta viene ricevuta con la stessa chiamata all'API CoreHTTP,`HTTPClient_Send`.

Demo di caricamento di base CoreHTTP su Amazon S3

> **Important**
>
> Questa demo è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Consigliamo di iniziare da qui quando si crea un nuovo progetto. Se hai già un progetto FreeRTOS esistente basato sull'ormai obsoleto repository Amazon-FreeRTOS, consulta ilGuida alla migrazione del repository Github di Amazon-FreeRTOS.

Introduzione

Questo esempio dimostra come inviare una richiesta PUT al server HTTP di Amazon Simple Storage Service (Amazon Simple Simple Storage Service) e caricare un file di piccole dimensioni. Esegue inoltre 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 indicatiNosu FreeRTOS.

Filettatura singola o 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 richiamare ripetutamente la libreria HTTP, le applicazioni multithread possono invece inviare richieste HTTP in background all'interno di un'attività di agente (o demone).

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 di Amazon S3 Simple Simple

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 parametri nella query URL prefirmata. Segui la procedura riportata di seguito per configurare la connessione aAWS.

1. Crea unAWS account:

   a. Se non lo hai già fatto, crea unAWS account.
   b. Gli account e le autorizzazioni vengono impostati utilizzandoAWS Identity and Access Management (IAM). Usi IAM per gestire le autorizzazioni per ogni utente del tuo account. Per impostazione predefinita, un utente non dispone delle autorizzazioni finché non le concede il proprietario principale.

      i. Per aggiungere un utente al tuoAWS account, consulta la Guida per l'utente IAM.

ii. Concedi l'autorizzazione al tuoAWS account per accedere a FreeRTOS eAWS IoT aggiungendo questa politica:

   - Amazon S3FullAccess

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. Caricamento di un file in Amazon Simple Simple Simple Simple Simple Simple Simple Simple Simple Simple Simple Simple Simple Simple Simple .
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 le istruzioni d'uso, consulta il`FreeRTOS-Plus/Demo/coreHTTP_Windows_Simulator/Common/presigned_url_generator/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 con successo richiedendo la dimensione del file. Il codice sorgente della demo è disponibile sul GitHubsito web.

Connessione al server HTTP di Amazon S3 Simple

La connectToServerWithBackoffRetriesfunzione tenta di stabilire una connessione TCP al server HTTP. Se la connessione fallisce, riprova dopo un timeout. Il valore di timeout aumenterà in modo esponenziale fino al raggiungimento del numero massimo di tentativi o fino al raggiungimento del valore massimo di timeout. 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`FreeRTOS-Plus/Source/Application-Protocols/network_transport/freertos_plus_tcp/using_mbedtls/using_mbedtls.c` file. La definizione di`prvConnectToServer` può essere trovata sul GitHubsito web.

Caricamento dei 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 funzione`HTTPClient_Send` 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 caricamenti

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 viene richiesto lo 0° byte. La dimensione del file è contenuta nel campo dell'`Content-Range`intestazione della risposta. 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 demo è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Consigliamo di iniziare da qui quando si crea un nuovo progetto. Se hai già un progetto FreeRTOS esistente basato sull'ormai obsoleto repository Amazon-FreeRTOS, consulta ilGuida alla migrazione del repository Github di Amazon-FreeRTOS.

Introduzione

Questa demo mostra come utilizzare le richieste di intervallo per scaricare file dal server HTTP Amazon S3. Le richieste di intervallo sono supportate in modo nativo nell'API CoreHTTP quando vengono utilizzate`HTTPClient_AddRangeHeader` per creare la richiesta HTTP. Per un ambiente con microcontrollori, le richieste di range sono altamente incoraggiate. 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, che richiedono 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 indicatiNosu FreeRTOS.

Filettatura singola o 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 richiamare ripetutamente la libreria HTTP, le applicazioni multithread possono invece inviare richieste HTTP in background all'interno di un'attività di agente (o demone).

Organizzazione del codice sorgente

Il progetto demo 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 di Amazon S3 Simple Simple

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 parametri nella query URL prefirmata. Segui la procedura riportata di seguito per configurare la connessione aAWS.

1. Crea unAWS account:
   a. Se non lo hai già fatto, crea e attiva unAWS account.
   b. Gli account e le autorizzazioni vengono impostati utilizzandoAWS 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 finché non le concede il proprietario principale.

   i. Per aggiungere un utente al tuoAWS account, consulta la Guida per l'utente IAM.

   ii. Concedi al tuoAWS account l'autorizzazione per accedere a FreeRTOS eAWS IoT aggiungendo queste politiche:

   - Amazon S3FullAccess

2. Crea un bucket in Simple Simple Simple Simple Simple Simple Simple Simple Simple Simple Simple Simple? nella Guida per l'utente della console Amazon Simple Storage Service.
3. Caricamento di un file in Simple Simple Simple Simple Simple Simple .
4. Genera un URL preregistrato utilizzando lo script che si trova in`FreeRTOS-Plus/Demo/coreHTTP_Windows_Simulator/Common/presigned_url_generator/presigned_urls_gen.py`. Per le istruzioni d'uso, vedere`FreeRTOS-Plus/Demo/coreHTTP_Windows_Simulator/Common/presigned_url_generator/README.md`.

Funzionalità

La demo recupera prima 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 GitHubsito web.

Connessione al server HTTP di Amazon S3 Simple

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 in modo esponenziale fino al raggiungimento del numero massimo di tentativi o fino al raggiungimento del valore massimo di timeout. `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-Plus/Source/Application-Protocols/Network_Transport/FreerTOS_Plus_TCP/Using_mbedtls/using_mbedtls.c.

Il codice sorgente per`prvConnectToServer()` può essere trovato su GitHub.

Creazione di una richiesta di intervallo

La funzione API`HTTPClient_AddRangeHeader()` supporta la serializzazione di un intervallo di byte nelle intestazioni delle richieste HTTP per formare una richiesta di intervallo. Le richieste di intervallo vengono utilizzate in questa demo per recuperare le dimensioni 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 viene richiesto lo 0° byte. La dimensione del file è contenuta nel campo dell'`Content-Range`intestazione della risposta. È prevista una`206 Partial Content` risposta dal server; qualsiasi altro codice di stato della risposta ricevuto è un errore.

Il codice sorgente per`prvGetS3ObjectFileSize()` può essere trovato 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 continuo 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`. Il codice di stato viene quindi verificato`206 Partial Content` e il numero di byte scaricati finora viene incrementato del valore dell'`Content-Length`intestazione.

Il codice sorgente per`prvDownloadS3ObjectFile()` può essere trovato su GitHub.

Demo multithread di base CoreHTTP

> **Important**
>
> Questa demo è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Consigliamo di iniziare da qui quando si crea un nuovo progetto. Se hai già un progetto FreeRTOS esistente basato sull'ormai obsoleto repository Amazon-FreeRTOS, consulta ilGuida alla migrazione del repository Github di Amazon-FreeRTOS.

Introduzione

Questa demo utilizza le code thread-safe di FreeRTOS per contenere richieste e risposte in attesa di essere elaborate. In questa demo, ci sono tre attività da prendere in considerazione.

- L'attività principale attende che le richieste vengano visualizzate 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 vengano visualizzate nella coda delle risposte. Registra ogni risposta che riceve.

Questa demo multithread di base è configurata per utilizzare una connessione TLS solo con 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 preimpostata.

Organizzazione del codice sorgente

Il progetto demo è denominato`http_demo_s3_download_multithreaded.c` e può essere trovato nella*`freertos`*`/demos/coreHTTP/` directory e nel GitHubsito Web.

Creazione del progetto demo

Il progetto demo utilizza l'edizione comunitaria gratuita di Visual Studio. Per creare la demo:

1. Aprire il file della soluzione di`mqtt_multitask_demo.sln` Visual Studio dall'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 -> Proprietà RTOSdemos -> Platform Toolset.

Configurazione del progetto demo

La demo utilizza lo stack TCP/IP FreeRTOS+TCP, quindi segui le istruzioni fornite per il progetto iniziale TCP/IP per:

1. Installa i componenti prerequisiti (come WinPCap).
2. Facoltativamente, imposta un indirizzo IP statico o dinamico, un indirizzo gateway e una maschera di rete.
3. Facoltativamente, 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 tentare di eseguire la demo HTTP.

Configurazione della connessione al server HTTP di Amazon S3 Simple Simple

Segui le istruzioni riportateConfigurazione della connessione al server HTTP di Amazon S3 Simple Simple nella demo di download di base di CoreHTTP.

Funzionalità

La demo crea tre attività in totale:

- Uno che invia richieste e riceve risposte in rete.
- Uno che crea richieste da inviare.
- Uno che elabora le risposte ricevute.

In questa demo, l'attività principale:

1. Crea le code di richieste e risposte.
2. Crea la connessione al server.
3. Crea le attività di richiesta e risposta.
4. Attende che la coda delle richieste invii le richieste in rete.
5. Inserisce le risposte ricevute tramite la rete nella coda delle risposte.

L'attività di richiesta:

1. Crea ciascuna delle richieste dell'intervallo.

L'attività di risposta:

1. Elabora ciascuna delle risposte ricevute.

Typedef

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 riceve una risposta in 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 preregistrato per il percorso degli oggetti nel bucket Simple Simple Simple Simple.
3. Si connette al server HTTP Amazon S3 utilizzando TLS con autenticazione del server.
4. Crea le code di richieste e risposte.
5. Crea le attività di richiesta e risposta.

La funzione`prvHTTPDemoTask()` esegue questa configurazione e fornisce lo stato 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()` può essere trovato 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 Simple Simple Simple. 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 viene richiesto lo 0° byte. La dimensione

del file è contenuta nel campo dell'`Content-Range`intestazione della risposta. 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` può essere trovato 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 in modo nativo nell'API della libreria client HTTP utilizzando la funzione`HTTPClient_AddRangeHeader`. La funzione`prvRequestS3ObjectRange` ne illustra l'uso`HTTPClient_AddRangeHeader()`.

Il codice sorgente della funzione`prvRequestS3ObjectRange` è disponibile su Github.

Attività di risposta HTTP

Le attività di risposta attendono nella coda di risposta le risposte ricevute in rete. L'attività principale compila 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 scrivendo il corpo della risposta nella memoria flash, ad esempio. 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 nella funzione`prvResponseTask`. Il codice sorgente di questa funzione è disponibile su Github.

AWS IoTDemo della libreria Jobs

> **Important**
>
> Questa demo è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Ti consigliamo di iniziare da qui quando crei un nuovo processo. Se disponi già di un progetto FreeRTOS esistente basato sull'ormai obsoleto repository Amazon-FreeRTOS, consulta ilGuida alla migrazione del repository Github di Amazon-FreeRTOS.

Introduzione

La demo della libreriaAWS IoT Jobs mostra come connettersi al servizioAWS IoT Jobs tramite una connessione MQTTAWS IoT, recuperare un lavoro ed elaborarlo su un dispositivo. Il progetto demo

diAWS IoT Jobs utilizza la porta Windows di FreeRTOS, quindi può essere creato e valutato con la versione Visual Studio Community su Windows. Non è necessario alcun microcontrollore hardware. La demo stabilisce una connessione sicura al brokerAWS IoT MQTT utilizzando TLS nello stesso modo in cuiDemo dell'autenticazione reciproca CoreMQTT.

> **Note**
>
> Per configurare ed eseguire le demo di FreeRTOS, segui i passaggi indicatiNosu 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.

Configurare la connessione al brokerAWS IoT MQTT

In questa demo, si utilizza una connessione MQTT al brokerAWS IoT MQTT. Questa connessione è configurata nello stesso modo diDemo dell'autenticazione reciproca CoreMQTT.

Funzionalità

La demo mostra il flusso di lavoro utilizzato per ricevereAWS IoT e processare i lavori su un dispositivo. La demo è interattiva e richiede di creare lavori utilizzando laAWS IoT console o ilAWS Command Line Interface (AWS CLI). Per ulteriori informazioni sulla creazione di un processo, consulta create-processo nel processo di riferimentoAWS CLI del processo. La demo richiede che il documento di lavoro abbia un set di`action` chiavi`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 comandoAWS 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. consulta il seguente formato per questo processo.

```
{
    "action": "publish",
    "message": "ADD_MESSAGE_HERE",
    "topic": "topic/name/here"
}
```

La demo si ripete 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 ingresso della demo di Jobs

Il codice sorgente della funzione di ingresso demo di Jobs è disponibile su GitHub. Questa funzione esegue le seguenti operazioni:

1. Stabilisci una connessione MQTT utilizzando le funzioni di supporto in`mqtt_demo_helpers.c`.
2. Iscriviti all'argomento MQTT per l'`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 libreriaAWS IoT Jobs.
3. Pubblica sull'argomento MQTT per l'`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 libreriaAWS IoT Jobs.
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, annulla l'iscrizione all'argomento MQTT e disconnettiti, utilizzando le funzioni di supporto presenti nel`mqtt_demo_helpers.c` file.

Richiamata per i messaggi MQTT ricevuti

La prvEventCallbackfunzione chiama`Jobs_MatchTopic` dalla libreriaAWS IoT Jobs per classificare il messaggio MQTT in entrata. 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 job. Di particolare interesse è la`prvSendUpdateForJob` funzione.

Inviare un aggiornamento per un lavoro in esecuzione

La funzione 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 demo è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Consigliamo di iniziare da questa sezione quando si crea un nuovo progetto. Se hai già un progetto FreeRTOS esistente basato sull'ormai obsoleto repository Amazon-FreeRTOS, consulta ilGuida alla migrazione del repository Github di Amazon-FreeRTOS.

Queste demo in questa sezione ti aiuteranno a imparare a utilizzare la libreria CoreMQTT.

Argomenti

- Demo dell'autenticazione reciproca CoreMQTT
- Dimostrazione di condivisione della connessione dell'agente CoreMQTT

Demo dell'autenticazione reciproca CoreMQTT

> **Important**
>
> Questa demo è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Consigliamo di iniziare da questa sezione quando si crea un nuovo progetto. Se hai già un progetto FreeRTOS esistente basato sull'ormai obsoleto repository Amazon-FreeRTOS, consulta ilGuida alla migrazione del repository Github di Amazon-FreeRTOS.

Introduzione

Il progetto demo di autenticazione reciproca CoreMQTT mostra come stabilire una connessione a un broker MQTT utilizzando TLS con autenticazione reciproca tra client e 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 e pubblicazione di MQTT a livello QoS 1. Sottoscrive 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 al broker e ricezione dello stesso messaggio dal broker si ripete all'infinito. I messaggi in questa demo vengono inviati con QoS 1, il che garantisce almeno una consegna in base alle specifiche MQTT.

> **Note**
>
> Per configurare ed eseguire le demo di FreeRTOS, segui i passaggi indicatiNosu FreeRTOS.

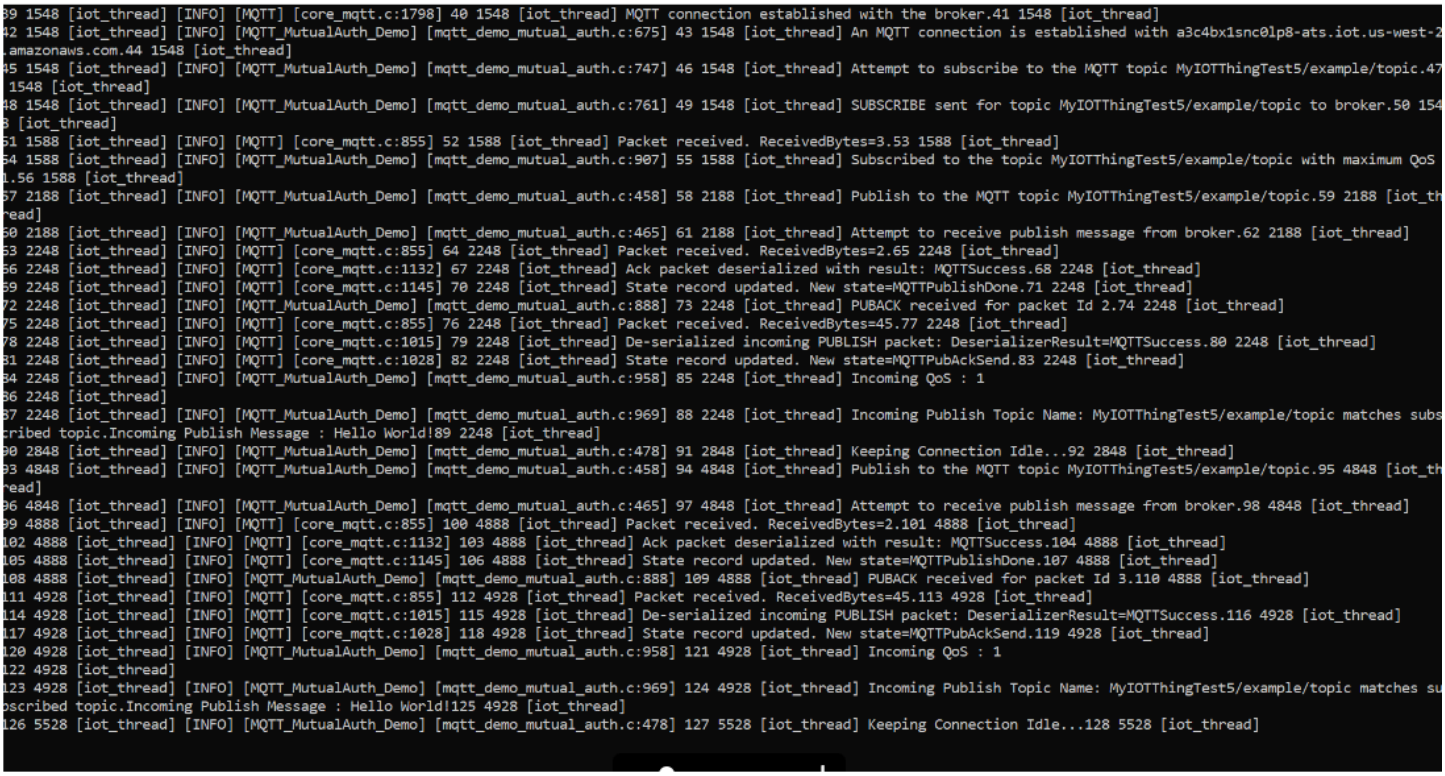
Codice sorgente

Il file sorgente della demo è denominato`mqtt_demo_mutual_auth.c` e può essere trovato nella`freertos`/`demos/coreMQTT/` directory e nel GitHubsito Web.

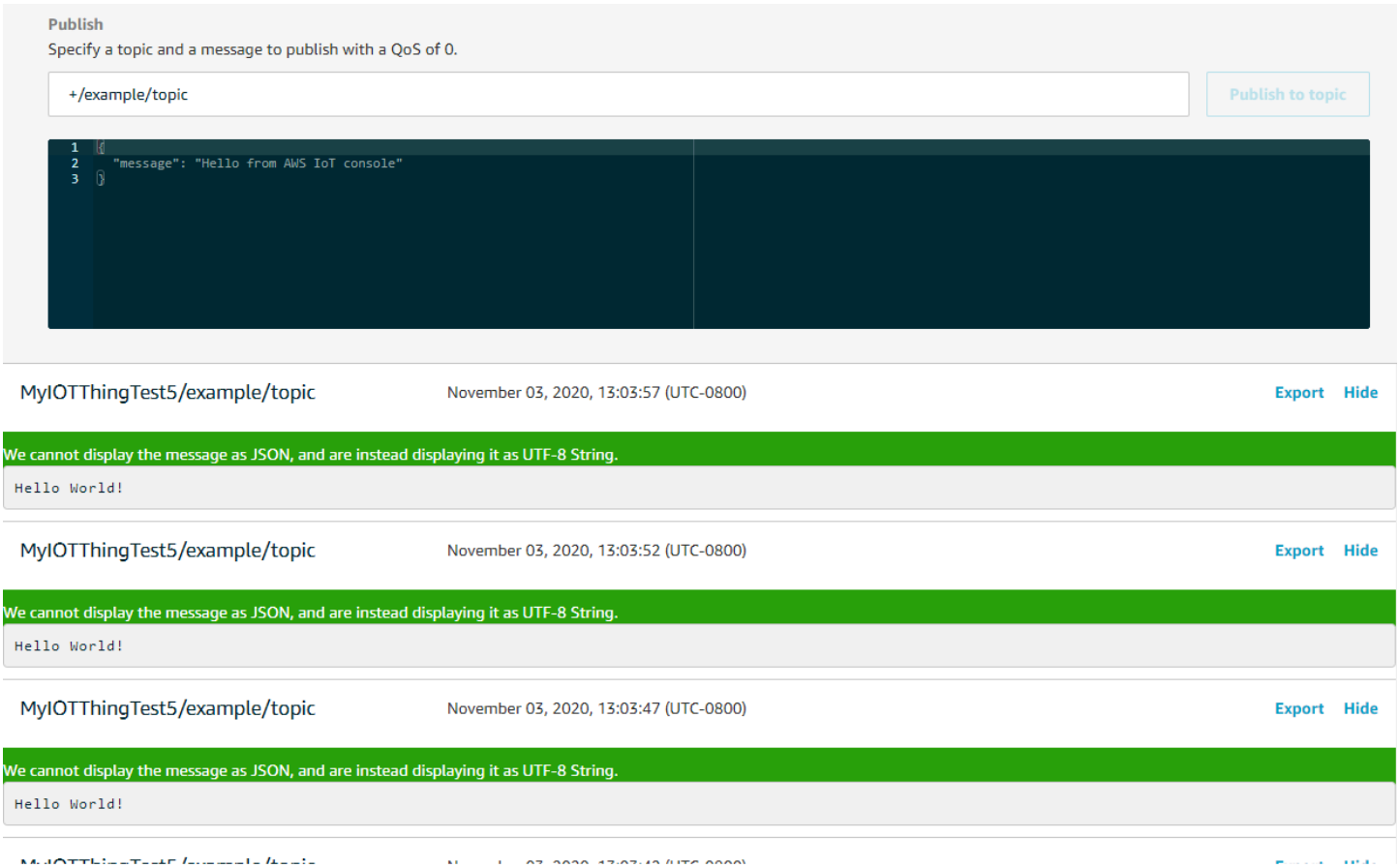
Funzionalità

La demo crea una singola attività applicativa che passa in rassegna una serie di esempi che dimostrano come connettersi al broker, iscriversi a un argomento sul broker, pubblicare su un argomento sul broker e infine disconnettersi dal broker. L'applicazione demo si iscrive e pubblica sullo stesso argomento. Ogni volta che la demo pubblica un messaggio al broker MQTT, il broker invia lo stesso messaggio all'applicazione demo.

Il completamento della demo ti aiuteranno a ottenere un risultato simile al seguente.



LaAWS IoT console genererà un risultato simile al seguente.



Riprova la logica con backoff e jitter esponenziali

La funzione prvBackoffForRetry mostra come le operazioni di rete fallite con il server, ad esempio connessioni TLS o richieste di iscrizione MQTT, possano essere ritentate con backoff e jitter esponenziali. La funzione calcola il periodo di backoff per il tentativo successivo ed esegue il ritardo di 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 PKCS11 per generare il numero casuale. L'uso del modulo PKCS11 consente l'accesso a un True Random Number Generator (TRNG) se la piattaforma del fornitore lo supporta. Si consiglia di inviare al generatore di numeri casuali una fonte di entropia specifica del dispositivo in modo da ridurre la probabilità di collisioni tra dispositivi durante i tentativi di connessione.

Connessione al broker MQTT

La prvConnectToServerWithBackoffRetriesfunzione tenta di creare una connessione TLS reciprocamente autenticata 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 fino al raggiungimento del periodo massimo di backoff.

La`BackoffAlgorithm_GetNextBackoff` funzione fornisce un valore di backoff crescente in modo esponenziale e ritorna`RetryUtilsRetriesExhausted` quando è stato raggiunto il numero massimo di tentativi. La`prvConnectToServerWithBackoffRetries` funzione restituisce uno stato di errore se non è possibile stabilire la connessione TLS al broker dopo il numero di tentativi configurato.

LaConnectionWithBroker funzione prvCreateMqtt dimostra come stabilire una connessione MQTT a un broker MQTT con una sessione pulita. Utilizza l'interfaccia di trasporto TLS, implementata nel`FreeRTOS-Plus/Source/Application-Protocols/platform/freertos/transport/src/tls_freertos.c` file. 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 temporale sono impostate in un contesto MQTT utilizzando la`MQTT_Init` funzione. Mostra anche come è impostata una funzione di callback di eventi pointer (`prvEventCallback`). Questo callback viene utilizzato per segnalare i messaggi in arrivo.

Iscrizione a un argomento MQTT

LaSubscribeWithBackoffRetries funzione prvMqtt dimostra come iscriversi a un filtro per argomenti sul broker MQTT. L'esempio dimostra come iscriversi a un filtro di argomento, ma è possibile passare un elenco di filtri di argomento nella stessa chiamata API di iscrizione per iscriversi a 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

LaPublishToTopic funzione PrvMqtt 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 messaggi in arrivo. Quando viene ricevuto un messaggio MQTT in entrata, chiama la funzione di callback dell'evento 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`prvMQTTProcessIncomingPublish()` la gestione dei messaggi di pubblicazione in entrata o`prvMQTTProcessResponse()` la gestione dei riconoscimenti (ACK).

Elaborazione dei pacchetti di pubblicazione MQTT in entrata

LaProcessIncomingPublish funzione PrvMqtt 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 prvMqttUnsubscribeFromTopic.

Modifica della CA principale utilizzata nella demo

Per impostazione predefinita, le demo di FreeRTOS utilizzano il certificato Amazon Root CA 1 (chiave RSA a 2048 bit) per l'autenticazione con ilAWS IoT Core server. 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, individua la linea seguente.

   ```
   * #define democonfigROOT_CA_PEM    "...insert here..."
   ```

   Rimuovi il commento da 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.

Dimostrazione di condivisione della connessione dell'agente CoreMQTT

> **Important**
>
> Questa demo è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Consigliamo di iniziare da questa sezione quando si crea un nuovo progetto. Se hai già un progetto FreeRTOS esistente basato sull'ormai obsoleto repository Amazon-FreeRTOS, consulta ilGuida alla migrazione del repository Github di Amazon-FreeRTOS.

Introduzione

Il progetto demo di condivisione della connessione CoreMQTT mostra come utilizzare un'applicazione multithread per stabilire una connessione al brokerAWS MQTT utilizzando TLS con autenticazione reciproca tra client e 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 e 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 i messaggi dal server a livello QoS 1. Questo ciclo di pubblicazione sul broker e ricezione dello stesso messaggio dal broker viene ripetuto più volte per ogni attività creata. I messaggi in questa demo vengono inviati in modalità QoS 1, che garantisce almeno una consegna in base alle specifiche MQTT.

> **Note**
>
> Per configurare ed eseguire le demo di FreeRTOS, segui i passaggi indicatiNosu FreeRTOS.

Questa demo utilizza una coda thread safe per contenere i comandi per interagire con l'API MQTT. Ci sono due attività da prendere in considerazione in questa demo.

- Un'attività MQTT Agent (principale) elabora i comandi dalla coda dei comandi mentre altre attività li mettono in coda. Questa attività entra in un ciclo, durante il quale elabora i comandi dalla coda dei comandi. Se viene ricevuto un comando di terminazione, questa attività uscirà dal ciclo.
- Un'attività dimostrativa di subpub crea una sottoscrizione a un argomento MQTT, quindi crea operazioni di pubblicazione e le inserisce nella coda dei comandi. Queste operazioni di pubblicazione vengono quindi eseguite dall'attività MQTT Agent. L'attività demo subpub attende il completamento della pubblicazione, indicata dall'esecuzione del callback di completamento del comando, quindi inserisce un breve ritardo prima di iniziare la pubblicazione successiva. Questa attività mostra esempi di come le attività dell'applicazione utilizzerebbero l'API CoreMQTT Agent.

Per i messaggi di pubblicazione in entrata, l'agente CoreMqtt richiama una singola funzione di callback. Questa demo include anche un gestore di sottoscrizioni che consente alle attività di specificare un callback da invocare per i messaggi di pubblicazione in entrata sugli argomenti a cui si sono abbonati. Il callback di pubblicazione in entrata dell'agente in questa demo richiama il gestore degli abbonamenti per pubblicare a ventaglio qualsiasi attività che abbia registrato un abbonamento.

Questa demo utilizza una connessione TLS con autenticazione reciproca a cui connettersiAWS. Se la rete si disconnette inaspettatamente durante la demo, il client tenta di riconnettersi utilizzando una logica di backoff esponenziale. Se il client si riconnette con successo, ma il broker non riesce a riprendere la sessione precedente, il cliente si iscriverà nuovamente agli stessi argomenti della sessione precedente.

A thread singolo o multithread

Esistono due modelli di utilizzo di CoreMQTT, a thread singolo e multithread (multitasking). Il modello a thread singolo utilizza la libreria CoreMQTT esclusivamente da un thread e richiede di effettuare chiamate esplicite ripetute nella libreria MQTT. I casi d'uso multithread possono invece eseguire il protocollo MQTT in background all'interno di un'attività agente (o demone), come mostrato nella demo documentata qui. Quando si esegue il protocollo MQTT in un'attività agente, non è necessario gestire in modo esplicito alcuno stato MQTT o chiamare la funzione`MQTT_ProcessLoop` API. Inoltre, quando 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

I file sorgente dimostrativi sono denominati`mqtt_agent_task.c`e `simple_sub_pub_demo.c` e possono essere trovati nella*`freertos`*`/demos/coreMQTT_Agent/` directory e nel GitHubsito Web.

Funzionalità

Questa demo crea almeno due attività: una principale che elabora le richieste di chiamate API MQTT e un numero configurabile di sottoattività che creano tali richieste. In questa demo, l'attività principale crea le sottoattività, chiama il ciclo di elaborazione e successivamente pulisce. L'attività principale crea una singola connessione MQTT al broker condivisa tra le sottoattività. Le sottoattività creano un abbonamento MQTT con il broker e quindi pubblicano i messaggi su di esso. Ogni sottoattività utilizza un argomento univoco per le sue pubblicazioni.

Attività principale

L'attività principale dell'applicazione, RunCoreMQTTAgentDemo, stabilisce una sessione MQTT, crea le sottoattività ed esegue il ciclo di elaborazione MQTTAgent_CommandLoop fino alla ricezione di un comando di terminazione. Se la rete si disconnette inaspettatamente, la demo si riconnetterà al broker in background e ristabilirà gli abbonamenti con il broker. Al termine del ciclo di elaborazione, si disconnette dal broker.

Comandi

Quando si richiama un'API dell'agente CoreMQTT, viene creato un comando che viene inviato alla coda dell'attività dell'agente, che viene elaborato in`MQTTAgent_CommandLoop()`. Al momento della creazione del comando, è possibile passare parametri di callback di completamento e di contesto opzionali. Una volta completato il comando corrispondente, il callback di completamento verrà richiamato con il contesto passato e tutti i valori restituiti creati come risultato del comando. La firma per il callback di completamento è la seguente:

```
typedef void (* MQTTAgentCommandCallback_t )( void * pCmdCallbackContext,
                                              MQTTAgentReturnInfo_t * pReturnInfo );
```

Il contesto di completamento dei comandi è definito dall'utente; per questa demo, è: struct MQTTAgentCommandContext.

I comandi sono considerati completati quando:

- Abbonamento, annullamento dell'iscrizione e pubblicazione con QoS > 0: una volta ricevuto il pacchetto di conferma corrispondente.
- Tutte le altre operazioni: una volta richiamata l'API CoreMQTT corrispondente.

Qualsiasi struttura utilizzata dal comando, incluse le informazioni di pubblicazione, le informazioni sulla sottoscrizione e i contesti di completamento, deve rimanere nell'ambito fino al completamento del comando. Un'attività di chiamata non deve riutilizzare nessuna delle strutture di un comando prima dell'invocazione del callback di completamento. Nota che, poiché il callback di completamento viene richiamato dall'agente MQTT, verrà eseguito con il contesto del thread dell'attività dell'agente, non con l'attività che ha creato il comando. I meccanismi di comunicazione tra processi, come le notifiche delle attività o le code, possono essere utilizzati per segnalare il completamento dell'operazione di chiamata del comando.

Eseguire il ciclo di comandi

I comandi vengono elaborati continuamente in`MQTTAgent_CommandLoop()`. Se non ci sono comandi da elaborare, il ciclo attenderà che ne venga aggiunto un massimo alla coda e, se non viene aggiunto alcun comando, eseguirà una singola iterazione di`MQTT_ProcessLoop()`.`MQTT_AGENT_MAX_EVENT_QUEUE_WAIT_TIME` Ciò garantisce sia la gestione di MQTT Keep-Alive sia la ricezione di eventuali pubblicazioni in entrata anche quando non ci sono comandi nella coda.

La funzione Command Loop verrà restituita per i seguenti motivi:

- Un comando restituisce inoltre qualsiasi codice di stato`MQTTSuccess`. Lo stato di errore viene restituito dal ciclo di comandi, quindi puoi decidere come gestirlo. In questa demo, la connessione TCP viene ristabilita e viene effettuato un tentativo di riconnessione. In caso di errore, è possibile eseguire una riconnessione in background senza l'intervento di altre attività che utilizzano MQTT.
- Viene elaborato un comando di disconnessione (da`MQTTAgent_Disconnect`). Il ciclo di comandi esce in modo che TCP possa essere disconnesso.
- Viene elaborato un comando di terminazione (da`MQTTAgent_Terminate`). Questo comando contrassegna anche qualsiasi comando ancora in coda o in attesa di un pacchetto di conferma come errore, con un codice di ritorno di`MQTTRecvFailed`.

Gestione delle sottoscrizioni

Poiché la demo utilizza più argomenti, un gestore di sottoscrizioni è un modo conveniente per associare gli argomenti sottoscritti a richiami o attività univoci. Il gestore degli abbonamenti in questa demo è a thread singolo, quindi non deve essere utilizzato da più attività contemporaneamente. In questa demo, le funzioni del gestore delle sottoscrizioni vengono richiamate solo dalle funzioni di callback passate all'agente MQTT ed eseguite solo con il contesto del thread dell'attività dell'agente.

Semplice operazione di sottoscrizione e pubblicazione

Ogni istanza di prvSimpleSubscribePublishTaskcrea una sottoscrizione a un argomento MQTT e crea operazioni di pubblicazione per quell'argomento. Per dimostrare più tipi di pubblicazione, le attività con numeri pari utilizzano QoS 0 (che sono complete una volta inviato il pacchetto di pubblicazione) e le attività dispari utilizzano QoS 1 (che vengono completate alla ricezione di un pacchetto PUBACK).

Over-the-air aggiorna l'applicazione demo

FreeRTOS include un'applicazione demo che dimostra la funzionalità della libreria over-the-air (OTA). L'applicazione demo OTA si trova nel file*freertos*`/demos/ota/ota_demo_core_mqtt/ota_demo_core_mqtt.c` o nel*freertos*`/demos/ota/ota_demo_core_http/ota_demo_core_http.c` file.

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 per esercitare la libreria OTA utilizzandovRun`OTAUpdateDemo()`.
3. Crea un client MQTT utilizzando `_establishMqttConnection()`.
4. Si connette al brokerAWS 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 e quindi riprende l'agente OTA.

Prima di poter utilizzare gli aggiornamenti OTA, completa tutti i prerequisiti inAggiornamenti via etere di FreeRTOS

Dopo aver completato la configurazione per gli aggiornamenti OTA, scarica, crea, esegui il flashing ed esegui la demo OTA di FreeRTOS su una piattaforma che supporti la funzionalità OTA. Sono disponibili istruzioni dimostrative specifiche per i seguenti dispositivi qualificati per FreeRTOS:

- Texas Instruments CC3220SF-LAUNCHXL

- Microchip Curiosity PIC32MZEF
- Espressif ESP32
- Scarica, crea, esegui il flashing ed esegui la demo OTA di FreeRTOS sul Renesas RX65N

Dopo aver creato, memorizzato nella flash ed eseguito l'applicazione demo OTA, puoi utilizzare la console AWS IoT o AWS CLI per creare un processo di aggiornamento OTA. 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.

```
    249 21207 [iot_thread] [ota_demo_core_mqtt.c:1850] [INFO] [MQTT]  Received: 0
 Queued: 0   Processed: 0   Dropped: 0
    250 21247 [MQTT Agent Task] [core_mqtt.c:886] [INFO] [MQTT] Packet received.
 ReceivedBytes=601.
    251 21247 [MQTT Agent Task] [core_mqtt.c:1045] [INFO] [MQTT] De-serialized incoming
 PUBLISH packet: DeserializerResult=MQTTSuccess.
    252 21248 [MQTT Agent Task] [core_mqtt.c:1058] [INFO] [MQTT] State record updated.
 New state=MQTTPubAckSend.
    253 21249 [MQTT Agent Task] [ota_demo_core_mqtt.c:976] [INFO] [MQTT] Received job
 message callback, size 548.
    254 21252 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
 value]=[execution.jobId: AFR_OTA-9702f1a3-b747-4c3e-a0eb-a3b0cf83ddbb]
    255 21253 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
 value]=[execution.jobDocument.afr_ota.streamname: AFR_OTA-945d320b-a18b-441b-
b435-4a18d4e7671f]
    256 21255 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
 value]=[execution.jobDocument.afr_ota.protocols: ["MQTT"]]
    257 21256 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
 value]=[filepath: aws_demos.bin]
    258 21257 [OTA Agent Task] [ota.c:1684] [INFO] [OTA] Extracted parameter: [key:
 value]=[filesize: 1164016]
    259 21258 [OTA Agent Task] [ota.c:1684] [INFO] [OTA] Extracted parameter: [key:
 value]=[fileid: 0]
    260 21259 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
 value]=[certfile: ecdsa-sha256-signer.crt.pem]
    261 21260 [OTA Agent Task] [ota.c:1575] [INFO] [OTA] Extracted parameter [ sig-
sha256-ecdsa: MEQCIE1SFkIHHiZAvkPpu6McJtx7SYoD... ]
    262 21261 [OTA Agent Task] [ota.c:1684] [INFO] [OTA] Extracted parameter: [key:
 value]=[fileType: 0]
```

```
    263 21262 [OTA Agent Task] [ota.c:2199] [INFO] [OTA] Job document was accepted.
 Attempting to begin the update.
    264 21263 [OTA Agent Task] [ota.c:2323] [INFO] [OTA] Job parsing success:
 OtaJobParseErr_t=OtaJobParseErrNone, Job name=AFR_OTA-9702f1a3-b747-4c3e-a0eb-
a3b0cf83ddbb
    265 21318 [iot_thread] [ota_demo_core_mqtt.c:1850] [INFO] [MQTT]  Received: 0
 Queued: 0   Processed: 0   Dropped: 0
    266 21418 [iot_thread] [ota_demo_core_mqtt.c:1850] [INFO] [MQTT]  Received: 0
 Queued: 0   Processed: 0   Dropped: 0
    267 21469 [OTA Agent Task] [ota.c:938] [INFO] [OTA] Setting OTA data interface.
    268 21470 [OTA Agent Task] [ota.c:2839] [INFO] [OTA] Current State=[CreatingFile],
 Event=[ReceivedJobDocument], New state=[CreatingFile]
    269 21482 [MQTT Agent Task] [core_mqtt.c:886] [INFO] [MQTT] Packet received.
 ReceivedBytes=3.
    270 21483 [OTA Agent Task] [ota_demo_core_mqtt.c:1503] [INFO] [MQTT] SUBSCRIBED
 to topic $aws/things/__test_infra_thing71/streams/AFR_OTA-945d320b-a18b-441b-
b435-4a18d4e7671f/data/cbor to bro
    271 21484 [OTA Agent Task] [ota.c:2839] [INFO] [OTA] Current
 State=[RequestingFileBlock], Event=[CreateFile], New state=[RequestingFileBlock]
    272 21518 [iot_thread] [ota_demo_core_mqtt.c:1850] [INFO] [MQTT]  Received: 0
 Queued: 0   Processed: 0   Dropped: 0
    273 21532 [MQTT Agent Task] [core_mqtt_agent_command_functions.c:76] [INFO] [MQTT]
 Publishing message to $aws/things/__test_infra_thing71/streams/AFR_OTA-945d320b-
a18b-441b-b435-4a18d4e7671f/
    274 21534 [OTA Agent Task] [ota_demo_core_mqtt.c:1553] [INFO] [MQTT] Sent PUBLISH
 packet to broker $aws/things/__test_infra_thing71/streams/AFR_OTA-945d320b-a18b-441b-
b435-4a18d4e7671f/get/cbor
    275 21534 [OTA Agent Task] [ota_mqtt.c:1112] [INFO] [OTA] Published to MQTT
 topic to request the next block: topic=$aws/things/__test_infra_thing71/streams/
AFR_OTA-945d320b-a18b-441b-b435-4a1
    276 21537 [OTA Agent Task] [ota.c:2839] [INFO] [OTA] Current
 State=[WaitingForFileBlock], Event=[RequestFileBlock], New state=[WaitingForFileBlock]
    277 21558 [MQTT Agent Task] [core_mqtt.c:886] [INFO] [MQTT] Packet received.
 ReceivedBytes=4217.
    278 21559 [MQTT Agent Task] [core_mqtt.c:1045] [INFO] [MQTT] De-serialized incoming
 PUBLISH packet: DeserializerResult=MQTTSuccess.
    279 21560 [MQTT Agent Task] [core_mqtt.c:1058] [INFO] [MQTT] State record updated.
 New state=MQTTPublishDone.
    280 21561 [MQTT Agent Task] [ota_demo_core_mqtt.c:1026] [INFO] [MQTT] Received data
 message callback, size 4120.
    281 21563 [OTA Agent Task] [ota.c:2464] [INFO] [OTA] Received valid file block:
 Block index=0, Size=4096
    282 21566 [OTA Agent Task] [ota.c:2683] [INFO] [OTA] Number of blocks remaining:
 284
```

```
   ... // Output removed for brevity

   3672 42745 [OTA Agent Task] [ota.c:2464] [INFO] [OTA] Received valid file block:
Block index=284, Size=752
   3673 42747 [OTA Agent Task] [ota.c:2633] [INFO] [OTA] Received final block of the
update.
   (428298) ota_pal: No such certificate file: ecdsa-sha256-signer.crt.pem. Using
certificate in ota_demo_config.h.
   3674 42818 [iot_thread] [ota_demo_core_mqtt.c:1850] [INFO] [MQTT]  Received: 285
Queued: 285   Processed: 284   Dropped: 0
   3675 42918 [iot_thread] [ota_demo_core_mqtt.c:1850] [INFO] [MQTT]  Received: 285
Queued: 285   Processed: 284   Dropped: 0

   ... // Output removed for brevity

   3678 43197 [OTA Agent Task] [ota.c:2654] [INFO] [OTA] Received entire update and
validated the signature.
   3685 43215 [OTA Agent Task] [ota_demo_core_mqtt.c:862] [INFO] [MQTT] Received
OtaJobEventActivate callback from OTA Agent.

   ... // Output removed for brevity

   2 39 [iot_thread] [INFO ][DEMO][390] ---------STARTING DEMO---------

   [0;32mI (3633) WIFI: WIFI_EVENT_STA_CONNECTED
   [0;32mI (4373) WIFI: SYSTEM_EVENT_STA_GOT_IP

   ... // Output removed for brevity

   4 351 [sys_evt] [INFO ][DEMO][3510] Connected to WiFi access point, ip address:
255.255.255.0.
   5 351 [iot_thread] [INFO ][DEMO][3510] Successfully initialized the demo. Network
type for the demo: 1
   6 351 [iot_thread] [ota_demo_core_mqtt.c:1902] [INFO] [MQTT] OTA over MQTT demo,
Application version 0.9.1
   7 351 [iot_thread] [ota_demo_core_mqtt.c:1323] [INFO] [MQTT] Creating a TLS
connection to <endpoint>-ats.iot.us-west-2.amazonaws.com:8883.
   9 718 [iot_thread] [core_mqtt.c:886] [INFO] [MQTT] Packet received.
ReceivedBytes=2.
   10 718 [iot_thread] [core_mqtt_serializer.c:970] [INFO] [MQTT] CONNACK session
present bit not set.
   11 718 [iot_thread] [core_mqtt_serializer.c:912] [INFO] [MQTT] Connection accepted.
```

```
    ... // Output removed for brevity

    17 736 [OTA Agent Task] [ota_demo_core_mqtt.c:1503] [INFO] [MQTT] SUBSCRIBED to
 topic $aws/things/__test_infra_thing71/jobs/notify-next to broker.
    18 737 [OTA Agent Task] [ota_mqtt.c:381] [INFO] [OTA] Subscribed to MQTT topic:
 $aws/things/__test_infra_thing71/jobs/notify-next
    30 818 [iot_thread] [ota_demo_core_mqtt.c:1850] [INFO] [MQTT]  Received: 0
 Queued: 0   Processed: 0   Dropped: 0
    31 819 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
 value]=[execution.jobId: AFR_OTA-9702f1a3-b747-4c3e-a0eb-a3b0cf83ddbb]
    32 820 [OTA Agent Task] [ota.c:1684] [INFO] [OTA] Extracted parameter: [key:
 value]=[execution.statusDetails.updatedBy: 589824]
    33 822 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
 value]=[execution.jobDocument.afr_ota.streamname: AFR_OTA-945d320b-a18b-441b-
b435-4a18d4e7671f]
    34 823 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
 value]=[execution.jobDocument.afr_ota.protocols: ["MQTT"]]
    35 824 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
 value]=[filepath: aws_demos.bin]
    36 825 [OTA Agent Task] [ota.c:1684] [INFO] [OTA] Extracted parameter: [key:
 value]=[filesize: 1164016]
    37 826 [OTA Agent Task] [ota.c:1684] [INFO] [OTA] Extracted parameter: [key:
 value]=[fileid: 0]
    38 827 [OTA Agent Task] [ota.c:1645] [INFO] [OTA] Extracted parameter: [key:
 value]=[certfile: ecdsa-sha256-signer.crt.pem]
    39 828 [OTA Agent Task] [ota.c:1575] [INFO] [OTA] Extracted parameter [ sig-sha256-
ecdsa: MEQCIE1SFkIHHiZAvkPpu6McJtx7SYoD... ]
    40 829 [OTA Agent Task] [ota.c:1684] [INFO] [OTA] Extracted parameter: [key:
 value]=[fileType: 0]
    41 830 [OTA Agent Task] [ota.c:2102] [INFO] [OTA] In self test mode.
    42 830 [OTA Agent Task] [ota.c:1936] [INFO] [OTA] New image has a higher version
 number than the current image: New image version=0.9.1, Previous image version=0.9.0
    43 832 [OTA Agent Task] [ota.c:2120] [INFO] [OTA] Image version is valid: Begin
 testing file: File ID=0
    53 896 [OTA Agent Task] [ota.c:794] [INFO] [OTA] Beginning self-test.
    62 971 [OTA Agent Task] [ota_demo_core_mqtt.c:1553] [INFO] [MQTT] Sent PUBLISH
 packet to broker $aws/things/__test_infra_thing71/jobs/AFR_OTA-9702f1a3-b747-4c3e-
a0eb-a3b0cf83ddbb/update to br63 971 [MQTT Agent Task] [core_mqtt.c:1045] [INFO] [MQTT]
 De-serialized incoming PUBLISH packet: DeserializerResult=MQTTSuccess.
    65 973 [MQTT Agent Task] [core_mqtt.c:1058] [INFO] [MQTT] State record updated. New
 state=MQTTPublishDone.
    64 973 [OTA Agent Task] [ota_demo_core_mqtt.c:902] [INFO] [MQTT] Successfully
 updated with the new image.
```

O configurazioniver-the-air demo

Le configurazioni demo OTA sono opzioni di configurazione specifiche 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 il completamento 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 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 aggiunto anche un ritardo casuale, fino a un massimo di questo ritardo base.

OTA_DEMO_CONN_RETRY_INTERVAL_MAX_SECONDI

L'intervallo massimo, in secondi, prima di riprovare la connessione di rete. Il ritardo di riconnessione viene raddoppiato ad ogni tentativo fallito, ma può arrivare solo fino a questo valore massimo, più un jitter dello stesso intervallo.

Scarica, crea, esegui il flashing ed esegui la demo OTA FreeRTOS su Texas Instruments CC3220SF-LAUNCHXL

> **Important**
>
> Questa integrazione di riferimento è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Consigliamo di iniziare da qui quando crei un nuovo progetto. Se disponi già di un progetto FreeRTOS esistente basato sull'ormai obsoleto repository Amazon-FreeRTOS, consulta ilGuida alla migrazione del repository Github di Amazon-FreeRTOS.

Per scaricare FreeRTOS e il codice demo OTA

- È possibile scaricare il codice sorgente dal GitHub sito all'indirizzo https://github.com/FreeRTOS/FreeRTOS.

Per creare l'applicazione demo

1. Seguire le istruzioni in Nosu FreeRTOS per importare il progetto `aws_demos` in Code Composer Studio, configurare l'endpoint AWS IoT, l'SSID Wi-Fi e la password, una chiave privata e un certificato per la scheda.
2. Apri*`freertos`*`/vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/aws_demo_config.h`, commenta`#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED` e definisci`CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED` o`CONFIG_OTA_HTTP_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 1000 [Tmr Svc] Simple Link task created
Device came up in Station mode
1 2534 [Tmr Svc] Write certificate...
2 5486 [Tmr Svc] [ERROR] Failed to destroy object. PKCS11_PAL_DestroyObject failed.
3 5486 [Tmr Svc] Write certificate...
4 5776 [Tmr Svc] Security alert threshold = 15
5 5776 [Tmr Svc] Current number of alerts = 1
6 5778 [Tmr Svc] Running Demos.
7 5779 [iot_thread] [INFO ][DEMO][5779] ---------STARTING DEMO---------
```

```
    8 5779 [iot_thread] [INFO ][INIT][5779] SDK successfully initialized.
    Device came up in Station mode
    [WLAN EVENT] STA Connected to the AP: afrlab-pepper , BSSID: 74:83:c2:b4:46:27
    [NETAPP EVENT] IP acquired by the device
    Device has connected to afrlab-pepper
    Device IP Address is 192.168.36.176
    9 8283 [iot_thread] [INFO ][DEMO][8282] Successfully initialized the demo. Network
 type for the demo: 1
    10 8283 [iot_thread] [INFO] OTA over MQTT demo, Application version 0.9.0
    11 8283 [iot_thread] [INFO] Creating a TLS connection to <endpoint>-ats.iot.us-
west-2.amazonaws.com:8883.
    12 8852 [iot_thread] [INFO] Creating an MQTT connection to <endpoint>-ats.iot.us-
west-2.amazonaws.com.
    13 8914 [iot_thread] [INFO] Packet received. ReceivedBytes=2.
    14 8914 [iot_thread] [INFO] CONNACK session present bit not set.
    15 8914 [iot_thread] [INFO] Connection accepted.
    16 8914 [iot_thread] [INFO] Received MQTT CONNACK successfully from broker.
    17 8914 [iot_thread] [INFO] MQTT connection established with the broker.
    18 8915 [iot_thread] [INFO]  Received: 0   Queued: 0   Processed: 0   Dropped: 0
    19 8953 [OTA Agent T] [INFO] Current State=[RequestingJob], Event=[Start], New
 state=[RequestingJob]
    20 9008 [MQTT Agent ] [INFO] Packet received. ReceivedBytes=3.
    21 9015 [OTA Agent T] [INFO] SUBSCRIBED to topic $aws/things/__test_infra_thing73/
jobs/notify-next to broker.
    22 9015 [OTA Agent T] [INFO] Subscribed to MQTT topic: $aws/things/
__test_infra_thing73/jobs/notify-next
    23 9504 [MQTT Agent ] [INFO] Publishing message to $aws/things/
__test_infra_thing73/jobs/$next/get.
    24 9535 [MQTT Agent ] [INFO] Packet received. ReceivedBytes=2.
    25 9535 [MQTT Agent ] [INFO] Ack packet deserialized with result: MQTTSuccess.
    26 9536 [MQTT Agent ] [INFO] State record updated. New state=MQTTPublishDone.
    27 9537 [OTA Agent T] [INFO] Sent PUBLISH packet to broker $aws/things/
__test_infra_thing73/jobs/$next/get to broker.
    28 9537 [OTA Agent T] [WARN] OTA Timer handle NULL for Timerid=0, can't stop.
    29 9537 [OTA Agent T] [INFO] Current State=[WaitingForJob],
 Event=[RequestJobDocument], New state=[WaitingForJob]
    30 9539 [MQTT Agent ] [INFO] Packet received. ReceivedBytes=120.
    31 9539 [MQTT Agent ] [INFO] De-serialized incoming PUBLISH packet:
 DeserializerResult=MQTTSuccess.
    32 9540 [MQTT Agent ] [INFO] State record updated. New state=MQTTPublishDone.
    33 9540 [MQTT Agent ] [INFO] Received job message callback, size 62.
    34 9616 [OTA Agent T] [INFO] Failed job document content check: Required job
 document parameter was not extracted: parameter=execution
```

```
   35 9616 [OTA Agent T] [INFO] Failed job document content check: Required job
document parameter was not extracted: parameter=execution.jobId
   36 9617 [OTA Agent T] [INFO] Failed job document content check: Required job
document parameter was not extracted: parameter=execution.jobDocument
   37 9617 [OTA Agent T] [INFO] Failed job document content check: Required job
document parameter was not extracted: parameter=execution.jobDocument.afr_ota
   38 9617 [OTA Agent T] [INFO] Failed job document content
check: Required job document parameter was not extracted:
parameter=execution.jobDocument.afr_ota.protocols
   39 9618 [OTA Agent T] [INFO] Failed job document content check: Required job
document parameter was not extracted: parameter=execution.jobDocument.afr_ota.files
   40 9618 [OTA Agent T] [INFO] Failed job document content check: Required job
document parameter was not extracted: parameter=filesize
   41 9618 [OTA Agent T] [INFO] Failed job document content check: Required job
document parameter was not extracted: parameter=fileid
   42 9619 [OTA Agent T] [INFO] Failed to parse JSON document as AFR_OTA job:
DocParseErr_t=7
   43 9619 [OTA Agent T] [INFO] No active job available in received job document:
OtaJobParseErr_t=OtaJobParseErrNoActiveJobs
   44 9619 [OTA Agent T] [ERROR] Failed to execute state transition handler: Handler
returned error: OtaErr_t=OtaErrJobParserError
   45 9620 [OTA Agent T] [INFO] Current State=[WaitingForJob],
Event=[ReceivedJobDocument], New state=[CreatingFile]
   46 9915 [iot_thread] [INFO]  Received: 0   Queued: 0   Processed: 0   Dropped: 0
   47 10915 [iot_thread] [INFO]  Received: 0   Queued: 0   Processed: 0   Dropped: 0
   48 11915 [iot_thread] [INFO]  Received: 0   Queued: 0   Processed: 0   Dropped: 0
   49 12915 [iot_thread] [INFO]  Received: 0   Queued: 0   Processed: 0   Dropped: 0
   50 13915 [iot_thread] [INFO]  Received: 0   Queued: 0   Processed: 0   Dropped: 0
   51 14915 [iot_thread] [INFO]  Received: 0   Queued: 0   Processed: 0   Dropped: 0
```

Scarica, crea, esegui il flashing ed esegui la demo OTA FreeRTOS su Microchip Curiosity PIC32MZEF

> **Important**
>
> Questa integrazione di riferimento è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Consigliamo di iniziare da qui quando crei un nuovo progetto. Se disponi già di un progetto FreeRTOS esistente basato sull'ormai obsoleto repository Amazon-FreeRTOS, consulta ilGuida alla migrazione del repository Github di Amazon-FreeRTOS.

> **Note**
>
> In accordo con Microchip, stiamo rimuovendo il Curiosity PIC32MZEF (DM320104) dal ramo principale del repository FreeRTOS Reference Integration e non lo includeremo più nelle nuove versioni. Microchip ha emesso un avviso ufficiale secondo cui il PIC32MZEF (DM320104) non è più consigliato per i nuovi progetti. È ancora possibile accedere ai progetti e al codice sorgente PIC32MZEF tramite i tag della versione precedente. Microchip consiglia ai clienti di utilizzare la scheda di sviluppo Curiosity PIC32MZ-EF-2.0 (DM320209) per nuovi progetti. La piattaforma PIC32mzv1 è ancora disponibile nella versione 202012.00 del repository FreeRTOS Reference Integration. Tuttavia, la piattaforma non è più supportata dalla versione 202107.00 del FreeRTOS Reference.

Per scaricare il codice demo OTA di FreeRTOS

- È possibile scaricare il codice sorgente dal GitHub sito all'indirizzo https://github.com/FreeRTOS/FreeRTOS.

Per creare l'applicazione demo dell'aggiornamento OTA

1. Seguire le istruzioni in Nosu FreeRTOS per importare il progetto `aws_demos` in MPLAB X IDE, configurare l'endpoint AWS IoT l'SSID e la password della rete Wi-Fi e una chiave privata e un certificato per la scheda.
2. Apri il`vendors/vendor/boards/board/aws_demos/config_files/ota_demo_config.h` file e inserisci il tuo certificato.

   ```
   [] = "your-certificate-key";
   ```
3. Incolla qui il contenuto del tuo certificato di firma del codice:

   ```
   #define otapalconfigCODE_SIGNING_CERTIFICATE [] = "your-certificate-key";
   ```

   Segui lo stesso formato di`aws_clientcredential_keys.h`: ogni riga deve terminare con il nuovo carattere di 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. Apri*`freertos`*`/vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/aws_demo_config.h`, commenta`#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED` e definisci`CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED` o`CONFIG_OTA_HTTP_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. Scollegare il debugger dalla scheda ed eseguire il progetto sulla scheda per accertarsi che è in grado di connettersi alla rete 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. Verificare che la scheda ICD4 sia selezionata. Dovrebbe essere visualizzato l'output riportato di seguito.

```
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 la scheda Microchip Curiosity PIC32MZEF riesce a connettersi a AWS IoT e a 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 integrazione di riferimento è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Consigliamo di iniziare da qui quando crei un nuovo progetto. Se disponi già di

un progetto FreeRTOS esistente basato sull'ormai obsoleto repository Amazon-FreeRTOS, consulta ilGuida alla migrazione del repository Github di Amazon-FreeRTOS.

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. Apri*freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h, commenta#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED e definisciCONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED oCONFIG_OTA_HTTP_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:
   - Apri*freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h, commenta#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED e definisciCONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED oCONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED.
   - Aprivendors/*vendor*/boards/*board*/aws_demos/config_files/ota_demo_config.h e copia il tuo certificato di firma del codice SHA-256/ECDSA in:

     ```
     #define otapalconfigCODE_SIGNING_CERTIFICATE [] = "your-certificate-key";
     ```

Scarica, crea, esegui il flashing ed esegui la demo OTA di FreeRTOS sul Renesas RX65N

> **Important**
>
> Questa integrazione di riferimento è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Consigliamo di iniziare da qui quando si crea un nuovo progetto. Se disponi già di

un progetto FreeRTOS esistente basato sull'ormai obsoleto repository Amazon-FreeRTOS, consulta ilGuida alla migrazione del repository Github di Amazon-FreeRTOS.

Questo capitolo mostra come scaricare, creare, eseguire il flashing ed eseguire le applicazioni demo OTA FreeRTOS sul Renesas RX65N.

Argomenti

- Configurazione dell'ambiente operativo
- Configura le tueAWS risorse
- Importa, configura il file di intestazione e crea aws_demos e boot_loader

Configurazione dell'ambiente operativo

Le procedure in questa sezione utilizzano gli ambienti seguenti:

- IDE: e2 studio 7.8.0, e 2 studio 2020-07
- Catene di strumenti: compilatore CCRX v3.0.1
- Dispositivi di destinazione: RSKRX65N-2 MB
- Debugger: emulatore E 2, E 2 Lite
- Software: Programmatore Flash Renesas, Renesas Secure Flash Programmer.exe, Tera Term

Per configurare l'hardware

1. Collega l'emulatore E 2 Lite e la porta seriale USB alla scheda RX65N e al PC.
2. Connect la fonte di alimentazione all'RX65N.

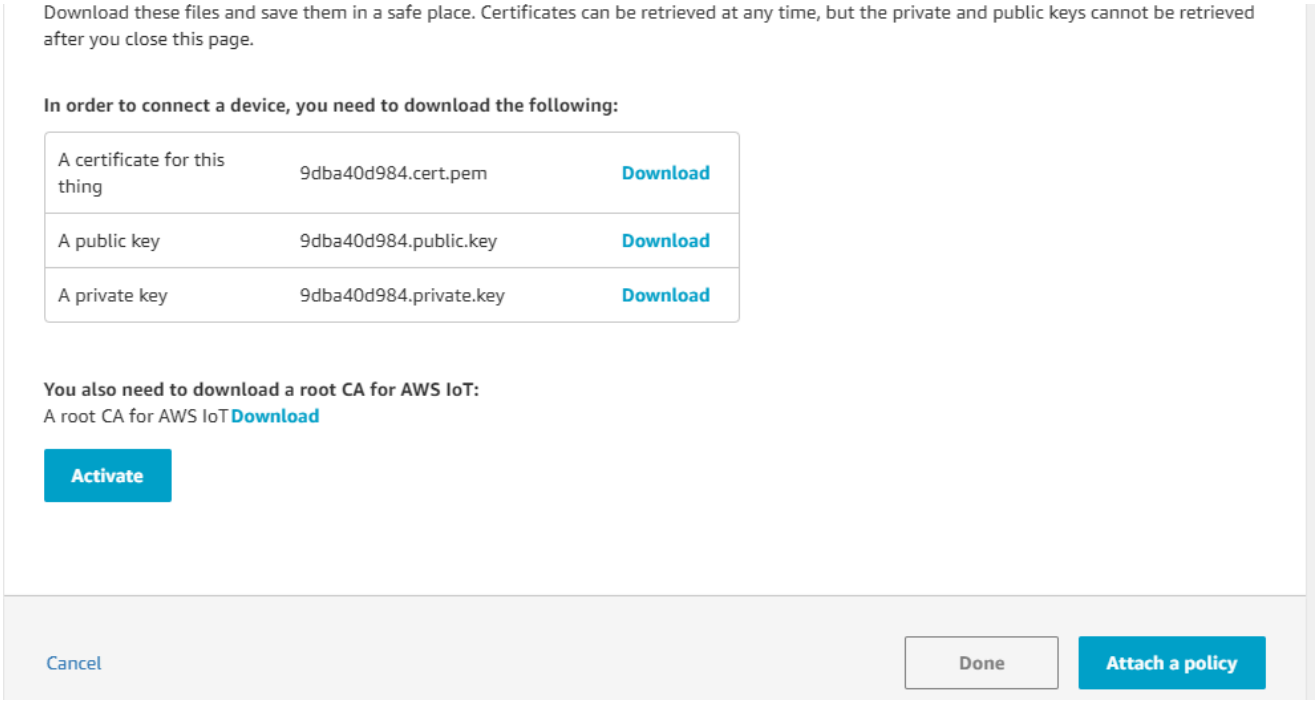
Configura le tueAWS risorse

1. Per eseguire le demo di FreeRTOS, è necessario disporre di unAWS account con un utente IAM autorizzato ad accedere aiAWS IoT servizi. Se non lo hai già fatto, segui la procedura inAWSConfigurazione dell'account e delle autorizzazioni.
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. Aprire la console AWS IoT.
4. Nel pannello di navigazione sinistro, scegli Gestisci, quindi scegli Oggetti per creare un oggetto.

   Una cosa è una rappresentazione di un dispositivo o di un'entità logica inAWS IoT. Può trattarsi di un dispositivo fisico o un sensore, ad esempio una lampadina o un interruttore su un muro. Può anche trattarsi di un'entità logica ad esempio un'istanza di un'applicazioneAWS IoT, o un'entità fisica che non si connette ad ma che è correlata ai dispositivi che si connettono (ad esempio un'auto con sensori per il motore o un pannello di controllo). AWS IoTfornisce un registro degli oggetti che ti aiuta con la gestione degli oggetti.

   a. Scegli Crea, quindi scegli Crea una singola cosa.
   b. Inserisci un nome per la tua cosa, quindi scegli Avanti.
   c. Scegli Crea certificato.
   d. Scarica i tre file creati e scegli Attiva.
   e. Scegliere Attach a policy (Collega policy).



   f. Selezionare la policy creata inPolicy dei dispositivi.

      Ogni dispositivo che riceve un aggiornamento OTA tramite MQTT deve essere registrato come oggettoAWS 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 autorizzazioni `iot:Connect` consentono al dispositivo di connettersi a AWS IoT tramite MQTT.
- 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 autorizzazioni `iot:Publish` e `iot:Subscribe` sugli argomenti dei flussi OTA AWS IoT (`.../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 autorizzazione `iot:Receive` consentono a AWS IoT Core di pubblicare messaggi su tali argomenti per il 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 suAWS.

   a. Per creare le chiavi e la certificazione, vedere la sezione 7.3 «Generazione di coppie di chiavi ECDSA-SHA256 con OpenSSL» nella politica di progettazione dell'aggiornamento del firmware MCU di Renesas.

   b. Aprire la console AWS IoT. Nel pannello di navigazione sinistro selezionare Gestisci, quindi Lavori. Seleziona Crea un Job, quindi Crea processo di aggiornamento OTA.

   c. In Seleziona dispositivi da aggiornare scegli Seleziona, quindi scegli l'elemento che hai creato in precedenza. Selezionare Next (Successivo).

   d. Nella pagina Crea un processo di aggiornamento OTA per FreeRTOS completa le seguenti operazioni:

      i. In 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 di firma del codice, scegli Crea.

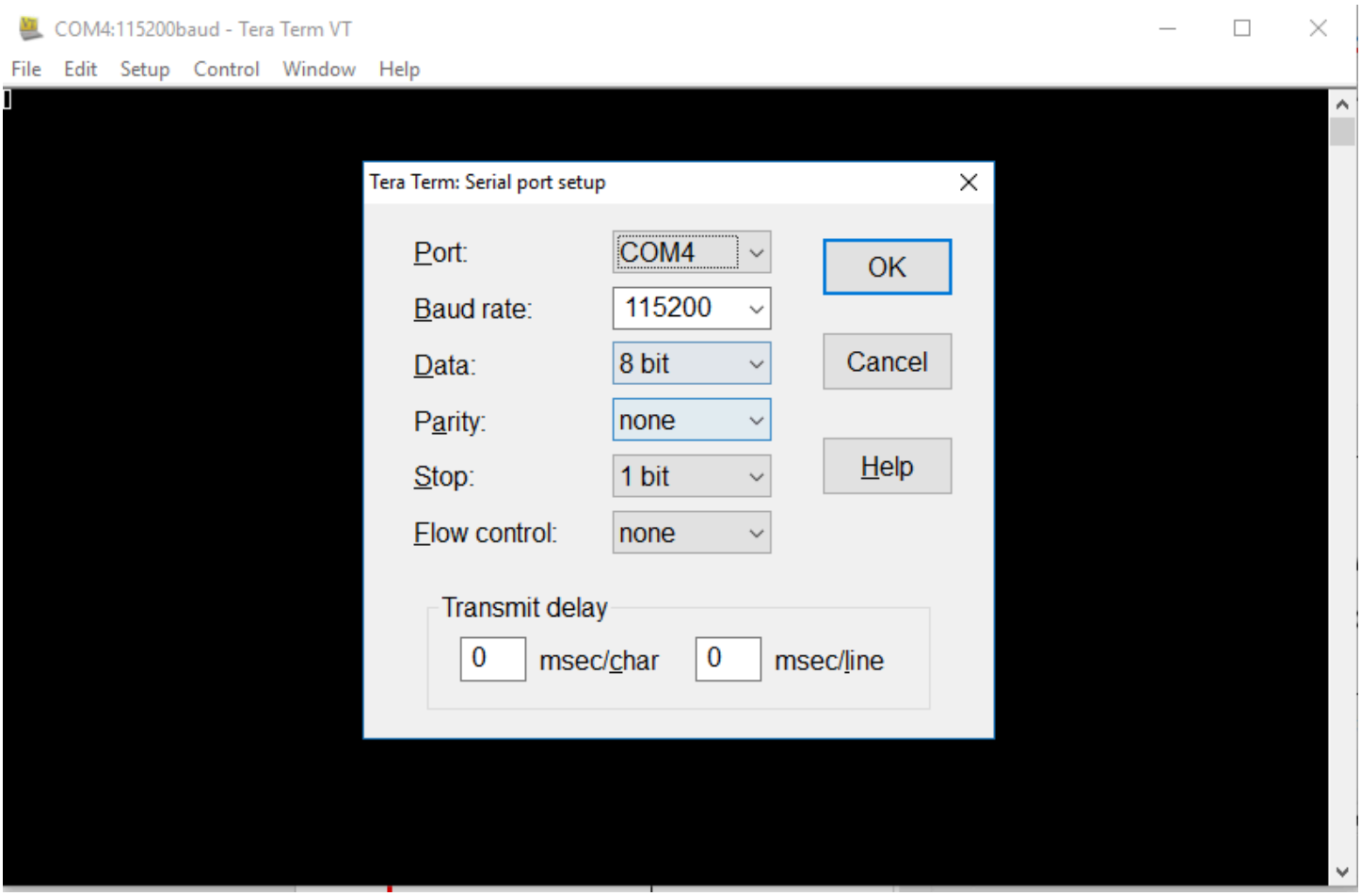
      iv. Nella finestra Crea un profilo di firma con codice, inserisci un nome per il profilo. Per la piattaforma hardware del dispositivo, seleziona Windows Simulator. Per il certificato di firma del codice, scegli Importa.

v. Sfoglia per selezionare il certificato (`secp256r1.crt`), la chiave privata del certificato (`secp256r1.key`) e la catena di certificati (`ca.crt`).

vi. Inserisci il percorso del certificato di firma del codice sul dispositivo. Quindi, scegli Create (Crea).

6. Per concedere l'accesso al codice di firma perAWS IoT, segui i passaggi indicatiConcessione dell'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.

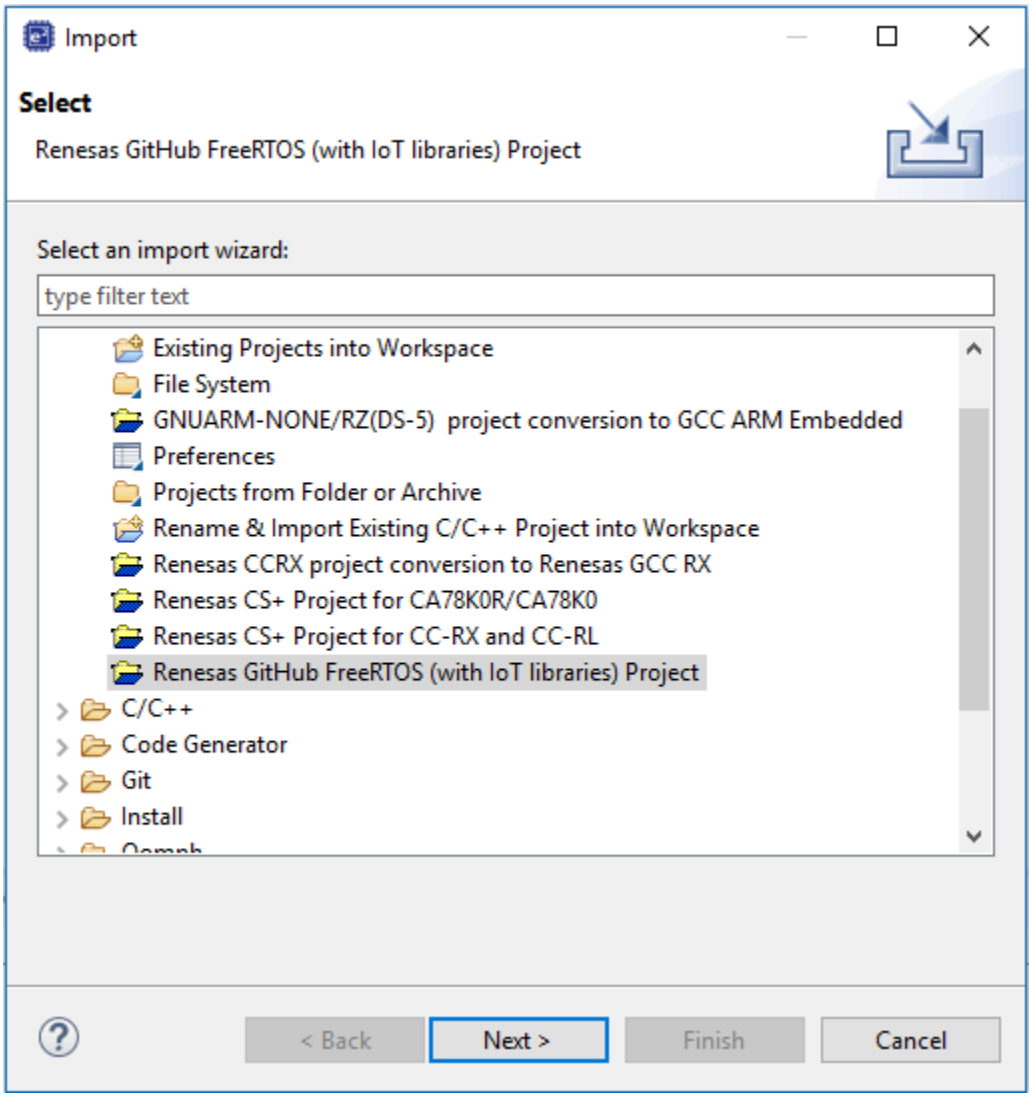


Importa, configura il file di intestazione e crea aws_demos e boot_loader

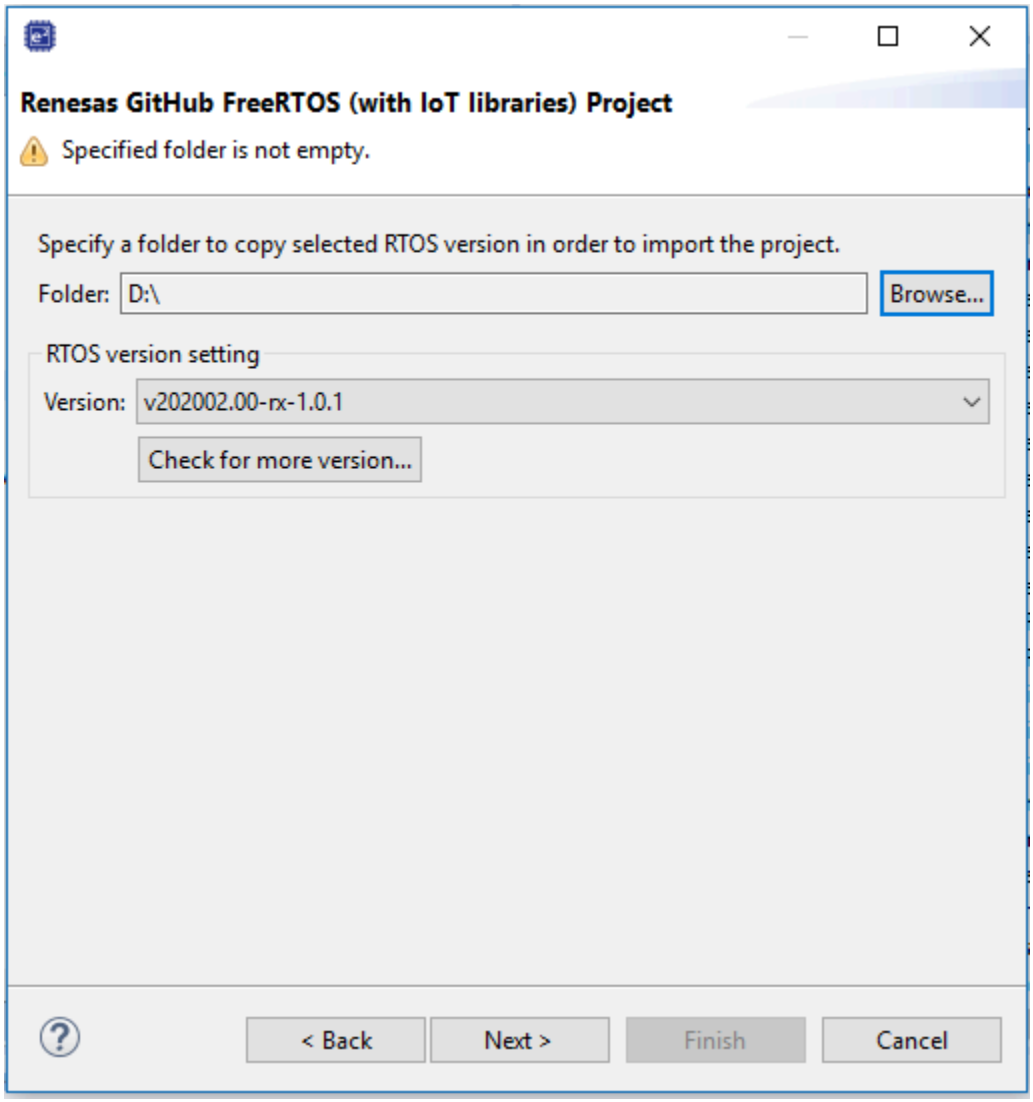
Per iniziare, si seleziona l'ultima versione del pacchetto FreeRTOS, che verrà scaricata GitHub e importata automaticamente nel progetto. In questo modo puoi concentrarti sulla configurazione di FreeRTOS e sulla scrittura del codice dell'applicazione.

1. Avvia e ^{2} studio.

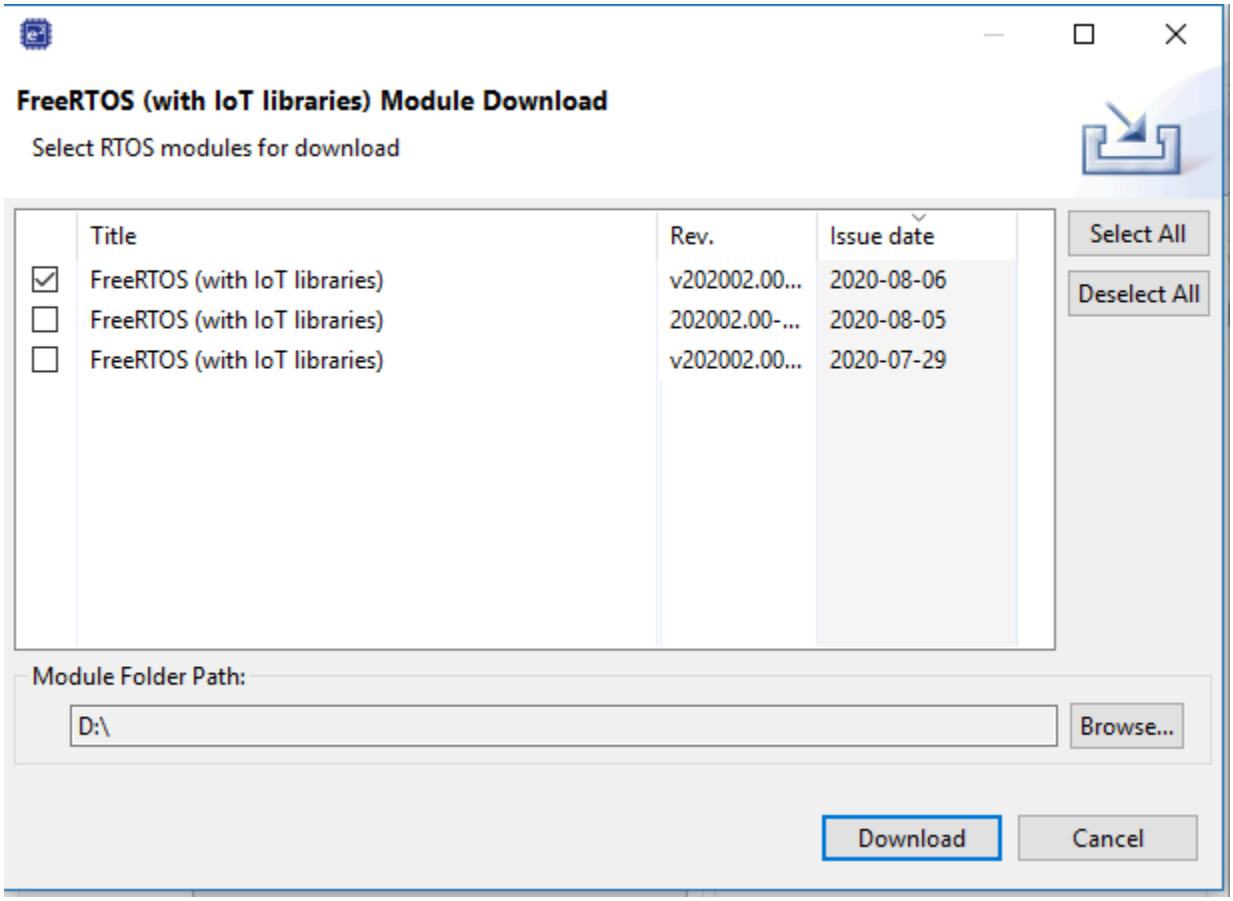
2. Scegli File, quindi scegli Importa...
3. Seleziona il progetto Renesas GitHub FreeRTOS (con librerie IoT).



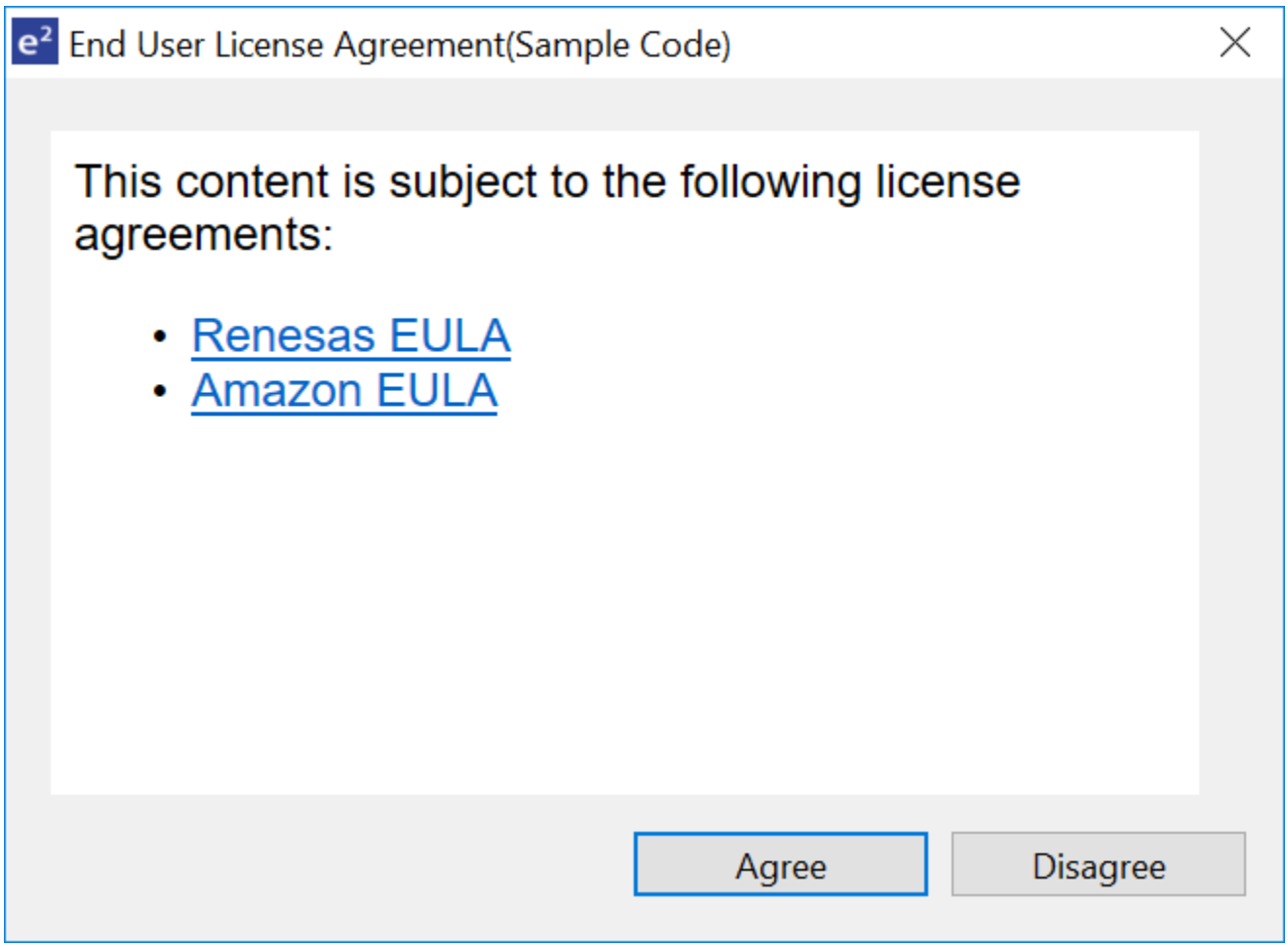
4. Scegli Verifica altre versioni... per mostrare la finestra di dialogo per il download.



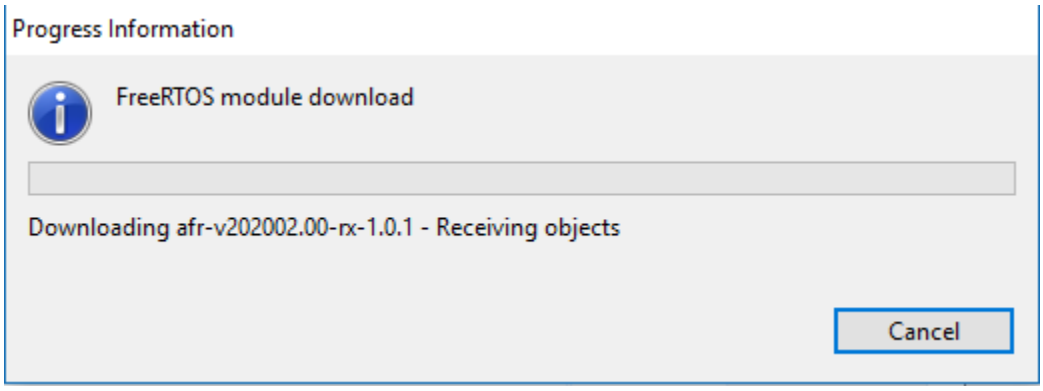
5. Seleziona il pacchetto più recente.



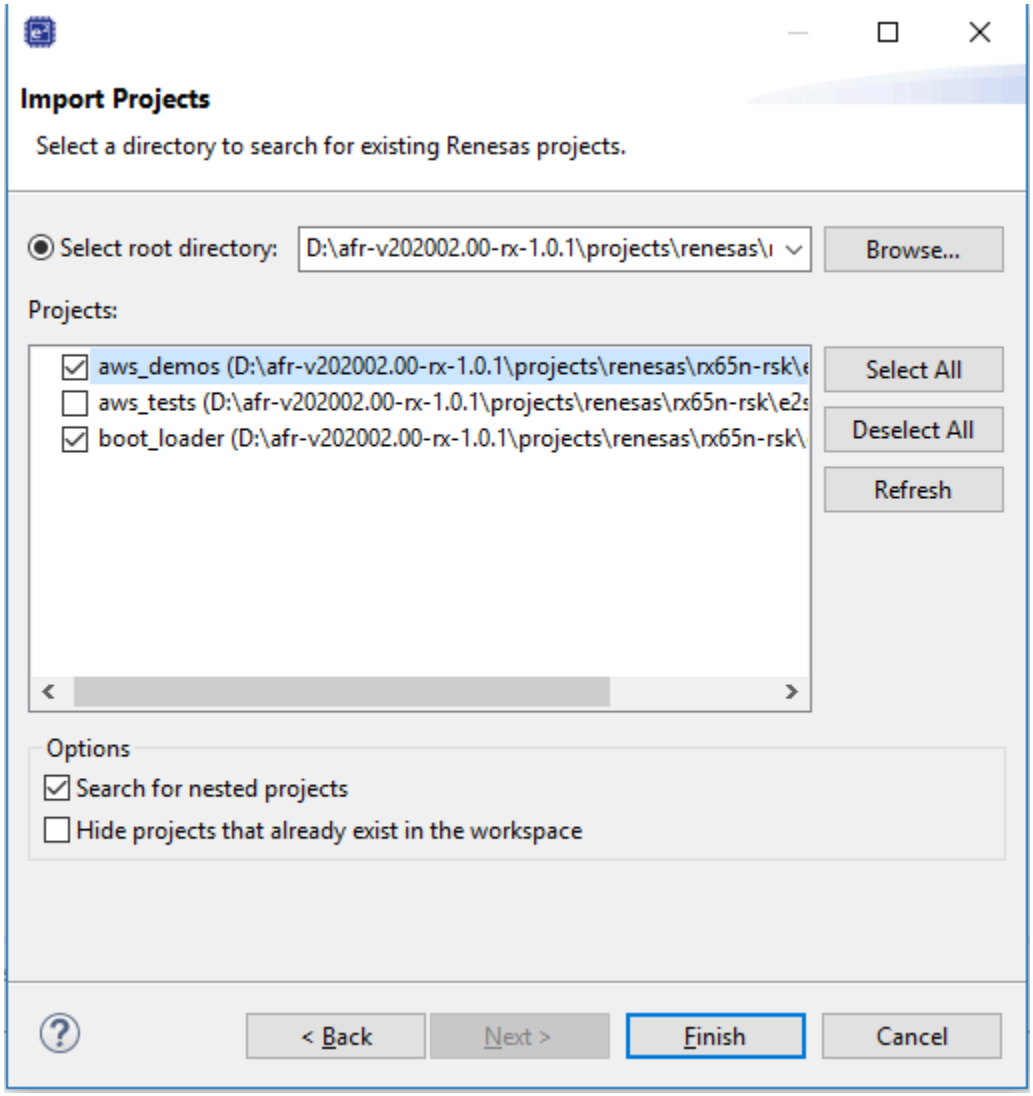
6. Scegli Accetto per accettare il contratto di licenza per l'utente finale.



7. Attendi il completamento del download.



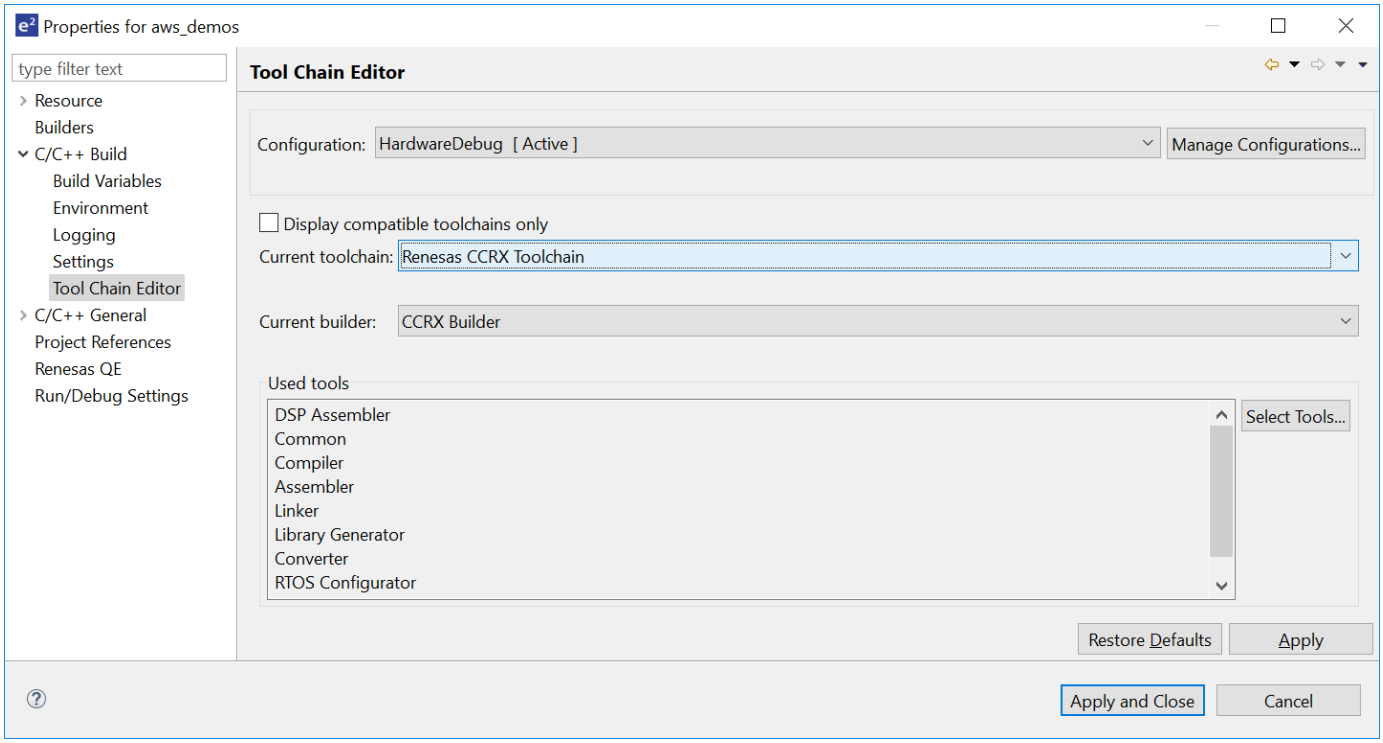
8. Selezionate i progetti aws_demos e boot_loader, quindi scegliete Finish per importarli.



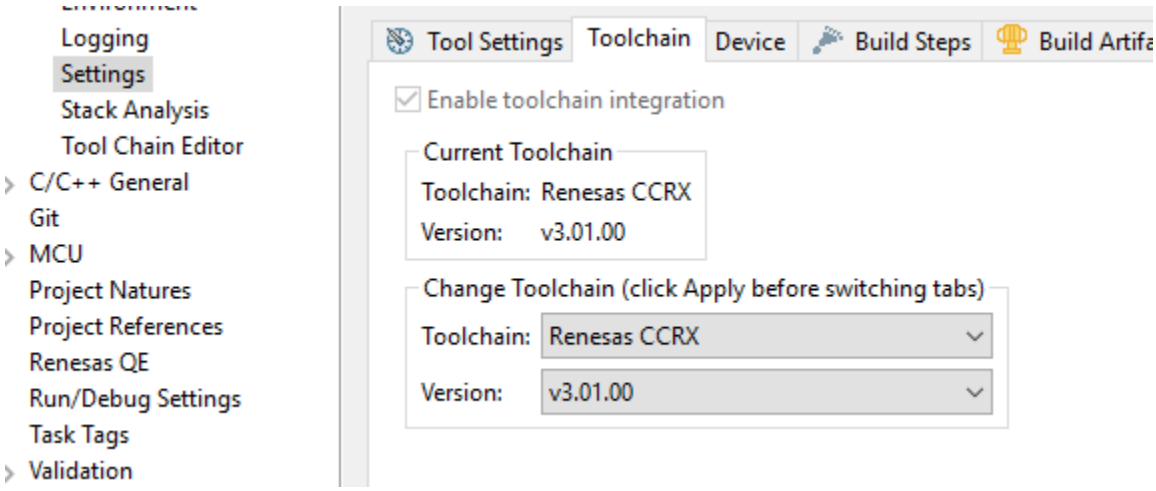
9. Per entrambi i progetti, apri le proprietà del progetto. Nel pannello di navigazione, scegli Editor della catena degli strumenti.

    a. Scegliete la toolchain attuale.

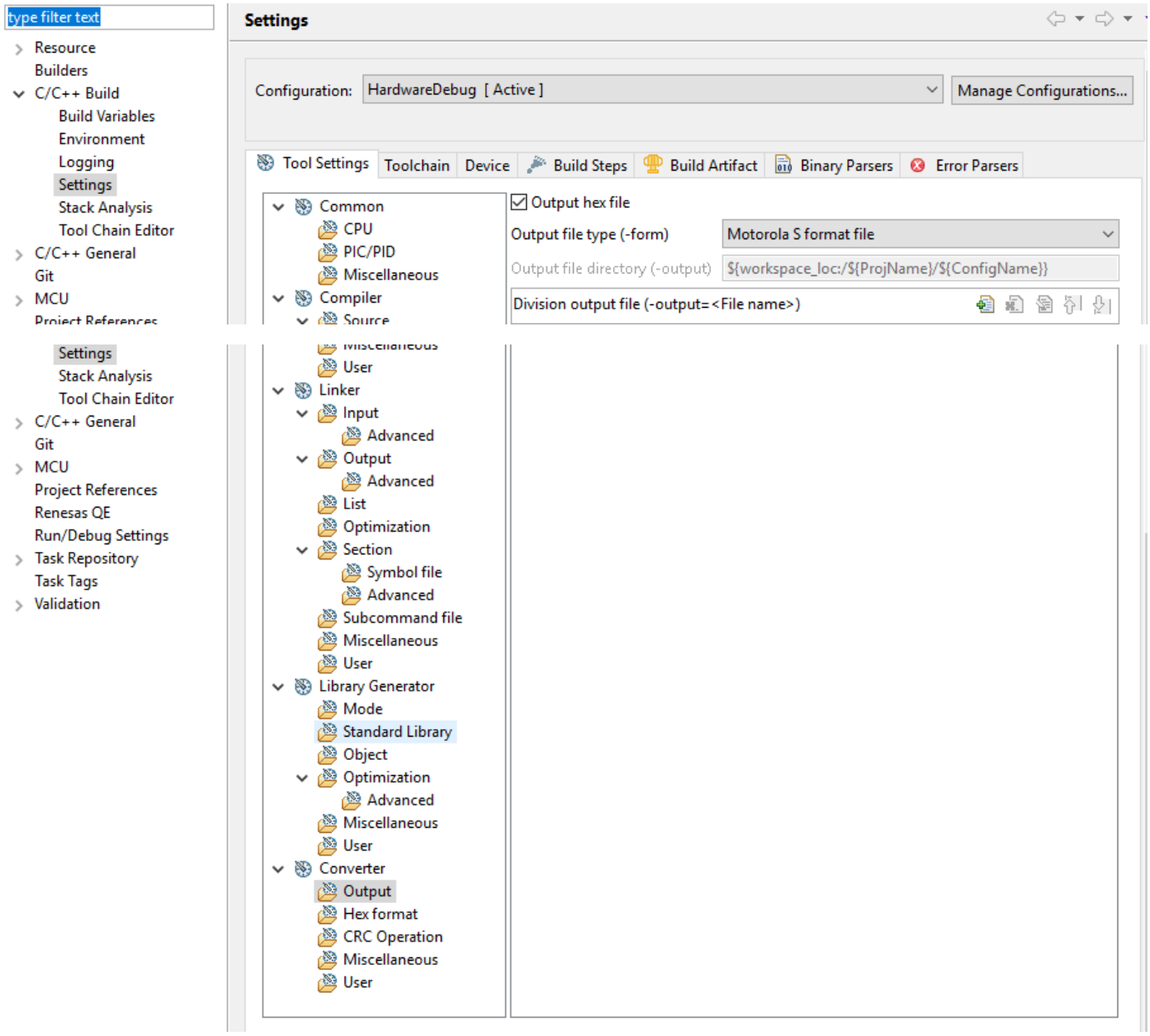
    b. Scegli il generatore attuale.



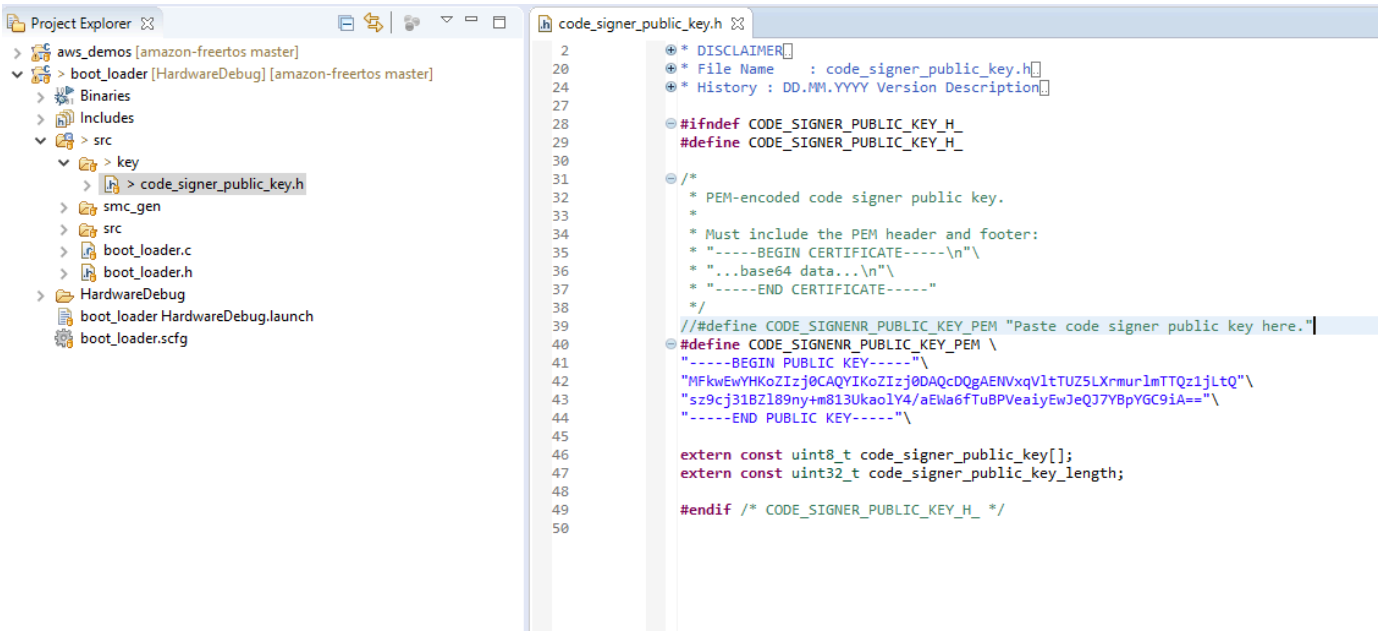
10. Nel pannello di navigazione scegli Settings (Impostazioni). Scegliete la scheda Toolchain, quindi scegliete la versione della toolchain.



Scegli la scheda Impostazioni dello strumento, espandi Convertitore e quindi scegli Output. Nella finestra principale, assicurati che sia selezionato il file esadecimale di output, 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, vedi Come implementare FreeRTOS OTA utilizzando Amazon Web Services su RX65N e la sezione 7.3 «Generazione di coppie di chiavi ECDSA-SHA256 con OpenSSL» nella politica di progettazione degli aggiornamenti del firmware MCU di Renesas.



Quindi crea il progetto da creare`boot_loader.mot`.

12. Apri il`aws_demos` progetto.

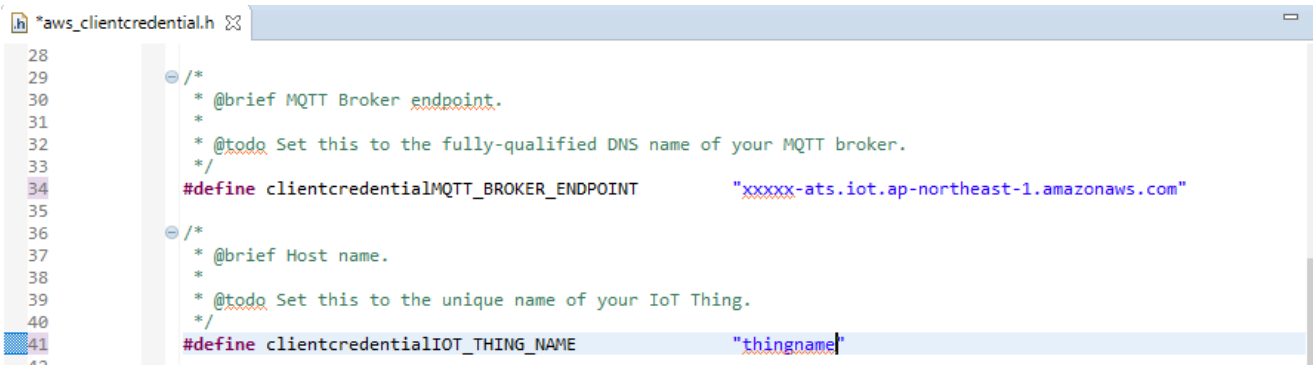
    a. Aprire la console AWS IoT.

    b. Nel riquadro di navigazione a sinistra scegliere Impostazioni. Prendi nota del tuo endpoint personalizzato nella casella di testo Device data endpoint.

    c. Scegli Gestisci, quindi scegli Cose. Prendi nota del nome dell'AWS IoToggetto nella tua bacheca.

    d. Nel`aws_demos` progetto, apri`demos/include/aws_clientcredential.h` e specificate i seguenti valori.

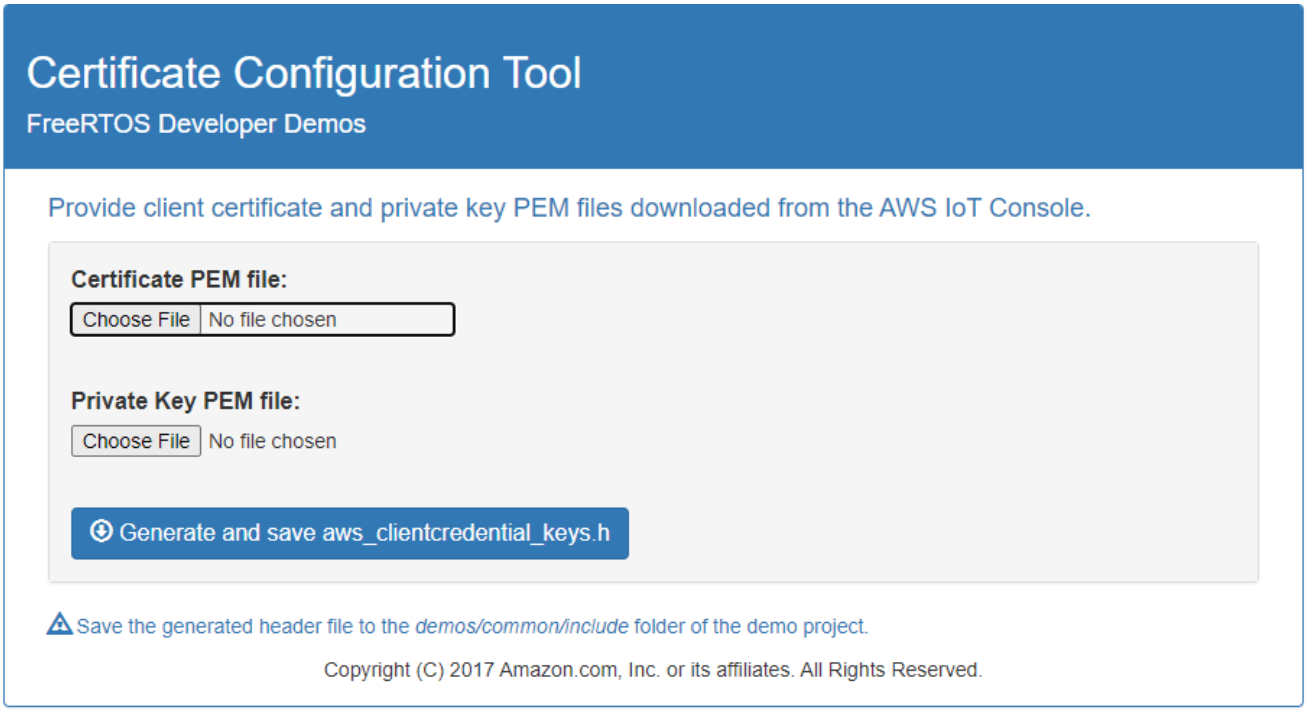
```
#define clientcredentialMQTT_BROKER_ENDPOINT[] = "Your AWS IoT endpoint";
#define clientcredentialIOT_THING_NAME "The AWS IoT thing name of your board"
```



e. Apri il file `tools/certificate_configuration/CertificateConfigurator.html`.

f. Importa il file PEM del certificato e il file PEM con chiave privata che hai scaricato in precedenza.

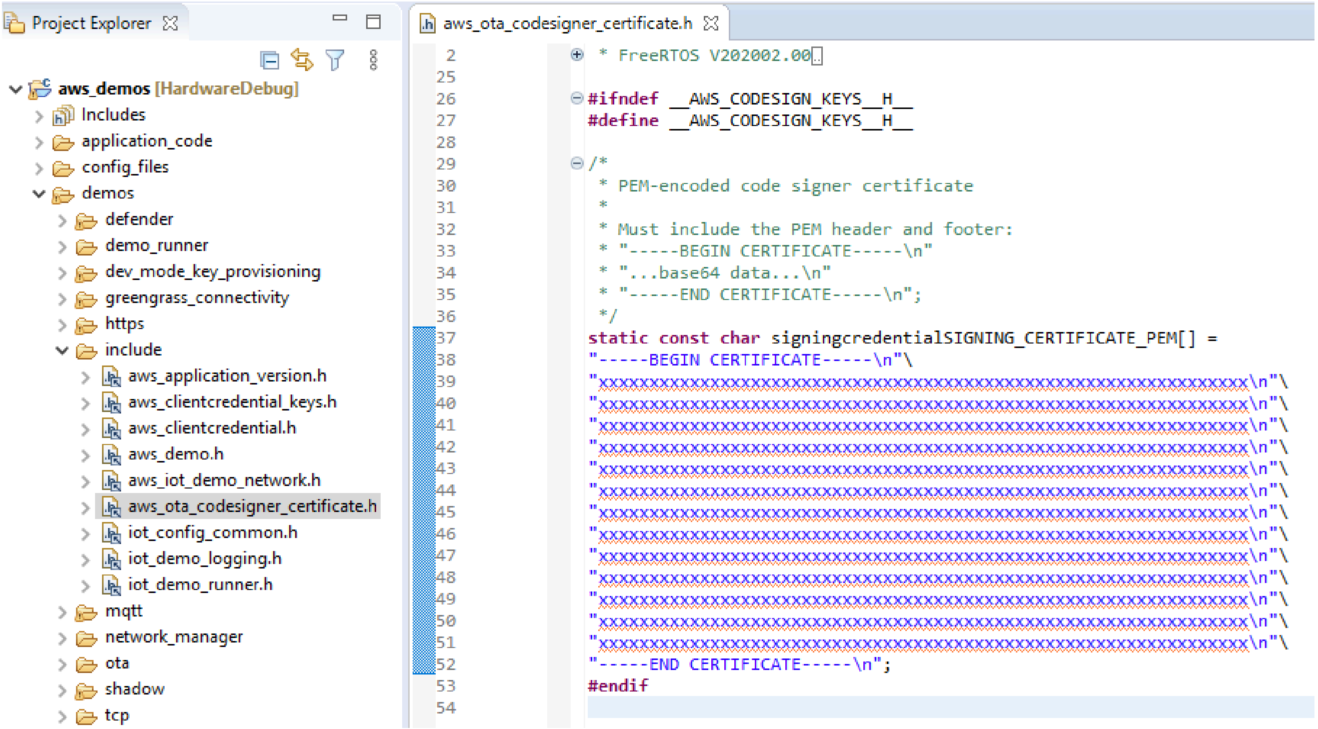
g. Scegli Genera e salva aws_clientcredential_keys.h e sostituisci questo file nella`demos/include/` directory.



h. Aprite il`vendors/renesas/boards/rx65n-rsk/aws_demos/config_files/ota_demo_config.h` file e specificate questi valori.

```
#define otapalconfigCODE_SIGNING_CERTIFICATE [] = "your-certificate-key";
```

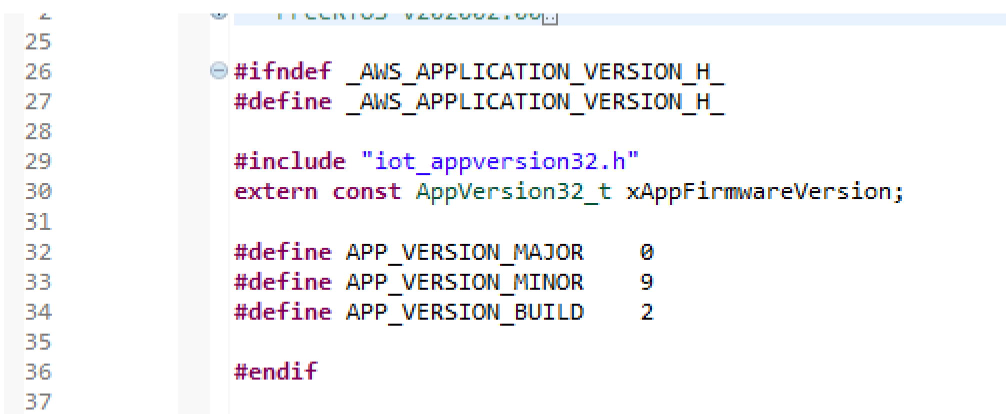
*your-certificate-key*Dov'è il valore del file`secp256r1.crt`. Ricordati di aggiungere «\» dopo ogni riga della certificazione. Per ulteriori informazioni sulla creazione del`secp256r1.crt` file, vedi Come implementare FreeRTOS OTA utilizzando Amazon Web Services su RX65N e la sezione 7.3 «Generazione di coppie di chiavi ECDSA-SHA256 con OpenSSL» nella politica di progettazione degli aggiornamenti del firmware MCU di Renesas.



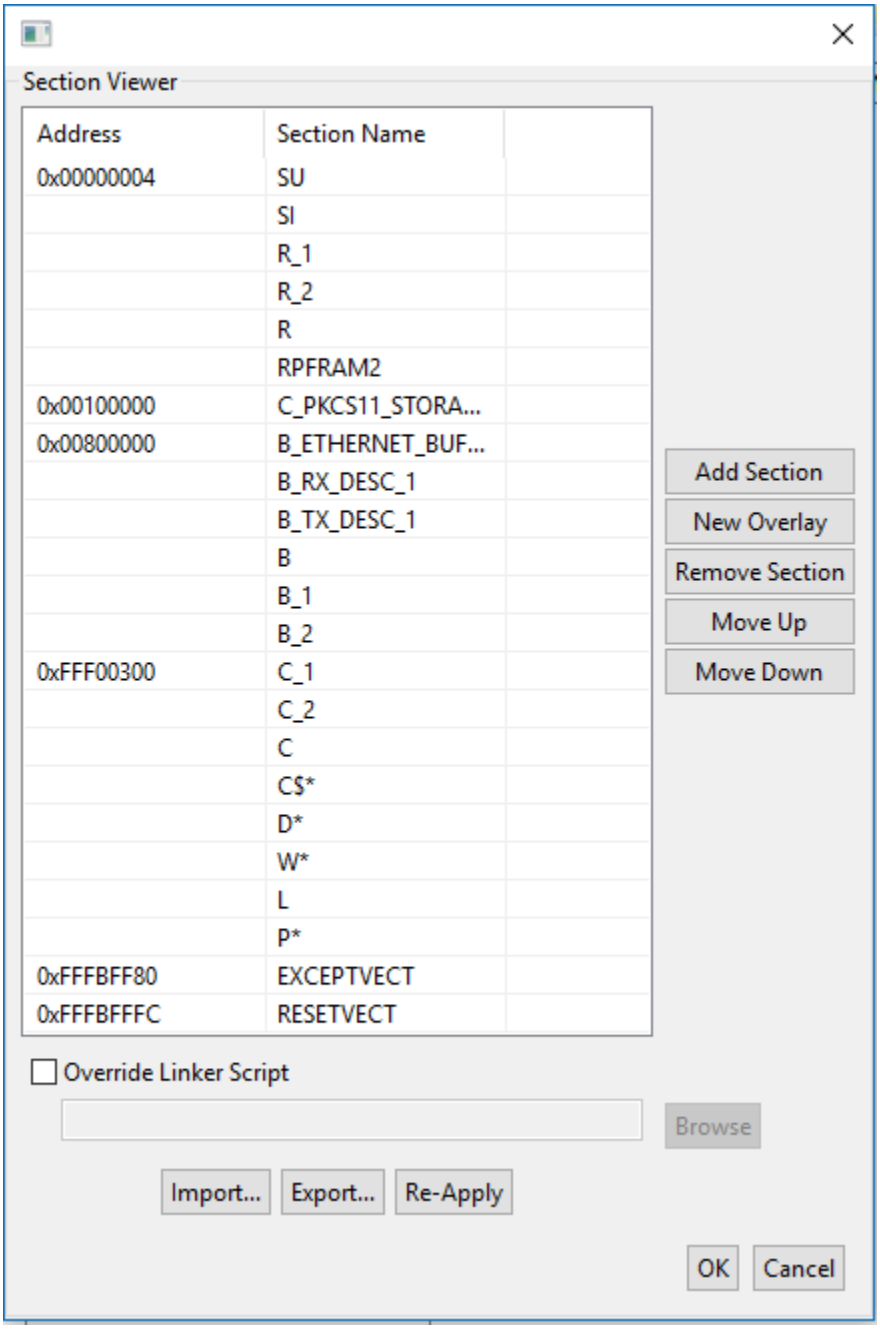
13. Attività A: installare la versione iniziale del firmware

   a. Apri il`vendors/renesas/boards/board/aws_demos/config_files/aws_demo_config.h` file, commenta`#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED` e definisci`CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED` o`CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED`.

   b. Apri il`demos/include/ aws_application_version.h` file e imposta la versione iniziale del firmware su`0.9.2`.



   c. Modificate le seguenti impostazioni nel Visualizzatore di sezioni.



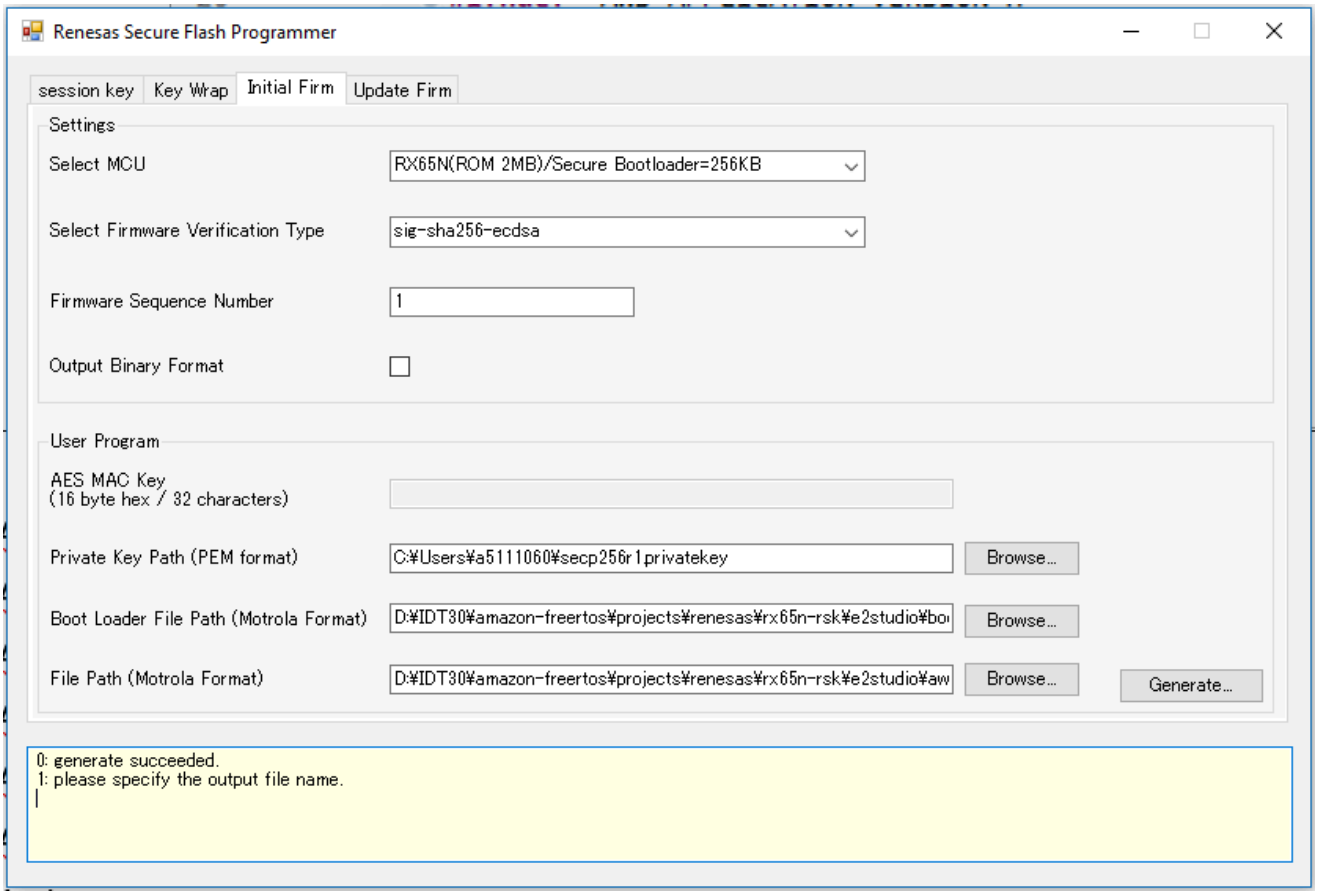
d. Scegliete Costruisci per creare il`aws_demos.mot` file.

14. Crea il file`userprog.mot` con Renesas Secure Flash Programmer. `userprog.mot`è una combinazione di`aws_demos.mot` e`boot_loader.mot`. È possibile eseguire il flashing di questo file sull'RX65N-RSK per installare il firmware iniziale.

    a. Scarica https://github.com/renesas/Amazon-FreeRTOS-Tools e apri`Renesas Secure Flash Programmer.exe`.

    b. Scegliete la scheda Initial Firm e impostate i seguenti parametri:

- Percorso chiave privata: la posizione di`secp256r1.privatekey`.
- Percorso del file Boot Loader: 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`).



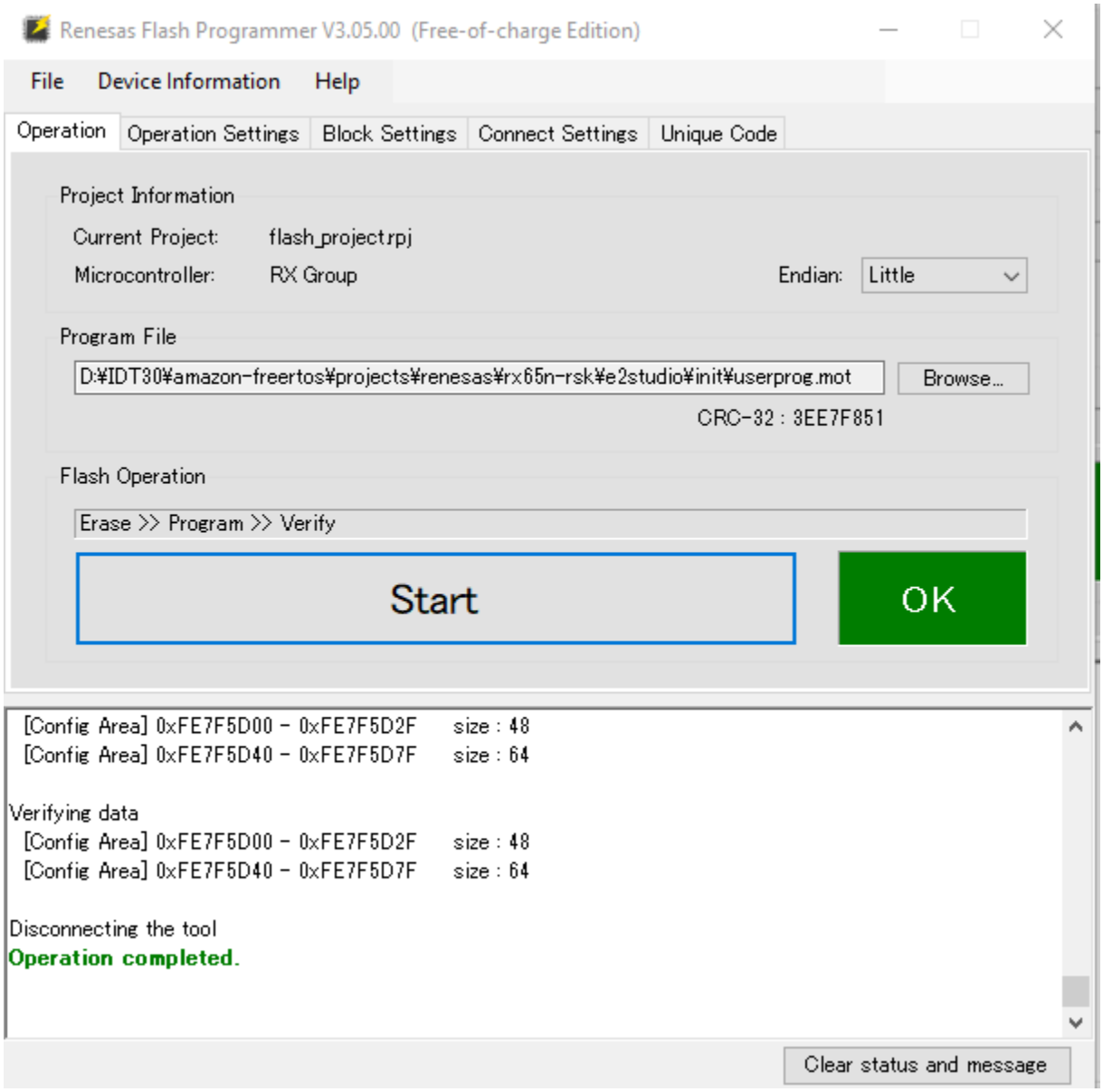
c. Crea una directory`init_firmware` denominata Generare`userprog.mot` e salvala nella`init_firmware` directory. Verifica che la generazione abbia avuto esito positivo.

15. Installare il firmware iniziale sull'RX65N-RSK.

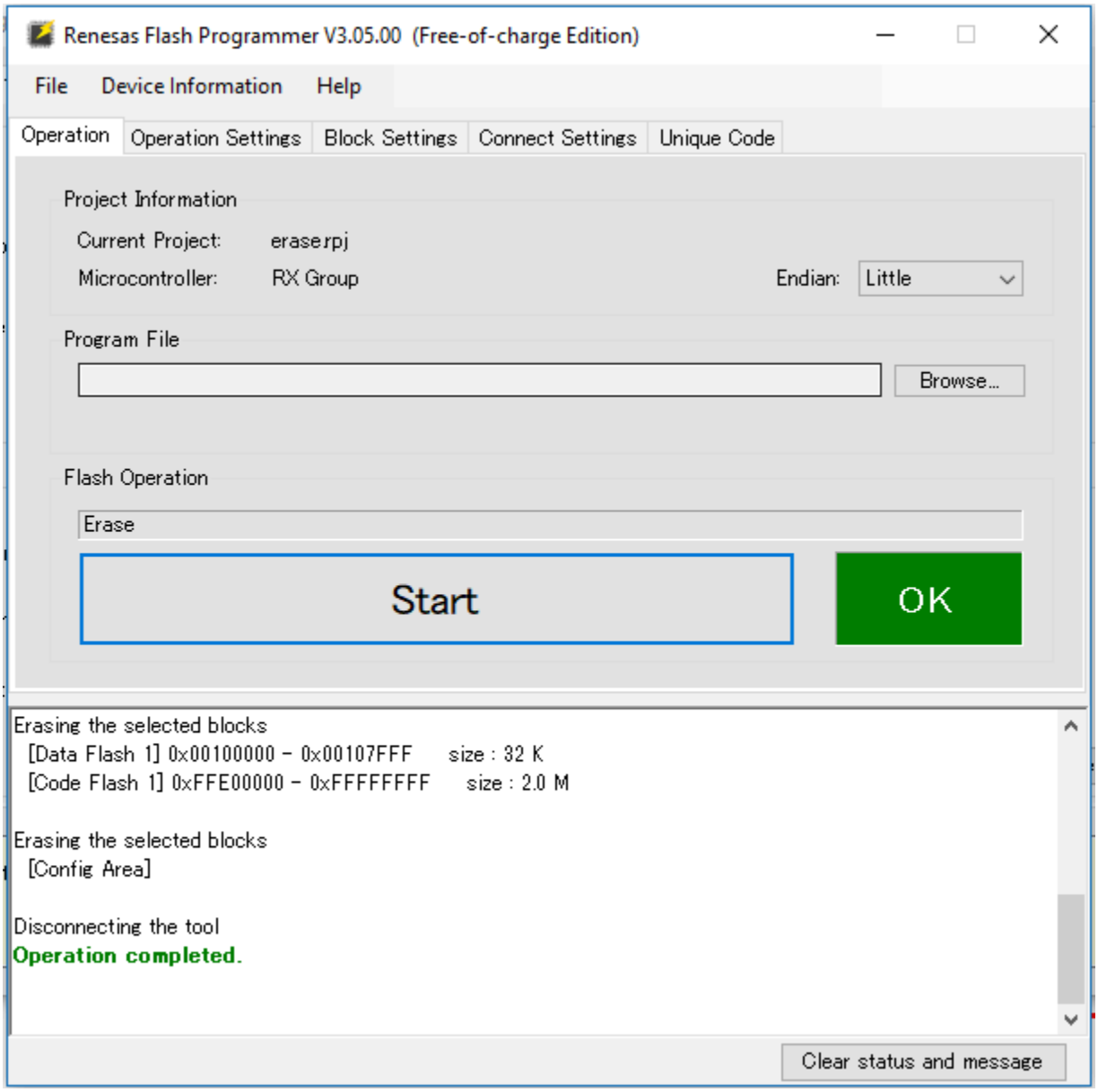
a. Scarica l'ultima versione di Renesas Flash Programmer (GUI di programmazione) da https://www.renesas.com/tw/en/products/software-tools/tools/programmer/renesas-flash-programmer-programming -gui.html.

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 il banco.



d. Per flashareuserprog.mot, scegli Sfoglia... e accedi allainit_firmware directory, seleziona iluserprog.mot file e scegli Avvia.



16. La versione 0.9.2 (versione iniziale) del firmware è stata installata sull'RX65N-RSK. La scheda RX65N-RSK è ora in attesa di aggiornamenti OTA. Se hai aperto Tera Term sul tuo PC, vedrai qualcosa di simile al seguente quando viene eseguito il firmware iniziale.

```
------------------------------------------------
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: Aggiornare la versione del firmware

    a. Apri il`demos/include/aws_application_version.h` file e incrementa il valore del`APP_VERSION_BUILD` token in`0.9.3`.

b. Ricreare il progetto.

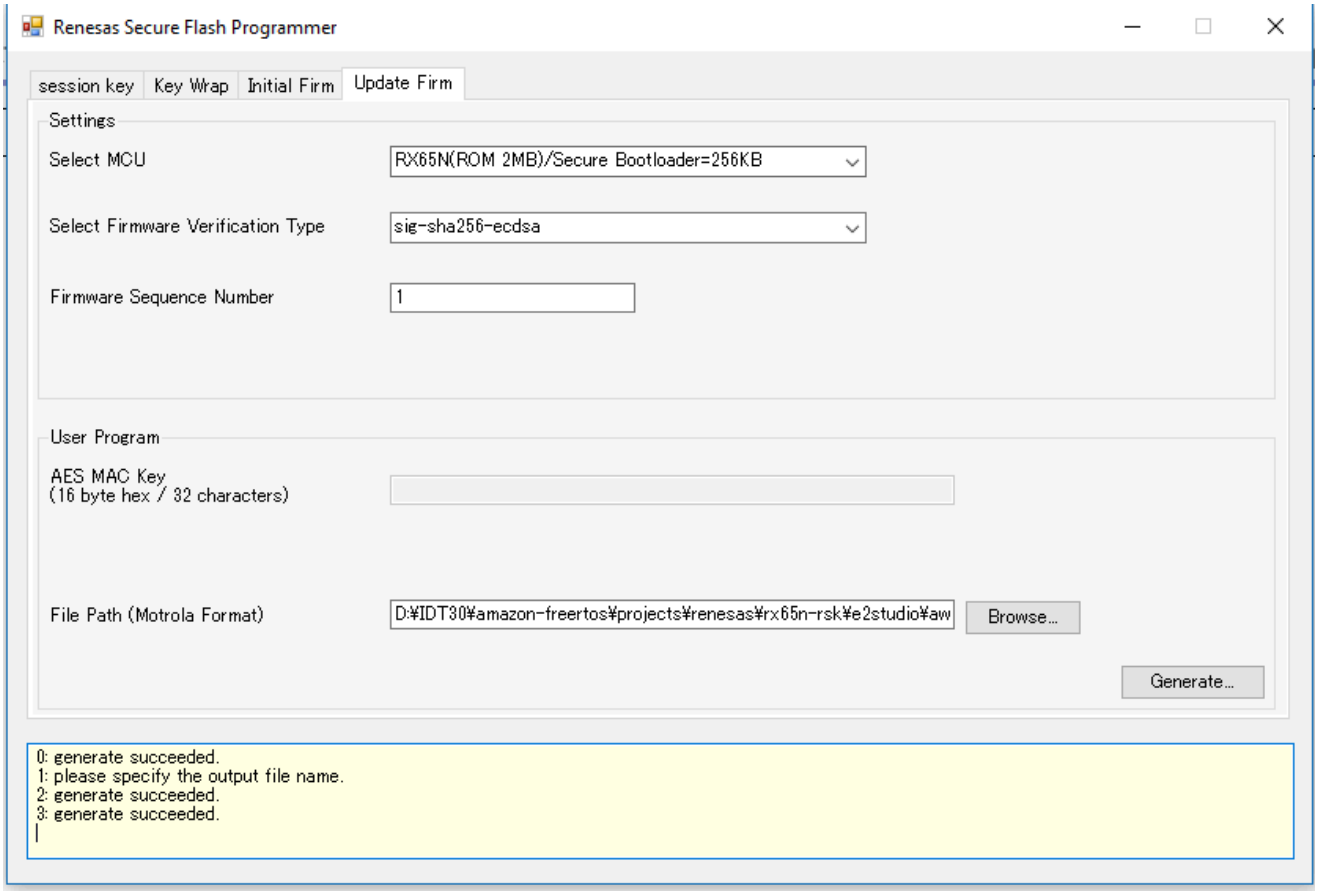
18. Crea il`userprog.rsu` file con Renesas Secure Flash Programmer per aggiornare la versione del tuo firmware.

    a. Apri il file `Amazon-FreeRTOS-Tools\Renesas Secure Flash Programmer.exe`.

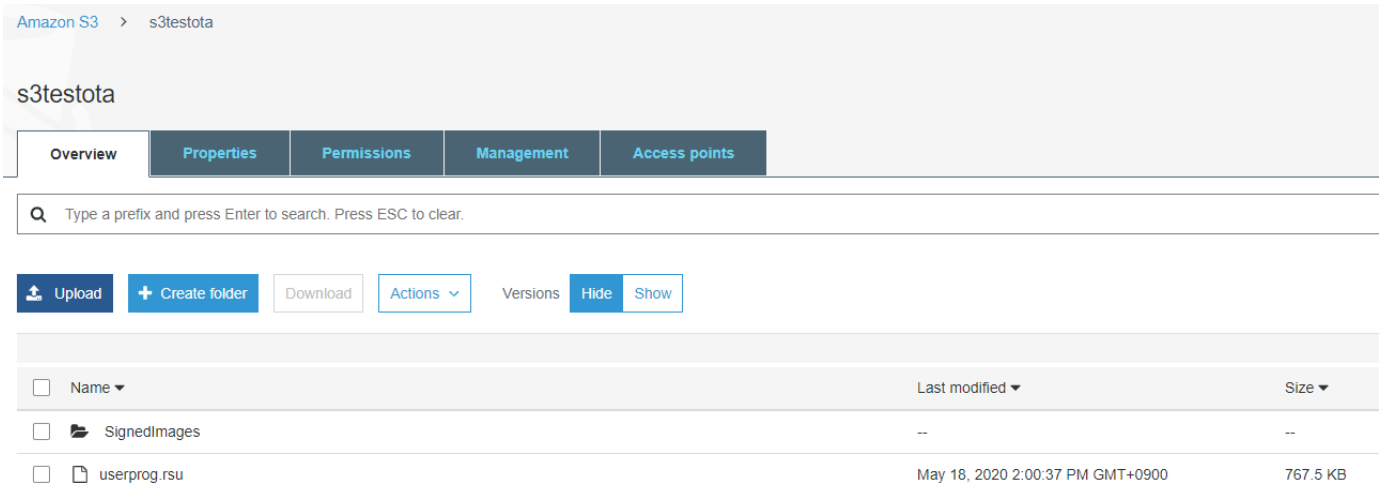
    b. Scegli la scheda Aggiorna azienda e imposta i seguenti parametri:

    - 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`. Generale`userprog.rsu` e salvalo nella`update_firmware` directory. Verifica che la generazione abbia avuto esito positivo.



19. Carica l'aggiornamento del firmware in un bucket Amazon S3 come descritto in[Crea un bucket Amazon S3 per archiviare l'aggiornamento](#).`userproj.rsu`



20. Creare un job per aggiornare il firmware sull'RX65N-RSK.

    AWS IoTJobs 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 diagnostica.

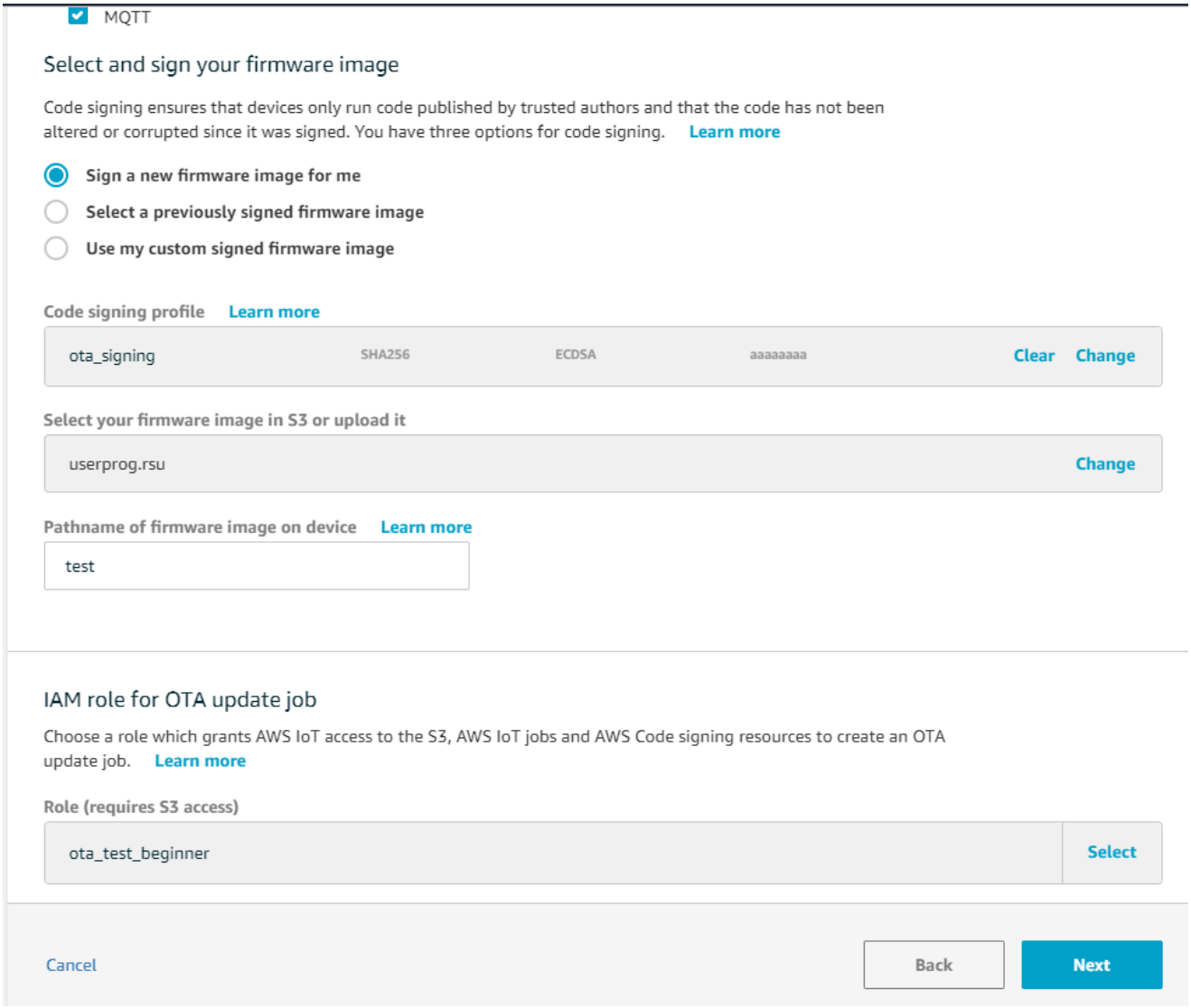
    a. Accedi alla console AWS IoT. Nel pannello di navigazione, scegli Gestisci, quindi Lavori.

    b. Scegli Crea un lavoro, quindi scegli Crea lavoro di aggiornamento OTA. Seleziona un elemento, quindi scegli Avanti.

    c. Crea un processo di aggiornamento OTA FreeRTOS come segue:

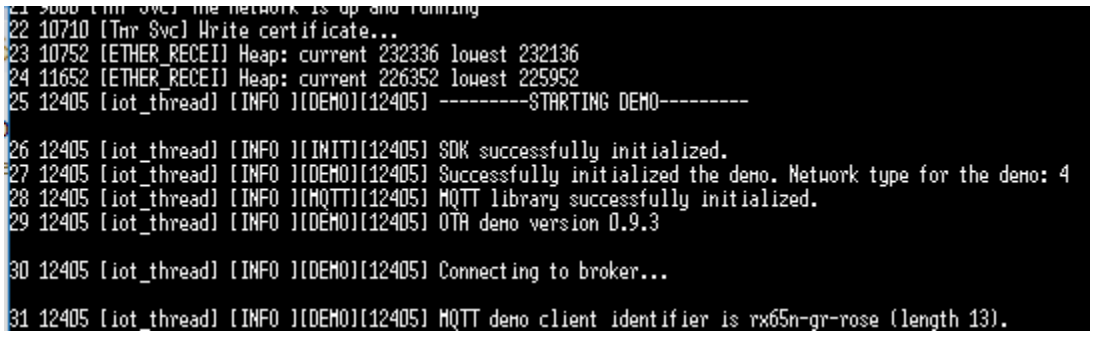
    - Scegli MQTT.
    - Selezionare il profilo di firma con codice creato nella sezione precedente.
    - Selezionare l'immagine del firmware caricata in un bucket Amazon S3.
    - Per Nome del percorso dell'immagine del firmware sul dispositivo, immettere**`test`**.
    - Scegliere laruolo IAM creato nella sezione precedente.

    d. Seleziona Successivo.

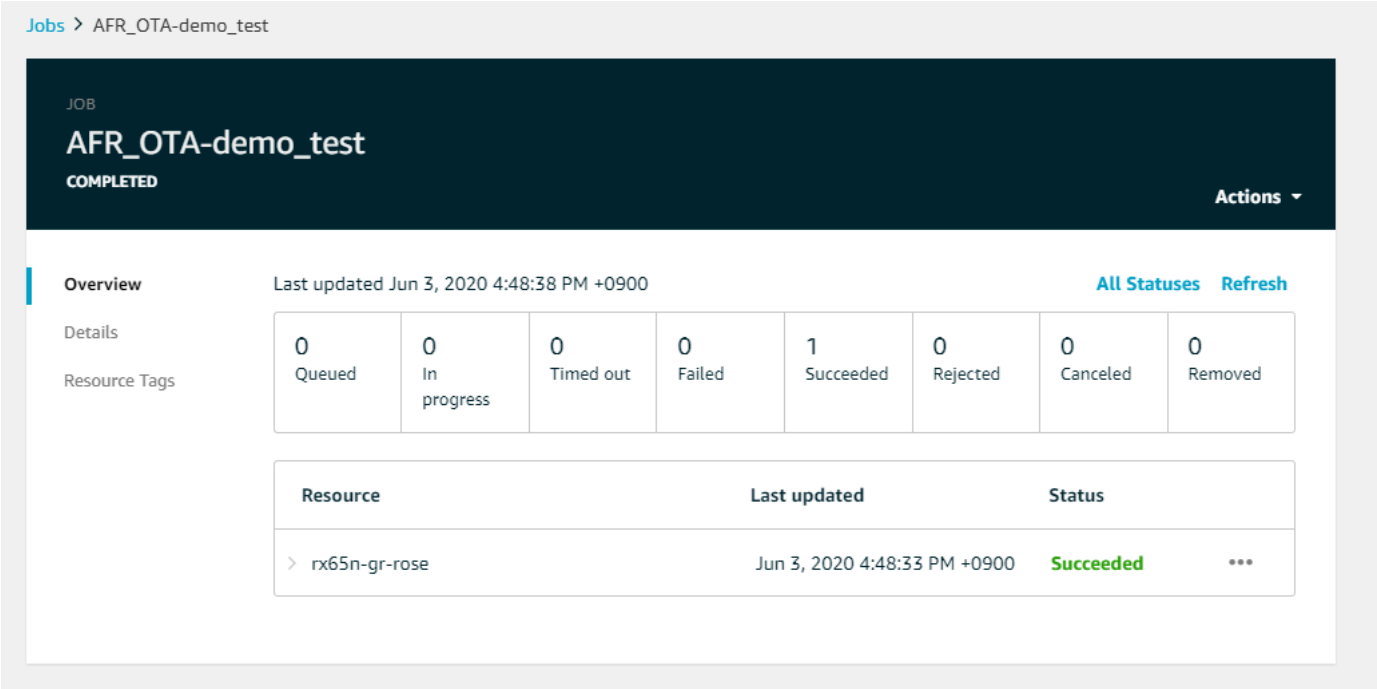


e. Inserisci un ID e scegli Crea.

21. Riapri Tera Term per verificare che il firmware sia stato aggiornato correttamente alla versione demo OTA 0.9.3.



22. SullaAWS IoT console, verifica che lo stato del processo sia Completato.



Tutorial: esegui aggiornamenti OTA su Espressif ESP32 utilizzando FreeRTOS Bluetooth Low Energy

> **Important**
>
> Questa integrazione di riferimento è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Consigliamo di iniziare da qui quando crei un nuovo progetto. Se disponi già di un progetto FreeRTOS esistente basato sull'ormai obsoleto repository Amazon-FreeRTOS, consulta ilGuida alla migrazione del repository Github di Amazon-FreeRTOS.

Questo tutorial mostra come aggiornare un microcontrollore Espressif ESP32 collegato a un proxy MQTT Bluetooth Low Energy su un dispositivo Android. Aggiorna il dispositivo utilizzando i processi di aggiornamentoAWS IoT Over-the-air (OTA). Il dispositivo si connetteAWS IoT utilizzando le credenziali Amazon Cognito inserite nell'app demo Android. Un operatore autorizzato avvia l'aggiornamento OTA dal cloud. Quando il dispositivo si connette tramite l'app demo Android, viene avviato l'aggiornamento OTA e il firmware viene aggiornato sul dispositivo.

Le versioni FreeRTOS 2019.06.00 Major e successive includono il supporto proxy Bluetooth Low Energy MQTT che può essere utilizzato per il provisioning Wi-Fi e le connessioni sicure aiAWS IoT servizi. Utilizzando la funzione Bluetooth Low Energy, puoi creare dispositivi a basso consumo che

possono essere accoppiati a un dispositivo mobile per la connettività senza richiedere il Wi-Fi. I dispositivi possono comunicare utilizzando MQTT connettendosi tramite SDK Bluetooth Low Energy Android o iOS che utilizzano profili di accesso generici (GAP) e attributi generici (GATT).

Ecco i passaggi che seguiremo per consentire gli aggiornamenti OTA tramite Bluetooth Low Energy:

1. Configurazione dello storage: crea un bucket e delle politiche Amazon S3 e configura un utente in grado di eseguire gli aggiornamenti.
2. Crea un certificato di firma del codice: crea un certificato di firma e consenti all'utente di firmare gli aggiornamenti del firmware.
3. Configura l'autenticazione Amazon Cognito: crea un provider di credenziali, un pool di utenti e l'accesso dell'applicazione al pool di utenti.
4. Configura FreeRTOS: configura Bluetooth Low Energy, le credenziali del client e il certificato pubblico di firma del codice.
5. Configura un'app Android: configura il provider di credenziali, il pool di utenti e distribuisci l'applicazione su un dispositivo Android.
6. Esegui lo script di aggiornamento OTA: per avviare un aggiornamento OTA, usa lo script di aggiornamento OTA.

Per ulteriori informazioni sul funzionamento degli aggiornamenti, consultareAggiornamenti via etere di FreeRTOS. Per ulteriori informazioni su come configurare la funzionalità proxy Bluetooth Low Energy MQTT, consulta il post Utilizzo di Bluetooth Low Energy con FreeRTOS su Espressif ESP32 di Richard Kang.

Prerequisiti

Per eseguire queste fasi in questo tutorial, hai bisogno di:

- Una scheda di sviluppo ESP32.
- Un cavo da microUSB a USB A.
- UnAWS account (il piano gratuito è sufficiente).
- Un telefono Android con Android v 6.0 o successivo e Bluetooth versione 4.2 o successiva.

Sul tuo computer di sviluppo hai bisogno di:

- Spazio su disco sufficiente (~500 Mb) per la toolchain di Xtensa e il codice sorgente e gli esempi di FreeRTOS.

- Android Studio installato.
- L'AWS CLIinstallato.
- Python3 installato.
- Il kit di sviluppoAWS software (SDK) boto3 per Python.

I passaggi di questo tutorial presuppongono che la toolchain Xtensa, ESP-IDF e il codice FreeRTOS siano installati nella`/esp` directory della home directory. È necessario aggiungere`~/esp/xtensa-esp32-elf/bin` alla`$PATH` variabile.

Fase 1: configurare l'archiviazione

1. Crea un bucket Amazon S3 per archiviare l'aggiornamentocon il controllo delle versioni abilitato per contenere le immagini del firmware.
2. Creazione di un ruolo del servizio per gli aggiornamenti OTAe aggiungere le policy gestite seguenti al ruolo:
   - AWSIotLogging
   - AWSIotRuleActions
   - AWSIotThingsRegistration
   - AWSFreeRTOSOTAUpdate
3. Crea un utente in grado di eseguire aggiornamenti OTA. Questo utente può firmare e distribuire gli aggiornamenti del firmware per i dispositivi IoT nell'account e ha accesso agli aggiornamenti OTA su tutti i dispositivi. L'accesso deve essere limitato a entità attendibili.
4. Segui i passaggi perCreazione di una policy utente OTA collegarlo al tuo utente.

Fase 2: Creazione del certificato di firma del codice

1. Crea un bucket Amazon S3 con il controllo delle versioni abilitato per contenere le immagini del firmware.
2. Crea un certificato di firma del codice che può essere utilizzato per firmare il firmware. Nota il certificato Amazon Resource Name (ARN) al momento dell'importazione.

```
aws acm import-certificate --profile=ota-update-user --certificate file://
ecdsasigner.crt --private-key file://ecdsasigner.key
```

Output di esempio:

```
{
"CertificateArn": "arn:aws:acm:us-east-1:<account>:certificate/<certid>"
}
```

Utilizzerai l'ARN in seguito per creare un profilo di firma. Se lo desideri, puoi creare il profilo ora con il seguente comando:

```
aws signer put-signing-profile --profile=ota-update-user --profile-
name esp32Profile --signing-material certificateArn=arn:aws:acm:us-
east-1:account:certificate/certid --platform AmazonFreeRTOS-Default --signing-
parameters certname=/cert.pem
```

Output di esempio:

```
{
"arn": "arn:aws:signer::<account>:/signing-profiles/esp32Profile"
}
```

Fase 3: configurazione dell'autenticazione Amazon Cognito

Creazione di una policy AWS IoT

1. Accedi alla console AWS IoT.
2. Nell'angolo superiore destro della console, scegliere Il mio account. In Impostazioni dell'account, annota l'ID account di 12 cifre.
3. Nel riquadro di navigazione a sinistra scegliere Impostazioni. Nell'endpoint dei dati del dispositivo, prendi nota del valore dell'endpoint. L'endpoint dovrebbe essere simile`xxxxxxxxxxxxxx.iot.us-west-2.amazonaws.com`. In questo esempio, laAWS regione è «us-west-2".
4. Nel riquadro di navigazione a sinistra, scegli Sicuro, scegli Criteri e quindi scegli Crea. Se non hai alcuna politica nel tuo account, vedrai il messaggio «Non hai ancora alcuna politica» e puoi scegliere Crea una politica.
5. Inserisci un nome per la tua politica, ad esempio «esp32_mqtt_proxy_iot_policy».

6. Nella sezione Add statements (Aggiungi istruzioni), scegliere Advanced mode (Modalità avanzata). Copiare e incollare il seguente JSON nella finestra dell'editor policy. Sostituisciloaws-account-id con l'ID del tuo account eaws-region con la tua regione (ad esempio, «us-west-2").

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:Connect",
      "Resource": "arn:aws:iot:aws-region:aws-account-id:*"
    },
    {
      "Effect": "Allow",
      "Action": "iot:Publish",
      "Resource": "arn:aws:iot:aws-region:aws-account-id:*"
    },
    {
      "Effect": "Allow",
      "Action": "iot:Subscribe",
      "Resource": "arn:aws:iot:aws-region:aws-account-id:*"
    },
    {
      "Effect": "Allow",
      "Action": "iot:Receive",
      "Resource": "arn:aws:iot:aws-region:aws-account-id:*"
    }
  ]
}
```

7. Seleziona Create (Crea).

Crea qualsiasiAWS IoT cosa

1. Accedi alla console AWS IoT.
2. Nel riquadro di navigazione a sinistra nel pannello di controllo scegliere Manage (Gestisci), quindi Things (Oggetti).
3. Nell'angolo in alto a destra, scegli Crea. Se non hai nessun elemento registrato nel tuo account, viene visualizzato il messaggio «Non hai ancora nulla» e puoi scegliere Registra un oggetto.

4. Nella pagina Creazione oggetti AWS IoT, scegliere Crea un oggetto singolo.
5. Nella pagina Aggiungi il tuo dispositivo al registro degli oggetti, inserisci un nome per il tuo oggetto (ad esempio, «esp32-ble»). Sono consentiti solo A-Z, «_» (trattino basso) e «-» (trattino). Seleziona Successivo.
6. Nella pagina Aggiungi un certificato per il tuo oggetto, in Ignora certificato e crea oggetto, scegli Crea oggetto senza certificato. Poiché utilizziamo l'app mobile proxy BLE che utilizza una credenziale Amazon Cognito per l'autenticazione e l'autorizzazione, non è richiesto alcun certificato del dispositivo.

Crea un client Amazon Cognito App

1. Accedi alla console Amazon Cognito.
2. Nel banner di navigazione in alto a destra, scegli Crea un pool di utenti.
3. Immettete il nome del pool (ad esempio, «esp32_mqtt_proxy_user_pool»).
4. Scegli Review defaults (Esamina impostazioni predefinite).
5. In App Client, scegli Aggiungi app client, quindi scegli Aggiungi un client per app.
6. Inserisci il nome del client dell'app (ad esempio «mqtt_app_client»).
7. Assicurati che sia selezionata l'opzione Genera segreto del cliente.
8. Scegli Create app client (Crea client dell'app).
9. Scegli Return to pool details (Torna ai dettagli del pool).
10. Nella pagina Revisione del pool di utenti, scegli Crea pool. Dovresti visualizzare un messaggio che dice «Il pool di utenti è stato creato correttamente» (trattino basso). Prendi nota dell'ID del pool.
11. Nel riquadro di navigazione, scegliere Client per app.
12. Scegli Mostra dettagli. Prendi nota dell'ID del client dell'app e del segreto del client dell'app.

Creazione di un pool di identità in Amazon Cognito

1. Accedi alla console Amazon Cognito.
2. Scegli Create new identity pool (Crea un nuovo pool di identità).
3. Immettete un nome per il pool di identità (ad esempio, «mqtt_proxy_identity_pool»).
4. Espandi i provider di autenticazione.

5. Scegli la scheda Cognito.
6. Inserisci l'ID del pool di utenti e l'ID del client dell'app che hai annotato nei passaggi precedenti.
7. Seleziona Create Pool (Crea pool).
8. Nella pagina successiva, per creare nuovi ruoli per identità autenticate e non autenticate, scegli Consenti.
9. Prendi nota dell'ID del pool di identità, che è nel formato`us-east-1:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx`.

Collegamento di una policy IAM all'identità autenticata

1. Apri la console Amazon Cognito.
2. Seleziona il pool di identità che hai appena creato (ad esempio, «mqtt_proxy_identity_pool»).
3. Scegli Modifica pool di identità.
4. Prendi nota del ruolo IAM assegnato al ruolo autenticato (ad esempio, «COGNITO_MQTT_PROXY_IDENTITY_POOLAUTH_ROLE»).
5. Aprire la console IAM.
6. Nel pannello di navigazione, selezionare Ruoli.
7. Cerca il ruolo assegnato (ad esempio, «COGNITO_MQTT_PROXY_IDENTITY_POOLAUTH_ROLE»), quindi selezionalo.
8. Scegli Aggiungi policy in linea, quindi scegli JSON.
9. Immettere la seguente policy:

```
{
       "Version": "2012-10-17",
       "Statement": [
       {
          "Effect": "Allow",
          "Action": [
             "iot:AttachPolicy",
             "iot:AttachPrincipalPolicy",
             "iot:Connect",
             "iot:Publish",
             "iot:Subscribe"
          ],
          "Resource": "*"
       }]
```

```
    }
```

10. Scegliere Review policy (Esamina policy).
11. Immettere un nome di policy (ad esempio, "-mqttProxyCognito trattino basso).
12. Scegli Create Policy (Crea policy).

Fase 4: configurare Amazon FreeRTOS

1. Scarica la versione più recente del codice Amazon FreeRTOS dal GitHub repository FreeRTOS.
2. Per abilitare la demo di aggiornamento OTA, segui i passaggi indicatiGuida introduttiva all'Espressif ESP32- DevKit C e all'ESP-WROVER-KIT.
3. Apporta queste modifiche aggiuntive nei seguenti file:

   a. Aprivendors/espressif/boards/esp32/aws_demos/config_files/aws_demo_config.h e definisciCONFIG_OTA_UPDATE_DEMO_ENABLED.

   b. Aprivendors/espressif/boards/esp32/aws_demos/common/config_files/aws_demo_config.h e cambiademoconfigNETWORK_TYPES inAWSIOT_NETWORK_TYPE_BLE.

   c. Apridemos/include/aws_clientcredential.h e inserisci l'URL dell'endpoint perclientcredentialMQTT_BROKER_ENDPOINT.

      Inserisci il nome dell'oggetto perclientcredentialIOT_THING_NAME (ad esempio, «esp32-ble»). Non è necessario aggiungere certificati quando si utilizzano le credenziali Amazon Cognito.

   d. Aprivendors/espressif/boards/esp32/aws_demos/config_files/aws_iot_network_config.h e modificaconfigSUPPORTED_NETWORKS e soloconfigENABLED_NETWORKS da includereAWSIOT_NETWORK_TYPE_BLE.

   e. Apri ilvendors/*vendor*/boards/*board*/aws_demos/config_files/ota_demo_config.h file e inserisci il tuo certificato.

      ```
      #define otapalconfigCODE_SIGNING_CERTIFICATE [] = "your-certificate-key";
      ```

   L'applicazione dovrebbe avviarsi e stampare la versione demo:

   ```
   11 13498 [iot_thread] [INFO ][DEMO][134980] Successfully initialized the demo.
    Network type for the demo: 2
   ```

```
12 13498 [iot_thread] [INFO ][MQTT][134980] MQTT library successfully initialized.
13 13498 [iot_thread] OTA demo version 0.9.20
14 13498 [iot_thread] Creating MQTT Client...
```

Fase 5: configurare un'app per Android

1. Scarica l'SDK Android Bluetooth Low Energy e un'app di esempio dal GitHub repository amazon-freertos-ble-android-sdk.
2. Apri il file`app/src/main/res/raw/awsconfiguration.json` e inserisci l'ID del pool AppClientId, la regione e AppClientSecret utilizzando le istruzioni nel seguente esempio JSON.

```
{
  "UserAgent": "MobileHub/1.0",
  "Version": "1.0",
  "CredentialsProvider": {
    "CognitoIdentity": {
      "Default": {
        "PoolId": "Cognito->Manage Identity Pools->Federated Identities-
>mqtt_proxy_identity_pool->Edit Identity Pool->Identity Pool ID",
        "Region": "Your region (for example us-east-1)"
      }
    }
  },

  "IdentityManager": {
    "Default": {}
  },

  "CognitoUserPool": {
    "Default": {
      "PoolId": "Cognito-> Manage User Pools -> esp32_mqtt_proxy_user_pool ->
 General Settings -> PoolId",
      "AppClientId": "Cognito-> Manage User Pools -> esp32_mqtt_proxy_user_pool ->
 General Settings -> App clients ->Show Details",
      "AppClientSecret": "Cognito-> Manage User Pools -> esp32_mqtt_proxy_user_pool
 -> General Settings -> App clients ->Show Details",
      "Region": "Your region (for example us-east-1)"
    }
  }
}
```

3. Apri`app/src/main/java/software/amazon/freertos/DemoConstants.java` e inserisci il nome della policy che hai creato in precedenza (ad esempio, *`esp32_mqtt_proxy_iot_policy`*) e anche la regione (ad esempio, *`us-east-1`*).
4. Crea e installa l'app demo.
   a. In Android Studio, scegli Build, quindi l'app Make Module.
   b. Scegli Esegui, quindi Esegui app. Puoi accedere al riquadro della finestra logcat in Android Studio per monitorare i messaggi di registro.
   c. Sul dispositivo Android, crea un account dalla schermata di accesso.
   d. Creare un utente. Se esiste già un utente, inserisci le credenziali.
   e. Consenti alla demo Amazon FreeRTOS di accedere alla posizione del dispositivo.
   f. Cerca dispositivi Bluetooth Low Energy.
   g. Sposta il cursore del dispositivo trovato su Attivato.
   h. Premere y sulla console di debug della porta seriale per l'ESP32.
   i. Scegli Pair & Connect.
5. Quanto più... il collegamento diventa attivo dopo aver stabilito la connessione. Lo stato della connessione dovrebbe cambiare in «BLE_CONNECTED» nel logcat del dispositivo Android quando la connessione è completa:

```
2019-06-06 20:11:32.160 23484-23497/software.amazon.freertos.demo I/FRD: BLE
 connection state changed: 0; new state: BLE_CONNECTED
```

6. Prima che i messaggi possano essere trasmessi, il dispositivo Amazon FreeRTOS e il dispositivo Android negoziano l'MTU. Dovresti vedere il seguente output in logcat:

```
2019-06-06 20:11:46.720 23484-23497/software.amazon.freertos.demo I/FRD:
 onMTUChanged : 512 status: Success
```

7. Il dispositivo si connette all'app e inizia a inviare messaggi MQTT utilizzando il proxy MQTT. Per confermare che il dispositivo è in grado di comunicare, assicurati che il valore dei dati caratteristici MQTT_CONTROL cambi a 01:

```
2019-06-06 20:12:28.752 23484-23496/software.amazon.freertos.demo D/FRD: <-<-<-
 Writing to characteristic: MQTT_CONTROL with data: 01
2019-06-06 20:12:28.839 23484-23496/software.amazon.freertos.demo D/FRD:
 onCharacteristicWrite for: MQTT_CONTROL; status: Success; value: 01
```

8. Quando i dispositivi sono associati, verrà visualizzato un messaggio sulla console ESP32. Per abilitare BLE, premere y. La demo non funzionerà finché non esegui questo passaggio.

```
E (135538) BT_GATT: GATT_INSUF_AUTHENTICATION: MITM Required
W (135638) BT_L2CAP: l2cble_start_conn_update, the last connection update command
 still pending.
E (135908) BT_SMP: Value for numeric comparison = 391840
15 13588 [InputTask] Numeric comparison:391840
16 13589 [InputTask] Press 'y' to confirm
17 14078 [InputTask] Key accepted
W (146348) BT_SMP: FOR LE SC LTK IS USED INSTEAD OF STK
18 16298 [iot_thread] Connecting to broker...
19 16298 [iot_thread] [INFO ][MQTT][162980] Establishing new MQTT connection.
20 16298 [iot_thread] [INFO ][MQTT][162980] (MQTT connection 0x3ffd5754, CONNECT
 operation 0x3ffd586c) Waiting for operation completion.
21 16446 [iot_thread] [INFO ][MQTT][164450] (MQTT connection 0x3ffd5754, CONNECT
 operation 0x3ffd586c) Wait complete with result SUCCESS.
22 16446 [iot_thread] [INFO ][MQTT][164460] New MQTT connection 0x3ffc0ccc
 established.
23 16446 [iot_thread] Connected to broker.
```

Fase 6: eseguire lo script di aggiornamento OTA

1. Per installare i prerequisiti, esegui questi comandi:

```
pip3 install boto3
```

```
pip3 install pathlib
```

2. Incrementa la versione dell'applicazione FreeRTOS in`demos/include/aws_application_version.h`.
3. Crea un nuovo file.bin.
4. Scarica lo script python start_ota.py. Per visualizzare il contenuto della guida per lo script, esegui il comando seguente in una finestra del terminale:

```
python3 start_ota.py -h
```

   Viene visualizzato quanto segue:

```
usage: start_ota.py [-h] --profile PROFILE [--region REGION]
                    [--account ACCOUNT] [--devicetype DEVICETYPE] --name NAME
                    --role ROLE --s3bucket S3BUCKET --otasigningprofile
                    OTASIGNINGPROFILE --signingcertificateid
                    SIGNINGCERTIFICATEID [--codelocation CODELOCATION]
Script to start OTA update
optional arguments:
-h, --help            show this help message and exit
--profile PROFILE     Profile name created using aws configure
--region REGION       Region
--account ACCOUNT     Account ID
--devicetype DEVICETYPE thing|group
--name NAME           Name of thing/group
--role ROLE           Role for OTA updates
--s3bucket S3BUCKET   S3 bucket to store firmware updates
--otasigningprofile OTASIGNINGPROFILE
                      Signing profile to be created or used
--signingcertificateid SIGNINGCERTIFICATEID
                      certificate id (not arn) to be used
--codelocation CODELOCATION
                      base folder location (can be relative)
```

5. Se hai utilizzato ilAWS CloudFormation modello fornito per creare risorse, esegui il comando seguente:

```
python3 start_ota_stream.py --profile otausercf --name esp32-ble --role
 ota_ble_iot_role-sample --s3bucket afr-ble-ota-update-bucket-sample --
otasigningprofile abcd --signingcertificateid certificateid
```

Dovresti vedere l'avvio dell'aggiornamento nella console di debug ESP32:

```
38 2462 [OTA Task] [prvParseJobDoc] Job was accepted. Attempting to start transfer.
---
49 2867 [OTA Task] [prvIngestDataBlock] Received file block 1, size 1024
50 2867 [OTA Task] [prvIngestDataBlock] Remaining: 1290
51 2894 [OTA Task] [prvIngestDataBlock] Received file block 2, size 1024
52 2894 [OTA Task] [prvIngestDataBlock] Remaining: 1289
53 2921 [OTA Task] [prvIngestDataBlock] Received file block 3, size 1024
54 2921 [OTA Task] [prvIngestDataBlock] Remaining: 1288
55 2952 [OTA Task] [prvIngestDataBlock] Received file block 4, size 1024
56 2953 [OTA Task] [prvIngestDataBlock] Remaining: 1287
```

```
57 2959 [iot_thread] State: Active  Received: 5   Queued: 5   Processed: 5
 Dropped: 0
```

6. Quando l'aggiornamento OTA è completo, il dispositivo si riavvia come richiesto dal processo di aggiornamento OTA. Quindi tenta di connettersi utilizzando il firmware aggiornato. Se l'aggiornamento è riuscito, il firmware aggiornato viene contrassegnato come attivo e dovresti vedere la versione aggiornata nella console:

```
13 13498 [iot_thread] OTA demo version 0.9.21
```

Applicazione demo AWS IoT Device Shadow

> **Important**
>
> Questa demo è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Consigliamo di procedere da qui quando procedere alla creazione di un nuovo progetto. Se disponi già di un progetto FreeRTOS esistente basato sull'ormai obsoleto repository Amazon-FreeRTOS, consulta ilGuida alla migrazione del repository Github di Amazon-FreeRTOS.

Introduzione

Questa demo mostra come utilizzare la libreriaAWS IoT Device Shadow per connettersi al servizioAWS Device Shadow. Utilizza illibreria CoreMQTT per stabilire una connessione MQTT con TLS (Mutual Authentication) al brokerAWS IoT MQTT e al parser della libreria CoreJSON per analizzare i documenti shadow ricevuti dal servizioAWS Shadow. La demo mostra le operazioni shadow di base, ad esempio 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 shadow/`update` e`/update/delta` messaggi inviati dal servizioAWS IoT Device Shadow.

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 da remoto, anche se il dispositivo non è attualmente connesso. Il dispositivo confermerà la richiesta di aggiornamento quando sarà connesso.

> **Note**
>
> Per configurare ed eseguire le demo di FreeRTOS, segui i passaggi indicatiNosu FreeRTOS.

Funzionalità

La demo crea una singola attività applicativa che passa in rassegna una serie di esempi che mostrano shadow`/update` e`/update/delta` callback per simulare l'alternanza dello stato di un dispositivo remoto. Invia un aggiornamento shadow con il nuovo`desired` stato e attende che il dispositivo cambi`reported` stato in risposta al nuovo`desired` stato. Inoltre, viene utilizzato un`/update` callback shadow per stampare gli stati delle ombre che cambiano. Questa demo utilizza anche una connessione MQTT sicura al brokerAWS IoT MQTT e presuppone che ci sia uno`power0n` stato nell'ombra del dispositivo.

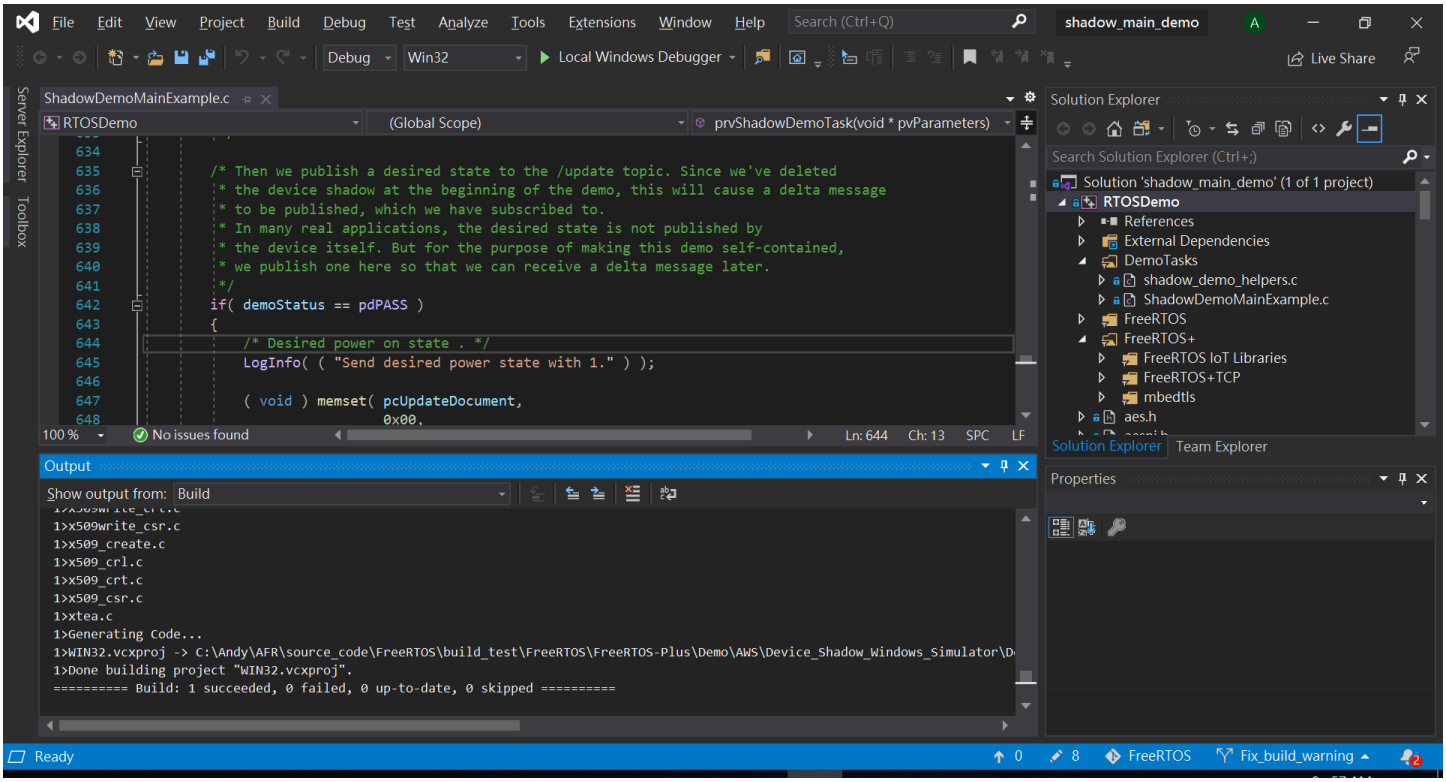
La demo esegue le seguenti operazioni:

1. Stabilisci una connessione MQTT utilizzando le funzioni di supporto in`shadow_demo_helpers.c`.
2. Assembla le stringhe degli argomenti MQTT per le operazioni shadow del dispositivo, utilizzando le macro definite dalla libreriaAWS IoT Device Shadow.
3. Pubblica sull'argomento MQTT utilizzato per eliminare un'ombra del dispositivo per eliminare qualsiasi ombra del dispositivo esistente.
4. Iscriviti agli argomenti MQTT per`/update/delta` `/update/accepted` e`/update/rejected` utilizza 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 causerà l'invio di un`/update/delta` messaggio al dispositivo.
6. Gestisci i messaggi MQTT in`prvEventCallback` entrata e determina se il messaggio è correlato all'ombra del dispositivo utilizzando una funzione definita dalla libreriaAWS IoT Device Shadow (`Shadow_MatchTopic`). Se il messaggio è un`/update/delta` messaggio shadow del dispositivo, la funzione demo principale pubblicherà un secondo messaggio per aggiornare lo stato segnalato`power0n`. Se viene ricevuto un`/update/accepted` messaggio, verifica che sia lo`clientToken` stesso pubblicato in precedenza nel messaggio di aggiornamento. Questo segnerà la fine della demo.

```
82 9136 [ShadowDemo] [INFO] [SHADOW] [prvShadowDemoTask:641] 83 9136 [ShadowDemo] Send desired power state with 1.84 9136 [ShadowDemo]
85 9296 [ShadowDemo] [INFO] [SHADOW] [prvEventCallback:482] 86 9296 [ShadowDemo] pPublishInfo->pTopicName:$aws/things/testClient16:34:41/shadow/update/delta.87 9296 [Shad
owDemo]
88 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateDeltaHandler:256] 89 9296 [ShadowDemo] /update/delta json payload:{"version":1,"timestamp":1602751002,"state":{"powerOn":1}
,"metadata":{"powerOn":{"timestamp":1602751002}},"clientToken":"009136"}.90 9296 [ShadowDemo]
91 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateDeltaHandler:298] 92 9296 [ShadowDemo] version: 193 9296 [ShadowDemo]
94 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateDeltaHandler:308] 95 9296 [ShadowDemo] version:1, ulCurrentVersion:0
96 9296 [ShadowDemo]
97 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateDeltaHandler:342] 98 9296 [ShadowDemo] The new power on state newState:1, ulCurrentPowerOnState:0
99 9296 [ShadowDemo]
100 9296 [ShadowDemo] [INFO] [SHADOW] [prvEventCallback:482] 101 9296 [ShadowDemo] pPublishInfo->pTopicName:$aws/things/testClient16:34:41/shadow/update/accepted.102 9296
 [ShadowDemo]
103 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateAcceptedHandler:376] 104 9296 [ShadowDemo] /update/accepted json payload:{"state":{"desired":{"powerOn":1}},"metadata":{"d
esired":{"powerOn":{"timestamp":1602751002}}},"version":1,"timestamp":1602751002,"clientToken":"009136"}.105 9296 [ShadowDemo]
106 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateAcceptedHandler:424] 107 9296 [ShadowDemo] clientToken: 009136108 9296 [ShadowDemo]
109 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateAcceptedHandler:429] 110 9296 [ShadowDemo] receivedToken:9136, clientToken:0
111 9296 [ShadowDemo]
112 9296 [ShadowDemo] [WARN] [SHADOW] [prvUpdateAcceptedHandler:442] 113 9296 [ShadowDemo] The received clientToken=9136 is not identical with the one=0 we sent 114 9296
[ShadowDemo]
115 9696 [ShadowDemo] [INFO] [SHADOW] [prvShadowDemoTask:670] 116 9696 [ShadowDemo] Report to the state change: 1117 9696 [ShadowDemo]
118 9856 [ShadowDemo] [INFO] [SHADOW] [prvEventCallback:482] 119 9856 [ShadowDemo] pPublishInfo->pTopicName:$aws/things/testClient16:34:41/shadow/update/accepted.120 9856
 [ShadowDemo]
121 9856 [ShadowDemo] [INFO] [SHADOW] [prvUpdateAcceptedHandler:376] 122 9856 [ShadowDemo] /update/accepted json payload:{"state":{"reported":{"powerOn":1}},"metadata":{"
reported":{"powerOn":{"timestamp":1602751003}}},"version":2,"timestamp":1602751003,"clientToken":"009696"}.123 9856 [ShadowDemo]
124 9856 [ShadowDemo] [INFO] [SHADOW] [prvUpdateAcceptedHandler:424] 125 9856 [ShadowDemo] clientToken: 009696126 9856 [ShadowDemo]
127 9856 [ShadowDemo] [INFO] [SHADOW] [prvUpdateAcceptedHandler:429] 128 9856 [ShadowDemo] receivedToken:9696, clientToken:9696
129 9856 [ShadowDemo]
130 9856 [ShadowDemo] [INFO] [SHADOW] [prvUpdateAcceptedHandler:437] 131 9856 [ShadowDemo] Received response from the device shadow. Previously published update with clie
ntToken=9696 has been accepted. 132 9856 [ShadowDemo]
133 10256 [ShadowDemo] [INFO] [SHADOW] [prvShadowDemoTask:698] 134 10256 [ShadowDemo] Start to unsubscribe shadow topics and disconnect from MQTT.
135 10256 [ShadowDemo]
136 12036 [ShadowDemo] [INFO] [SHADOW] [prvShadowDemoTask:747] 137 12036 [ShadowDemo] Demo completed successfully.138 12036 [ShadowDemo]
139 12036 [ShadowDemo] [INFO] [SHADOW] [prvShadowDemoTask:756] 140 12036 [ShadowDemo] Deleting Shadow Demo task.141 12036 [ShadowDemo]
```

La demo è disponibile nel file *freertos*`/demos/device_shadow_for_aws/shadow_demo_main.c` o su GitHub.

Lo screenshot seguente mostra l'output previsto quando la demo avrà successo.



Connect al brokerAWS IoT MQTT

Per connetterci al brokerAWS IoT MQTT, utilizziamo lo stesso metodo`MQTT_Connect()` di[Demo dell'autenticazione reciproca CoreMQTT](#).

Eliminare il documento shadow

Per eliminare il documento shadow, chiama`xPublishToTopic` con un messaggio vuoto, utilizzando le macro definite dalla libreriaAWS IoT Device Shadow. Viene utilizzato`MQTT_Publish` per pubblicare sull'`/delete`argomento. La seguente sezione del codice seguente mostra come procedere 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 oscuri

Iscriviti agli argomenti Device Shadow per ricevere notifiche dalAWS IoT broker sulle modifiche alle ombre. Gli argomenti Device Shadow sono assemblati mediante macro definite nella libreria Device Shadow. La seguente sezione del codice seguente mostra come procedere 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 su Shadow

Per inviare un aggiornamento shadow, la demo chiamaxPublishToTopic con un messaggio in formato JSON, utilizzando le macro definite dalla libreria Device Shadow. Viene utilizzatoMQTT_Publish per pubblicare sull'/deleteargomento. La seguente sezione del codice seguente mostra come procedere nellaprvShadowDemoTask 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 di aggiornamento shadow

La funzione di callback dell'utente, che è stata registrata nella libreria client CoreMQTT utilizzando laMQTT_Init funzione, ci informerà di un evento di pacchetto in arrivo. Vedi la funzione di callback prvEventCallbackattiva GitHub.

La funzione di callback conferma che il pacchetto in arrivo 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 shadow 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 lo`powerOn` stato e lo confronta con lo stato corrente del dispositivo mantenuto localmente. Se sono diversi, lo stato locale del dispositivo viene aggiornato in modo da riflettere il nuovo valore dello`powerOn` stato del documento shadow.

Se il messaggio in arrivo è un messaggio shadow di tipo`ShadowMessageTypeUpdateAccepted`, chiamiamo `prvUpdateAcceptedHandler` per gestire questo messaggio. Il gestore`prvUpdateAcceptedHandler` analizza il messaggio utilizzando la libreria CoreJSON per ottenere il`clientToken` file dal messaggio. 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 demo è ospitata nel repository Amazon-FreeRTOS che è obsoleto. Consigliamo di iniziare da qui quando crei un nuovo progetto. Se hai già un progetto FreeRTOS esistente basato sull'ormai obsoleto repository Amazon-FreeRTOS, consulta ilGuida alla migrazione del repository Github di Amazon-FreeRTOS.

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 esegui i progetti demo FreeRTOS sul tuo microcontrollore e di aver creato ed esegui i progetti demo di FreeRTOS sul 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 indicatiNosu FreeRTOS. Le demo del client e del server TCP non sono al momento supportate nei kit di sviluppo Cypress CYW943907AEVAL1F e CYW954907AEVAL1F.

1. Segui le istruzioni in Configurazione del server TLS Echo nella Guida alla portabilità di FreeRTOS.

   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. Apri*`freertos`*`/vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/aws_demo_config.h`, commenta`#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED` e

definisci`CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED` o`CONFIG_OTA_HTTP_UPDATE_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`.

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.