aws

Panduan Pengguna

FreeRTOS

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

FreeRTOS: Panduan Pengguna

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

Merek dagang dan tampilan dagang Amazon tidak boleh digunakan sehubungan dengan produk atau layanan apa pun yang bukan milik Amazon, dengan cara apa pun yang dapat menyebabkan kebingungan di antara pelanggan, atau dengan cara apa pun yang merendahkan atau mendiskreditkan Amazon. Semua merek dagang lain yang tidak dimiliki oleh Amazon merupakan hak milik masing-masing pemiliknya, yang mungkin atau tidak terafiliasi, terkait dengan, atau disponsori oleh Amazon.

Table of Contents

Apa itu FreeRTOS? 1
Unduh kode sumber FreeRTOS 1
Platform perangkat keras yang memenuhi syarat Freertos 1
Sumber daya tambahan 2
Versi FreeRTOS 3
Dukungan Jangka Panjang FreeRTOS 3
Rencana Pemeliharaan Diperpanjang FreeRTOS 3
Arsitektur FreeRTOS 4
Alur kerja pengembangan 5
Dasar-dasar kernel FreeRTOS 6
Penjadwal kernel FreeRTOS 6
Alokasi memori kernel 7
Kelola memori aplikasi 7
Koordinasi antar tugas 8
Antrean 8
Semaphores dan mutex 9
Direct-to-task pemberitahuan 9
Buffer aliran 10
Buffer pesan 11
Dukungan multiprocessing simetris (SMP) 13
Pengatur waktu perangkat lunak 14
Dukungan daya rendah 14
`FreeRTOSConfig.h` 14
AWS IoT Perangkat SDK untuk Embedded C 16
Memahami FreeRTOS Common IO APIs 17
Perpustakaan 17
IO umum - dasar 17
IO Umum - BLE 19
IO Umum untuk Perangkat Lunak Umum Amazon 19
Apa itu ACS? 19
Program Kualifikasi 20
Memulai dengan FreeRTOS 21
Memulai dengan Quick Connect 21
Jelajahi perpustakaan FreeRTOS 21

Membangun AWS IoT produk yang aman dan kuat 22
Kembangkan AWS IoT aplikasi Anda 22
AWS IoT Device Tester untuk FreeRTOS 23
Suite kualifikasi FreeRTOS 23
Memahami rangkaian pengujian khusus 24
Versi IDT yang didukung untuk FreeRTOS 25
Versi terbaru dari IDT untuk FreeRTOS 25
Versi IDT sebelumnya 27
Versi IDT yang tidak didukung 33
Unduh IDT untuk FreeRTOS 67
Unduh IDT secara manual 67
Unduh IDT secara terprogram 68
IDT dengan suite kualifikasi FreeRTOS 2.0 (FRQ 2.0) 73
Siapkan prasyarat kualifikasi LTS 74
Tes pertama papan mikrokontroler Anda 84
Gunakan UI IDT untuk menjalankan suite kualifikasi FreeRTOS 100
Jalankan suite kualifikasi FreeRTOS 2.0 115
Lihat IDT secara Gratis RTOSresults 119
Menafsirkan IDT untuk hasil FreeRTOS 119
Lihat IDT secara Gratis RTOSlogs 122
IDT dengan suite kualifikasi FreeRTOS 1.0 (FRQ 1.0) 123
Siapkan prasyarat kualifikasi 1.0 124
Tes pertama papan mikrokontroler Anda 128
Gunakan UI IDT untuk menjalankan suite kualifikasi FreeRTOS 148
Jalankan tes Bluetooth Low Energy 159
Jalankan suite kualifikasi FreeRTOS 164
Lihat IDT untuk hasil FreeRTOS 170
Menafsirkan IDT untuk hasil FreeRTOS 171
Lihat IDT untuk log FreeRTOS 174
Kembangkan dan jalankan rangkaian pengujian IDT Anda sendiri 174
Unduh versi terbaru IDT untuk FreeRTOS 175
Alur kerja test suite 175
Tutorial: Bangun dan jalankan sampel rangkaian tes IDT 176
Tutorial: Kembangkan rangkaian tes IDT sederhana 182
Versi rangkaian tes 269
Memecahkan masalah kesalahan 270

Memecahkan masalah konfigurasi perangkat 271
Memecahkan masalah kesalahan batas waktu 285
Fitur seluler dan AWS biaya 285
Kebijakan pembuatan laporan kualifikasi 286
AWS Kebijakan terkelola untuk AWS IoT Device Tester 286
Kebijakan terkelola 286
Pembaruan kebijakan 293
Kebijakan Support 296
Keamanan di AWS 298
Identity and Access Management 298
Audiens 299
Mengautentikasi dengan identitas 300
Mengelola akses menggunakan kebijakan 303
Bagaimana FreeRTOS bekerja dengan IAM 306
Contoh kebijakan berbasis identitas 313
Pemecahan Masalah 316
Validasi kepatuhan 318
Ketahanan 319
Keamanan infrastruktur 320
Panduan Migrasi Repositori Github Amazon-freertos 321
Lampiran 321
Arsip 327
Arsip Panduan Pengguna FreeRTOS 327
Konten Panduan Pengguna FreeRTOS sebelumnya 327
Memulai dengan FreeRTOS 327
Over-the-Air Update 523
Perpustakaan FreeRTOS 609
Demo FreeRTOS 676
..... dcccii

Apa itu FreeRTOS?

Dikembangkan dalam kemitraan dengan perusahaan chip terkemuka di dunia selama periode 15 tahun, dan sekarang diunduh setiap 170 detik, FreeRTOS adalah sistem operasi real-time (RTOS) terkemuka di pasar untuk mikrokontroler dan mikroprosesor kecil. Didistribusikan secara bebas di bawah lisensi open source MIT, FreeRTOS mencakup kernel dan kumpulan perpustakaan yang berkembang yang cocok untuk digunakan di semua sektor industri. FreeRTOS dibangun dengan penekanan pada keandalan dan kemudahan penggunaan.

FreeRTOS mencakup pustaka untuk pembaruan konektivitas, keamanan, over-the-air dan (OTA). FreeRTOS juga mencakup aplikasi demo yang menunjukkan fitur FreeRTOS pada papan yang memenuhi syarat.

FreeRTOS adalah proyek sumber terbuka. Anda dapat mengunduh kode sumber, berkontribusi perubahan atau penyempurnaan, atau melaporkan masalah di GitHub situs di https://github.com/FreeRTOS/ FreeRTOS.

Kami merilis kode FreeRTOS di bawah lisensi sumber terbuka MIT, sehingga Anda dapat menggunakannya dalam proyek komersial dan pribadi.

Kami juga menyambut kontribusi untuk dokumentasi FreerTOS (Panduan Pengguna FreeRTOS, Panduan Porting FreeRTOS, dan Panduan Kualifikasi FreerTOS). Untuk melihat sumber penurunan harga untuk dokumentasi, lihat https://github.com/awsdocs/aws-freertos-docs. Ini dirilis di bawah lisensi Creative Commons (CC BY-ND).

Unduh kode sumber FreeRTOS

Unduh paket FreeRTOS dan Long Term Support (LTS) terbaru dari halaman Unduhan di freertos.org.

Platform perangkat keras yang memenuhi syarat Freertos

Platform perangkat keras berikut memenuhi syarat untuk FreeRTOS:

- ATECC608A Zero Touch Kit Penyediaan untuk AWS IoT
- Kit Pengembangan Cypress CYW9439 07 AEVAL1 F
- Kit Pengembangan Cypress CYW9549 07 AEVAL1 F
- Cypress CY8 CCIT-064S0S2-4343W Kit
- Espressif ESP32 - C DevKit

- Espressif ESP-WROVER-KIT
- Espressif ESP-WROOM-32SE
- Espressif ESP32 -S2-Saola-1
- Kit Konektivitas IoT Infineon XMC48 00
- Kit Pemula Marvell MW32 0 AWS IoT
- Kit Pemula Marvell MW322 AWS IoT
- MediaTek MT7697Kit Pengembangan Hx
- Bundel MZEF Keingintahuan PIC32 Microchip
- Nordik n 0-DK RF5284
- NuMaker-IoT-M487
- Modul LPC54 IoT NXP 018
- Solusi Keamanan OPTIGA Trust X
- Modul IoT Renesas RX65 N RSK
- STMicroelectronicsSTM32Kit Penemuan L4 IoT Node
- Texas Instruments CC322 0SF-LAUNCHXL
- Microsoft Windows 7 atau yang lebih baru, dengan setidaknya dual core dan koneksi Ethernet terprogram
- Kit IoT Industri Xilinx Avnet MicroZed

Perangkat yang memenuhi syarat juga tercantum di Katalog Perangkat AWS Mitra.

Untuk informasi tentang kualifikasi perangkat baru, lihat Panduan Kualifikasi FreeRTOS.

Sumber daya tambahan

Sumber daya ini mungkin bermanfaat bagi Anda.

- Untuk Dokumentasi FreeRTOS tambahan, lihat freertos.org.
- Untuk pertanyaan tentang FreeRTOS untuk tim teknik FreeRTOS, Anda dapat membuka masalah di halaman FreeRTOS. GitHub
- Untuk pertanyaan teknis tentang FreeRTOS, lihat Forum Komunitas FreeRTOS.
- Untuk informasi selengkapnya tentang menghubungkan perangkat AWS IoT, lihat Penyediaan Perangkat di Panduan AWS IoT Core Pengembang.

- Untuk dukungan teknis AWS, lihat AWS Support.
- Untuk pertanyaan tentang AWS penagihan, layanan akun, acara, penyalahgunaan, atau masalah lain dengan AWS, lihat halaman Hubungi Kami.

Versi FreeRTOS

Pustaka individu menggunakan nomor versi gaya xy.z, mirip dengan versi semantik. x adalah nomor versi utama, y nomor versi minor, dan mulai dari 2022, z adalah nomor tambalan. Sebelum 2022, z adalah nomor rilis poin, yang mengharuskan pustaka LTS pertama memiliki nomor tambalan dari bentuk “x.y.z LTS Patch 2".

Paket perpustakaan menggunakan nomor versi stempel tanggal gaya yyyymm.x. yyyy adalah tahun, mm bulan, dan x nomor urut opsional yang menunjukkan urutan rilis dalam sebulan. Dalam kasus paket LTS, x adalah nomor patch sekuensial untuk rilis LTS tersebut. Pustaka individu yang terkandung dalam sebuah paket adalah apa pun versi terbaru dari perpustakaan itu pada tanggal tersebut. Untuk paket LTS, ini adalah versi patch terbaru dari pustaka LTS yang awalnya dirilis sebagai versi LTS pada tanggal tersebut.

Dukungan Jangka Panjang FreeRTOS

FreeRTOS Long Term Support (LTS) rilis menerima keamanan dan perbaikan bug kritis (jika diperlukan) setidaknya selama dua tahun setelah rilis. Dengan pemeliharaan berkelanjutan ini, Anda dapat menggabungkan perbaikan bug di seluruh siklus pengembangan dan penerapan tanpa gangguan mahal untuk memperbarui ke versi utama pustaka FreeRTOS yang baru.

Dengan FreeRTOS LTS, Anda mendapatkan kumpulan pustaka lengkap yang diperlukan untuk membangun IoT yang terhubung dan produk tertanam yang aman. LTS membantu mengurangi biaya pemeliharaan dan pengujian yang terkait dengan memperbarui pustaka pada perangkat Anda yang sudah diproduksi.

FreeRTOS LTS mencakup kernel FreeRTOS dan pustaka IoT: Freertos+TCP, CoreMQTT, CoreHTTP, core, CoreJson, OTA, Jobs,, dan Device Shadow. PKCS11 AWS IoT AWS IoT AWS IoT Device Defender AWS IoT Untuk informasi lebih lanjut, lihat pustaka FreerTOS LTS.

Rencana Pemeliharaan Diperpanjang FreeRTOS

AWS juga menawarkan FreerTOS Extended Maintenance Plan (EMP), yang menyediakan patch keamanan dan perbaikan bug penting pada versi FreerTOS Long Term Support (LTS) pilihan

Anda hingga sepuluh tahun tambahan. Dengan FreeRTOS EMP, perangkat tahan lama berbasis FreeRTOS Anda dapat mengandalkan versi yang memiliki stabilitas fitur dan menerima pembaruan keamanan selama bertahun-tahun. Anda menerima pemberitahuan tepat waktu tentang tambalan yang akan datang di pustaka FreeRTOS, sehingga Anda dapat merencanakan penyebaran patch keamanan di perangkat Internet of Things (IoT) Anda.

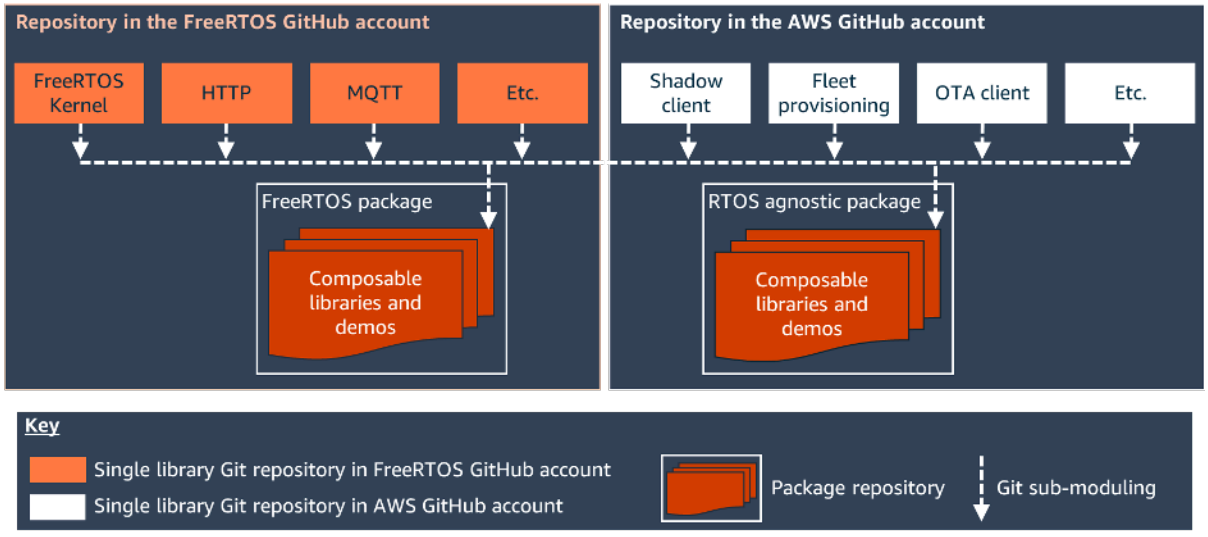
Untuk mempelajari lebih lanjut tentang FreeRTOS EMP, lihat halaman Fitur.

Arsitektur FreeRTOS

FreeRTOS berisi dua jenis repositori, repositori pustaka tunggal dan repositori paket. Setiap repositori pustaka tunggal berisi kode sumber untuk satu pustaka tanpa proyek atau contoh build apa pun. Package repositori berisi beberapa pustaka, dan dapat berisi proyek yang telah dikonfigurasi sebelumnya yang menunjukkan penggunaan pustaka.

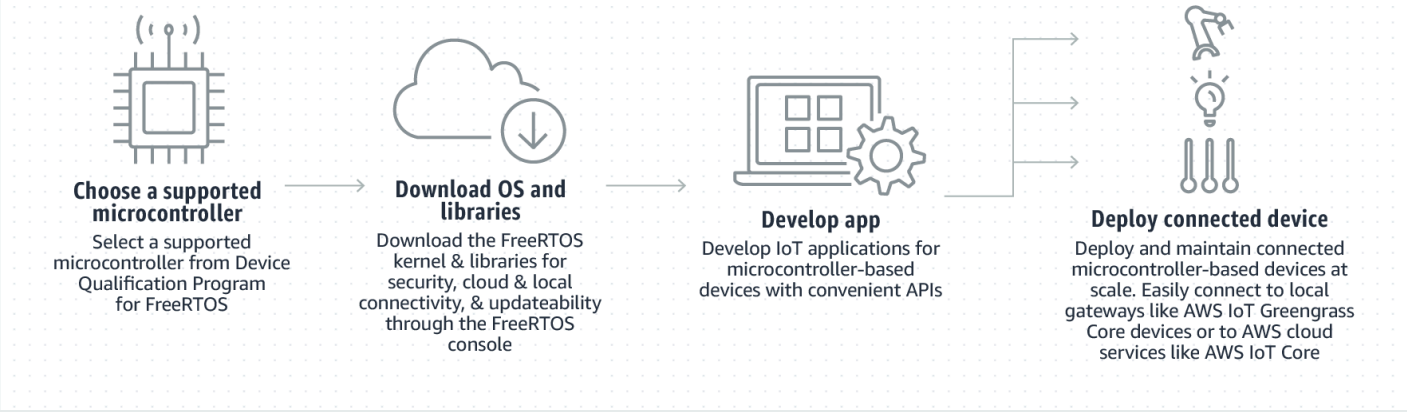
Meskipun repositori paket berisi beberapa pustaka, mereka tidak berisi salinan pustaka tersebut. Sebaliknya, repositori paket mereferensikan pustaka yang dikandungnya sebagai submodul git. Menggunakan submodul memastikan bahwa ada satu sumber kebenaran untuk setiap perpustakaan individu.

Repositori git pustaka individu dibagi antara dua GitHub organisasi. Repositori yang berisi pustaka khusus FreeRTOS (seperti Freertos+TCP) atau pustaka generik (seperti CoreMQTT, yang agnostik cloud karena berfungsi dengan broker MQTT mana pun) ada di organisasi FreeRTOS. GitHub Repositori yang berisi pustaka AWS IoT tertentu (seperti klien AWS IoT over-the-air pembaruan) ada di organisasi. AWS GitHub Diagram berikut menjelaskan strukturnya.



Alur kerja pengembangan

Anda memulai pengembangan dengan mengunduh FreeRTOS. Anda membuka zip paket dan mengimpornya ke IDE Anda. Anda kemudian dapat mengembangkan aplikasi pada platform perangkat keras yang Anda pilih dan memproduksi serta menyebarkan perangkat ini menggunakan proses pengembangan yang sesuai untuk perangkat Anda. Perangkat yang digunakan dapat terhubung ke AWS IoT layanan atau AWS IoT Greengrass sebagai bagian dari solusi IoT lengkap.



Dasar-dasar kernel FreeRTOS

Kernel FreeRTOS adalah sistem operasi real-time yang mendukung berbagai arsitektur. Dasar-dasarnya ideal untuk membangun aplikasi mikrokontroler tertanam. Ini menyediakan:

- Penjadwal multitasking.
- Beberapa opsi alokasi memori (termasuk kemampuan untuk membuat sistem yang sepenuhnya dialokasikan secara statis).
- Primitif koordinasi antar tugas, termasuk pemberitahuan tugas, antrian pesan, beberapa jenis semaphore, dan buffer aliran dan pesan.
- Support untuk multiprocessing simetris (SMP) pada mikrokontroler multi-core.

Kernel FreeRTOS tidak pernah melakukan operasi non-deterministik, seperti berjalan di daftar tertaut, di dalam bagian kritis atau interupsi. Kernel FreeRTOS mencakup implementasi pengatur waktu perangkat lunak yang efisien yang tidak menggunakan waktu CPU apa pun kecuali timer membutuhkan servis. Tugas yang diblokir tidak memerlukan servis berkala yang memakan waktu. Direct-to-tasknotifikasi memungkinkan pensinyalan tugas cepat, dengan praktis tidak ada overhead RAM. Mereka dapat digunakan di sebagian besar skenario antar tugas dan interrupt-to-task pensinyalan.

Kernel FreeRTOS dirancang untuk menjadi kecil, sederhana, dan mudah digunakan. Gambar biner kernel RTOS yang khas berada di kisaran 4000 hingga 9000 byte.

Untuk up-to-date dokumentasi terbanyak tentang kernel FreeRTOS, lihat Freertos.org. Freertos.org menawarkan sejumlah tutorial dan panduan terperinci tentang penggunaan kernel FreerTOS, termasuk Panduan Memulai Cepat FreeRTOS Kernel FreeRTOS dan Implementasi RTOS yang lebih mendalam dalam Dokumentasi FreerTOS.

Penjadwal kernel FreeRTOS

Aplikasi tertanam yang menggunakan RTOS dapat disusun sebagai serangkaian tugas independen. Setiap tugas dijalankan dalam konteksnya sendiri, tanpa ketergantungan pada tugas lain. Hanya satu tugas dalam aplikasi yang berjalan kapan saja. Penjadwal RTOS real-time menentukan kapan setiap tugas harus dijalankan. Setiap tugas dilengkapi dengan tumpukannya sendiri. Ketika tugas ditukar sehingga tugas lain dapat dijalankan, konteks eksekusi tugas disimpan ke tumpukan tugas sehingga dapat dipulihkan ketika tugas yang sama kemudian ditukar kembali untuk melanjutkan eksekusinya.

Untuk memberikan perilaku real-time deterministik, penjadwal tugas FreeRTOS memungkinkan tugas diberi prioritas yang ketat. RTOS memastikan tugas prioritas tertinggi yang dapat dijalankan diberikan waktu pemrosesan. Ini membutuhkan waktu pemrosesan berbagi antara tugas-tugas dengan prioritas yang sama jika mereka siap dijalankan secara bersamaan. FreeRTOS juga membuat tugas idle yang dijalankan hanya ketika tidak ada tugas lain yang siap dijalankan.

Alokasi memori kernel

Kernel RTOS membutuhkan RAM setiap kali tugas, antrian, atau objek RTOS lainnya dibuat. RAM dapat dialokasikan:

- Secara statis pada waktu kompilasi.
- Secara dinamis dari heap RTOS oleh fungsi pembuatan objek API RTOS.

Ketika objek RTOS dibuat secara dinamis, menggunakan pustaka `malloc()` dan `free()` fungsi C standar tidak selalu sesuai karena sejumlah alasan:

- Mereka mungkin tidak tersedia pada sistem tertanam.
- Mereka mengambil ruang kode yang berharga.
- Mereka biasanya tidak aman untuk benang.
- Mereka tidak deterministik.

Untuk alasan ini, FreeRTOS menyimpan API alokasi memori di lapisan portabelnya. Lapisan portabel berada di luar file sumber yang mengimplementasikan fungsionalitas inti RTOS, sehingga Anda dapat memberikan implementasi khusus aplikasi yang sesuai untuk sistem real-time yang Anda kembangkan. Ketika kernel RTOS membutuhkan RAM, ia memanggil `pvPortMalloc()` bukan `malloc()` (). Ketika RAM sedang dibebaskan, kernel RTOS memanggil `vPortFree()` bukan. `free()`

Kelola memori aplikasi

Ketika aplikasi membutuhkan memori, mereka dapat mengalokasikannya dari tumpukan FreeRTOS. FreeRTOS menawarkan beberapa skema manajemen heap yang berkisar dalam kompleksitas dan fitur. Anda juga dapat memberikan implementasi heap Anda sendiri.

Kernel FreeRTOS mencakup lima implementasi heap:

`heap_1`

Adalah implementasi yang paling sederhana. Tidak mengizinkan memori dibebaskan.

`heap_2`

Memungkinkan memori dibebaskan, tetapi tidak menyatukan blok bebas yang berdekatan.

`heap_3`

Membungkus standar `malloc()` dan `free()` untuk keamanan benang.

`heap_4`

Menggabungkan blok bebas yang berdekatan untuk menghindari fragmentasi. Termasuk opsi penempatan alamat absolut.

`heap_5`

Mirip dengan heap_4. Dapat menjangkau tumpukan di beberapa area memori yang tidak berdekatan.

Koordinasi antar tugas

Bagian ini berisi informasi tentang FreeRTOS primitif.

Topik

- Antrean
- Semaphores dan mutex
- Direct-to-task pemberitahuan
- Buffer aliran
- Buffer pesan

Antrean

Antrian adalah bentuk utama komunikasi antar tugas. Mereka dapat digunakan untuk mengirim pesan antar tugas dan antara interupsi dan tugas. Dalam kebanyakan kasus, mereka digunakan sebagai buffer thread-safe, First In First Out (FIFO) dengan data baru dikirim ke bagian belakang antrian. (Data juga dapat dikirim ke depan antrian.) Pesan dikirim melalui antrian dengan salinan, yang berarti data (yang dapat menjadi penunjuk ke buffer yang lebih besar) itu sendiri disalin ke dalam antrian daripada hanya menyimpan referensi ke data.

Antrian APIs mengizinkan waktu blok yang akan ditentukan. Saat tugas mencoba membaca dari antrian kosong, tugas ditempatkan ke status Diblokir hingga data tersedia di antrian atau waktu pemblokiran berlalu. Tugas dalam status Diblokir tidak menghabiskan waktu CPU apa pun, memungkinkan tugas lain berjalan. Demikian pula, ketika tugas mencoba menulis ke antrian penuh, tugas ditempatkan ke status Diblokir sampai ruang tersedia dalam antrian atau waktu blok berlalu. Jika lebih dari satu blok tugas pada antrian yang sama, tugas dengan prioritas tertinggi akan diblokir terlebih dahulu.

Primitif FreeRTOS lainnya, direct-to-task seperti notifikasi dan streaming dan buffer pesan, menawarkan alternatif ringan untuk antrian dalam banyak skenario desain umum.

Semaphores dan mutex

Kernel FreeRTOS menyediakan semaphores biner, menghitung semaphores, dan mutex untuk tujuan saling eksklusi dan sinkronisasi.

Semaphores biner hanya dapat memiliki dua nilai. Mereka adalah pilihan yang baik untuk menerapkan sinkronisasi (baik antara tugas atau antara tugas dan interupsi). Menghitung semaphores membutuhkan lebih dari dua nilai. Mereka memungkinkan banyak tugas untuk berbagi sumber daya atau melakukan operasi sinkronisasi yang lebih kompleks.

Mutexes adalah semaphores biner yang mencakup mekanisme pewarisan prioritas. Ini berarti bahwa jika tugas prioritas tinggi memblokir saat mencoba mendapatkan mutex yang saat ini dipegang oleh tugas prioritas yang lebih rendah, prioritas tugas yang memegang token untuk sementara dinaikkan ke tugas pemblokiran. Mekanisme ini dirancang untuk memastikan tugas prioritas yang lebih tinggi disimpan dalam keadaan Diblokir untuk waktu sesingkat mungkin, untuk meminimalkan inversi prioritas yang telah terjadi.

Direct-to-task pemberitahuan

Pemberitahuan tugas memungkinkan tugas untuk berinteraksi dengan tugas lain, dan untuk menyinkronkan dengan rutinitas layanan interupsi (ISRs), tanpa perlu objek komunikasi terpisah seperti semaphore. Setiap tugas RTOS memiliki nilai notifikasi 32-bit yang digunakan untuk menyimpan konten notifikasi, jika ada. Pemberitahuan tugas RTOS adalah peristiwa yang dikirim langsung ke tugas yang dapat membuka blokir tugas penerima dan secara opsional memperbarui nilai pemberitahuan tugas penerima.

Pemberitahuan tugas RTOS dapat digunakan sebagai alternatif yang lebih cepat dan ringan untuk biner dan menghitung semaphores dan, dalam beberapa kasus, antrian. Pemberitahuan tugas

memiliki keunggulan kecepatan dan jejak RAM dibandingkan fitur FreeRTOS lainnya yang dapat digunakan untuk melakukan fungsionalitas yang setara. Namun, pemberitahuan tugas hanya dapat digunakan ketika hanya ada satu tugas yang dapat menjadi penerima acara.

Buffer aliran

Stream buffer memungkinkan aliran byte diteruskan dari rutinitas layanan interupsi ke tugas, atau dari satu tugas ke tugas lainnya. Aliran byte dapat memiliki panjang yang sewenang-wenang dan tidak harus memiliki awal atau akhir. Sejumlah byte dapat ditulis pada satu waktu, dan sejumlah byte dapat dibaca pada satu waktu. Anda mengaktifkan fungsionalitas buffer streaming dengan menyertakan file `stream_buffer.c` sumber dalam proyek Anda.

Stream buffer menganggap hanya ada satu tugas atau interupsi yang menulis ke buffer (penulis), dan hanya satu tugas atau interupsi yang dibaca dari buffer (pembaca). Aman bagi penulis dan pembaca untuk menjadi tugas yang berbeda atau mengganggu rutinitas layanan, tetapi tidak aman untuk memiliki banyak penulis atau pembaca.

Implementasi buffer aliran menggunakan direct-to-task notifikasi. Oleh karena itu, memanggil API buffer aliran yang menempatkan tugas pemanggilan ke status Diblokir dapat mengubah status dan nilai notifikasi tugas pemanggilan.

Mengirim data

`xStreamBufferSend()`digunakan untuk mengirim data ke buffer aliran dalam tugas. `xStreamBufferSendFromISR()`digunakan untuk mengirim data ke buffer aliran dalam interrupt service routine (ISR).

`xStreamBufferSend()`memungkinkan waktu blok ditentukan. Jika `xStreamBufferSend()` dipanggil dengan waktu blok bukan nol untuk menulis ke buffer aliran dan buffer penuh, tugas ditempatkan ke status Diblokir sampai ruang tersedia atau waktu blok berakhir.

`sbSEND_COMPLETED()`dan `sbSEND_COMPLETED_FROM_ISR()` merupakan makro yang dipanggil (secara internal, oleh FreeRTOS API) saat data ditulis ke buffer aliran. Dibutuhkan pegangan buffer aliran yang diperbarui. Kedua makro ini memeriksa untuk melihat apakah ada tugas yang diblokir pada buffer aliran menunggu data, dan jika demikian, menghapus tugas dari status Diblokir.

Anda dapat mengubah perilaku default ini dengan menyediakan implementasi `sbSEND_COMPLETED()` in Anda sendiri`FreeRTOSConfig.h`. Ini berguna ketika buffer aliran digunakan untuk meneruskan data antar inti pada prosesor multicore. Dalam

skenario itu, `sbSEND_COMPLETED()` dapat diimplementasikan untuk menghasilkan interupsi di inti CPU lainnya, dan rutin layanan interupsi kemudian dapat menggunakan `xStreamBufferSendCompletedFromISR()` API untuk memeriksa, dan jika perlu membuka blokir, tugas yang menunggu data.

Menerima data

`xStreamBufferReceive()`digunakan untuk membaca data dari buffer aliran dalam tugas. `xStreamBufferReceiveFromISR()`digunakan untuk membaca data dari buffer aliran dalam rutinitas layanan interupsi (ISR).

`xStreamBufferReceive()`memungkinkan waktu blok ditentukan. Jika `xStreamBufferReceive()` dipanggil dengan waktu blok bukan nol untuk membaca dari buffer aliran dan buffer kosong, tugas ditempatkan ke status Diblokir sampai jumlah data tertentu tersedia di buffer aliran, atau waktu blok berakhir.

Jumlah data yang harus ada di buffer aliran sebelum tugas diblokir disebut tingkat pemicu buffer aliran. Tugas yang diblokir dengan tingkat pemicu 10 tidak diblokir ketika setidaknya 10 byte ditulis ke buffer atau waktu pemblokiran tugas berakhir. Jika waktu blok tugas membaca berakhir sebelum tingkat pemicu tercapai, tugas akan menerima data apa pun yang ditulis ke buffer. Tingkat pemicu tugas harus diatur ke nilai antara 1 dan ukuran buffer aliran. Tingkat pemicu buffer aliran diatur saat `xStreamBufferCreate()` dipanggil. Itu bisa diubah dengan meneleponxStreamBufferSetTriggerLevel().

`sbRECEIVE_COMPLETED()`dan `sbRECEIVE_COMPLETED_FROM_ISR()` merupakan makro yang dipanggil (secara internal, oleh FreeRTOS API) saat data dibaca dari buffer aliran. Makro memeriksa untuk melihat apakah ada tugas yang diblokir pada buffer aliran menunggu ruang tersedia di dalam buffer, dan jika demikian, mereka menghapus tugas dari status Diblokir. Anda dapat mengubah perilaku default `sbRECEIVE_COMPLETED()` dengan menyediakan implementasi alternatif di`FreeRTOSConfig.h`.

Buffer pesan

Buffer pesan memungkinkan pesan diskrit panjang variabel diteruskan dari rutinitas layanan interupsi ke tugas, atau dari satu tugas ke tugas lainnya. Misalnya, pesan dengan panjang 10, 20, dan 123 byte semuanya dapat ditulis ke, dan dibaca dari, buffer pesan yang sama. Pesan 10-byte hanya dapat dibaca sebagai pesan 10-byte, bukan sebagai byte individual. Buffer pesan dibangun di atas implementasi buffer aliran. Anda dapat mengaktifkan fungsionalitas buffer pesan dengan menyertakan file `stream_buffer.c` sumber dalam proyek Anda.

Buffer pesan menganggap hanya ada satu tugas atau interupsi yang menulis ke buffer (penulis), dan hanya satu tugas atau interupsi yang dibaca dari buffer (pembaca). Aman bagi penulis dan pembaca untuk menjadi tugas yang berbeda atau mengganggu rutinitas layanan, tetapi tidak aman untuk memiliki banyak penulis atau pembaca.

Implementasi buffer pesan menggunakan direct-to-task notifikasi. Oleh karena itu, memanggil API buffer aliran yang menempatkan tugas pemanggilan ke status Diblokir dapat mengubah status dan nilai notifikasi tugas pemanggilan.

Untuk mengaktifkan buffer pesan untuk menangani pesan berukuran variabel, panjang setiap pesan ditulis ke dalam buffer pesan sebelum pesan itu sendiri. Panjang disimpan dalam variabel tipe`size_t`, yang biasanya 4 byte pada arsitektur 32-byte. Oleh karena itu, menulis pesan 10-byte ke buffer pesan sebenarnya menghabiskan 14 byte ruang buffer. Demikian juga, menulis pesan 100-byte ke buffer pesan sebenarnya menggunakan 104 byte ruang buffer.

Mengirim data

`xMessageBufferSend()`digunakan untuk mengirim data ke buffer pesan dari tugas. `xMessageBufferSendFromISR()`digunakan untuk mengirim data ke buffer pesan dari interrupt service routine (ISR).

`xMessageBufferSend()`memungkinkan waktu blok ditentukan. Jika `xMessageBufferSend()` dipanggil dengan waktu blok bukan nol untuk menulis ke buffer pesan dan buffer penuh, tugas ditempatkan ke status Diblokir sampai salah satu ruang tersedia di buffer pesan, atau waktu blok berakhir.

`sbSEND_COMPLETED()`dan `sbSEND_COMPLETED_FROM_ISR()` merupakan makro yang dipanggil (secara internal, oleh FreeRTOS API) saat data ditulis ke buffer aliran. Dibutuhkan satu parameter, yang merupakan pegangan buffer aliran yang diperbarui. Kedua makro ini memeriksa untuk melihat apakah ada tugas yang diblokir pada buffer aliran menunggu data, dan jika demikian, mereka menghapus tugas dari status Diblokir.

Anda dapat mengubah perilaku default ini dengan menyediakan implementasi `sbSEND_COMPLETED()` in Anda sendiri`FreeRTOSConfig.h`. Ini berguna ketika buffer aliran digunakan untuk meneruskan data antar inti pada prosesor multicore. Dalam skenario itu, `sbSEND_COMPLETED()` dapat diimplementasikan untuk menghasilkan interupsi di inti CPU lainnya, dan rutin layanan interupsi kemudian dapat menggunakan `xStreamBufferSendCompletedFromISR()` API untuk memeriksa, dan jika perlu membuka blokir, tugas yang sedang menunggu data.

Menerima data

`xMessageBufferReceive()`digunakan untuk membaca data dari buffer pesan dalam tugas. `xMessageBufferReceiveFromISR()`digunakan untuk membaca data dari buffer pesan dalam interrupt service routine (ISR). `xMessageBufferReceive()`memungkinkan waktu blok ditentukan. Jika `xMessageBufferReceive()` dipanggil dengan waktu blok bukan nol untuk membaca dari buffer pesan dan buffer kosong, tugas ditempatkan ke status Diblokir sampai data tersedia, atau waktu pemblokiran berakhir.

`sbRECEIVE_COMPLETED()`dan `sbRECEIVE_COMPLETED_FROM_ISR()` merupakan makro yang dipanggil (secara internal, oleh FreeRTOS API) saat data dibaca dari buffer aliran. Makro memeriksa untuk melihat apakah ada tugas yang diblokir pada buffer aliran menunggu ruang tersedia di dalam buffer, dan jika demikian, mereka menghapus tugas dari status Diblokir. Anda dapat mengubah perilaku default `sbRECEIVE_COMPLETED()` dengan menyediakan implementasi alternatif di`FreeRTOSConfig.h`.

Dukungan multiprocessing simetris (SMP)

Dukungan SMP di Freertos Kernel memungkinkan satu contoh kernel FreerTOS untuk menjadwalkan tugas di beberapa inti prosesor yang identik. Arsitektur inti harus identik dan berbagi memori yang sama.

FreeRTOS API secara substansional tetap sama antara versi single-core dan SMP, kecuali untuk tambahan ini. APIs Oleh karena itu, aplikasi yang ditulis untuk versi single-core FreeRTOS harus dikompilasi dengan versi SMP dengan sedikit atau tanpa usaha. Namun, mungkin ada beberapa masalah fungsional, karena beberapa asumsi yang benar untuk aplikasi single-core mungkin tidak lagi berlaku untuk aplikasi multi-core.

Salah satu asumsi umum adalah bahwa tugas prioritas yang lebih rendah tidak dapat berjalan saat tugas prioritas yang lebih tinggi sedang berjalan. Meskipun ini benar pada sistem single-core, itu tidak lagi berlaku untuk sistem multi-core karena banyak tugas dapat berjalan secara bersamaan. Jika aplikasi bergantung pada prioritas tugas relatif untuk memberikan pengecualian bersama, aplikasi mungkin mengamati hasil yang tidak terduga di lingkungan multi-inti.

Satu asumsi umum lainnya adalah bahwa tidak ISRs dapat berjalan secara bersamaan satu sama lain atau dengan tugas lain. Ini tidak lagi berlaku di lingkungan multi-core. Penulis aplikasi perlu memastikan pengecualian timbal balik yang tepat saat mengakses data yang dibagikan antara tugas dan. ISRs

Pengatur waktu perangkat lunak

Timer perangkat lunak memungkinkan fungsi untuk dieksekusi pada waktu yang ditentukan di masa depan. Fungsi yang dijalankan oleh timer disebut fungsi callback timer. Waktu antara timer yang dimulai dan fungsi callback yang sedang dijalankan disebut periode timer. Kernel FreeRTOS menyediakan implementasi timer perangkat lunak yang efisien karena:

- Itu tidak menjalankan fungsi callback timer dari konteks interupsi.
- Itu tidak menghabiskan waktu pemrosesan apa pun kecuali timer telah benar-benar kedaluwarsa.
- Itu tidak menambahkan overhead pemrosesan apa pun ke interupsi centang.
- Itu tidak berjalan struktur daftar tautan apa pun sementara interupsi dinonaktifkan.

Dukungan daya rendah

Seperti kebanyakan sistem operasi tertanam, kernel FreeRTOS menggunakan timer perangkat keras untuk menghasilkan interupsi tick periodik, yang digunakan untuk mengukur waktu. Penghematan daya implementasi timer perangkat keras reguler dibatasi oleh kebutuhan untuk keluar secara berkala dan kemudian memasukkan kembali status daya rendah untuk memproses interupsi centang. Jika frekuensi interupsi kutu terlalu tinggi, energi dan waktu yang dikonsumsi masuk dan keluar dari status daya rendah untuk setiap tick melebihi potensi keuntungan hemat daya untuk semua kecuali mode hemat daya paling ringan.

Untuk mengatasi batasan ini, FreeRTOS menyertakan mode timer tickless untuk aplikasi berdaya rendah. Mode idle tickless FreeRTOS menghentikan interupsi centang periodik selama periode idle (periode ketika tidak ada tugas aplikasi yang dapat dijalankan), dan kemudian membuat penyesuaian koreksi ke nilai hitungan centang RTOS saat interupsi centang dimulai ulang. Menghentikan interupsi centang memungkinkan mikrokontroler tetap dalam keadaan hemat daya yang dalam sampai interupsi terjadi, atau sekarang saatnya kernel RTOS untuk mentransisikan tugas ke status siap.

Konfigurasikan kernel (Gratis RTOSConfig .h)

Anda dapat mengonfigurasi kernel FreeRTOS untuk papan dan aplikasi tertentu dengan `FreeRTOSConfig.h` file header. Setiap aplikasi yang dibangun di atas kernel harus memiliki file `FreeRTOSConfig.h` header di preprocessor include path. `FreeRTOSConfig.h`khusus aplikasi dan harus ditempatkan di bawah direktori aplikasi, dan bukan di salah satu direktori kode sumber kernel FreeRTOS.

`FreeRTOSConfig.h`File untuk aplikasi demo dan pengujian FreeRTOS terletak di dan. *`freertos`*`/vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/FreeRTOSConfig.h` *`freertos`*`/vendors/`*`vendor`*`/boards/`*`board`*`/aws_tests/config_files/FreeRTOSConfig.h`

Untuk daftar parameter konfigurasi yang tersedia untuk ditentukan`FreeRTOSConfig.h`, lihat Freertos.org.

AWS IoT Perangkat SDK untuk Embedded C

> **Note**
>
> SDK ini dimaksudkan untuk digunakan oleh pengembang perangkat lunak tertanam berpengalaman.

AWS IoT Device SDK for Embedded C C-SDK adalah kumpulan file sumber C di bawah lisensi sumber terbuka MIT yang dapat digunakan dalam aplikasi tertanam untuk menghubungkan perangkat IoT dengan aman. AWS IoT Core Ini termasuk klien MQTT, klien HTTP, JSON Parser, dan Device Shadow, AWS IoT Jobs, AWS IoT Fleet AWS IoT Provisioning, dan perpustakaan. AWS IoT Device Defender SDK ini didistribusikan dalam bentuk sumber dan dapat dibangun ke dalam firmware pelanggan bersama dengan kode aplikasi, perpustakaan lain, dan sistem operasi (OS) pilihan Anda.

Umumnya AWS IoT Device SDK for Embedded C ditargetkan pada perangkat terbatas sumber daya yang memerlukan runtime bahasa C yang dioptimalkan. Anda dapat menggunakan SDK pada sistem operasi apa pun dan menghostingnya pada jenis prosesor apa pun (misalnya, MCUs dan MPUs). Namun, jika perangkat Anda memiliki memori dan sumber daya pemrosesan yang cukup, kami sarankan Anda menggunakan salah satu AWS IoT Perangkat tingkat tinggi SDKs.

Untuk informasi selengkapnya, lihat berikut ini:

- AWS IoT Perangkat SDK untuk Embedded C
- AWS IoT Perangkat SDK untuk Embedded C aktif GitHub
- AWS IoT SDK Perangkat untuk Readme C Tertanam
- AWS IoT SDK Perangkat untuk Sampel C Tertanam

Memahami FreeRTOS Common IO APIs

IO umum APIs bertindak sebagai lapisan abstraksi perangkat keras (HAL) yang menyediakan antarmuka umum antara driver dan kode aplikasi tingkat tinggi. FreeRTOS Common IO menyediakan seperangkat APIs standar untuk mengakses perangkat serial umum pada papan referensi yang didukung; implementasi ini tidak disertakan. APIs Ini umum APIs berkomunikasi dan berinteraksi dengan periferal ini dan memungkinkan kode Anda berfungsi di seluruh platform. Tanpa Common IO, menulis kode untuk bekerja dengan perangkat tingkat rendah adalah khusus vendor silikon.

> **Note**
>
> FreeRTOS tidak memerlukan implementasi Common APIs IO agar berfungsi, tetapi akan mencoba menggunakan Common APIs IO sebagai cara untuk berinteraksi dengan periferal tertentu pada papan berbasis mikrokontroler alih-alih khusus vendor. APIs

Secara umum, driver perangkat independen dari sistem operasi yang mendasarinya dan khusus untuk konfigurasi perangkat keras tertentu. HAL mengabstraksi detail tentang cara kerja driver tertentu dan menyediakan API yang seragam untuk mengontrol perangkat tersebut. Anda dapat menggunakan hal yang sama APIs untuk mengakses berbagai driver perangkat di beberapa papan referensi berbasis mikrokontroler- (MCU-).

Perpustakaan

Saat ini, FreeRTOS menyediakan dua pustaka Common IO: Common IO - basic dan Common IO - BLE.

IO umum - dasar

Gambaran Umum

Common IO - basic menyediakan APIs yang berhubungan dengan periferal I/O dasar dan fungsi yang mungkin Anda temukan di papan berbasis MCU. Common IO - repositori dasar tersedia di. GitHub

Periferal yang didukung

- ADC

- GPIO
- I2C
- PWM
- SPI
- UART
- Pengawas
- Kilat
- RTC
- MEREDAM
- Mengatur ulang
- I2S
- Penghitung kinerja
- Informasi platform perangkat keras

Fitur yang didukung

- Baca/tulis sinkron

  Fungsi tidak kembali sampai jumlah data yang diminta ditransfer.

- Baca/tulis asinkron

  Fungsi segera kembali dan transfer data terjadi secara asinkron. Ketika tindakan selesai, callback pengguna terdaftar dipanggil.

Spesifik periferal

- I2C

  Gabungkan beberapa operasi menjadi satu transaksi. Digunakan untuk menulis kemudian membaca tindakan dalam satu transaksi.

- SPI

  Transfer data antara primer dan sekunder, yang berarti menulis dan membaca terjadi secara bersamaan.

Referensi API

Untuk referensi API lengkap, lihat Common IO - referensi API dasar.

IO Umum - BLE

Gambaran Umum

Common IO - BLE menyediakan abstraksi dari tumpukan Bluetooth Low Energy pabrikan. Ini menyediakan antarmuka berikut yang dapat digunakan untuk mengontrol perangkat, dan melakukan operasi GAP dan GATT. Repositori Common IO - BLE tersedia di. GitHub

Pengelola Perangkat Bluetooth:

Ini menyediakan antarmuka untuk mengontrol perangkat Bluetooth, melakukan operasi penemuan perangkat, dan tugas terkait konektivitas lainnya.

Manajer Adaptor BLE:

Ini menyediakan antarmuka untuk fungsi API GAP yang khusus untuk BLE.

Manajer Adaptor Klasik Bluetooth:

Ini menyediakan antarmuka untuk mengontrol fungsionalitas klasik BT perangkat.

Pelayan GATT:

Ini menyediakan antarmuka untuk menggunakan fitur server Bluetooth GATT.

Klien GATT:

Ini menyediakan antarmuka untuk menggunakan fitur klien Bluetooth GATT.

Antarmuka Koneksi A2DP:

Ini menyediakan antarmuka untuk profil Sumber A2DP untuk perangkat lokal.

Referensi API

Untuk referensi API lengkap, lihat referensi Common IO - BLE API.

IO Umum untuk Perangkat Lunak Umum Amazon

Common IO APIs adalah bagian dari implementasi yang diperlukan oleh Amazon Common Software for Devices, khususnya untuk diimplementasikan dalam kit porting perangkat vendor (DPK).

Apa itu ACS?

Amazon Common Software (ACS) untuk Perangkat adalah perangkat lunak yang membuatnya lebih cepat bagi Anda untuk mengintegrasikan Perangkat Amazon SDKs di perangkat Anda. ACS

menyediakan lapisan integrasi API terpadu, komponen pra-validasi dan efisien memori untuk fungsi umum seperti konektivitas, perangkat porting kit (DPK), dan rangkaian pengujian multi-tier.

Program Kualifikasi

Program kualifikasi Perangkat Lunak Umum Amazon untuk Perangkat memverifikasi bahwa build ACS DPK (Device Porting Kit) yang berjalan pada papan pengembangan berbasis mikrokontroler tertentu kompatibel dengan praktik terbaik program yang diterbitkan dan cukup kuat untuk lulus tes yang diamanatkan ACS yang ditentukan oleh program kualifikasi.

Vendor yang memenuhi syarat dalam program ini tercantum di halaman Vendor Chipset ACS.

Untuk informasi tentang kualifikasi, hubungi ACS untuk Perangkat.

Memulai dengan FreeRTOS

Topik

- Memulai dengan Quick Connect
- Jelajahi perpustakaan FreeRTOS
- Membangun AWS IoT produk yang aman dan kuat
- Kembangkan AWS IoT aplikasi Anda

Memulai dengan Quick Connect

Untuk menjelajah dengan cepat AWS IoT, mulailah dengan Demo AWS Quick Connect. Demo Quick Connect mudah diatur dan menghubungkan mitra yang disediakan, papan yang memenuhi syarat FreeRTOS. AWS IoT

Ikuti tutorial AWS IoT Memulai untuk pemahaman yang lebih baik tentang AWS IoT dan AWS IoT konsol. Anda dapat memodifikasi kode sumber demo yang disediakan dengan demo Quick Connect menggunakan sistem build board yang dipilih dan alat untuk terhubung ke AWS akun Anda. Aliran data dari AWS IoT konsol di akun Anda terlihat sekarang.

Jelajahi perpustakaan FreeRTOS

Setelah Anda memahami bagaimana perangkat IoT dan AWS IoT bekerja bersama, Anda dapat mulai menjelajahi pustaka FreeRTOS, dan pustaka Long-Term Support (LTS).

Beberapa pustaka yang umum digunakan untuk perangkat berbasis AWS IoT FreeRTOS adalah:

- FreeRTOS Kernel
- CoreMQTT
- AWS IoT Over-the-Air (OTA)

Kunjungi freertos.org untuk dokumentasi teknis dan demo khusus perpustakaan.

Membangun AWS IoT produk yang aman dan kuat

Lihat Integrasi FreeRTOS AWS IoT Unggulan untuk mempelajari praktik terbaik dalam membuat perangkat lunak perangkat IoT lebih aman dan kuat. Integrasi FreeRTOS IoT ini dirancang untuk meningkatkan keamanan menggunakan kombinasi perangkat lunak FreeRTOS, dan papan yang disediakan mitra dengan fitur keamanan perangkat keras. Gunakan mereka dalam produksi apa adanya, atau gunakan sebagai model untuk desain Anda sendiri.

Kembangkan AWS IoT aplikasi Anda

Ikuti langkah-langkah ini untuk membuat proyek aplikasi untuk AWS IoT produk Anda:

1. Unduh versi FreerTOS atau Long Term Support (LTS) terbaru dari freertos.org, atau kloning dari repositori Freertos-LTS. GitHub Anda juga dapat mengintegrasikan pustaka FreeRTOS yang diperlukan ke dalam proyek Anda dari toolchain vendor MCU jika tersedia.
2. Ikuti panduan Porting FreeRTOS untuk membuat proyek, mengatur lingkungan pengembangan, dan mengintegrasikan pustaka FreeRTOS ke dalam proyek Anda. Gunakan repositori GitHub Freertos-Libraries-Integration-Tests untuk memvalidasi porting.

AWS IoT Device Tester untuk FreeRTOS

IDT untuk FreeRTOS adalah alat untuk memenuhi syarat tingkat throughput data dengan sistem operasi FreeRTOS. Device tester (IDT) pertama kali membuka koneksi USB atau UART ke perangkat. Kemudian mem-flash gambar FreeRTOS yang dikonfigurasi untuk menguji fungsionalitas perangkat dalam berbagai kondisi. AWS IoT Device Tester suite dapat diperluas dan IDT digunakan untuk orkestrasi AWS IoT pengujian pelanggan.

IDT untuk FreeRTOS berjalan pada komputer host (Windows, macOS, atau Linux) yang terhubung ke perangkat yang sedang diuji. IDT mengonfigurasi dan mengatur kasus uji, dan mengumpulkan hasil. Ini juga menyediakan antarmuka baris perintah untuk mengelola eksekusi uji.

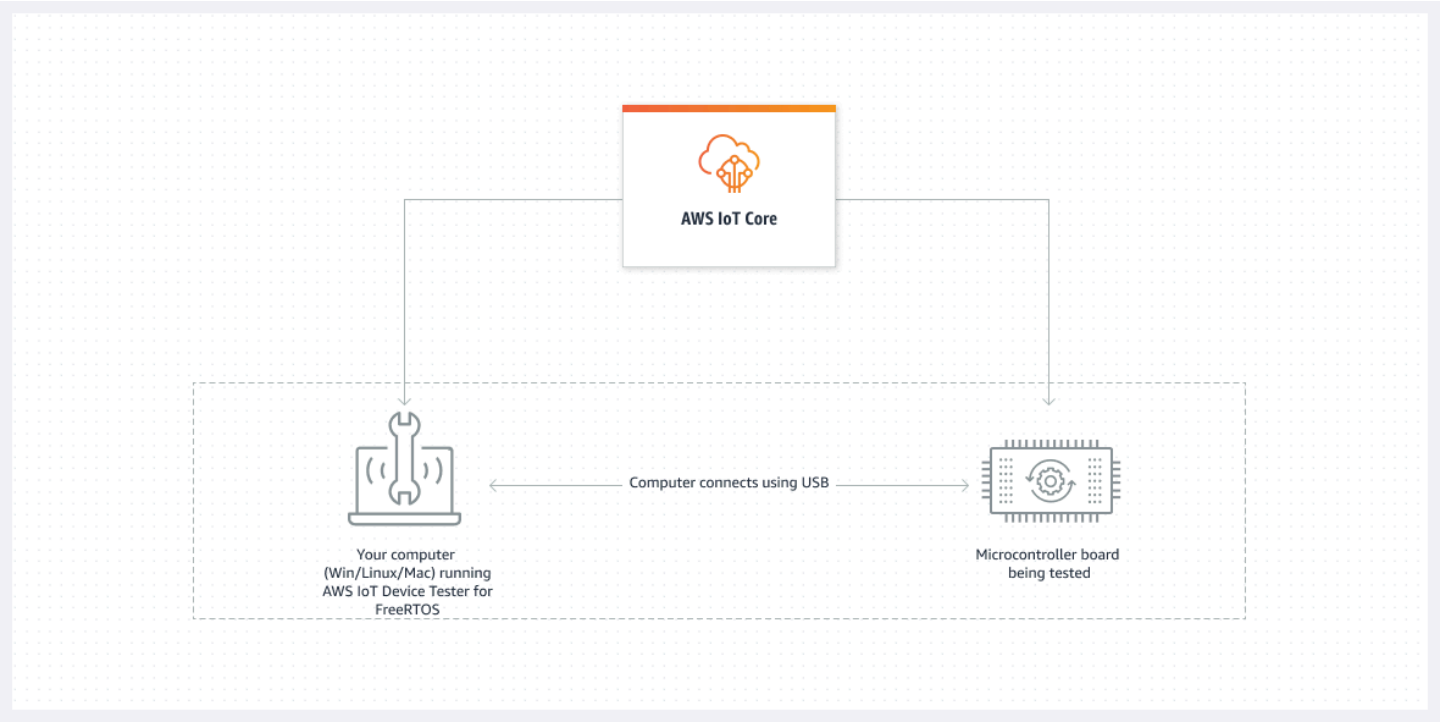
Suite kualifikasi FreeRTOS

IDT untuk FreeRTOS memverifikasi port FreeRTOS pada mikro-kontroler Anda, dan jika dapat berkomunikasi secara efektif dengan cara yang andal dan aman. AWS IoT Secara khusus, ini memverifikasi apakah antarmuka lapisan porting untuk pustaka FreerTOS diimplementasikan dengan benar. Ini juga melakukan end-to-end tes dengan AWS IoT Core. Misalnya, ini memverifikasi apakah papan Anda dapat mengirim dan menerima pesan MQTT dan memprosesnya dengan benar.

Kualifikasi FreeRTOS (FRQ) 2.0 suite menggunakan kasus pengujian FreeRTOS-Libraries-Integration-Tests dari dan Device Advisor yang ditentukan dalam Panduan Kualifikasi FreeRTOS.

IDT untuk FreeRTOS menghasilkan laporan pengujian yang dapat Anda kirimkan AWS ke Jaringan Mitra (APN) untuk dimasukkan perangkat FreeRTOS Anda dalam Katalog Perangkat Mitra. AWS Untuk informasi selengkapnya, lihat Program Kualifikasi Perangkat AWS.

Diagram berikut menunjukkan pengaturan infrastruktur pengujian untuk kualifikasi FreeRTOS.



IDT untuk FreeRTOS mengatur sumber daya pengujian ke dalam rangkaian pengujian dan grup pengujian:

- Rangkaian pengujian adalah kumpulan grup pengujian yang digunakan untuk memverifikasi bahwa perangkat berfungsi dengan versi FreeRTOS tertentu.
- Kelompok uji adalah kumpulan kasus uji individu yang terkait dengan fitur tertentu, seperti pesan BLE dan MQTT.

Untuk informasi selengkapnya, silakan lihat Versi rangkaian tes

Memahami rangkaian pengujian khusus

IDT untuk FreeRTOS menggabungkan pengaturan konfigurasi standar dan format hasil dengan lingkungan rangkaian pengujian. Lingkungan ini memungkinkan Anda mengembangkan rangkaian pengujian khusus untuk perangkat dan perangkat lunak perangkat Anda. Anda dapat menambahkan pengujian kustom untuk validasi internal Anda sendiri, atau memberikannya kepada pelanggan Anda untuk verifikasi perangkat.

Cara Anda mengonfigurasi rangkaian pengujian kustom menentukan konfigurasi pengaturan yang harus Anda berikan kepada pengguna untuk menjalankan rangkaian pengujian kustom Anda. Untuk informasi selengkapnya, lihat Kembangkan dan jalankan rangkaian pengujian IDT Anda sendiri.

Versi yang didukung AWS IoT Device Tester

Topik ini mencantumkan versi yang didukung AWS IoT Device Tester untuk FreeRTOS. Sebagai praktik terbaik, kami menyarankan Anda menggunakan IDT versi terbaru untuk FreeRTOS yang mendukung FreeRTOS versi target Anda. Setiap versi IDT untuk FreeRTOS memiliki satu atau lebih versi FreeRTOS yang sesuai yang didukungnya. Kami menyarankan Anda mengunduh versi baru IDT untuk FreeRTOS ketika versi baru FreeRTOS dirilis.

Dengan mengunduh perangkat lunak, Anda menyetujui Perjanjian AWS IoT Device Tester Lisensi yang terkandung dalam arsip unduhan.

> **Note**
>
> Ketika Anda menggunakan AWS IoT Device Tester untuk FreeRTOS, kami sarankan Anda memperbarui ke rilis patch terbaru dari versi Freertos-LTS terbaru.

> **Important**
>
> Per Oktober 2022, AWS IoT Device Tester untuk Kualifikasi AWS IoT FreerTOS (FRQ) 1.0 tidak menghasilkan laporan kualifikasi yang ditandatangani. Anda tidak dapat memenuhi syarat perangkat AWS IoT FreeRTOS baru untuk dicantumkan di AWS Katalog Perangkat Mitra melalui Program Kualifikasi Perangkat menggunakan versi AWS IDT FRQ 1.0. Meskipun Anda tidak dapat memenuhi syarat perangkat FreerTOS menggunakan IDT FRQ 1.0, Anda dapat terus menguji perangkat FreeRTOS Anda dengan FRQ 1.0. Kami menyarankan Anda menggunakan IDT FRQ 2.0 untuk memenuhi syarat dan daftar perangkat FreeRTOS di Katalog Perangkat Mitra.AWS

Versi terbaru dari AWS IoT Device Tester untuk FreeRTOS

Gunakan tautan berikut untuk mengunduh versi terbaru IDT untuk FreeRTOS.

Versi terbaru dari AWS IoT Device Tester untuk FreeRTOS

AWS IoT Device Tester versi	Versi suite uji	Versi FreeRTOS yang didukung	Tautan unduhan	Tanggal rilis	Catatan rilis
IDT v4.9.0	FRQ_2.5.0	• 202112.00 • 202212.00 • 202212.01 • Semua tambalan FreerTOS 202210-LTS yang menggunakan pustaka FreerTOS LTS.	• Linux • macOS • Jendela	2023.04.04	• Mendukung pengujian terhadap FreerTOS 202112, 202212, 202212.01 dan semua tambalan FreeRTOS 202210-LTS yang menggunakan pustaka FreeRTOS. Lihat README.md untuk informasi lebih lanjut. Anda harus menyertakan versi patch untuk Freertos-LTS di file Anda. `manifest.yml`

AWS IoT Device Tester versi	Versi suite uji	Versi FreeRTOS yang didukung	Tautan unduhan	Tanggal rilis	Catatan rilis
					• Peningkatan waktu uji OTA E2E. • Membatasi jumlah perangkat yang tercantum dalam `device.json` 1. • Penyempurnaan dan perbaikan bug kecil.

> **Note**
>
> Kami tidak menyarankan agar beberapa pengguna menjalankan IDT dari lokasi bersama, seperti direktori NFS atau folder bersama jaringan Windows. Praktik ini dapat mengakibatkan crash atau korupsi data. Kami sarankan Anda mengekstraksi paket IDT ke drive lokal dan menjalankan biner IDT pada workstation lokal Anda.

Versi IDT sebelumnya untuk FreeRTOS

Versi IDT sebelumnya untuk FreeRTOS juga didukung.

Versi sebelumnya AWS IoT Device Tester untuk FreeRTOS

AWS IoT Device Tester versi	Versi suite uji	Versi FreeRTOS yang didukung	Tautan unduhan	Tanggal rilis	Catatan rilis
IDT v4.8.1	FRQ_2.4.0	• 202112.00 • 202212.00 • 202212.01 • Semua tambalan FreerTOS 202210-LTS yang menggunakan pustaka FreerTOS LTS.	• Linux • macOS • Jendela	2023.01.23	• Lihat README.MD untuk informasi lebih lanjut. Anda harus menyertakan versi patch untuk Freertos-LTS di file Anda. `manifest.yml` • Penyempurnaan dan perbaikan bug kecil.
IDT v4.6.0	FRQ_2.3.0	• 202112.00 • 202212.00 • 202212.01 • 202210-LTS yang menggunakan pustaka FreeRTOS LTS.	• Linux • macOS • Jendela	2022.11.16	• Lihat README.MD untuk informasi lebih lanjut. Anda harus menyertakan versi patch untuk Freertos-LTS di

AWS IoT Device Tester versi	Versi suite uji	Versi FreeRTOS yang didukung	Tautan unduhan	Tanggal rilis	Catatan rilis
					file Anda. `manifest.yml` • Untuk informasi lebih lanjut tentang apa yang termasuk dalam rilis FreeRTOS 202210-LTS, lihat file ChangeLog.md di. GitHub • Menambahkan kemampuan untuk mengkonfigurasi dan menjalankan AWS IoT Device Tester FreeRTOS melalui antarmuka pengguna berbasis

AWS IoT Device Tester versi	Versi suite uji	Versi FreeRTOS yang didukung	Tautan unduhan	Tanggal rilis	Catatan rilis
					web. Lihat UI untuk IDT untuk suite kualifikasi FreeRTOS 2.0 (FRQ 2.0) untuk memulai. • Menambahkan opsi untuk menyimpan salinan modifikasi dari kode sumber yang dibuat dan digunakan saat runtime untuk debugging pasca-pengujian. Untuk informasi selengkapnya, lihat Konfigurasikan

AWS IoT Device Tester versi	Versi suite uji	Versi FreeRTOS yang didukung	Tautan unduhan	Tanggal rilis	Catatan rilis
					pengaturan build, flash, dan uji. • Menambahkan dukungan SDK Klien IDT untuk Java. Untuk informasi selengkapnya tentang SDK Klien IDT, lihat. Kembangkan dan jalankan rangkaian pengujian IDT Anda sendiri

AWS IoT Device Tester versi	Versi suite uji	Versi FreeRTOS yang didukung	Tautan unduhan	Tanggal rilis	Catatan rilis
IDT v4.5.11	FRQ_2.2.0	• 202112.00 • 202212.00 • 202212.01 • 202210-LTS yang menggunakan pustaka FreeRTOS LTS.	• Linux • macOS • Jendela	2022.10.14	• Lihat README.MD untuk informasi lebih lanjut. Anda harus menyertakan versi patch untuk Freertos-LTS di file Anda. `manifest.yml` • Untuk informasi lebih lanjut tentang apa yang termasuk dalam rilis FreeRTOS 202210-LTS, lihat file ChangeLog.md di. GitHub • Penyempurnaan dan perbaikan bug kecil.

Untuk informasi selengkapnya, lihat Memahami kebijakan dukungan untuk AWS IoT Device Tester.

Versi IDT yang tidak didukung untuk FreeRTOS

Bagian ini mencantumkan versi IDT yang tidak didukung untuk FreeRTOS. Versi yang tidak didukung tidak menerima perbaikan bug atau pembaruan. Untuk informasi selengkapnya, lihat Memahami kebijakan dukungan untuk AWS IoT Device Tester.

Versi IDT-Freertos berikut tidak lagi didukung.

Versi FreeRTOS yang tidak AWS IoT Device Tester didukung

AWS IoT Device Tester versi	Versi suite uji	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
IDT v4.5.10	FRQ_2.1.4	• 202112.00 • 202012-LTS yang menggunakan pustaka FreeRTOS LTS.	2022.09.02	• Untuk informasi selengkapnya tentang apa yang termasuk dalam rilis FreeRTOS 202012-LTS, lihat file ChangeLog.md di. GitHub • Menyelesaikan masalah yang memengaruhi grup `OTA End to End` uji. • Dihapus `FullTransportInterfacePlainText` dari berjalan dalam kualifikasi

AWS IoT Device Tester versi	Versi suite uji	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
				berjalan. Teks biasa masih dapat dijalankan sebagai grup uji pengembangan dengan menggunakan `-\-group-id` bendera. • Meningkatkan logging dan keterbacaan konsol dan output file. • Penyempurnaan dan perbaikan bug kecil.

AWS IoT Device Tester versi	Versi suite uji	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
IDT v4.5.9	FRQ_2.1.3	• 202112.00 • 202012.04 -LTS yang menggunakan pustaka FreeRTOS LTS.	2022.08.17	• Untuk informasi selengkapnya tentang apa yang termasuk dalam rilis FreeRTOS 202012.04-LTS, lihat file ChangeLog .md di. GitHub • Menyelesaikan masalah yang memengaruhi grup `FreeRTOSIntegrity` uji. • Grup `FullCloudIoT` pengujian yang diperbarui dengan menghapus kasus uji “MQTT Connect Exponential Backoff Retries”. • Penyempurnaan dan

AWS IoT Device Tester versi	Versi suite uji	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
				perbaikan bug kecil.
IDT v4.5.6	FRQ_2.1.2	• 202112.00 • 202012.04 -LTS yang menggunakan pustaka FreeRTOS LTS.	2022.06.29	• Untuk informasi selengkapnya tentang apa yang termasuk dalam rilis FreeRTOS 202012.04-LTS, lihat file ChangeLog .md di. GitHub • Menambahkan grup uji baru `FullCloud IoT` yang menguji papan AWS IoT Core Device Advisor. • Menyelesaikan masalah yang memengaruhi kasus uji OTA E2E. • Penyempurnaan dan perbaikan bug kecil.

AWS IoT Device Tester versi	Versi suite uji	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
IDT v4.5.5	FRQ_2.1.1	• 202112.00 • 202012.04 -LTS yang menggunakan pustaka FreeRTOS LTS.	2022.06.06	• Untuk informasi selengkapnya tentang apa yang termasuk dalam rilis FreeRTOS 202012.04-LTS, lihat file ChangeLog .md di. GitHub • Menambahkan grup uji baru `FullCloud IoT` yang menguji papan AWS IoT Core Device Advisor. • Menyelesaikan masalah yang memengaruhi kasus RTOSIntegrity uji Gratis RTOSVersion dan Gratis. • Penyempurnaan dan perbaikan bug kecil.

AWS IoT Device Tester versi	Versi suite uji	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
IDT v4.5.5	FRQ_2.1.0	• 202107.00 • 202112.00 • 202012.04 -LTS yang menggunakan pustaka FreeRTOS LTS.	2022.05.31	• Untuk informasi selengkapnya tentang apa yang termasuk dalam rilis FreeRTOS 202012.04-LTS, lihat file ChangeLog .md di. GitHub • Menambahkan grup uji baru `FullCloudIoT` yang menguji papan AWS IoT Core Device Advisor. • Penyempurnaan dan perbaikan bug kecil.

AWS IoT Device Tester versi	Versi suite uji	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
IDT v4.5.4	FRQ_2.0.0	• 202112.00 • 202012.04 -LTS yang menggunakan pustaka FreeRTOS LTS.	2022.05.09	• Untuk informasi selengkapnya tentang apa yang termasuk dalam rilis FreeRTOS 202012.04-LTS, lihat file ChangeLog .md di. GitHub • Menghapus persyaratan untuk memenuhi syarat papan hanya menggunakan versi Amazon FreeRTOS dari `aws/amazon-freertos` GitHub repositori. • Penyempurnaan dan perbaikan bug kecil.

AWS IoT Device Tester versi	Versi suite uji	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
IDT v4.5.2	FRQ_1.6.2	202107.00	2022.01.25	• Untuk informasi lebih lanjut tentang apa yang termasuk dalam rilis FreeRTOS 202107.00 , lihat file ChangeLog .md di. GitHub • Mengimple mentasikan orkestrator pengujian IDT baru untuk mengonfig urasi rangkaian pengujian kustom. Untuk informasi selengkap nya, lihat Mengonfig urasi orkestrat or pengujian IDT. • Penyempur naan dan perbaikan bug kecil.

AWS IoT Device Tester versi	Versi suite uji	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
IDT v4.0.3	FRQ_1.5.1	202012.00	2021.07.30	• Support untuk kualifikasi perangkat dengan kredensil terkunci pada Modul Keamanan Perangkat Keras. • Penyempurnaan dan perbaikan bug kecil.

AWS IoT Device Tester versi	Versi suite uji	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
IDT v4.3.0	FRQ_1.6.1	202107.00	2021.07.26	• Untuk informasi lebih lanjut tentang apa yang termasuk dalam rilis FreeRTOS 202107.00, lihat file ChangeLog.md di. GitHub • Menambahkan kemampuan untuk mengkonfigurasi dan menjalankan AWS IoT Device Tester FreeRTOS melalui antarmuka pengguna berbasis web. Lihat Gunakan antarmuka pengguna IDT untuk menjalankan rangkaian kualifikasi FreeRTOS

AWS IoT Device Tester versi	Versi suite uji	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
				untuk memulai.

AWS IoT Device Tester versi	Versi suite uji	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis

IDT v4.1.0	FRQ_1.6.0	202107.00	2021.07.21	• Untuk informasi lebih lanjut tentang apa yang termasuk dalam rilis FreeRTOS 202107.00 , lihat file ChangeLog .md di. GitHub
				• Menghapus kasus uji berikut dari kualifikasi OTA:
				  • Agen OTA
				  • OTA Nama File Hilang
				  • Jumlah Blok yang Dikonfigurasi OTA Max
				• Menghapus kelompok `Both` uji Dataplane OTA dari Kualifikasi OTA. Dalam `device.js`

AWS IoT Device Tester versi	Versi suite uji	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
				on file, `OTADataPlaneProtocol` konfigurasi sekarang hanya menerima `HTTP` atau `MQTT` sebagai nilai yang didukung. • Mengimplementasikan perubahan berikut pada `freertosFileConfiguration` konfigurasi dalam `userdata.json` file untuk perubahan kode sumber FreeRTOS: • Mengubah nama file yang ditentukan untuk `otaAgentTestsConfi`

AWS IoT Device Tester versi	Versi suite uji	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
				g dan otaAgentDemosConfig dari aws_ota_agent_config.h keota_config.h . • Menambahkan konfigurasi otaDemosConfig opsional baru untuk menentukan path file ke ota_demo_config.h file baru. • Menambahkan bidang baru testStartDelayms userdata.json untuk menentukan penundaan antara waktu perangkat di-flash untuk menjalankan

AWS IoT Device Tester versi	Versi suite uji	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
				grup pengujian FreeRTOS dan kapan perangkat mulai menjalankan pengujian. Nilai harus diberikan dalam milidetik. Penundaan ini dapat digunakan untuk memberi IDT kesempatan untuk terhubung sehingga tidak ada output tes yang terlewatkan.

AWS IoT Device Tester versi	Versi suite uji	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
IDT v4.0.1	FRQ_1.4.1	202012.00	2021.01.19	• Untuk informasi selengkapnya tentang apa yang termasuk dalam rilis FreeRTOS 202012.00 , lihat file ChangeLog .md di. GitHub • Memperkenalkan kasus uji OTA (Over-the-air) E2E (end-to-end) tambahan. • Mendukung kualifikasi papan pengembangan yang menjalankan FreerTOS 202012.00 yang menggunakan perpustakaan FreerTOS LTS. • Menambahkan dukungan untuk kualifika

AWS IoT Device Tester versi	Versi suite uji	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
				si papan pengembangan FreeRTOS menggunakan konektivitas seluler. • Memperbaiki bug dalam konfigurasi server echo. • Memungkinkan Anda untuk mengembangkan dan menjalankan suite pengujian kustom Anda sendiri menggunakan AWS IoT Device Tester untuk FreeRTOS. Untuk informasi selengkapnya, lihat Kembangkan dan jalankan rangkaian

AWS IoT Device Tester versi	Versi suite uji	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
				pengujian IDT Anda sendiri. • Menyediakan aplikasi IDT yang ditandatangani kode, sehingga Anda tidak perlu memberikan izin saat menjalankannya di Windows atau macOS. • Menyempurnakan logika penguraian hasil tes BLE.

AWS IoT Device Tester versi	Versi suite uji	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
IDT v3.4.0	FRQ_1.3.0	202011.01	2020.11.05	• Untuk detail selengkap nya, lihat file ChangelOG .md di file. GitHub • Memperbaiki bug di mana 'RSA' bukan opsi PKCS11 konfigurasi yang valid. • Memperbai ki bug di mana bucket Amazon S3 tidak dibersihk an dengan benar setelah pengujian OTA. • Pembaruan untuk mendukung kasus uji baru di dalam grup uji FullMQTT.

AWS IoT Device Tester versi	Versi suite uji	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
IDT v3.3.0	FRQ_1.2.0	202007.00	2020.09.17	• Untuk detail selengkapnya, lihat file ChangeIOG.md di file. GitHub • end-to-endTes baru untuk memvalidasi fitur penangguhan dan lanjutkan pembaruan Over The Air (OTA). • Memperbaiki bug yang menyebabkan pengguna di Wilayah eu-central-1 tidak dapat lulus validasi konfigurasi untuk pengujian OTA. • Ditambahkan `--update-idt` parameter ke `run-suite` perintah.

AWS IoT Device Tester versi	Versi suite uji	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
				Anda dapat menggunakan opsi ini untuk mengatur respons untuk prompt pembaruan IDT. • Ditambahkan `--update-managed-policy` parameter ke `run-suite` perintah. Anda dapat menggunakan opsi ini untuk mengatur respons untuk prompt pembaruan kebijakan terkelola. • Perbaikan internal dan perbaikan bug, termasuk: • Untuk pembaruan rangkaian pengujian otomatis,

AWS IoT Device Tester versi	Versi suite uji	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
				peningkatan pada peningkatan file konfigurasi.

AWS IoT Device Tester versi	Versi suite uji	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
IDT v3.0.2	FRQ_1.0.1	202002.00		• Untuk informasi selengkapnya, lihat file ChangeLOG.md di file. GitHub • Menambahkan pembaruan otomatis suite pengujian dalam IDT. IDT sekarang dapat mengunduh suite pengujian terbaru yang tersedia untuk versi FreeRTOS Anda. Dengan fitur ini, Anda dapat: • Unduh rangkaian pengujian terbaru menggunakan `upgrade-test-`

AWS IoT Device Tester versi	Versi suite uji	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
				`suite` perintah. • Unduh rangkaian pengujian terbaru dengan menyetel tanda saat Anda memulai IDT. Gunakan `-u` *`flag`* opsi di mana *`flag`* bisa 'y' untuk selalu mengunduh atau 'n' untuk menggunakan versi yang ada. Jika ada beberapa versi rangkaian pengujian yang tersedia, versi terbaru akan

AWS IoT Device Tester versi	Versi suite uji	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
				digunakan kecuali Anda menentukan ID rangkaian pengujian saat memulai IDT. • Gunakan `list-supported-versions` opsi baru untuk mencantumkan versi FreeRTOS dan test suite yang didukung oleh versi IDT yang diinstal. • Buat daftar kasus uji dalam grup dan jalankan pengujian individual. Test suite diberi versi menggunak

AWS IoT Device Tester versi	Versi suite uji	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
				an a. `major` `minor`. `patch`format mulai dari 1.0.0. • Menambahkan `list-supported-products` perintah - Daftar FreeRTOS dan versi test suite yang didukung oleh versi IDT yang diinstal. • Menambahkan `list-test-cases` perintah - Daftar kasus uji yang tersedia dalam kelompok uji. • Menambahkan `test-id` opsi untuk `run-suite` perintah - Gunakan opsi ini untuk menjalank

AWS IoT Device Tester versi	Versi suite uji	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
				an kasus uji individual dalam grup uji.

AWS IoT Device Tester versi	Versi suite uji	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
IDT v1.7.1	FRQ_1.0.0	202002.00		• Untuk detail selengkapnya, lihat file ChangelOG.md di file. GitHub • Mendukung metode penandatanganan kode khusus untuk kasus end-to-end uji over-the-air (OTA) sehingga Anda dapat menggunakan perintah dan skrip penandatanganan kode Anda sendiri untuk menandatangani muatan OTA. • Menambahkan precheck untuk port serial sebelum dimulainya pengujian. Pengujian

AWS IoT Device Tester versi	Versi suite uji	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
				akan gagal dengan cepat dengan pesan kesalahan yang ditingkatkan jika port serial salah dikonfigurasi dalam `device.json` file. • Menambahkan Kebijakan AWS Terkelola `AWSIoTDeviceTesterForFreeRTOSFullAccess` dengan izin yang diperlukan untuk menjalankan AWS IoT Device Tester. Jika rilis baru memerlukan izin tambahan, kami menambahkannya ke kebijakan terkelola ini sehingga Anda

AWS IoT Device Tester versi	Versi suite uji	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
				tidak perlu memperbarui izin IAM secara manual. • File yang dinamai `AFQ_Report.xml` dalam direktori hasil sekarang`FRQ_Report.xml` .

AWS IoT Device Tester versi	Versi suite uji	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
IDT v1.6.2	FRQ_1.0.0	202002.00		• Mendukung tes opsional untuk OTA melalui HTTPS untuk memenuhi syarat papan pengembangan FreeRTOS Anda. • Mendukung titik akhir AWS IoT ATS dalam pengujian. • Mendukung kemampuan untuk memberi tahu pengguna tentang versi IDT terbaru sebelum memulai rangkaian pengujian.

AWS IoT Device Tester versi	Versi suite uji	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
IDT v1.5.2	FRQ_1.0.0	201910.00		• Mendukung kualifikasi perangkat FreeRTOS dengan elemen aman (kunci onboard). • Mendukung port server gema yang dapat dikonfigurasi untuk Secure Sockets dan grup uji Wi-Fi. • Mendukung tanda pengganda batas waktu untuk meningkatkan batas waktu, yang berguna saat Anda memecahkan masalah untuk kesalahan terkait batas waktu. • Menambahkan perbaikan bug untuk

AWS IoT Device Tester versi	Versi suite uji	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
				penguraian log. • Mendukung titik akhir iot ats dalam pengujian.
IDT v1.4.1	FRQ_1.0.0	201908.00		• Menambahkan dukungan untuk PKCS11 pustaka baru dan pembaruan kasus uji. • Memperkenalkan kode kesalahan yang dapat ditindaklanjuti. Untuk informasi selengkapnya, silakan lihat Kode kesalahan IDT • Kebijakan IAM yang diperbarui digunakan untuk menjalankan IDT.

AWS IoT Device Tester versi	Versi suite uji	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
IDT v1.3.2	FRQ_1.0.0	201906.00		• Menambahkan dukungan untuk menguji Bluetooth Low Energy (BLE). • Pengalaman pengguna yang ditingkatkan untuk perintah antarmuka baris perintah IDT (CLI). • Kebijakan IAM yang diperbarui digunakan untuk menjalankan IDT.
IDT-Freertos v1.2	FRQ_1.0.0	• FreeRTOS v1.4.8 • FreeRTOS v1.4.9		Menambahkan dukungan untuk menguji perangkat FreerTOS dengan sistem build CMAKE.
IDT-Freertos v1.1	FRQ_1.0.0			
IDT-Freertos v1.0	FRQ_1.0.0			

Unduh IDT untuk FreeRTOS

Topik ini menjelaskan opsi untuk mengunduh IDT untuk FreeRTOS. Anda dapat menggunakan salah satu tautan unduhan perangkat lunak berikut atau Anda dapat mengikuti petunjuk untuk mengunduh IDT secara terprogram.

> **Important**
>
> Per Oktober 2022, AWS IoT Device Tester untuk Kualifikasi AWS IoT FreerTOS (FRQ) 1.0 tidak menghasilkan laporan kualifikasi yang ditandatangani. Anda tidak dapat memenuhi syarat perangkat AWS IoT FreeRTOS baru untuk dicantumkan di AWS Katalog Perangkat Mitra melalui Program Kualifikasi Perangkat menggunakan versi AWS IDT FRQ 1.0. Meskipun Anda tidak dapat memenuhi syarat perangkat FreerTOS menggunakan IDT FRQ 1.0, Anda dapat terus menguji perangkat FreeRTOS Anda dengan FRQ 1.0. Kami menyarankan Anda menggunakan IDT FRQ 2.0 untuk memenuhi syarat dan daftar perangkat FreeRTOS di Katalog Perangkat Mitra.AWS

Topik

- Unduh IDT secara manual
- Unduh IDT secara terprogram

Dengan mengunduh perangkat lunak, Anda menyetujui Perjanjian AWS IoT Device Tester Lisensi yang terkandung dalam arsip unduhan.

> **Note**
>
> IDT tidak mendukung untuk dijalankan oleh beberapa pengguna dari lokasi bersama, seperti direktori NFS atau folder bersama jaringan Windows. Kami sarankan Anda mengekstraksi paket IDT ke drive lokal dan menjalankan biner IDT pada workstation lokal Anda.

Unduh IDT secara manual

Topik ini mencantumkan versi IDT yang didukung untuk FreeRTOS. Sebagai praktik terbaik, kami menyarankan Anda menggunakan versi terbaru AWS IoT Device Tester yang mendukung FreeRTOS versi target Anda. Rilis baru FreeRTOS mungkin mengharuskan Anda mengunduh versi baru. AWS

IoT Device Tester Anda menerima pemberitahuan ketika Anda memulai uji coba jika tidak AWS IoT Device Tester kompatibel dengan versi FreeRTOS yang Anda gunakan.

Lihat Versi yang didukung AWS IoT Device Tester

Unduh IDT secara terprogram

IDT menyediakan operasi API yang dapat Anda gunakan untuk mengambil URL tempat Anda dapat mengunduh IDT secara terprogram. Anda juga dapat menggunakan operasi API ini untuk memeriksa apakah Anda memiliki IDT versi terbaru. Operasi API ini memiliki titik akhir berikut.

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

Untuk memanggil operasi API ini, Anda harus memiliki izin untuk melakukan **`iot-device-tester:LatestIdt`** tindakan. Sertakan AWS tanda tangan Anda, dengan `iot-device-tester` sebagai Nama Layanan

Permintaan API

HostOs — Sistem operasi mesin host. Pilih dari salah satu pilihan berikut:

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

TestSuiteType — Jenis test suite. Pilih opsi berikut:

`FR`— IDT untuk FreeRTOS

ProductVersion

(Opsional) Versi FreeRTOS. Layanan ini mengembalikan versi IDT terbaru yang kompatibel untuk versi FreeRTOS tersebut. Jika Anda tidak menentukan opsi ini, layanan akan mengembalikan IDT versi terbaru.

Respon API

Respons API memiliki format berikut. `DownloadURL`Termasuk 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
    }
}
```

Contoh

Anda dapat mereferensikan contoh-contoh berikut untuk mengunduh IDT secara terprogram. Contoh-contoh ini menggunakan kredensil yang Anda simpan dalam variabel `AWS_ACCESS_KEY_ID` dan `AWS_SECRET_ACCESS_KEY` lingkungan. Untuk mengikuti praktik keamanan terbaik, jangan simpan kredensil Anda dalam kode Anda.

Example

Contoh: Unduh menggunakan cURL versi 7.75.0 atau yang lebih baru (Mac dan Linux)

Jika Anda memiliki cURL versi 7.75.0 atau yang lebih baru, Anda dapat menggunakan `aws-sigv4` flag untuk menandatangani permintaan API. Contoh ini menggunakan jq untuk mengurai URL unduhan dari respons.

> **Warning**
>
> `aws-sigv4`Bendera membutuhkan parameter kueri dari permintaan GET curl berada dalam urutan HostOs/ProductVersion/TestSuiteTypeatau HostOs/TestSuiteType. Pesanan yang tidak sesuai, akan mengakibatkan kesalahan mendapatkan tanda tangan yang tidak cocok untuk String Canonical dari API Gateway.
> Jika parameter ProductVersionopsional disertakan, Anda harus menggunakan versi produk yang didukung seperti yang didokumentasikan dalam versi yang didukung AWS IoT Device Tester untuk FreeRTOS.

- Ganti *`us-west-2`* dengan Anda Wilayah AWS. Untuk daftar kode Region, lihat Titik akhir Regional.
- Ganti *`linux`* dengan sistem operasi mesin host Anda.
- Ganti *`202107.00`* dengan versi FreeRTOS Anda.

```
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

Contoh: Unduh menggunakan cURL versi sebelumnya (Mac dan Linux)

Anda dapat menggunakan perintah cURL berikut dengan AWS tanda tangan yang Anda tanda tangani dan hitung. Untuk informasi selengkapnya tentang cara menandatangani dan menghitung AWS tanda tangan, lihat Menandatangani permintaan AWS API.

- Ganti *linux* dengan sistem operasi mesin host Anda.
- Ganti *Timestamp* dengan tanggal dan waktu, seperti**20220210T004606Z**.
- Ganti *Date* dengan tanggal, seperti**20220210**.
- Ganti *AWSRegion* dengan Anda Wilayah AWS. Untuk daftar kode Region, lihat Titik akhir Regional.
- Ganti *AWSSignature* dengan AWS tanda tangan yang Anda hasilkan.

```
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

Contoh: Unduh menggunakan skrip Python

Contoh ini menggunakan pustaka permintaan Python. Contoh ini diadaptasi dari contoh Python untuk Menandatangani permintaan AWS API di Referensi AWS Umum.

- Ganti *us-west-2* dengan wilayah Anda. Untuk daftar kode Region, lihat Titik akhir Regional.
- Ganti *linux* dengan sistem operasi mesin host Anda.

```
# 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)
```

IDT dengan suite kualifikasi FreeRTOS 2.0 (FRQ 2.0)

Suite kualifikasi FreeRTOS 2.0 adalah versi terbaru dari suite kualifikasi FreeRTOS. Kami merekomendasikan pengembang untuk menggunakan FRQ 2.0 karena terdiri dari kasus uji yang relevan untuk memenuhi syarat perangkat yang menjalankan pustaka FreerTOS Long Term Support (LTS).

IDT untuk FreeRTOS memverifikasi port FreeRTOS pada mikro-kontroler Anda, dan jika berkomunikasi secara efektif dengan. AWS IoT Secara khusus, ini memverifikasi antarmuka lapisan porting dengan pustaka FreerTOS, dan jika repositori pengujian FreeRTOS diimplementasikan dengan benar. Ini juga melakukan end-to-end tes dengan AWS IoT Core. Tes yang dijalankan oleh IDT untuk FreeRTOS didefinisikan dalam repositori FreerTOS. GitHub

IDT untuk FreeRTOS menjalankan tes sebagai aplikasi tertanam yang berkedip pada perangkat mikrokontroler yang sedang diuji. Gambar biner aplikasi termasuk FreeRTOS, antarmuka FreeRTOS yang diporting, dan driver perangkat papan. Tujuan dari pengujian ini adalah untuk memverifikasi bahwa antarmuka FreeRTOS yang di-porting berfungsi dengan benar di atas driver perangkat Anda.

IDT untuk FreeRTOS menghasilkan laporan pengujian yang dapat Anda kirimkan AWS IoT agar perangkat keras Anda terdaftar di AWS Katalog Perangkat Mitra. Untuk informasi selengkapnya, lihat Program Kualifikasi Perangkat AWS.

IDT untuk FreeRTOS berjalan pada komputer host (Windows, macOS, atau Linux) yang terhubung ke perangkat yang sedang diuji. IDT mengonfigurasi dan mengatur kasus uji dan mengumpulkan hasil. Ini juga menyediakan antarmuka baris perintah untuk mengelola menjalankan tes.

Untuk menguji perangkat Anda, IDT untuk FreeRTOS membuat sumber daya seperti AWS IoT hal-hal, grup FreeRTOS, fungsi Lambda. Untuk membuat sumber daya ini, IDT untuk FreeRTOS menggunakan kredensyal AWS yang dikonfigurasi untuk melakukan panggilan API `config.json` atas nama Anda. Sumber daya ini disediakan pada berbagai waktu selama tes.

Ketika Anda menjalankan IDT untuk FreeRTOS di komputer host Anda, ia melakukan langkah-langkah berikut:

1. Memuat dan memvalidasi konfigurasi perangkat dan kredensial Anda.
2. Melakukan tes yang dipilih dengan sumber daya lokal dan cloud yang diperlukan.
3. Membersihkan sumber daya lokal dan cloud.
4. Menghasilkan laporan tes yang menunjukkan jika forum Anda lulus tes yang diperlukan untuk kualifikasi.

Siapkan prasyarat kualifikasi LTS

Bagian ini menjelaskan prasyarat untuk menguji mikrokontroler dengan. AWS IoT Device Tester

Mempersiapkan kualifikasi FreeRTOS

> **Note**
>
> AWS IoT Device Tester untuk FreeRTOS sangat merekomendasikan menggunakan rilis patch terbaru dari versi Freertos-LTS terbaru.

IDT untuk FRQ 2.0 adalah kualifikasi untuk FreeRTOS. Sebelum menjalankan IDT FRQ 2.0 untuk kualifikasi, Anda harus menyelesaikan Kualifikasi papan Anda di Panduan Kualifikasi FreerTOS. Untuk mem-port pustaka, pengujian, dan penyiapan*`manifest.yml`*, lihat Mem-porting pustaka FreeRTOS di Panduan Porting FreerTOS. FRQ 2.0 berisi proses kualifikasi yang berbeda. Lihat Perubahan terbaru dalam kualifikasi dalam panduan kualifikasi FreeRTOS untuk detailnya.

Repositori Freertos-Libraries-Integration-Tests harus ada agar IDT dapat dijalankan. Lihat README.md tentang cara mengkloning dan mem-port repositori ini ke proyek sumber Anda. FreeRTOS-Libraries-Integration-Testsharus menyertakan yang `manifest.yml` terletak di root proyek Anda, agar IDT dapat berjalan.

> **Note**
>
> IDT tergantung pada implementasi repositori pengujian. `UNITY_OUTPUT_CHAR` Log keluaran pengujian dan log perangkat tidak boleh saling bertautan. Lihat Menerapkan bagian makro logging pustaka di Panduan Porting FreeRTOS untuk detail lebih lanjut.

Unduh IDT untuk FreeRTOS

Setiap versi FreeRTOS memiliki versi IDT yang sesuai untuk FreerTOS untuk melakukan tes kualifikasi. Unduh versi IDT yang sesuai untuk FreeRTOS dari versi yang didukung untuk FreeRTOS. AWS IoT Device Tester

Ekstrak IDT untuk FreeRTOS ke lokasi pada sistem file tempat Anda telah membaca dan menulis izin. Karena Microsoft Windows memiliki batas karakter untuk panjang jalur, ekstrak IDT untuk FreeRTOS ke direktori root seperti atau. `C:\` `D:\`

> **Note**
>
> Beberapa pengguna tidak boleh menjalankan IDT dari lokasi bersama, seperti direktori NFS atau folder bersama jaringan Windows. Ini akan mengakibatkan crash atau korupsi data. Kami menyarankan Anda mengekstrak paket IDT ke drive lokal.

Unduh Git

IDT harus menginstal Git sebagai prasyarat untuk memastikan integritas kode sumber.

Ikuti petunjuk dalam GitHubpanduan untuk menginstal Git. Untuk memverifikasi versi Git yang diinstal saat ini, masukkan perintah `git --version` di terminal.

> **Warning**
>
> IDT menggunakan Git untuk menyelaraskan dengan status direktori bersih atau kotor. Jika Git tidak diinstal, grup `FreeRTOSIntegrity` pengujian akan gagal, atau tidak akan berjalan seperti yang diharapkan. Jika IDT mengembalikan kesalahan seperti `git executable not found` ataugit `command not found`, instal atau instal ulang Git dan coba lagi.

Topik

- Buat AWS akun
- AWS IoT Device Tester kebijakan terkelola
- (Opsional) Instal AWS Command Line Interface

Buat AWS akun

> **Note**
>
> Suite kualifikasi IDT lengkap hanya didukung dalam hal berikut Wilayah AWS
>
> - AS Timur (Virginia Utara)
> - AS Barat (Oregon)
> - Asia Pasifik (Tokyo)

- Eropa (Irlandia)

Untuk menguji perangkat Anda, IDT untuk FreeRTOS membuat sumber daya seperti AWS IoT hal-hal, grup FreeRTOS, dan fungsi Lambda. Untuk membuat sumber daya tersebut, IDT untuk FreeRTOS mengharuskan Anda membuat dan mengonfigurasi AWS akun, dan kebijakan IAM yang memberikan izin IDT untuk FreeRTOS untuk mengakses sumber daya atas nama Anda saat menjalankan pengujian.

Langkah-langkah berikut adalah membuat dan mengkonfigurasi AWS akun Anda.

1. Jika Anda sudah memiliki AWS akun, lompat ke langkah berikutnya. Lain buat AWS akun.
2. Ikuti langkah-langkah dalam Membuat peran IAM. Jangan menambahkan izin atau kebijakan saat ini.
3. Untuk menjalankan tes kualifikasi OTA, lanjutkan ke Langkah 4. Lain pergi ke Langkah 5.
4. Lampirkan kebijakan inline izin IAM OTA ke peran IAM Anda.

   a.

   > **Important**
   >
   > Templat kebijakan berikut memberikan izin IDT untuk membuat peran, membuat kebijakan, dan melampirkan kebijakan ke peran. IDT untuk FreeRTOS menggunakan izin ini untuk pengujian yang membuat peran. Meskipun templat kebijakan tidak memberikan hak administrator kepada pengguna, izin tersebut dapat digunakan untuk mendapatkan akses administrator ke akun Anda AWS .

   b. Ikuti langkah-langkah di bawah ini untuk melampirkan izin yang diperlukan ke peran IAM Anda:

      i. Pada halaman Izin, pilih Tambahkan izin.

      ii. Pilih Buat kebijakan sebaris.

      iii. Pilih tab JSON dan salin izin berikut ke kotak teks JSON. Gunakan template di bawah Sebagian Besar Wilayah jika Anda tidak berada di wilayah Tiongkok. Jika Anda berada di wilayah Tiongkok, gunakan templat di bawah Wilayah Beijing dan 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

Template kebijakan berikut dapat digunakan di Wilayah Beijing dan 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. Setelah selesai, pilih Tinjau kebijakan.

v. Masukkan IDTFreeRTOSIAMPermissionssebagai nama kebijakan.

vi. Pilih Buat kebijakan.

5. Lampirkan AWSIoTDeviceTesterForFreeRTOSFullAkses ke peran IAM Anda.

   a. Untuk melampirkan izin yang diperlukan ke peran IAM Anda:

i. Pada halaman Izin, pilih Tambahkan izin.

ii. Pilih Lampirkan kebijakan.

iii. Cari kebijakan AWSIoTDeviceTesterForFreeRTOSFullAkses. Centang kotaknya.

b. Pilih Tambahkan izin.

6. Ekspor kredensi untuk IDT. Lihat Mendapatkan kredensi peran IAM untuk akses CLI untuk detailnya.

AWS IoT Device Tester kebijakan terkelola

Kebijakan `AWSIoTDeviceTesterForFreeRTOSFullAccess` terkelola berisi AWS IoT Device Tester izin berikut untuk pemeriksaan versi, fitur pembaruan otomatis, dan kumpulan metrik.

- `iot-device-tester:SupportedVersion`

  Memberikan AWS IoT Device Tester izin untuk mengambil daftar produk yang didukung, rangkaian pengujian, dan versi IDT.
- `iot-device-tester:LatestIdt`

  Memberikan AWS IoT Device Tester izin untuk mengambil versi IDT terbaru yang tersedia untuk diunduh.
- `iot-device-tester:CheckVersion`

  Memberikan AWS IoT Device Tester izin untuk memeriksa kompatibilitas versi untuk IDT, suite pengujian, dan produk.
- `iot-device-tester:DownloadTestSuite`

  Memberikan AWS IoT Device Tester izin untuk mengunduh pembaruan rangkaian pengujian.
- `iot-device-tester:SendMetrics`

  Memberikan AWS izin untuk mengumpulkan metrik tentang penggunaan AWS IoT Device Tester internal.

(Opsional) Instal AWS Command Line Interface

Anda mungkin lebih suka menggunakan AWS CLI untuk melakukan beberapa operasi. Jika Anda belum AWS CLI menginstal, ikuti petunjuk di Install the AWS CLI.

Konfigurasikan AWS CLI untuk AWS Wilayah yang ingin Anda gunakan dengan menjalankan aws configure dari baris perintah. Untuk informasi tentang AWS Wilayah yang mendukung IDT untuk FreeRTOS,AWS lihat Wilayah dan Titik Akhir. Untuk informasi selengkapnya, aws configure lihat Konfigurasi cepat dengan aws configure.

Tes pertama papan mikrokontroler Anda

Anda dapat menggunakan IDT untuk FreeRTOS untuk menguji implementasi pustaka FreeRTOS Anda. Setelah Anda mem-porting pustaka FreeRTOS untuk driver perangkat papan Anda, AWS IoT Device Tester gunakan untuk menjalankan tes kualifikasi di papan mikrokontroler Anda.

Tambahkan lapisan porting perpustakaan

Untuk mem-port FreeRTOS untuk perangkat Anda, lihat Panduan Porting FreeRTOS. Saat mengimplementasikan repositori pengujian FreerTOS dan mem-porting lapisan FreerTOS, Anda harus `manifest.yml` menyediakan jalur ke setiap pustaka, termasuk repositori tes. File ini akan berada di direktori root kode sumber Anda. Lihat instruksi file manifes untuk detailnya.

Konfigurasikan AWS kredensional Anda AWS IoT Device Tester untuk berkomunikasi dengan Cloud AWS

Anda perlu mengonfigurasi AWS kredensyal Anda AWS IoT Device Tester untuk berkomunikasi dengan Cloud. AWS Untuk informasi selengkapnya, lihat Menyiapkan AWS Kredensyal dan Wilayah untuk Pengembangan. AWS Kredensi yang valid ditentukan dalam file *devicetester_extract_location*`/devicetester_freertos_`*[win|mac|linux]*`/configs/config.json` konfigurasi.

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

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

`auth`Atribut `config.json` file memiliki bidang metode yang mengontrol AWS otentikasi, dan dapat dinyatakan sebagai file atau lingkungan. Menyetel bidang ke lingkungan akan menarik AWS kredensyal Anda dari variabel lingkungan mesin host Anda. Menyetel bidang ke file mengimpor profil tertentu dari file `.aws/credentials` konfigurasi.

Topik

- Buat kumpulan perangkat di IDT untuk FreeRTOS
- Konfigurasikan pengaturan build, flash, dan uji

Buat kumpulan perangkat di IDT untuk FreeRTOS

Perangkat yang akan diuji diatur dalam kumpulan perangkat. Setiap kumpulan perangkat terdiri dari satu atau lebih perangkat yang identik. Anda dapat mengonfigurasi IDT untuk FreeRTOS untuk menguji satu perangkat, atau beberapa perangkat dalam kumpulan. Untuk mempercepat proses kualifikasi, IDT untuk FreerTOS dapat menguji perangkat dengan spesifikasi yang sama secara paralel. Ini menggunakan metode round-robin untuk mengeksekusi kelompok uji yang berbeda pada setiap perangkat dalam kumpulan perangkat.

`device.json`File memiliki array di tingkat atasnya. Setiap atribut array adalah kumpulan perangkat baru. Setiap kumpulan perangkat memiliki atribut array perangkat, yang memiliki beberapa perangkat yang dideklarasikan. Dalam template, ada kumpulan perangkat dan hanya satu perangkat di kumpulan perangkat itu. Anda dapat menambahkan satu atau beberapa perangkat ke kumpulan perangkat dengan mengedit `devices` bagian `device.json` templat di `configs` folder.

> **Note**
>
> Semua perangkat di kolam yang sama harus memiliki spesifikasi teknis dan SKU yang sama. Untuk mengaktifkan build paralel dari kode sumber untuk grup pengujian yang berbeda, IDT untuk FreerTOS menyalin kode sumber ke folder hasil di dalam folder yang diekstrak IDT untuk FreeRTOS. Anda harus mereferensikan jalur kode sumber dalam perintah build atau flash menggunakan `testdata.sourcePath` variabel. IDT untuk FreeRTOS menggantikan variabel ini dengan jalur sementara dari kode sumber yang disalin. Untuk informasi selengkapnya, lihat IDT untuk variabel FreeRTOS.

Berikut ini adalah contoh `device.json` file yang digunakan untuk membuat kumpulan perangkat dengan beberapa perangkat.

```
[
    {
        "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"
              }
            ]
          }
        ]
    }
]
```

Atribut berikut digunakan dalam `device.json` file:

`id`

ID alfanumerik yang ditentukan pengguna yang secara unik mengidentifikasi kumpulan perangkat. Perangkat yang termasuk dalam kolam harus dari jenis yang sama. Saat serangkaian pengujian sedang berjalan, perangkat di kolam digunakan untuk memparalelkan beban kerja.

`sku`

Nilai alfanumerik yang secara unik mengidentifikasi papan yang Anda uji. SKU digunakan untuk melacak forum yang berkualitas.

> **Note**
>
> Jika Anda ingin mencantumkan papan Anda di Katalog Perangkat AWS Mitra, SKU yang Anda tentukan di sini harus cocok dengan SKU yang Anda gunakan dalam proses pencatatan.

`features`

Array yang berisi fitur yang didukung perangkat. AWS IoT Device Tester menggunakan informasi ini untuk memilih tes kualifikasi yang akan dijalankan.

Nilai yang didukung adalah:

`Wifi`

Menunjukkan apakah papan Anda memiliki kemampuan Wi-Fi.

`Cellular`

Menunjukkan apakah papan Anda memiliki kemampuan seluler.

`PKCS11`

Menunjukkan algoritma kriptografi kunci publik yang didukung papan. PKCS11 diperlukan untuk kualifikasi. Nilai yang didukung adalah`ECC`,`RSA`, dan`Both`. `Both`menunjukkan papan mendukung keduanya `ECC` dan`RSA`.

`KeyProvisioning`

Menunjukkan metode penulisan sertifikat klien X.509 tepercaya ke papan Anda.

Nilai yang valid adalah`Import`,`Onboard`, `Both` dan`No`. `Onboard`,`Both`, atau penyediaan `No` kunci diperlukan untuk kualifikasi. `Import`sendiri bukan pilihan yang valid untuk kualifikasi.

- Gunakan `Import` hanya jika papan Anda mengizinkan impor kunci pribadi. Memilih `Import` bukanlah konfigurasi yang valid untuk kualifikasi dan harus digunakan hanya untuk tujuan pengujian, khususnya dengan kasus PKCS11 uji. `Onboard`, `Both` atau `No` diperlukan untuk kualifikasi.
- Gunakan `Onboard` jika papan Anda mendukung kunci pribadi terpasang (misalnya, jika perangkat Anda memiliki elemen aman, atau jika Anda lebih suka membuat key pair dan sertifikat perangkat Anda sendiri). Pastikan Anda menambahkan `secureElementConfig` elemen di setiap bagian perangkat dan meletakkan jalur absolut ke file kunci publik di `publicKeyAsciiHexFilePath` bidang.
- Gunakan `Both` jika papan Anda mendukung pengimporan kunci pribadi dan pembuatan kunci on-board untuk penyediaan kunci.
- Gunakan `No` jika papan Anda tidak mendukung penyediaan kunci. `No`hanya opsi yang valid ketika perangkat Anda juga telah disediakan sebelumnya.

OTA

Menunjukkan apakah papan Anda mendukung fungsionalitas pembaruan over-the-air (OTA). `OtaDataPlaneProtocol`Atribut menunjukkan protokol dataplane OTA mana yang didukung perangkat. OTA dengan protokol jalur data HTTP atau MQTT diperlukan untuk kualifikasi. Untuk melewati menjalankan pengujian OTA saat pengujian, setel fitur OTA ke `No` dan `OtaDataPlaneProtocol` atribut ke`None`. Ini tidak akan menjadi kualifikasi.

BLE

Menunjukkan apakah papan Anda mendukung Bluetooth Low Energy (BLE).

devices.id

Pengenal unik yang ditetapkan pengguna untuk perangkat Anda.

devices.connectivity.serialPort

Port serial komputer host digunakan untuk terhubung ke perangkat yang sedang diuji.

devices.secureElementConfig.PublicKeyAsciiHexFilePath

Diperlukan jika papan Anda TIDAK `pre-provisioned` atau tidak `PublicDeviceCertificateArn` disediakan. Karena `Onboard` merupakan jenis Penyediaan Kunci yang diperlukan, bidang ini saat ini diperlukan untuk grup uji FullTransportInterface TLS. Jika perangkat Anda`pre-provisioned`, `PublicKeyAsciiHexFilePath` bersifat opsional dan tidak perlu disertakan.

Blok berikut adalah jalur absolut ke file yang berisi hex byte kunci publik yang diekstrak dari `Onboard` kunci pribadi.

```
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
```

Jika kunci publik Anda dalam format.der, Anda dapat menyandikan kunci publik secara langsung untuk menghasilkan file hex.

Untuk menghasilkan file hex dari kunci publik.der, masukkan perintah berikut: xxd

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

Jika kunci publik Anda dalam format.pem, Anda dapat mengekstrak header dan footer yang dikodekan base64 dan mendekodekannya ke dalam format biner. Kemudian, Anda hex menyandikan string biner untuk menghasilkan file hex.

Untuk menghasilkan file hex untuk kunci publik.pem, lakukan hal berikut:

1. Jalankan base64 perintah berikut untuk menghapus header dan footer base64 dari kunci publik. Kunci yang diterjemahkan, bernama`base64key`, kemudian output ke file: `pubkey.der`

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

2. Jalankan xxd perintah berikut untuk mengonversi `pubkey.der` ke format hex. Kunci yang dihasilkan disimpan sebagai *`outFile`*

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

`devices.secureElementConfig.PublicDeviceCertificateArn`

ARN sertifikat dari elemen aman Anda yang diunggah ke. AWS IoT CoreUntuk informasi tentang mengunggah sertifikat AWS IoT Core, lihat sertifikat klien X.509 di Panduan Pengembang.AWS IoT

`devices.secureElementConfig.SecureElementSerialNumber`

(Opsional) Nomor seri elemen aman. Nomor seri secara opsional digunakan untuk membuat sertifikat perangkat untuk penyediaan kunci JITR.

`devices.secureElementConfig.preProvisioned`

(Opsional) Setel ke “Ya” jika perangkat memiliki elemen aman yang telah disediakan sebelumnya dengan kredenal terkunci, yang tidak dapat mengimpor, membuat, atau menghancurkan objek. Jika atribut ini disetel ke Ya, Anda harus memberikan label pkcs11 yang sesuai.

`devices.secureElementConfig.pkcs11JITPCodeVerifyRootCertSupport`

(Opsional) Setel ke Ya jika PKCS11 implementasi inti perangkat mendukung penyimpanan untuk JITP. Ini akan memungkinkan pengujian JITP saat `codeverify` menguji inti PKCS 11, dan memerlukan kunci verifikasi kode, sertifikat JITP, dan label PKCS 11 sertifikat root untuk disediakan.

`identifiers`

(Opsional) Sebuah array pasangan nama-nilai arbitrer. Anda dapat menggunakan nilai-nilai ini dalam perintah build dan flash yang dijelaskan di bagian selanjutnya.

Konfigurasikan pengaturan build, flash, dan uji

IDT untuk FreeRTOS membangun dan mem-flash tes ke papan Anda secara otomatis. Untuk mengaktifkan ini, Anda harus mengkonfigurasi IDT untuk menjalankan perintah build dan flash untuk perangkat keras Anda. Pengaturan perintah build dan flash dikonfigurasi dalam file `userdata.json` template yang terletak di `config` folder.

Konfigurasikan pengaturan untuk perangkat pengujian

Pengaturan build, flash, dan test dibuat dalam `configs/userdata.json` file. Contoh JSON berikut menunjukkan bagaimana Anda dapat mengkonfigurasi IDT untuk FreeRTOS untuk menguji beberapa perangkat:

```
{
    "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>"
    }
  }
```

Berikut daftar atribut yang digunakan dalam`userdata.json`:

`sourcePath`

Jalur ke root kode sumber FreeRTOS yang diporting.

`retainModifiedSourceDirectories`

(Opsional) Memeriksa apakah akan mempertahankan direktori sumber yang dimodifikasi yang digunakan selama pembuatan dan flashing untuk tujuan debugging. Jika disetel ke`true`, direktori sumber yang dimodifikasi diberi nama RetainedSrc dan ditemukan dalam folder log hasil di setiap grup uji yang dijalankan. Jika tidak disertakan, bidang default ke. `false`

`freeRTOSTestParamConfigPath`

Jalur ke `test_param_config.h` file untuk FreeRTOS-Libraries-Integration-Tests integrasi. File ini harus menggunakan variabel `{{testData.sourcePath}}` placeholder untuk membuatnya relatif terhadap root kode sumber. AWS IoT Device Tester menggunakan parameter dalam file ini untuk mengkonfigurasi tes.

`freeRTOSTestExecutionConfigPath`

Jalur ke `test_execution_config.h` file untuk FreeRTOS-Libraries-Integration-Tests integrasi. File ini harus menggunakan variabel `{{testData.sourcePath}}` placeholder untuk membuatnya relatif terhadap root repositori. AWS IoT Device Tester menggunakan file ini untuk mengontrol tes mana yang harus dijalankan.

`freeRTOSVersion`

Versi FreeRTOS termasuk versi patch yang digunakan dalam implementasi Anda. Lihat Versi FreeRTOS yang didukung AWS IoT Device Tester untuk versi FreeRTOS yang kompatibel dengan untuk FreeRTOS. AWS IoT Device Tester

`buildTool`

Perintah untuk membangun kode sumber Anda. Semua referensi ke jalur kode sumber dalam perintah build harus diganti dengan AWS IoT Device Tester variabel`{{testData.sourcePath}}`. Gunakan `{{config.idtRootPath}}` placeholder untuk mereferensikan skrip build relatif terhadap jalur AWS IoT Device Tester root.

`flashTool`

Perintah untuk mem-flash gambar ke perangkat Anda. Semua referensi ke jalur kode sumber dalam perintah flash harus diganti dengan AWS IoT Device Tester variabel`{{testData.sourcePath}}`. Gunakan `{{config.idtRootPath}}` placeholder untuk referensi skrip flash relatif terhadap jalur AWS IoT Device Tester root.

> **Note**
>
> Struktur pengujian integrasi baru dengan FRQ 2.0 tidak memerlukan variabel jalur seperti `{{enableTests}}` dan`{{buildImageName}}`. Tes OTA End to End dijalankan dengan template konfigurasi yang disediakan di FreeRTOS-Libraries-Integration-Tests GitHub repositori. Jika file dalam GitHub repositori ada di proyek sumber induk Anda, kode sumber tidak berubah di antara pengujian. Jika image build yang berbeda untuk OTA End to End diperlukan, Anda harus membuat gambar ini dalam skrip build dan menentukannya di `userdata.json` file yang ditentukan di bawah`otaConfiguration`.

`testStartDelayms`

Menentukan berapa milidetik pelari uji FreeRTOS akan menunggu sebelum mulai menjalankan tes. Ini dapat berguna jika perangkat yang diuji mulai mengeluarkan informasi pengujian penting sebelum IDT memiliki kesempatan untuk terhubung dan mulai masuk karena jaringan atau masalah latensi lainnya. Nilai ini hanya berlaku untuk kelompok uji FreeRTOS, dan tidak untuk kelompok uji lain yang tidak menggunakan pelari uji FreeRTOS, seperti tes OTA. Jika Anda menerima kesalahan terkait dengan 10 yang diharapkan tetapi menerima 5, bidang ini harus disetel ke 5000.

`echoServerConfiguration`

Konfigurasi untuk mengatur server gema untuk tes TLS. Bidang ini wajib diisi.

`keyGenerationMethod`

Server gema dikonfigurasi dengan opsi ini. Pilihannya adalah EC, atau RSA.

`serverPort`

Nomor port tempat server echo berjalan.

`otaConfiguration`

Konfigurasi untuk tes OTA PAL dan OTA E2E. Bidang ini wajib diisi.

`otaE2EFirmwarePath`

Jalur ke gambar bin OTA yang digunakan IDT untuk pengujian OTA End to End.

`otaPALCertificatePath`

Jalur ke sertifikat untuk uji OTA PAL pada perangkat. Ini digunakan untuk memverifikasi tanda tangan. Misalnya, ecdsa-sha256-signer.crt.pem.

`deviceFirmwarePath`

Jalur ke nama kode keras untuk gambar firmware untuk boot. Jika perangkat Anda TIDAK menggunakan sistem file untuk boot firmware, tentukan bidang ini sebagai `'NA'`. Jika perangkat Anda menggunakan sistem file untuk boot firmware, tentukan jalur atau nama ke image boot firmware.

`codeSigningConfiguration`

`signingMethod`

Metode penandatanganan kode. Nilai yang mungkin adalah AWS atau Kustom.

> **Note**
>
> Untuk Wilayah Beijing dan Ningxia, gunakan Custom. AWS penandatanganan kode tidak didukung di wilayah itu.

`signerHashingAlgorithm`

Algoritma hashing didukung pada perangkat. Nilai yang mungkin adalah `SHA1` atau`SHA256`.

`signerSigningAlgorithm`

Algoritma penandatanganan didukung pada perangkat. Nilai yang mungkin adalah `RSA` atau`ECDSA`.

`signerCertificate`

Sertifikat tepercaya yang digunakan untuk OTA. Untuk metode penandatanganan AWS kode, gunakan Amazon Resource Name (ARN) untuk sertifikat tepercaya yang diunggah ke Certificate Manager AWS . Untuk metode penandatanganan kode kustom, gunakan jalur absolut ke file sertifikat penandatangan. Untuk informasi tentang membuat sertifikat tepercaya, lihat Membuat sertifikat penandatanganan kode.

`untrustedSignerCertificate`

ARN atau filepath untuk sertifikat kedua yang digunakan dalam beberapa tes OTA sebagai sertifikat yang tidak tepercaya. Untuk informasi tentang membuat sertifikat, lihat Membuat sertifikat penandatanganan kode.

`signerCertificateFileName`

Nama file sertifikat penandatanganan kode pada perangkat. Nilai ini harus sesuai dengan nama file yang Anda berikan saat menjalankan `aws acm import-certificate` perintah.

`compileSignerCertificate`

Nilai Boolean yang menentukan status sertifikat verifikasi tanda tangan. Nilai yang valid adalah `true` dan `false`.

Tetapkan nilai ini ke true jika sertifikat verifikasi tanda tangan penandatangan kode tidak disediakan atau di-flash. Itu harus dikompilasi ke dalam proyek. AWS IoT Device Tester mengambil sertifikat tepercaya dan mengkompilasinya menjadi. `aws_codesigner_certificate.h`

`signerPlatform`

Algoritma penandatanganan dan hashing yang digunakan AWS Code Signer saat membuat pekerjaan pembaruan OTA. Saat ini, nilai yang mungkin untuk bidang ini adalah `AmazonFreeRTOS-TI-CC3220SF` dan`AmazonFreeRTOS-Default`.

- Pilih `AmazonFreeRTOS-TI-CC3220SF` jika `SHA1` dan`RSA`.
- Pilih `AmazonFreeRTOS-Default` jika `SHA256` dan`ECDSA`.

- Jika Anda membutuhkan `SHA256 | RSA` atau `SHA1 | ECDSA` untuk konfigurasi Anda, hubungi kami untuk dukungan lebih lanjut.
- Konfigurasikan `signCommand` jika Anda memilih `Custom` untuk`signingMethod`.

`signCommand`

Dua placeholder `{{inputImageFilePath}}` dan `{{outputSignatureFilePath}}` diperlukan dalam perintah. `{{inputImageFilePath}}`adalah jalur file gambar yang dibangun oleh IDT untuk ditandatangani. `{{outputSignatureFilePath}}` adalah jalur file dari tanda tangan yang akan dihasilkan oleh skrip.

`pkcs11LabelConfiguration`

PKCS11 konfigurasi label memerlukan setidaknya satu set label label sertifikat perangkat, label kunci publik, dan label kunci pribadi untuk menjalankan grup PKCS11 pengujian. PKCS11 Label yang diperlukan didasarkan pada konfigurasi perangkat Anda dalam `device.json` file. Jika pre-provisioned disetel ke Yes in`device.json`, maka label yang diperlukan harus salah satu di bawah ini tergantung pada apa yang dipilih untuk fitur tersebut. PKCS11

- `PreProvisionedEC`
- `PreProvisionedRSA`

Jika pra-penyediaan disetel ke No in`device.json`, maka label yang diperlukan adalah:

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

Tiga label berikut diperlukan hanya jika Anda memilih Ya untuk `pkcs11JITPCodeVerifyRootCertSupport` dalam `device.json` file Anda.

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

Nilai untuk bidang ini harus sesuai dengan nilai yang ditentukan dalam Panduan Porting FreerTOS.

`pkcs11LabelDevicePrivateKeyForTLS`

(Opsional) Label ini digunakan untuk label PKCS #11 dari kunci pribadi. Untuk perangkat dengan dukungan onboard dan impor dari penyediaan kunci, label ini digunakan untuk pengujian. Label ini mungkin berbeda dari yang ditentukan untuk kasus yang telah disediakan

sebelumnya. Jika Anda memiliki penyediaan kunci yang disetel ke Tidak dan disetel sebelumnya ke Ya, di`device.json`, ini akan menjadi tidak terdefinisi.

`pkcs11LabelDevicePublicKeyForTLS`

(Opsional) Label ini digunakan untuk label PKCS #11 dari kunci publik. Untuk perangkat dengan dukungan onboard dan impor dari penyediaan kunci, label ini digunakan untuk pengujian. Label ini mungkin berbeda dari yang ditentukan untuk kasus yang telah disediakan sebelumnya. Jika Anda memiliki penyediaan kunci yang disetel ke Tidak dan disetel sebelumnya ke Ya, di`device.json`, ini akan menjadi tidak terdefinisi.

`pkcs11LabelDeviceCertificateForTLS`

(Opsional) Label ini digunakan untuk label PKCS #11 dari sertifikat perangkat. Untuk perangkat dengan dukungan onboard dan impor dari penyediaan kunci, label ini akan digunakan untuk pengujian. Label ini mungkin berbeda dari yang ditentukan untuk kasus yang telah disediakan sebelumnya. Jika Anda memiliki penyediaan kunci yang disetel ke Tidak dan disetel sebelumnya ke Ya, di`device.json`, ini akan menjadi tidak terdefinisi.

`pkcs11LabelPreProvisionedECDevicePrivateKeyForTLS`

(Opsional) Label ini digunakan untuk label PKCS #11 dari kunci pribadi. Untuk perangkat dengan elemen aman atau keterbatasan perangkat keras, ini akan memiliki label berbeda untuk mempertahankan AWS IoT kredensyal. Jika perangkat Anda mendukung pra-penyediaan dengan kunci EC, berikan label ini. Ketika preProvisionEd disetel ke Yes in`device.json`, label ini,`pkcs11LabelPreProvisionedRSADevicePrivateKeyForTLS`, atau keduanya harus disediakan. Label ini mungkin berbeda dari yang ditentukan untuk kasus onboard dan impor.

`pkcs11LabelPreProvisionedECDevicePublicKeyForTLS`

(Opsional) Label ini digunakan untuk label PKCS #11 dari kunci publik. Untuk perangkat dengan elemen aman atau keterbatasan perangkat keras, ini akan memiliki label berbeda untuk mempertahankan AWS IoT kredensyal. Jika perangkat Anda mendukung pra-penyediaan dengan kunci EC, berikan label ini. Ketika preProvisionEd disetel ke Yes in`device.json`, label ini,`pkcs11LabelPreProvisionedRSADevicePublicKeyForTLS`, atau keduanya harus disediakan. Label ini mungkin berbeda dari yang ditentukan untuk kasus onboard dan impor.

`pkcs11LabelPreProvisionedECDeviceCertificateForTLS`

(Opsional) Label ini digunakan untuk label PKCS #11 dari sertifikat perangkat. Untuk perangkat dengan elemen aman atau keterbatasan perangkat keras,

ini akan memiliki label berbeda untuk mempertahankan AWS IoT kredensyal. Jika perangkat Anda mendukung pra-penyediaan dengan kunci EC, berikan label ini. Ketika preProvisionEd disetel ke Yes in`device.json`, label ini,`pkcs11LabelPreProvisionedRSADeviceCertificateForTLS`, atau keduanya harus disediakan. Label ini mungkin berbeda dari yang ditentukan untuk kasus onboard dan impor.

`pkcs11LabelPreProvisionedRSADevicePrivateKeyForTLS`

(Opsional) Label ini digunakan untuk label PKCS #11 dari kunci pribadi. Untuk perangkat dengan elemen aman atau keterbatasan perangkat keras, ini akan memiliki label berbeda untuk mempertahankan AWS IoT kredensyal. Jika perangkat Anda mendukung pra-penyediaan dengan kunci RSA, berikan label ini. Ketika preProvisionEd disetel ke Yes in`device.json`, label ini,`pkcs11LabelPreProvisionedECDevicePrivateKeyForTLS`, atau keduanya harus disediakan.

`pkcs11LabelPreProvisionedRSADevicePublicKeyForTLS`

(Opsional) Label ini digunakan untuk label PKCS #11 dari kunci publik. Untuk perangkat dengan elemen aman atau keterbatasan perangkat keras, ini akan memiliki label berbeda untuk mempertahankan AWS IoT kredensyal. Jika perangkat Anda mendukung pra-penyediaan dengan kunci RSA, berikan label ini. Ketika preProvisionEd disetel ke Yes in`device.json`, label ini,`pkcs11LabelPreProvisionedECDevicePublicKeyForTLS`, atau keduanya harus disediakan.

`pkcs11LabelPreProvisionedRSADeviceCertificateForTLS`

(Opsional) Label ini digunakan untuk label PKCS #11 dari sertifikat perangkat. Untuk perangkat dengan elemen aman atau keterbatasan perangkat keras, ini akan memiliki label berbeda untuk mempertahankan AWS IoT kredensyal. Jika perangkat Anda mendukung pra-penyediaan dengan kunci RSA, berikan label ini. Ketika preProvisionEd disetel ke Yes in`device.json`, label ini,`pkcs11LabelPreProvisionedECDeviceCertificateForTLS`, atau keduanya harus disediakan.

`pkcs11LabelCodeVerifyKey`

(Opsional) Label ini digunakan untuk label PKCS #11 dari kunci verifikasi kode. Jika perangkat Anda memiliki dukungan penyimpanan PKCS #11 dari sertifikat JITP, kunci verifikasi kode, dan sertifikat root, berikan label ini. Ketika `pkcs11JITPCodeVerifyRootCertSupport` in `device.json` diatur ke Ya, label ini harus disediakan.

`pkcs11LabelJITPCertificate`

(Opsional) Label ini digunakan untuk label PKCS #11 dari sertifikat JITP. Jika perangkat Anda memiliki dukungan penyimpanan PKCS #11 dari sertifikat JITP, kunci verifikasi kode, dan sertifikat root, berikan label ini. Ketika `pkcs11JITPCodeVerifyRootCertSupport` in `device.json` diatur ke Ya, label ini harus disediakan.

IDT untuk variabel FreeRTOS

Perintah untuk membangun kode Anda dan mem-flash perangkat mungkin memerlukan konektivitas atau informasi lain tentang perangkat Anda agar berhasil berjalan. AWS IoT Device Tester memungkinkan Anda untuk mereferensikan informasi perangkat dalam flash dan membangun perintah menggunakan JsonPath. Dengan menggunakan JsonPath ekspresi sederhana, Anda dapat mengambil informasi yang diperlukan yang ditentukan dalam `device.json` file Anda.

Variabel jalur

IDT untuk FreeRTOS mendefinisikan variabel jalur berikut yang dapat digunakan dalam baris perintah dan file konfigurasi:

`{{testData.sourcePath}}`

Memperluas ke jalur kode sumber. Jika Anda menggunakan variabel ini, itu harus digunakan dalam perintah flash dan build.

`{{device.connectivity.serialPort}}`

Memperluas ke port serial.

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

Memperluas ke nomor seri perangkat Anda.

`{{config.idtRootPath}}`

Memperluas ke jalur AWS IoT Device Tester root.

UI untuk IDT untuk suite kualifikasi FreeRTOS 2.0 (FRQ 2.0)

AWS IoT Device Tester untuk FreeRTOS (IDT untuk FreeRTOS) termasuk antarmuka pengguna berbasis web (UI) di mana Anda dapat berinteraksi dengan aplikasi baris perintah IDT dan file konfigurasi terkait. Anda menggunakan IDT untuk FreeRTOS UI untuk membuat konfigurasi baru,

atau memodifikasi yang sudah ada, untuk perangkat Anda. Anda juga dapat menggunakan UI untuk memanggil aplikasi IDT dan menjalankan tes FreeRTOS terhadap perangkat Anda.

Untuk informasi tentang cara menggunakan baris perintah untuk menjalankan tes kualifikasi, lihatTes pertama papan mikrokontroler Anda.

Bagian ini menjelaskan prasyarat untuk IDT untuk FreeRTOS UI dan cara menjalankan pengujian kualifikasi dari UI.

Topik

- Mengatur prasyarat IDT
- Konfigurasikan AWS kredensional untuk menggunakan UI IDT
- Buka IDT untuk FreeRTOS UI
- Buat konfigurasi baru
- Ubah konfigurasi yang ada
- Jalankan tes kualifikasi

Mengatur prasyarat IDT

Untuk menjalankan tes melalui AWS IoT Device Tester (IDT) untuk FreeRTOS UI, Anda harus menyelesaikan prasyarat pada Siapkan prasyarat kualifikasi LTS halaman untuk Kualifikasi Freertos IDT (FRQ) 2.x.

Konfigurasikan AWS kredensional untuk menggunakan UI IDT

Anda harus mengonfigurasi kredensi pengguna IAM Anda untuk AWS pengguna yang Anda buat. Buat AWS akun Anda dapat menentukan kredensial Anda dengan salah satu dari dua cara berikut:

- Di file kredensial
- Sebagai variabel lingkungan

Konfigurasikan AWS kredensional dengan file kredensial

IDT menggunakan file kredensials yang sama sebagai AWS CLI. Untuk informasi selengkapnya, lihat File konfigurasi dan kredensial.

Lokasi file kredensial bervariasi berdasarkan sistem operasi yang Anda gunakan:

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

Tambahkan AWS kredensi Anda ke `credentials` file dalam format berikut:

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

> **Note**
>
> Jika Anda tidak menggunakan `default` AWS profil, Anda harus menentukan nama profil di IDT untuk FreeRTOS UI. Untuk informasi selengkapnya tentang profil, lihat Konfigurasi dan setelan file kredensi.

Konfigurasikan AWS kredensional dengan variabel lingkungan

Variabel lingkungan adalah variabel yang dikelola oleh sistem operasi dan digunakan oleh perintah sistem. Mereka tidak disimpan jika Anda menutup sesi SSH. IDT untuk FreeRTOS UI menggunakan variabel `AWS_SECRET_ACCESS_KEY` dan lingkungan untuk menyimpan `AWS_ACCESS_KEY_ID` kredensi Anda. AWS

Untuk mengatur variabel ini di Linux, macOS, atau Unix, gunakan export:

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

Untuk menetapkan variabel ini di Windows, gunakan set:

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

Buka IDT untuk FreeRTOS UI

Topik ini menjelaskan cara membuka IDT untuk FreeRTOS UI untuk menggunakan suite kualifikasi FreeRTOS.

Untuk membuka IDT untuk FreeRTOS UI

1. Unduh IDT yang didukung untuk versi FreeRTOS. Kemudian ekstrak arsip yang diunduh ke direktori yang telah Anda baca dan tulis izin.
2. Arahkan ke IDT untuk direktori instalasi FreerTOS:

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

3. Jalankan perintah berikut untuk membuka IDT untuk FreeRTOS UI:

   Linux

   ```
   .devicetester_ui_linux_x86-64
   ```

   Windows

   ```
   ./devicetester_ui_win_x64-64
   ```

   macOS

   ```
   ./devicetester_ui_mac_x86-64
   ```

   > **Note**
   >
   > Di macOS, untuk memungkinkan sistem Anda menjalankan UI, buka System Preferences -> Security & Privacy. Ketika Anda menjalankan tes, Anda mungkin perlu melakukan ini tiga kali lagi. ini

   IDT untuk FreeRTOS UI terbuka di browser default Anda. Tiga versi utama terbaru dari browser berikut mendukung UI:

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

> **Note**
>
> Untuk pengalaman yang lebih baik, kami merekomendasikan Google Chrome atau Mozilla Firefox untuk mengakses IDT untuk FreeRTOS UI. Microsoft Internet Explorer tidak didukung oleh UI.

> **Important**
>
> Anda harus mengonfigurasi AWS kredensional Anda sebelum membuka UI. Jika Anda belum mengonfigurasi kredensialnya, tutup jendela browser IDT untuk FreeRTOS UI, ikuti langkah-langkahnyaKonfigurasikan AWS kredensional untuk menggunakan UI IDT, lalu buka kembali IDT untuk FreeRTOS UI.

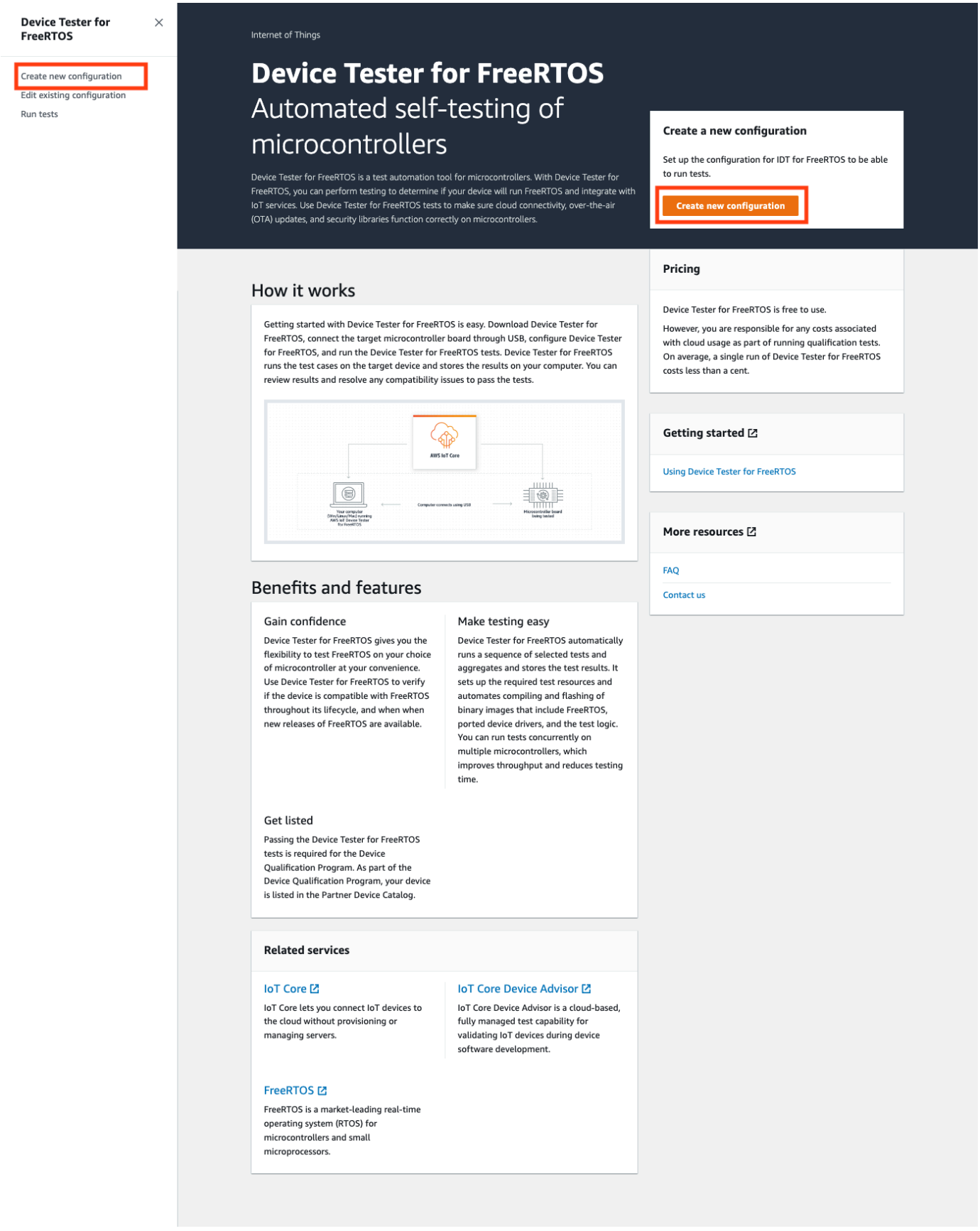
Buat konfigurasi baru

Jika Anda pengguna pertama kali, Anda harus membuat konfigurasi baru untuk mengatur file konfigurasi JSON yang IDT untuk FreeRTOS memerlukan untuk menjalankan pengujian. Anda kemudian dapat menjalankan tes atau memodifikasi konfigurasi yang dibuat.

Untuk contoh`config.json`,`device.json`, dan `userdata.json` file, lihatTes pertama papan mikrokontroler Anda.

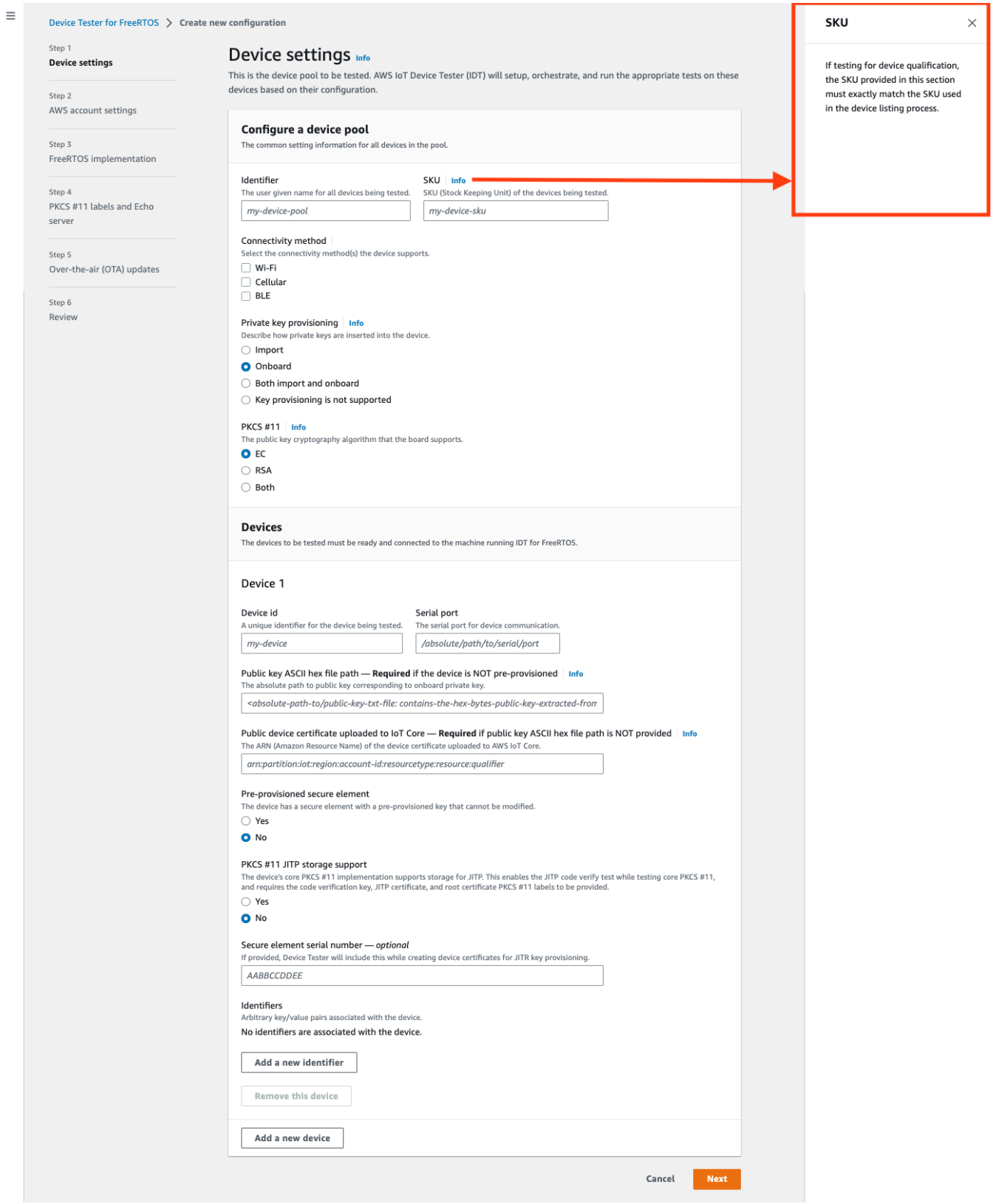
Untuk membuat konfigurasi baru

1. Di IDT untuk FreeRTOS UI, buka menu navigasi, dan pilih Buat konfigurasi baru.

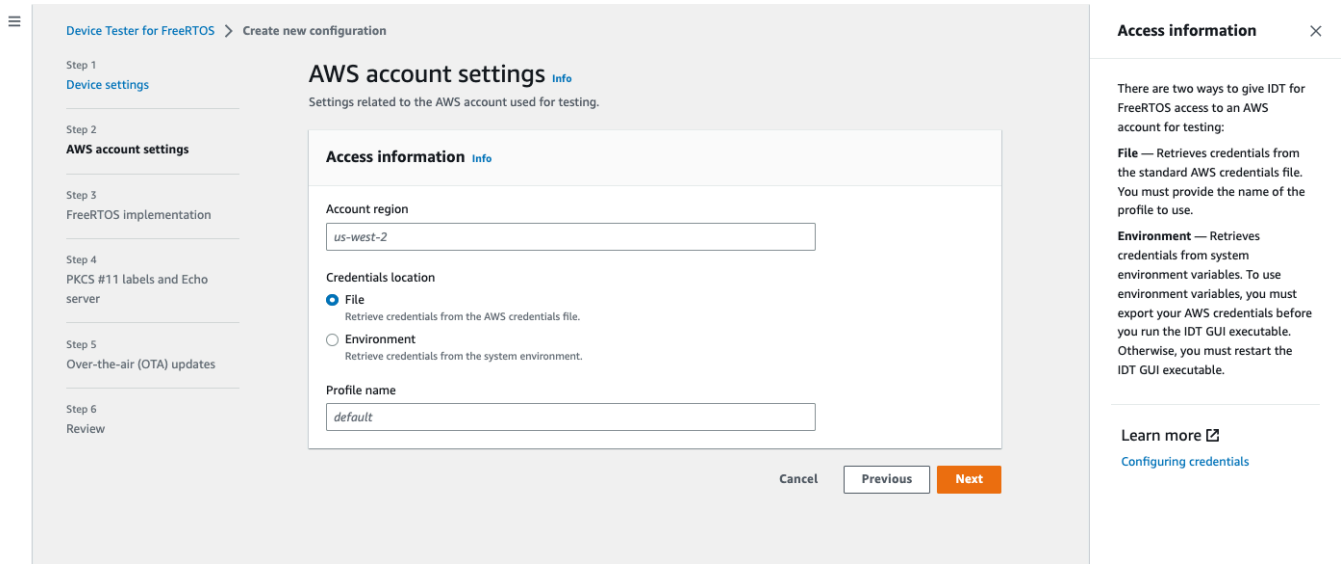

Device Tester for FreeRTOS
Create new configuration
Edit existing configuration
Run tests
Internet of Things
Device Tester for FreeRTOS
Automated self-testing of microcontrollers
Device Tester for FreeRTOS is a test automation tool for microcontrollers. With Device Tester for FreeRTOS, you can perform 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. Ikuti panduan konfigurasi untuk masuk ke pengaturan konfigurasi IDT yang digunakan untuk menjalankan tes kualifikasi. Wizard mengkonfigurasi pengaturan berikut dalam file konfigurasi JSON yang terletak di direktori. *devicetester-extract-location*`/config`

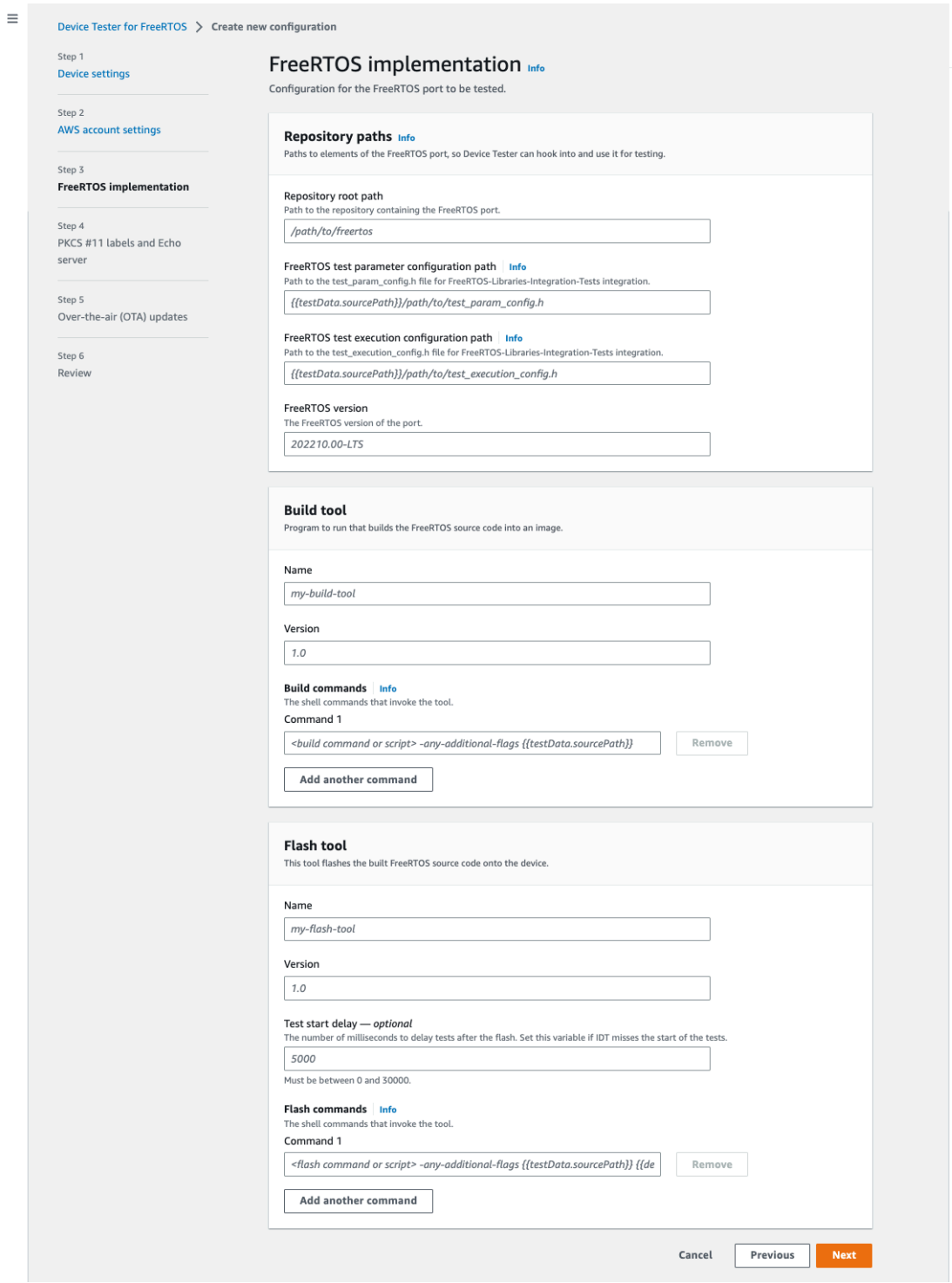
   - Pengaturan perangkat — Pengaturan kumpulan perangkat untuk perangkat yang akan diuji. Pengaturan ini dikonfigurasi di `sku` bidang `id` dan, dan perangkat memblokir untuk kumpulan perangkat dalam `config.json` file.


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.

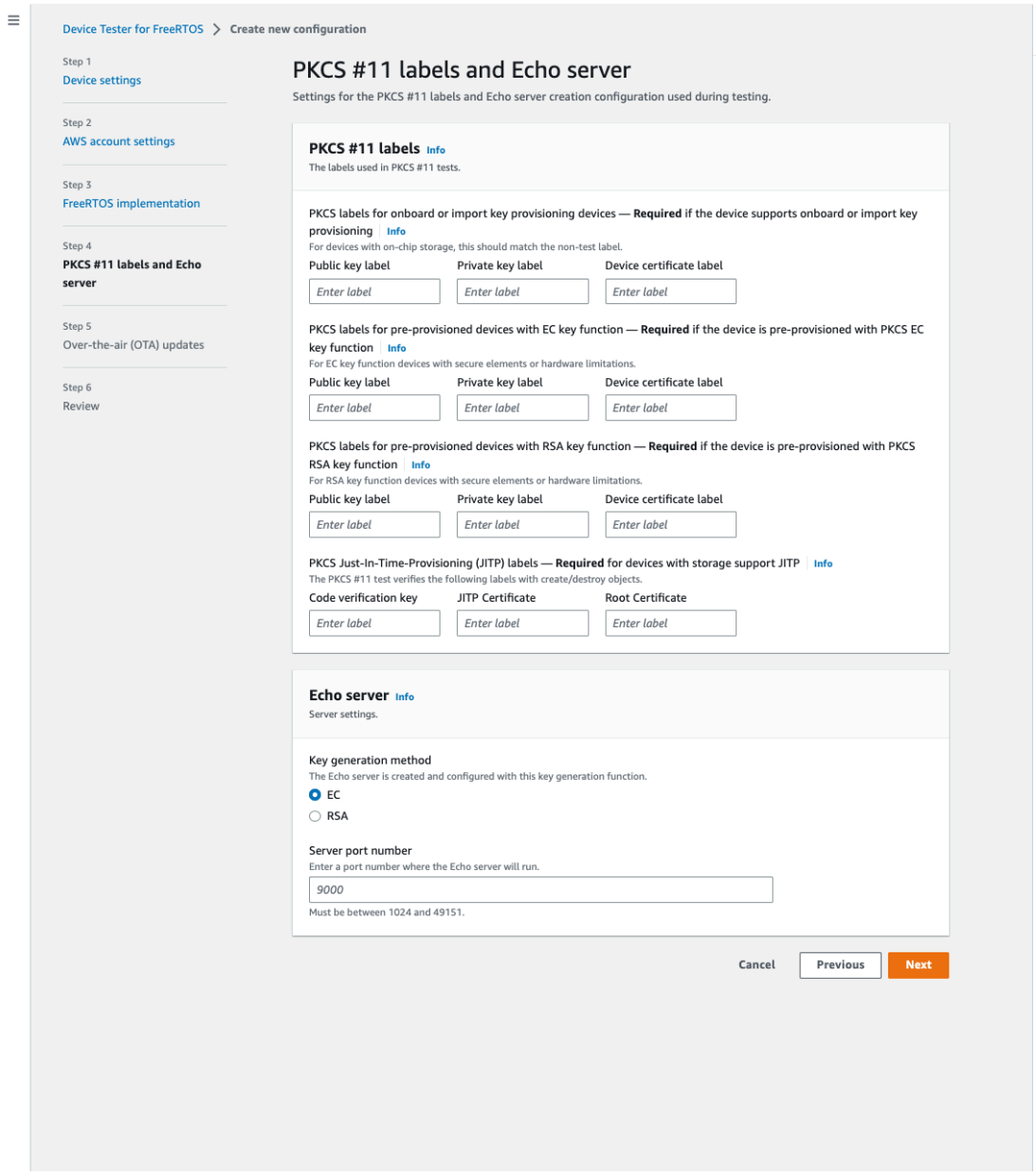
- AWS pengaturan akun — Akun AWS Informasi yang digunakan IDT untuk FreeRTOS untuk AWS membuat sumber daya selama pengujian dijalankan. Pengaturan ini dikonfigurasi dalam `config.json` file.



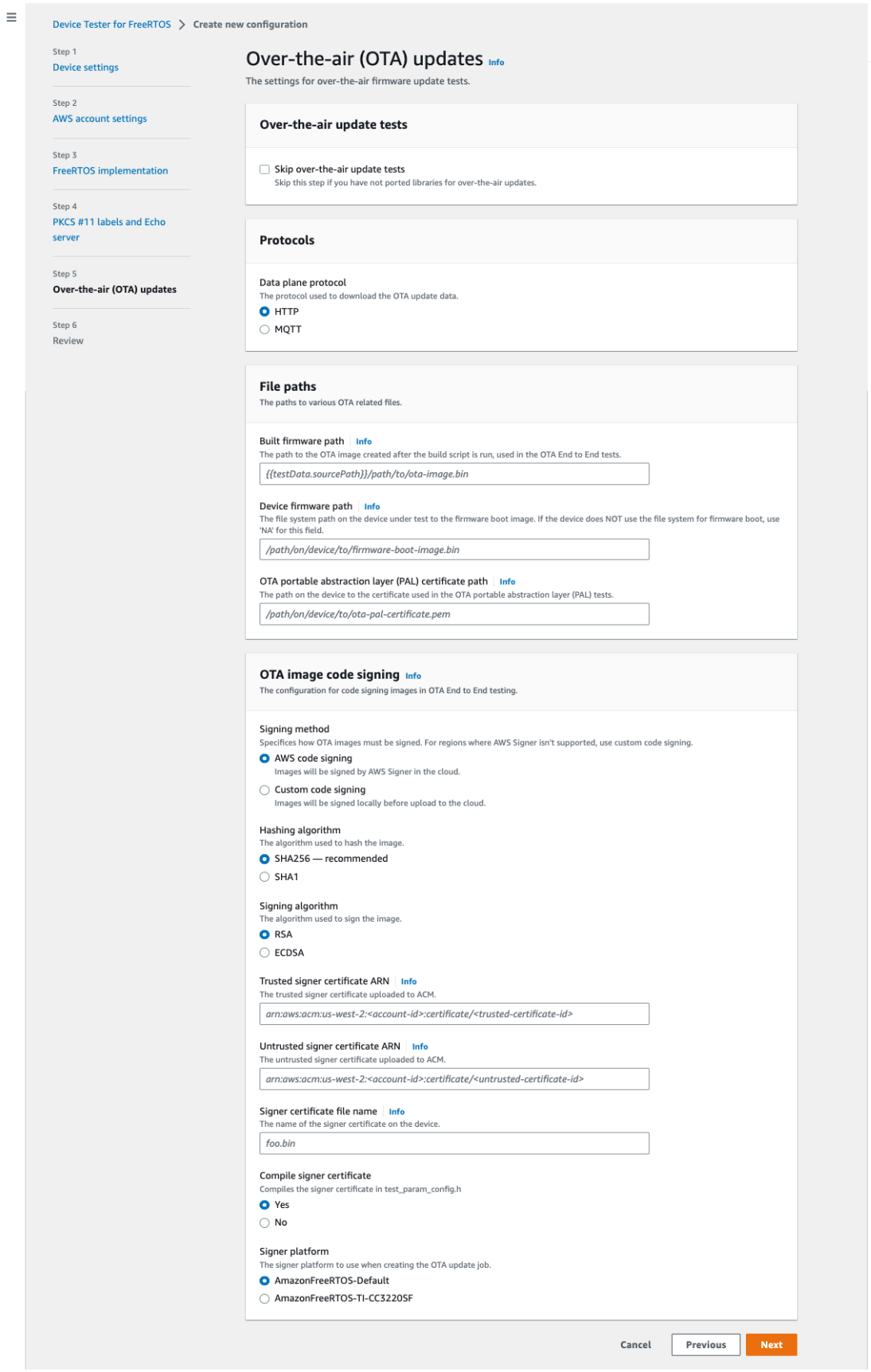
- Implementasi FreerTOS - Jalur absolut ke repositori FreerTOS dan kode porting, dan versi FreerTOS yang ingin Anda jalankan IDT FRQ. Jalur ke file header konfigurasi eksekusi dan parameter dari `FreeRTOS-Libraries-Integration-Tests` GitHub repositori. Perintah build dan flash untuk perangkat keras Anda yang memungkinkan IDT untuk membangun dan mem-flash tes ke papan Anda secara otomatis. Pengaturan ini dikonfigurasi dalam `userdata.json` file.



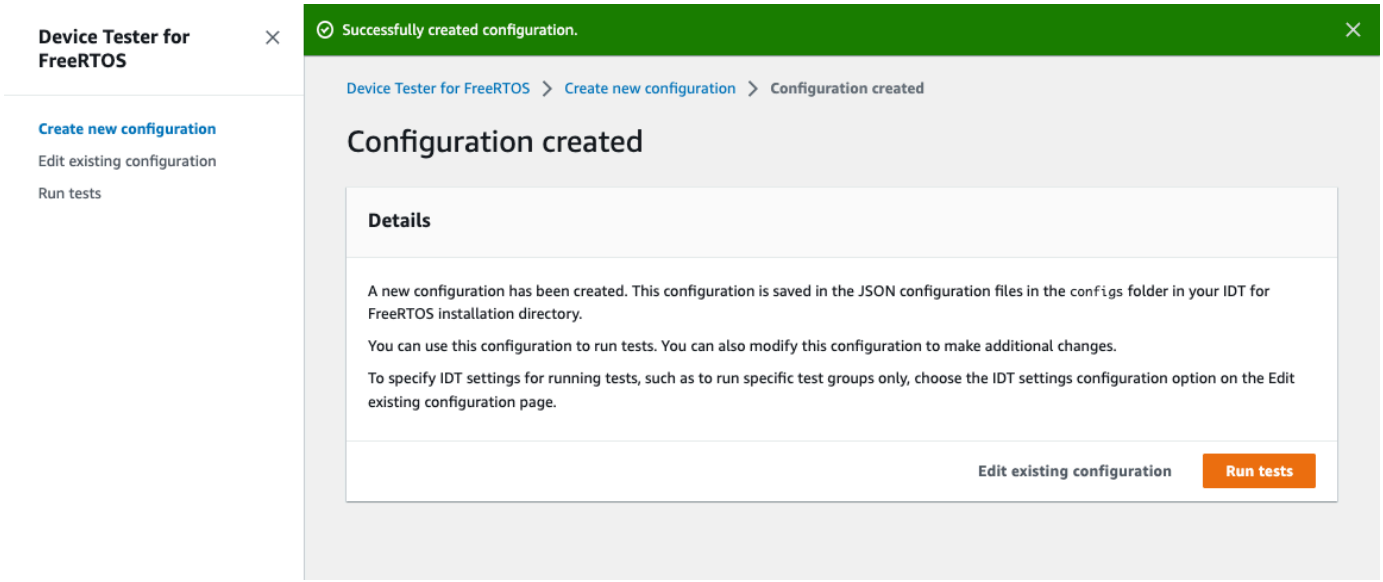
- Label PKCS #11 dan server Echo — Label PKCS #11 yang sesuai dengan kunci yang disediakan di perangkat keras Anda berdasarkan fungsionalitas utama dan metode penyediaan kunci. Pengaturan konfigurasi server echo untuk pengujian Transport Interface. Pengaturan ini dikonfigurasi dalam `device.json` file `userdata.json` dan.



- Over-the-air (OTA) update - Pengaturan yang mengontrol tes fungsionalitas OTA. Pengaturan ini dikonfigurasi di `features` blok `userdata.json` file `device.json` dan.


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. Pada halaman Tinjauan, verifikasi informasi konfigurasi Anda.



Setelah Anda selesai meninjau konfigurasi Anda, untuk menjalankan tes kualifikasi Anda, pilih Jalankan tes.

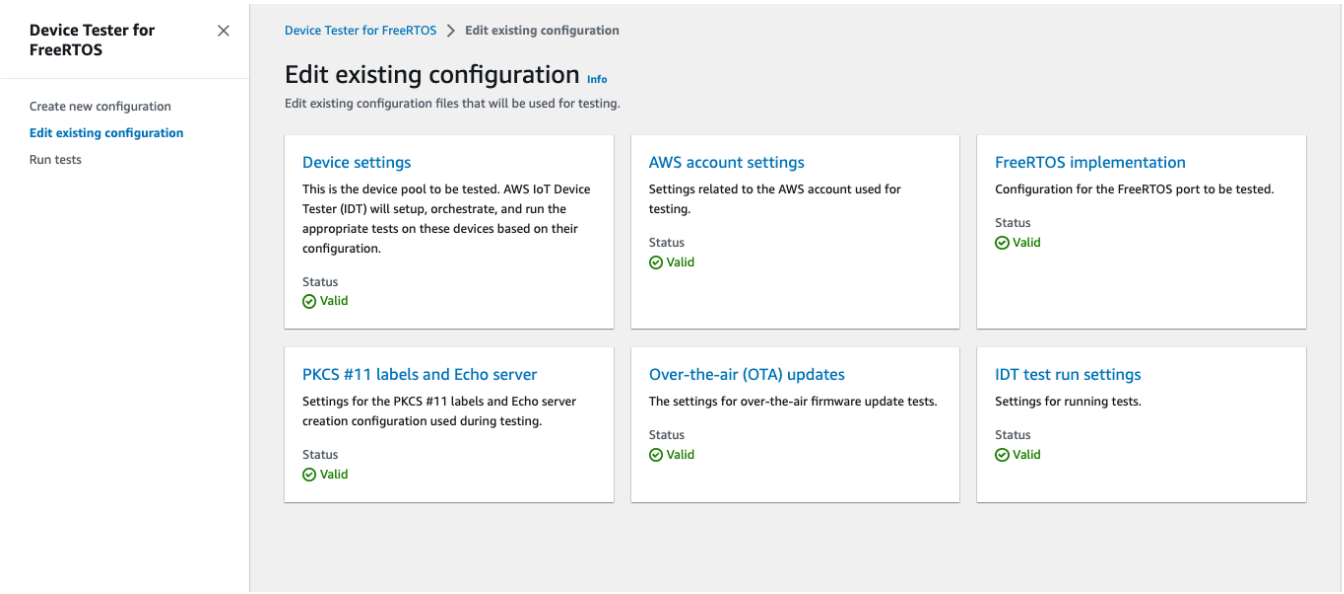
Ubah konfigurasi yang ada

Jika Anda telah menyiapkan file konfigurasi untuk IDT untuk FreeRTOS, Anda dapat menggunakan IDT untuk FreeRTOS UI untuk memodifikasi konfigurasi yang ada. File konfigurasi yang ada harus berada di *`devicetester-extract-location`*`/config` direktori.

Untuk memodifikasi konfigurasi

1. Di IDT untuk FreeRTOS UI, buka menu navigasi, dan pilih Edit konfigurasi yang ada.

   Dasbor konfigurasi menampilkan informasi tentang pengaturan konfigurasi yang ada. Jika konfigurasi salah atau tidak tersedia, status untuk konfigurasi tersebut adalah`Error validating configuration`.

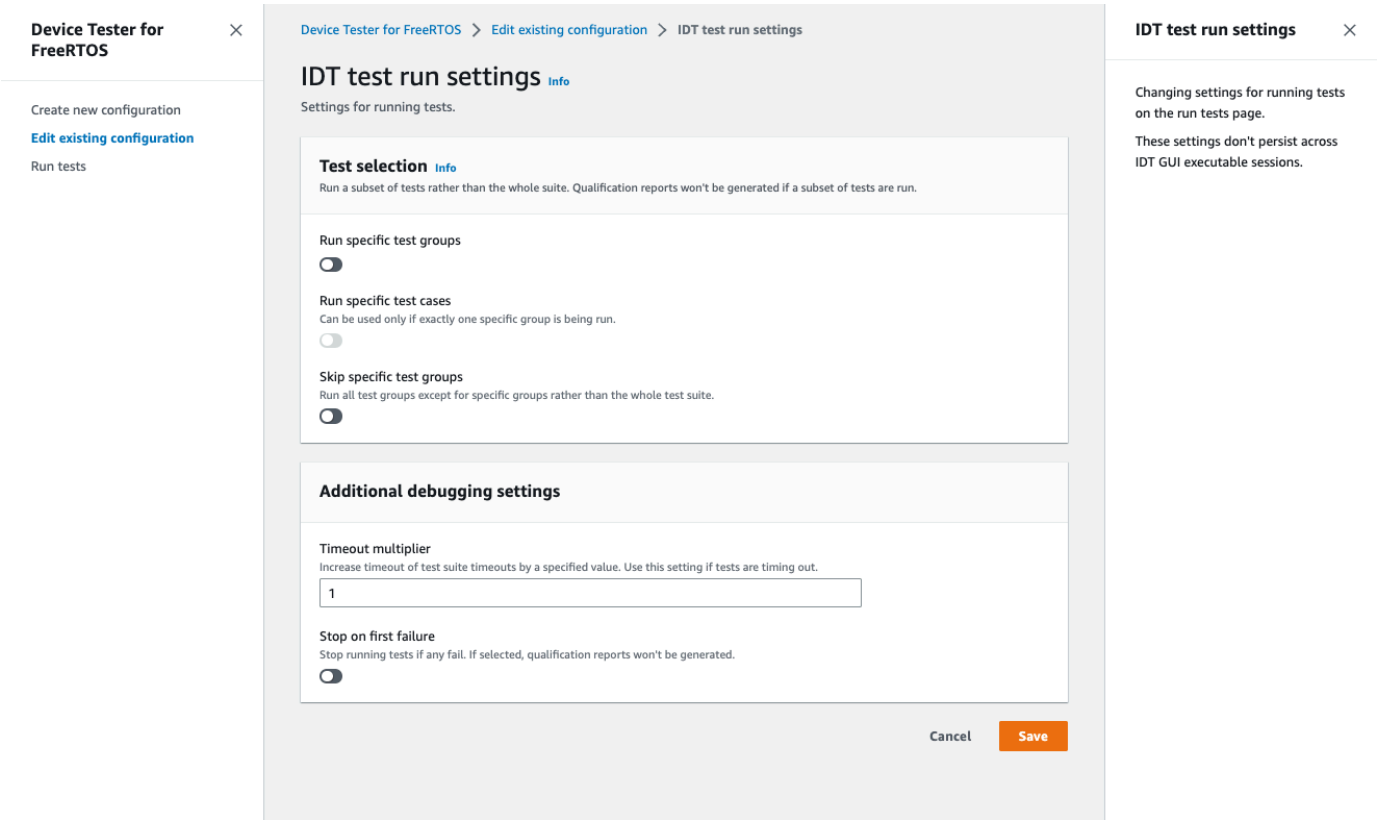


2. Untuk mengubah pengaturan konfigurasi yang ada, selesaikan langkah-langkah berikut:

   a. Pilih nama pengaturan konfigurasi untuk membuka halaman pengaturannya.

   b. Ubah pengaturan, lalu pilih Simpan untuk membuat ulang file konfigurasi yang sesuai.

3. Untuk memodifikasi IDT untuk pengaturan uji coba FreerTOS, pilih pengaturan uji coba IDT di tampilan edit:



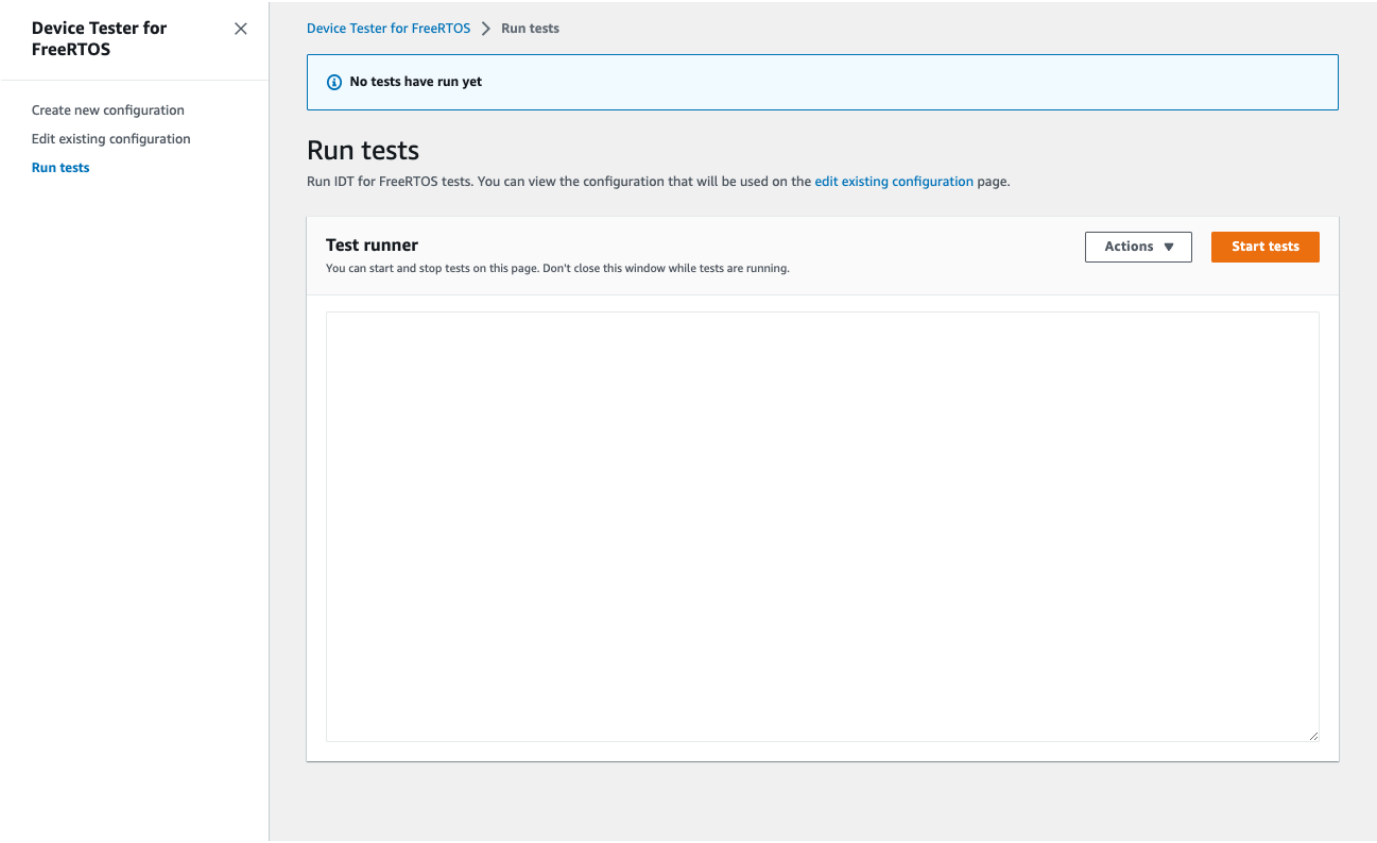
Setelah Anda selesai memodifikasi konfigurasi Anda, verifikasi bahwa semua pengaturan konfigurasi Anda lulus validasi. Jika status untuk setiap pengaturan konfigurasi adalah`Valid`, Anda dapat menjalankan tes kualifikasi Anda dengan konfigurasi ini.

Jalankan tes kualifikasi

Setelah Anda membuat konfigurasi untuk IDT untuk FreeRTOS UI Anda dapat menjalankan tes kualifikasi Anda.

Untuk menjalankan tes kualifikasi

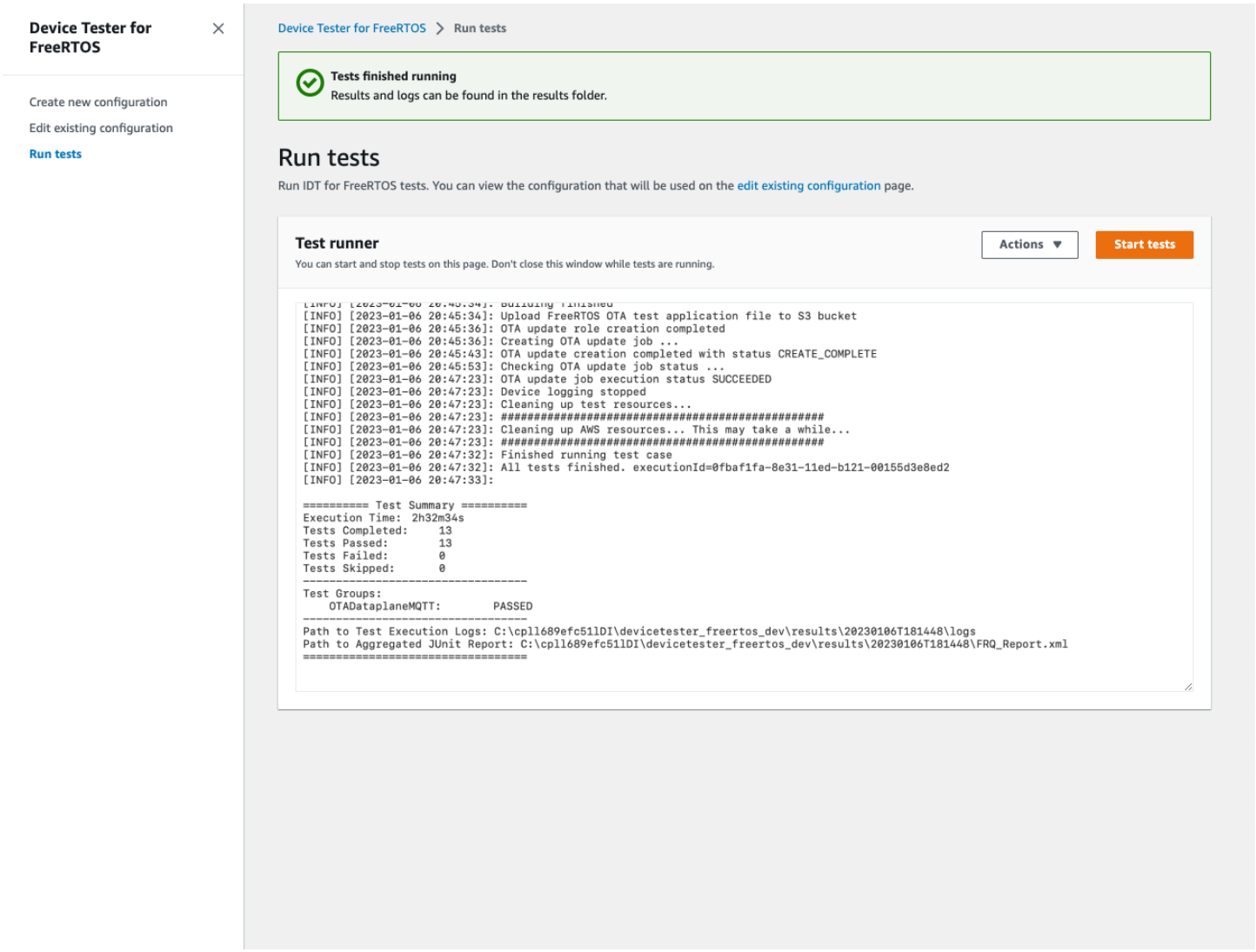
1. Di menu navigasi, pilih Jalankan tes.
2. Pilih Mulai tes untuk memulai uji coba. Secara default, semua pengujian yang berlaku dijalankan untuk konfigurasi perangkat Anda. IDT untuk FreeRTOS menghasilkan laporan kualifikasi ketika semua tes selesai.



IDT untuk FreeRTOS menjalankan tes kualifikasi. Ini kemudian menampilkan ringkasan uji coba dan kesalahan apa pun di konsol Test runner. Setelah uji coba selesai, Anda dapat melihat hasil pengujian dan log dari lokasi berikut:

- Hasil tes terletak di *devicetester-extract-location*/results/*execution-id* direktori.
- Log uji terletak di *devicetester-extract-location*/results/*execution-id*/logs direktori.

Untuk informasi selengkapnya tentang hasil pengujian dan log, lihat Lihat IDT secara Gratis RTOSresults danLihat IDT secara Gratis RTOSlogs.



Jalankan suite kualifikasi FreeRTOS 2.0

Gunakan AWS IoT Device Tester untuk FreeRTOS executable untuk berinteraksi dengan IDT untuk FreeRTOS. Contoh baris perintah berikut menunjukkan cara menjalankan tes kualifikasi untuk kumpulan perangkat (satu set perangkat yang identik).

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
```

Menjalankan serangkaian tes pada kolam perangkat. `userdata.json`File harus berada di *devicetester_extract_location*`/devicetester_freertos_`*[win|mac|linux]*`/configs/` direktori.

> **Note**
>
> Jika Anda menjalankan IDT untuk FreeRTOS di Windows, gunakan garis miring maju (/) untuk menentukan jalur ke file. `userdata.json`

Gunakan perintah berikut untuk menjalankan grup pengujian tertentu:

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

`pool-id`Parameter `suite-id` dan bersifat opsional jika Anda menjalankan rangkaian pengujian tunggal pada satu kumpulan perangkat (yaitu, Anda hanya memiliki satu kumpulan perangkat yang ditentukan dalam `device.json` file Anda).

Gunakan perintah berikut untuk menjalankan kasus uji tertentu dalam kelompok uji:

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

Anda dapat menggunakan `list-test-cases` perintah untuk daftar kasus uji dalam kelompok uji.

IDT untuk opsi baris perintah FreeRTOS

grup-id

(Opsional) Kelompok uji yang akan dijalankan, sebagai daftar yang dipisahkan koma. Jika tidak ditentukan, IDT akan menjalankan semua grup uji di rangkaian tes.

pool-id

(Opsional) Kumpulan perangkat untuk diuji. Ini diperlukan jika Anda menentukan beberapa kumpulan perangkat di`device.json`. Jika Anda hanya memiliki satu kumpulan perangkat, Anda dapat menghilangkan opsi ini.

suite-id

(Opsional) Versi test suite untuk dijalankan. Jika tidak ditentukan, IDT menggunakan versi terbaru dalam direktori tes pada sistem Anda.

tes-id

(Opsional) Tes yang akan dijalankan, sebagai daftar yang dipisahkan koma. Jika ditentukan, `group-id` harus menentukan satu grup.

Example

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

-h

Gunakan opsi bantuan untuk mempelajari lebih lanjut tentang `run-suite` opsi.

Example

Contoh

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

IDT untuk perintah FreeRTOS

Perintah IDT untuk FreeRTOS mendukung operasi berikut:

IDT v4.5.2 and later

`help`

Mencantumkan informasi tentang perintah yang ditentukan.

`list-groups`

Daftar grup dalam suite tertentu.

`list-suites`

Daftar suite yang tersedia.

`list-supported-products`

Daftar produk yang didukung dan versi rangkaian pengujian.

`list-supported-versions`

Daftar FreeRTOS dan versi test suite yang didukung oleh versi IDT saat ini.

`list-test-cases`

Daftar kasus uji dalam grup tertentu.

`run-suite`

Menjalankan serangkaian tes pada kolam perangkat.

Gunakan `--suite-id` opsi untuk menentukan versi suite pengujian, atau hilangkan untuk menggunakan versi terbaru di sistem Anda.

Gunakan `--test-id` untuk menjalankan kasus uji individual.

Example

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

> **Note**
>
> Mulai IDT v3.0.0, IDT memeriksa secara online untuk rangkaian pengujian yang lebih baru. Untuk informasi selengkapnya, lihat Versi rangkaian tes.

Lihat IDT secara Gratis RTOSresults

Saat berjalan, IDT menuliskan kesalahan ke konsol, file log, dan laporan tes. Setelah IDT menyelesaikan rangkaian pengujian kualifikasi, IDT menulis ringkasan uji coba ke konsol dan menghasilkan dua laporan pengujian. Laporan-laporan ini dapat ditemukan di *devicetester-extract-location*`/results/`*execution-id*`/`. Kedua laporan menangkap hasil dari pelaksanaan kualifikasi tes suite.

`awsiotdevicetester_report.xml`Ini adalah laporan pengujian kualifikasi yang Anda kirimkan AWS untuk mencantumkan perangkat Anda di Katalog Perangkat AWS Mitra. Laporan tersebut berisi elemen berikut:

- IDT untuk versi FreeRTOS.
- Versi FreeRTOS yang diuji.
- Fitur FreeRTOS yang didukung oleh perangkat berdasarkan tes yang dilewati.
- SKU dan nama perangkat yang ditentukan dalam `device.json` file.
- Fitur perangkat yang ditentukan dalam `device.json` file.
- Ringkasan agregat hasil uji kasus.
- Rincian hasil kasus uji oleh pustaka yang diuji berdasarkan fitur perangkat.

`FRQ_Report.xml`Ini adalah laporan dalam format JUnit XHTML standar. Anda dapat mengintegrasikannya ke dalam platform CI/CD seperti Jenkins, Bamboo, dan sebagainya. Laporan tersebut berisi elemen berikut:

- Ringkasan agregat hasil kasus uji.
- Rincian hasil kasus uji oleh pustaka yang diuji berdasarkan fitur perangkat.

Menafsirkan IDT untuk hasil FreeRTOS

Bagian laporan di `awsiotdevicetester_report.xml` atau `FRQ_Report.xml` mencantumkan hasil tes yang dijalankan.

Tag XHTML pertama `<testsuites>` berisi ringkasan keseluruhan eksekusi pengujian. Sebagai contoh:

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

Atribut yang digunakan dalam **<testsuites>** tag

name

Nama dari tes suite.

time

Waktu, dalam hitungan detik, yang dibutuhkan untuk menjalankan suite kualifikasi.

tests

Jumlah kasus uji yang dieksekusi.

failures

Jumlah kasus uji yang dijalankan, tetapi tidak lulus.

errors

Jumlah kasus uji yang tidak dapat dijalankan oleh IDT untuk FreeRTOS.

disabled

Atribut ini tidak digunakan dan bisa diabaikan.

Jika tidak ada kegagalan atau kesalahan kasus uji, perangkat Anda memenuhi persyaratan teknis untuk menjalankan FreeRTOS dan dapat beroperasi dengan layanan. AWS IoT Jika Anda memilih untuk mencantumkan perangkat Anda di Katalog Perangkat AWS Mitra, Anda dapat menggunakan laporan ini sebagai bukti kualifikasi.

Jika terjadi kegagalan atau kesalahan kasus uji, Anda dapat mengidentifikasi kasus uji yang gagal dengan meninjau tag `<testsuites>` XHTML. Tag `<testsuite>` XHTML di dalam `<testsuites>` tag menunjukkan ringkasan hasil kasus uji untuk grup uji.

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

Formatnya mirip dengan `<testsuites>` tag, tetapi dengan atribut `skipped` yang disebut yang tidak digunakan dan dapat diabaikan. Di dalam setiap tag `<testsuite>` XHTML, ada `<testcase>` tag untuk setiap kasus uji yang dieksekusi untuk kelompok uji. Sebagai contoh:

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

Atribut yang digunakan dalam **<awsproduct>** tag

name

Nama produk yang sedang diuji.

version

Versi produk yang sedang diuji.

features

Fitur divalidasi. Fitur yang ditandai sebagai `required` wajib mengirimkan forum Anda untuk kualifikasi. Cuplikan berikut menunjukkan bagaimana ini muncul dalam file. `awsiotdevicetester_report.xml`

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

Fitur yang ditandai sebagai `optional` tidak diperlukan untuk kualifikasi. Potongan berikut menunjukkan fitur opsional.

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

Jika tidak ada kegagalan pengujian atau kesalahan untuk fitur yang diperlukan, perangkat Anda memenuhi persyaratan teknis untuk menjalankan FreeRTOS dan dapat beroperasi dengan layanan. AWS IoT Jika ingin mencantumkan perangkat di Katalog Perangkat AWS Mitra, Anda dapat menggunakan laporan ini sebagai bukti kualifikasi.

Jika terjadi kegagalan atau kesalahan uji, Anda dapat mengidentifikasi pengujian yang gagal tersebut dengan meninjau tanda XML `<testsuites>`. Tag XML `<testsuite>` di dalam tag `<testsuites>` menunjukkan ringkasan hasil tes untuk grup uji. Sebagai contoh:

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

Formatnya mirip dengan `<testsuites>` tag, tetapi memiliki `skipped` atribut yang tidak digunakan dan dapat diabaikan. Di dalam masing-masing tanda XML `<testsuite>`, terdapat tanda `<testcase>` untuk setiap tes yang dieksekusi untuk suatu grup uji. Sebagai contoh:

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

Atribut yang digunakan dalam **`<testcase>`** tag

`name`

Nama kasus uji.

`attempts`

Berapa kali IDT untuk FreeRTOS mengeksekusi kasus uji.

Ketika tes gagal atau kesalahan terjadi, tanda `<failure>` atau `<error>` akan ditambahkan ke tanda `<testcase>` dengan informasi untuk pemecahan masalah. Sebagai contoh:

```
<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>
```

Untuk informasi selengkapnya, lihat Memecahkan masalah kesalahan.

Lihat IDT secara Gratis RTOSlogs

Anda dapat menemukan log yang dihasilkan IDT untuk FreeRTOS dari eksekusi pengujian. *devicetester-extract-location*`/results/`*execution-id*`/logs` Dua rangkaian log yang dihasilkan:

- `test_manager.log`

  Berisi log yang dihasilkan dari IDT untuk FreeRTOS (misalnya, konfigurasi terkait log dan pembuatan laporan).

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

  File log untuk kasus uji, termasuk output dari perangkat yang diuji. File log diberi nama sesuai dengan kelompok uji dan kasus uji yang dijalankan.

IDT dengan suite kualifikasi FreeRTOS 1.0 (FRQ 1.0)

> **Important**
>
> Per Oktober 2022, AWS IoT Device Tester untuk Kualifikasi AWS IoT FreerTOS (FRQ) 1.0 tidak menghasilkan laporan kualifikasi yang ditandatangani. Anda tidak dapat memenuhi syarat perangkat AWS IoT FreeRTOS baru untuk dicantumkan di AWS Katalog Perangkat Mitra melalui Program Kualifikasi Perangkat menggunakan versi AWS IDT FRQ 1.0. Meskipun Anda tidak dapat memenuhi syarat perangkat FreerTOS menggunakan IDT FRQ 1.0, Anda dapat terus menguji perangkat FreeRTOS Anda dengan FRQ 1.0. Kami menyarankan Anda menggunakan IDT FRQ 2.0 untuk memenuhi syarat dan daftar perangkat FreeRTOS di Katalog Perangkat Mitra.AWS

Anda dapat menggunakan IDT untuk kualifikasi FreeRTOS untuk memverifikasi bahwa sistem operasi FreerTOS bekerja secara lokal di perangkat Anda dan dapat berkomunikasi dengannya. AWS IoT Secara khusus, ini memverifikasi bahwa antarmuka lapisan porting untuk pustaka FreeRTOS diimplementasikan dengan benar. Ini juga melakukan end-to-end tes dengan AWS IoT Core. Misalnya, memverifikasi papan Anda dapat mengirim dan menerima pesan MQTT dan memprosesnya dengan benar. Pengujian yang dijalankan oleh IDT untuk FreeRTOS didefinisikan dalam repositori FreerTOS. GitHub

Pengujian dijalankan sebagai aplikasi tertanam yang di-flash ke papan Anda. Gambar biner aplikasi termasuk FreeRTOS, antarmuka FreerTOS porting vendor semikonduktor, dan driver perangkat papan. Tujuan dari pengujian ini adalah untuk memverifikasi fungsi antarmuka FreeRTOS yang diporting dengan benar di atas driver perangkat.

IDT untuk FreeRTOS menghasilkan laporan pengujian yang dapat Anda kirimkan AWS IoT untuk menambahkan perangkat keras Anda ke Katalog Perangkat Mitra. AWS Untuk informasi selengkapnya, lihat Program Kualifikasi Perangkat AWS.

IDT untuk FreeRTOS berjalan pada komputer host (Windows, macOS, atau Linux) yang terhubung ke papan untuk diuji. IDT mengeksekusi kasus uji dan mengumpulkan hasil. Ini juga menyediakan antarmuka baris perintah untuk mengelola eksekusi uji.

Selain perangkat pengujian, IDT untuk FreeRTOS menciptakan sumber daya (misalnya, AWS IoT hal-hal, grup FreeRTOS, fungsi Lambda, dan sebagainya) untuk memfasilitasi proses kualifikasi. Untuk membuat sumber daya ini, IDT untuk FreeRTOS menggunakan kredensil AWS yang

dikonfigurasi untuk melakukan panggilan API `config.json` atas nama Anda. Sumber daya ini disediakan pada berbagai waktu selama tes.

Ketika Anda menjalankan IDT untuk FreeRTOS di komputer host Anda, ia melakukan langkah-langkah berikut:

1. Memuat dan memvalidasi konfigurasi perangkat dan kredensial Anda.
2. Melakukan tes yang dipilih dengan sumber daya lokal dan cloud yang diperlukan.
3. Membersihkan sumber daya lokal dan cloud.
4. Menghasilkan laporan tes yang menunjukkan jika forum Anda lulus tes yang diperlukan untuk kualifikasi.

Topik

- Siapkan prasyarat kualifikasi 1.0
- Tes pertama papan mikrokontroler Anda
- Gunakan antarmuka pengguna IDT untuk menjalankan rangkaian kualifikasi FreeRTOS
- Jalankan tes Bluetooth Low Energy
- Jalankan suite kualifikasi FreeRTOS
- Lihat IDT untuk hasil FreeRTOS
- Menafsirkan IDT untuk hasil FreeRTOS
- Lihat IDT untuk log FreeRTOS

Siapkan prasyarat kualifikasi 1.0

Bagian ini menjelaskan prasyarat untuk menguji mikrokontroler dengan. AWS IoT Device Tester

Unduh FreeRTOS

Anda dapat mengunduh rilis FreeRTOS GitHubdari dengan perintah berikut:

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

di mana <FREERTOS_RELEASE_VERSION>adalah versi FreeRTOS (misalnya, 202007.00) sesuai dengan versi IDT yang tercantum dalam. Versi yang didukung AWS IoT Device Tester Ini memastikan Anda memiliki kode sumber lengkap, termasuk submodul, dan menggunakan versi IDT yang benar untuk versi FreeRTOS Anda, dan sebaliknya.

Windows memiliki batasan panjang jalur 260 karakter. Struktur jalur FreeRTOS memiliki banyak level dalam, jadi jika Anda menggunakan Windows, pertahankan jalur file Anda di bawah batas 260 karakter. Misalnya, kloning `C:\FreeRTOS` FreeRTOS ke bukan. `C:\Users\username\programs\projects\myproj\FreeRTOS\`

Kualifikasi FreeRTOS dengan perpustakaan LTS

- Agar mikrokontroler Anda ditetapkan sebagai mendukung versi FreeRTOS berbasis dukungan jangka panjang (LTS) di Katalog Perangkat AWS Mitra, Anda harus menyediakan file manifes. Untuk informasi lebih lanjut, lihat Daftar Periksa Kualifikasi FreerTOS di Panduan Kualifikasi FreerTOS.
- Untuk memvalidasi bahwa mikrokontroler Anda mendukung versi FreeRTOS berbasis LTS dan memenuhi syarat untuk diserahkan ke Katalog Perangkat AWS Mitra, Anda harus menggunakan (AWS IoT Device Tester IDT) dengan FreeRTOS Qualification (FRQ) test suite versi v1.4.x.
- Support untuk FreeRTOS versi berbasis LTS terbatas pada FreeRTOS versi 202012.xx.

Unduh IDT untuk FreeRTOS

Setiap versi FreeRTOS memiliki versi IDT yang sesuai untuk FreerTOS untuk melakukan tes kualifikasi. Unduh versi IDT yang sesuai untuk FreeRTOS dari. Versi yang didukung AWS IoT Device Tester

Ekstrak IDT untuk FreeRTOS ke lokasi pada sistem file tempat Anda telah membaca dan menulis izin. Karena Microsoft Windows memiliki batas karakter untuk panjang jalur, ekstrak IDT untuk FreeRTOS ke direktori root seperti atau. `C:\` `D:\`

> **Note**
>
> Kami tidak menyarankan agar beberapa pengguna menjalankan IDT dari lokasi bersama, seperti direktori NFS atau folder bersama jaringan Windows. Hal ini dapat mengakibatkan crash atau korupsi data. Kami menyarankan Anda mengekstrak paket IDT ke drive lokal.

Buat dan konfigurasikan AWS akun

Mendaftar untuk Akun AWS

Jika Anda tidak memiliki Akun AWS, selesaikan langkah-langkah berikut untuk membuatnya.

Untuk mendaftar untuk Akun AWS

1. Buka https://portal.aws.amazon.com/billing/pendaftaran.
2. Ikuti petunjuk online.

   Bagian dari prosedur pendaftaran melibatkan menerima panggilan telepon atau pesan teks dan memasukkan kode verifikasi pada keypad telepon.

   Saat Anda mendaftar untuk sebuah Akun AWS, sebuah Pengguna root akun AWSdibuat. Pengguna root memiliki akses ke semua Layanan AWS dan sumber daya di akun. Sebagai praktik keamanan terbaik, tetapkan akses administratif ke pengguna, dan gunakan hanya pengguna root untuk melakukan tugas yang memerlukan akses pengguna root.

AWS mengirimi Anda email konfirmasi setelah proses pendaftaran selesai. Kapan saja, Anda dapat melihat aktivitas akun Anda saat ini dan mengelola akun Anda dengan masuk https://aws.amazon.comke/ dan memilih Akun Saya.

Buat pengguna dengan akses administratif

Setelah Anda mendaftar Akun AWS, amankan Pengguna root akun AWS, aktifkan AWS IAM Identity Center, dan buat pengguna administratif sehingga Anda tidak menggunakan pengguna root untuk tugas sehari-hari.

Amankan Anda Pengguna root akun AWS

1. Masuk ke AWS Management Consolesebagai pemilik akun dengan memilih pengguna Root dan memasukkan alamat Akun AWS email Anda. Di laman berikutnya, masukkan kata sandi.

   Untuk bantuan masuk dengan menggunakan pengguna root, lihat Masuk sebagai pengguna root di AWS Sign-In Panduan Pengguna.
2. Mengaktifkan autentikasi multi-faktor (MFA) untuk pengguna root Anda.

   Untuk petunjuk, lihat Mengaktifkan perangkat MFA virtual untuk pengguna Akun AWS root (konsol) Anda di Panduan Pengguna IAM.

Buat pengguna dengan akses administratif

1. Aktifkan Pusat Identitas IAM.

   Untuk mendapatkan petunjuk, silakan lihat Mengaktifkan AWS IAM Identity Center di Panduan Pengguna AWS IAM Identity Center .
2. Di Pusat Identitas IAM, berikan akses administratif ke pengguna.

   Untuk tutorial tentang menggunakan Direktori Pusat Identitas IAM sebagai sumber identitas Anda, lihat Mengkonfigurasi akses pengguna dengan default Direktori Pusat Identitas IAM di Panduan AWS IAM Identity Center Pengguna.

Masuk sebagai pengguna dengan akses administratif

- Untuk masuk dengan pengguna Pusat Identitas IAM, gunakan URL masuk yang dikirim ke alamat email saat Anda membuat pengguna Pusat Identitas IAM.

  Untuk bantuan masuk menggunakan pengguna Pusat Identitas IAM, lihat Masuk ke portal AWS akses di Panduan AWS Sign-In Pengguna.

Tetapkan akses ke pengguna tambahan

1. Di Pusat Identitas IAM, buat set izin yang mengikuti praktik terbaik menerapkan izin hak istimewa paling sedikit.

   Untuk petunjuknya, lihat Membuat set izin di Panduan AWS IAM Identity Center Pengguna.
2. Tetapkan pengguna ke grup, lalu tetapkan akses masuk tunggal ke grup.

   Untuk petunjuk, lihat Menambahkan grup di Panduan AWS IAM Identity Center Pengguna.

AWS IoT Device Tester kebijakan terkelola

Kebijakan `AWSIoTDeviceTesterForFreeRTOSFullAccess` terkelola berisi AWS IoT Device Tester izin berikut untuk pemeriksaan versi, fitur pembaruan otomatis, dan kumpulan metrik.

- `iot-device-tester:SupportedVersion`

  Memberikan AWS IoT Device Tester izin untuk mengambil daftar produk yang didukung, rangkaian pengujian, dan versi IDT.

- `iot-device-tester:LatestIdt`

  Memberikan AWS IoT Device Tester izin untuk mengambil versi IDT terbaru yang tersedia untuk diunduh.

- `iot-device-tester:CheckVersion`

  Memberikan AWS IoT Device Tester izin untuk memeriksa kompatibilitas versi untuk IDT, suite pengujian, dan produk.

- `iot-device-tester:DownloadTestSuite`

  Memberikan AWS IoT Device Tester izin untuk mengunduh pembaruan rangkaian pengujian.

- `iot-device-tester:SendMetrics`

  Memberikan AWS izin untuk mengumpulkan metrik tentang penggunaan AWS IoT Device Tester internal.

(Opsional) Instal AWS Command Line Interface

Anda mungkin lebih suka menggunakan AWS CLI untuk melakukan beberapa operasi. Jika Anda belum AWS CLI menginstal, ikuti petunjuk di Install the AWS CLI.

Konfigurasikan AWS CLI untuk AWS Wilayah yang ingin Anda gunakan dengan menjalankan aws configure dari baris perintah. Untuk informasi tentang AWS Wilayah yang mendukung IDT untuk FreeRTOS,AWS lihat Wilayah dan Titik Akhir. Untuk informasi selengkapnya, aws configure lihat Konfigurasi cepat dengan aws configure.

Tes pertama papan mikrokontroler Anda

Anda dapat menggunakan IDT untuk FreeRTOS untuk menguji saat Anda mem-port antarmuka FreerTOS. Setelah Anda mem-porting antarmuka FreeRTOS untuk driver perangkat papan Anda, Anda AWS IoT Device Tester gunakan untuk menjalankan tes kualifikasi di papan mikrokontroler Anda.

Tambahkan lapisan porting perpustakaan

Untuk mem-port FreeRTOS untuk perangkat Anda, ikuti petunjuk di Panduan Porting FreerTOS.

Konfigurasikan AWS kredensyal Anda

Anda perlu mengonfigurasi AWS kredensyal Anda AWS IoT Device Tester untuk berkomunikasi dengan Cloud. AWS Untuk informasi selengkapnya, lihat Menyiapkan AWS Kredensyal dan Wilayah untuk Pengembangan. AWS Kredensyal yang valid harus ditentukan dalam file *devicetester_extract_location*`/devicetester_afreertos_`*[win|mac|linux]*`/configs/config.json` konfigurasi.

Topik

- Buat kumpulan perangkat di IDT untuk FreeRTOS
- Konfigurasikan pengaturan build, flash, dan uji

Buat kumpulan perangkat di IDT untuk FreeRTOS

Perangkat yang akan diuji diatur dalam kumpulan perangkat. Setiap kumpulan perangkat terdiri dari satu atau lebih perangkat yang identik. Anda dapat mengonfigurasi IDT untuk FreeRTOS untuk menguji satu perangkat di kolam renang atau beberapa perangkat di kolam renang. Untuk mempercepat proses kualifikasi, IDT untuk FreerTOS dapat menguji perangkat dengan spesifikasi yang sama secara paralel. Ini menggunakan metode round-robin untuk mengeksekusi kelompok uji yang berbeda pada setiap perangkat dalam kumpulan perangkat.

Anda dapat menambahkan satu atau beberapa perangkat ke kumpulan perangkat dengan mengedit `devices` bagian `device.json` templat di `configs` folder.

> **Note**
>
> Semua perangkat di kolam yang sama harus memiliki spesifikasi teknis dan SKU yang sama.

Untuk mengaktifkan build paralel dari kode sumber untuk grup pengujian yang berbeda, IDT untuk FreerTOS menyalin kode sumber ke folder hasil di dalam folder yang diekstrak IDT untuk FreeRTOS. Jalur kode sumber dalam perintah build atau flash Anda harus direferensikan menggunakan `sdkPath` variabel `testdata.sourcePath` atau. IDT untuk FreeRTOS menggantikan variabel ini dengan jalur sementara dari kode sumber yang disalin. Untuk informasi selengkapnya, lihat IDT untuk variabel FreeRTOS.

Berikut ini adalah `device.json` file contoh yang digunakan untuk membuat kumpulan perangkat dengan beberapa perangkat.

```
[
    {
        "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"
              }
            ]
          }
        ]
    }
]
```

Atribut berikut digunakan dalam `device.json` file:

`id`

ID alfanumerik yang ditentukan pengguna yang secara unik mengidentifikasi kumpulan perangkat. Perangkat yang termasuk dalam kolam harus dari jenis yang sama. Saat serangkaian pengujian sedang berjalan, perangkat di kolam digunakan untuk memparalelkan beban kerja.

`sku`

Nilai alfanumerik yang secara unik mengidentifikasi papan yang Anda uji. SKU digunakan untuk melacak forum yang berkualitas.

> **Note**
>
> Jika Anda ingin mencantumkan papan Anda di Katalog Perangkat AWS Mitra, SKU yang Anda tentukan di sini harus cocok dengan SKU yang Anda gunakan dalam proses pencatatan.

`features`

Array yang berisi fitur yang didukung perangkat. AWS IoT Device Tester menggunakan informasi ini untuk memilih tes kualifikasi yang akan dijalankan.

Nilai yang didukung adalah:

`TCP/IP`

Menunjukkan apakah papan Anda mendukung TCP/IP tumpukan dan apakah itu didukung on-chip (MCU) atau diturunkan ke modul lain. TCP/IP diperlukan untuk kualifikasi.

`WIFI`

Menunjukkan apakah papan Anda memiliki kemampuan Wi-Fi. Harus disetel ke `No` if `Cellular` diatur ke`Yes`.

`Cellular`

Menunjukkan apakah papan Anda memiliki kemampuan seluler. Harus disetel ke `No` if `WIFI` diatur ke`Yes`. Ketika fitur ini disetel ke`Yes`, FullSecureSockets pengujian akan dijalankan menggunakan EC2 instans AWS t2.micro dan ini dapat menimbulkan biaya tambahan ke akun Anda. Untuk informasi selengkapnya, lihat EC2 harga Amazon.

`TLS`

Menunjukkan apakah papan Anda mendukung TLS. TLS diperlukan untuk kualifikasi.

`PKCS11`

Menunjukkan algoritma kriptografi kunci publik yang didukung papan. PKCS11 diperlukan untuk kualifikasi. Nilai yang didukung adalah`ECC`,`RSA`, `Both` dan`No`. `Both`menunjukkan papan mendukung kedua `ECC` dan `RSA` algoritma.

`KeyProvisioning`

Menunjukkan metode penulisan sertifikat klien X.509 tepercaya ke papan Anda. Nilai yang valid adalah`Import`, `Onboard` dan`No`. Penyediaan kunci diperlukan untuk kualifikasi.

- Gunakan `Import` jika papan Anda mengizinkan impor kunci pribadi. IDT akan membuat kunci pribadi dan membangun ini ke kode sumber FreeRTOS.
- Gunakan `Onboard` jika board Anda mendukung pembuatan kunci pribadi on-board (misalnya, jika perangkat Anda memiliki elemen aman, atau jika Anda lebih suka membuat key pair dan sertifikat perangkat Anda sendiri). Pastikan Anda menambahkan `secureElementConfig` elemen di setiap bagian perangkat dan meletakkan jalur absolut ke file kunci publik di `publicKeyAsciiHexFilePath` bidang.
- Gunakan `No` jika papan Anda tidak mendukung penyediaan kunci.

`OTA`

Menunjukkan apakah papan Anda mendukung fungsionalitas pembaruan over-the-air (OTA). `OtaDataPlaneProtocol`Atribut menunjukkan protokol dataplane OTA mana yang didukung perangkat. Atribut diabaikan jika fitur OTA tidak didukung oleh perangkat. Kapan `"Both"` dipilih, waktu eksekusi pengujian OTA diperpanjang karena menjalankan MQTT, HTTP, dan tes campuran.

> **Note**
>
> Dimulai dengan IDT v4.1.0, hanya `OtaDataPlaneProtocol` menerima `HTTP` dan `MQTT` sebagai nilai yang didukung.

`BLE`

Menunjukkan apakah papan Anda mendukung Bluetooth Low Energy (BLE).

`devices.id`

Pengenal unik yang ditetapkan pengguna untuk perangkat yang sedang diuji.

`devices.connectivity.protocol`

Protokol komunikasi yang digunakan untuk berkomunikasi dengan perangkat ini. Nilai yang didukung: `uart`.

`devices.connectivity.serialPort`

Port serial komputer host digunakan untuk terhubung ke perangkat yang sedang diuji.

`devices.secureElementConfig.PublicKeyAsciiHexFilePath`

Jalur absolut ke file yang berisi hex byte kunci publik yang diekstrak dari kunci pribadi onboard.

Contoh format:

```
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
```

Jika kunci publik Anda dalam format.der, Anda dapat menyandikan kunci publik secara langsung untuk menghasilkan file hex.

Contoh perintah untuk kunci publik.der untuk menghasilkan file hex:

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

Jika kunci publik Anda dalam format.pem, Anda dapat mengekstrak bagian yang dikodekan base64, mendekodekannya ke dalam format biner, dan kemudian hex menyandikannya untuk menghasilkan file hex.

Misalnya, gunakan perintah ini untuk menghasilkan file hex untuk kunci publik.pem:

1. Keluarkan bagian kunci yang dikodekan base64 (strip header dan footer) dan simpan dalam file, misalnya beri nama`base64key`, jalankan perintah ini untuk mengubahnya menjadi format.der:

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

2. Jalankan `xxd` perintah untuk mengubahnya menjadi format hex.

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

`devices.secureElementConfig.SecureElementSerialNumber`

(Opsional) Nomor seri elemen aman. Berikan bidang ini saat nomor seri dicetak bersama dengan kunci publik perangkat saat Anda menjalankan proyek FreeRTOS demo/test .

`devices.secureElementConfig.preProvisioned`

(Opsional) Setel ke “Ya” jika perangkat memiliki elemen aman yang telah disediakan sebelumnya dengan kredensil terkunci, yang tidak dapat mengimpor, membuat, atau menghancurkan objek.

Konfigurasi ini berlaku hanya ketika `features` telah `KeyProvisioning` disetel ke “Onboard”, bersama dengan `PKCS11` disetel ke “ECC”.

`identifiers`

(Opsional) Sebuah array pasangan nama-nilai arbitrer. Anda dapat menggunakan nilai-nilai ini dalam perintah build dan flash yang dijelaskan di bagian selanjutnya.

Konfigurasikan pengaturan build, flash, dan uji

Agar IDT untuk FreeRTOS dapat membangun dan mem-flash tes ke papan Anda secara otomatis, Anda harus mengonfigurasi IDT untuk menjalankan perintah build dan flash untuk perangkat keras Anda. Pengaturan perintah build dan flash dikonfigurasi dalam file `userdata.json` template yang terletak di `config` folder.

Konfigurasikan pengaturan untuk perangkat pengujian

Pengaturan build, flash, dan test dibuat dalam `configs/userdata.json` file. Kami mendukung konfigurasi Echo Server dengan memuat sertifikat dan kunci klien dan server di file. `customPath` Untuk informasi selengkapnya, lihat Menyiapkan server gema di Panduan Porting FreeRTOS. Contoh JSON berikut menunjukkan bagaimana Anda dapat mengkonfigurasi IDT untuk FreeRTOS untuk menguji beberapa perangkat:

```
{
    "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"
            }
        ]
    }
}
```

Berikut ini mencantumkan atribut yang digunakan dalam`userdata.json`:

`sourcePath`

Jalur ke root kode sumber FreeRTOS yang diporting. Untuk pengujian paralel dengan SDK, `sourcePath` dapat diatur menggunakan dudukan `{{userData.sdkConfiguration.path}}` tempat. Misalnya:

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

vendorPath

Jalur ke kode FreeRTOS khusus vendor. Untuk pengujian serial, `vendorPath` dapat diatur sebagai jalur absolut. Misalnya:

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

Untuk pengujian paralel, `vendorPath` dapat diatur menggunakan dudukan `{{testData.sourcePath}}` tempat. Misalnya:

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

`vendorPath`Variabel hanya diperlukan saat berjalan tanpa SDK, variabel dapat dihapus jika tidak.

> **Note**
>
> Saat menjalankan pengujian secara paralel tanpa SDK, `{{testData.sourcePath}}` placeholder harus digunakan di bidang`vendorPath`,,`buildTool`. `flashTool` Saat menjalankan pengujian dengan satu perangkat, jalur absolut harus digunakan di `flashTool` bidang `vendorPath``buildTool`,,. Saat menjalankan dengan SDK, `{{sdkPath}}` placeholder harus digunakan dalam perintah`sourcePath`,`buildTool`, dan. `flashTool`

sdkConfiguration

Jika Anda memenuhi syarat FreeRTOS dengan modifikasi apa pun pada file dan struktur folder di luar apa yang diperlukan untuk porting, maka Anda perlu mengonfigurasi informasi SDK Anda di blok ini. Jika Anda tidak memenuhi syarat dengan FreeRTOS porting di dalam SDK, maka Anda harus menghilangkan blok ini sepenuhnya.

sdkConfiguration.name

Nama SDK yang Anda gunakan dengan FreeRTOS. Jika Anda tidak menggunakan SDK, maka seluruh `sdkConfiguration` blok harus dihilangkan.

sdkConfiguration.version

Versi SDK yang Anda gunakan dengan FreeRTOS. Jika Anda tidak menggunakan SDK, maka seluruh `sdkConfiguration` blok harus dihilangkan.

sdkConfiguration.path

Jalur absolut ke direktori SDK Anda yang berisi kode FreeRTOS Anda. Jika Anda tidak menggunakan SDK, maka seluruh `sdkConfiguration` blok harus dihilangkan.

buildTool

Jalur lengkap ke skrip build Anda (.bat atau .sh) yang berisi perintah untuk membangun kode sumber Anda. Semua referensi ke jalur kode sumber dalam perintah build harus diganti dengan AWS IoT Device Tester variabel `{{testdata.sourcePath}}` dan referensi ke jalur SDK harus diganti dengan`{{sdkPath}}`. Gunakan `{{config.idtRootPath}}` placeholder untuk mereferensikan jalur IDT absolut atau relatif.

testStartDelayms

Menentukan berapa milidetik pelari uji FreeRTOS akan menunggu sebelum mulai menjalankan tes. Ini dapat berguna jika perangkat yang diuji mulai mengeluarkan informasi pengujian penting sebelum IDT memiliki kesempatan untuk terhubung dan mulai masuk karena jaringan atau latensi lainnya. Nilai maksimum yang diizinkan adalah 30000 ms (30 detik). Nilai ini hanya berlaku untuk kelompok uji FreeRTOS, dan tidak berlaku untuk kelompok uji lain yang tidak menggunakan pelari uji FreeRTOS, seperti tes OTA.

flashTool

Jalur lengkap ke skrip flash Anda (.sh atau.bat) yang berisi perintah flash untuk perangkat Anda. Semua referensi ke jalur kode sumber dalam perintah flash harus diganti oleh variabel IDT untuk FreeRTOS `{{testdata.sourcePath}}` dan semua referensi ke jalur SDK Anda harus diganti dengan IDT untuk variabel FreeRTOS. Gunakan placeholder untuk mereferensikan jalur IDT absolut atau relatif. `{{sdkPath}}` `{{config.idtRootPath}}`

buildImageInfo

testsImageName

Nama file yang dihasilkan oleh perintah build saat membuat pengujian dari *freertos-source*`/tests` folder.

`demosImageName`

Nama file yang dihasilkan oleh perintah build saat membuat pengujian dari *`freertos-source`*`/demos` folder.

`clientWifiConfig`

Konfigurasi Wi-Fi klien. Tes perpustakaan Wi-Fi memerlukan papan MCU untuk terhubung ke dua titik akses. (Dua titik akses bisa sama.) Atribut ini mengkonfigurasi pengaturan Wi-Fi untuk titik akses pertama. Beberapa kasus uji Wi-Fi mengharapkan titik akses memiliki keamanan dan tidak terbuka. Pastikan kedua titik akses berada pada subnet yang sama dengan komputer host yang menjalankan IDT.

`wifi_ssid`

Wi-Fi SSID.

`wifi_password`

Kata sandi Wi-Fi.

`wifiSecurityType`

Jenis keamanan Wi-Fi yang digunakan. Salah satu nilai:

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

> **Note**
>
> Jika papan Anda tidak mendukung Wi-Fi, Anda masih harus menyertakan `clientWifiConfig` bagian dalam `device.json` file Anda, tetapi Anda dapat menghilangkan nilai untuk atribut ini.

`testWifiConfig`

Konfigurasi Wi-Fi uji. Tes perpustakaan Wi-Fi memerlukan papan MCU untuk terhubung ke dua titik akses. (Dua titik akses bisa sama.) Atribut ini mengkonfigurasi pengaturan Wi-Fi untuk titik

akses kedua. Beberapa kasus uji Wi-Fi mengharapkan titik akses memiliki keamanan dan tidak terbuka. Pastikan kedua titik akses berada pada subnet yang sama dengan komputer host yang menjalankan IDT.

`wifiSSID`

Wi-Fi SSID.

`wifiPassword`

Kata sandi Wi-Fi.

`wifiSecurityType`

Jenis keamanan Wi-Fi yang digunakan. Salah satu nilai:

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

> **Note**
>
> Jika papan Anda tidak mendukung Wi-Fi, Anda masih harus menyertakan `testWifiConfig` bagian dalam `device.json` file Anda, tetapi Anda dapat menghilangkan nilai untuk atribut ini.

`echoServerCertificateConfiguration`

Placeholder pembuatan sertifikat server gema yang dapat dikonfigurasi untuk pengujian soket aman. Bidang ini wajib diisi.

`certificateGenerationMethod`

Menentukan apakah sertifikat server dihasilkan secara otomatis atau disediakan secara manual.

`customPath`

Jika `certificateGenerationMethod` “Kustom”, `certificatePath` dan `privateKeyPath` diperlukan.

`certificatePath`

Menentukan filepath untuk sertifikat server.

`privateKeyPath`

Menentukan filepath untuk kunci pribadi.

`eccCurveFormat`

Menentukan format kurva didukung oleh papan. Diperlukan saat `PKCS11` disetel ke “ecc” di`device.json`. Nilai yang valid adalah “P224", “P256", “P384", atau “P521".

`echoServerConfiguration`

Port server gema yang dapat dikonfigurasi untuk WiFi dan pengujian soket aman. Bidang ini bersifat opsional.

`securePortForSecureSocket`

Port yang digunakan untuk mengatur server gema dengan TLS untuk uji soket aman. Nilai defaultnya adalah 33333. Pastikan port yang dikonfigurasi tidak diblokir oleh firewall atau jaringan perusahaan Anda.

`insecurePortForSecureSocket`

Port yang digunakan untuk mengatur server gema tanpa TLS untuk uji soket aman. Nilai default yang digunakan dalam pengujian adalah 33334. Pastikan port yang dikonfigurasi tidak diblokir oleh firewall atau jaringan perusahaan Anda.

`insecurePortForWiFi`

Port yang digunakan untuk setup server echo tanpa TLS untuk WiFi pengujian. Nilai default yang digunakan dalam pengujian adalah 33335. Pastikan port yang dikonfigurasi tidak diblokir oleh firewall atau jaringan perusahaan Anda.

`otaConfiguration`

Konfigurasi OTA. [Opsional]

`otaFirmwareFilePath`

Jalur lengkap ke gambar OTA yang dibuat setelah build. Misalnya, `{{testData.sourcePath}}/`*`relative-path/to/ota/image/from/source/root`*.

`deviceFirmwareFileName`

Jalur file lengkap pada perangkat MCU tempat firmware OTA berada. Beberapa perangkat tidak menggunakan bidang ini, tetapi Anda masih harus memberikan nilai.

`otaDemoConfigFilePath`

Jalan lengkap ke`aws_demo_config.h`, ditemukan di*`afr-source`*`/vendors/vendor/boards/board/aws_demos/config_files/`. File-file ini termasuk dalam template kode porting yang disediakan FreeRTOS.

`codeSigningConfiguration`

Konfigurasi penandatanganan kode.

`signingMethod`

Metode penandatanganan kode. Nilai yang mungkin adalah `AWS` atau`Custom`.

> **Note**
>
> Untuk Wilayah Beijing dan Ningxia, gunakan`Custom`. `AWS`penandatanganan kode tidak didukung di Wilayah ini.

`signerHashingAlgorithm`

Algoritma hashing didukung pada perangkat. Nilai yang mungkin adalah `SHA1` atau`SHA256`.

`signerSigningAlgorithm`

Algoritma penandatanganan didukung pada perangkat. Nilai yang mungkin adalah `RSA` atau`ECDSA`.

`signerCertificate`

Sertifikat tepercaya yang digunakan untuk OTA.

Untuk metode penandatanganan AWS kode, gunakan Amazon Resource Name (ARN) untuk sertifikat tepercaya yang diunggah ke file. AWS Certificate Manager

Untuk metode penandatanganan kode kustom, gunakan jalur absolut ke file sertifikat penandatangan.

Untuk informasi selengkapnya tentang membuat sertifikat tepercaya, lihatBuat sertifikat penandatanganan kode.

`signerCertificateFileName`

Nama file sertifikat penandatanganan kode pada perangkat. Nilai ini harus sesuai dengan nama file yang Anda berikan saat menjalankan `aws acm import-certificate` perintah.

Untuk informasi selengkapnya, lihat Buat sertifikat penandatanganan kode.

`compileSignerCertificate`

Setel ke `true` jika sertifikat verifikasi tanda tangan penandatangan kode tidak disediakan atau di-flash, sehingga harus dikompilasi ke dalam proyek. AWS IoT Device Tester mengambil sertifikat tepercaya dan mengkompilasinya menjadi. `aws_codesigner_certifiate.h`

`untrustedSignerCertificate`

ARN atau filepath untuk sertifikat kedua yang digunakan dalam beberapa tes OTA sebagai sertifikat yang tidak tepercaya. Untuk informasi selengkapnya tentang membuat sertifikat, lihat Membuat sertifikat penandatanganan kode.

`signerPlatform`

Algoritma penandatanganan dan hashing yang digunakan AWS Code Signer saat membuat pekerjaan pembaruan OTA. Saat ini, nilai yang mungkin untuk bidang ini adalah `AmazonFreeRTOS-TI-CC3220SF` dan`AmazonFreeRTOS-Default`.

- Pilih `AmazonFreeRTOS-TI-CC3220SF` jika `SHA1` dan`RSA`.
- Pilih `AmazonFreeRTOS-Default` jika `SHA256` dan`ECDSA`.

Jika Anda membutuhkan `SHA256 | RSA` atau `SHA1 | ECDSA` untuk konfigurasi Anda, hubungi kami untuk dukungan lebih lanjut.

Konfigurasikan `signCommand` jika Anda memilih `Custom` untuk`signingMethod`.

`signCommand`

Perintah yang digunakan untuk melakukan penandatanganan kode kustom. Anda dapat menemukan template di `/configs/script_templates` direktori.

Dua placeholder `{{inputImageFilePath}}` dan `{{outputSignatureFilePath}}` diperlukan dalam perintah. `{{inputImageFilePath}}`adalah jalur file gambar yang

dibangun oleh IDT untuk ditandatangani. `{{outputSignatureFilePath}}`adalah jalur file dari tanda tangan yang akan dihasilkan oleh skrip.

`cmakeConfiguration`

CMake konfigurasi [Opsional]

> **Note**
>
> Untuk menjalankan kasus CMake uji, Anda harus memberikan nama papan, nama vendor, dan salah satu `frToolchainPath` atau`compilerName`. Anda juga dapat memberikan `cmakeToolchainPath` jika Anda memiliki jalur khusus ke CMake rantai alat.

`boardName`

Nama papan yang diuji. Nama papan harus sama dengan nama folder di bawah*`path/to/afr/source/code`*`/vendors/`*`vendor`*`/boards/`*`board`*.

`vendorName`

Nama vendor untuk papan yang diuji. Vendor harus sama dengan nama folder di bawah*`path/to/afr/source/code`*`/vendors/`*`vendor`*.

`compilerName`

Nama kompiler.

`frToolchainPath`

Jalur yang sepenuhnya memenuhi syarat ke toolchain kompiler

`cmakeToolchainPath`

Jalur yang sepenuhnya memenuhi syarat ke rantai alat. CMake Bidang ini opsional

`freertosFileConfiguration`

Konfigurasi file FreerTOS yang IDT memodifikasi sebelum menjalankan tes.

`required`

Bagian ini menentukan tes wajib yang file konfigurasinya telah Anda pindahkan, misalnya, TLS PKCS11, dan sebagainya.

`configName`

Nama tes yang sedang dikonfigurasi.

`filePath`

Jalur absolut ke file konfigurasi dalam *`freertos`* repo. Gunakan `{{testData.sourcePath}}` variabel untuk menentukan jalur.

`optional`

Bagian ini menentukan tes opsional yang file konfigurasinya telah Anda pindahkan, misalnya OTA, WiFi, dan sebagainya.

`configName`

Nama tes yang sedang dikonfigurasi.

`filePath`

Jalur absolut ke file konfigurasi dalam *`freertos`* repo. Gunakan `{{testData.sourcePath}}` variabel untuk menentukan jalur.

> **Note**
>
> Untuk menjalankan kasus CMake uji, Anda harus memberikan nama papan, nama vendor, dan salah satu `afrToolchainPath` atau`compilerName`. Anda juga dapat memberikan `cmakeToolchainPath` jika Anda memiliki jalur khusus ke CMake rantai alat.

IDT untuk variabel FreeRTOS

Perintah untuk membangun kode Anda dan mem-flash perangkat mungkin memerlukan konektivitas atau informasi lain tentang perangkat Anda agar berhasil berjalan. AWS IoT Device Tester memungkinkan Anda untuk mereferensikan informasi perangkat dalam flash dan membangun perintah menggunakan JsonPath. Dengan menggunakan JsonPath ekspresi sederhana, Anda dapat mengambil informasi yang diperlukan yang ditentukan dalam `device.json` file Anda.

Variabel jalur

IDT untuk FreeRTOS mendefinisikan variabel jalur berikut yang dapat digunakan dalam baris perintah dan file konfigurasi:

{{testData.sourcePath}}

Memperluas ke jalur kode sumber. Jika Anda menggunakan variabel ini, itu harus digunakan dalam perintah flash dan build.

{{sdkPath}}

Memperluas ke nilai dalam Anda `userData.sdkConfiguration.path` saat digunakan dalam perintah build dan flash.

{{device.connectivity.serialPort}}

Memperluas ke port serial.

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

Memperluas ke nomor seri perangkat Anda.

{{enableTests}}

Nilai integer yang menunjukkan apakah build untuk pengujian (nilai 1) atau demo (nilai 0).

{{buildImageName}}

Nama file gambar yang dibangun oleh perintah build.

{{otaCodeSignerPemFile}}

File PEM untuk penandatangan kode OTA.

{{config.idtRootPath}}

Memperluas ke jalur AWS IoT Device Tester root. Variabel ini menggantikan jalur absolut untuk IDT saat digunakan oleh perintah build dan flash.

Gunakan antarmuka pengguna IDT untuk menjalankan rangkaian kualifikasi FreeRTOS

Dimulai dengan IDT v4.3.0, untuk AWS IoT Device Tester FreeRTOS (IDT-FreeRTOS) termasuk antarmuka pengguna berbasis web yang memungkinkan Anda untuk berinteraksi dengan baris perintah IDT executable dan file konfigurasi terkait. Anda dapat menggunakan UI IDT-Freertos untuk membuat konfigurasi baru untuk menjalankan pengujian IDT, atau untuk memodifikasi konfigurasi yang ada. Anda juga dapat menggunakan UI untuk memanggil IDT executable dan menjalankan tes.

IDT-Freertos UI menyediakan fungsi-fungsi berikut:

- Sederhanakan pengaturan file konfigurasi untuk tes IDT-Freertos.
- Sederhanakan menggunakan IDT-Freertos untuk menjalankan tes kualifikasi.

Untuk informasi tentang menggunakan baris perintah untuk menjalankan tes kualifikasi, lihatTes pertama papan mikrokontroler Anda.

Bagian ini menjelaskan prasyarat untuk menggunakan UI IDT-Freertos, dan menunjukkan cara memulai menjalankan pengujian kualifikasi di UI.

Topik

- Siapkan prasyarat untuk menjalankan suite kualifikasi FreeRTOS
- Memulai dengan IDT-Freertos UI

Siapkan prasyarat untuk menjalankan suite kualifikasi FreeRTOS

Bagian ini menjelaskan prasyarat untuk menguji mikrokontroler dengan. AWS IoT Device Tester

Topik

- Gunakan browser web yang didukung
- Unduh FreeRTOS
- Unduh IDT untuk FreeRTOS
- Buat dan konfigurasikan AWS akun
- AWS IoT Device Tester kebijakan terkelola

Gunakan browser web yang didukung

UI IDT-Freertos mendukung browser web berikut.

Peramban	Versi
Google Chrome	Tiga versi utama terbaru
Mozilla Firefox	Tiga versi utama terbaru
Microsoft Edge	Tiga versi utama terbaru

Peramban	Versi
Apple Safari untuk macOS	Tiga versi utama terbaru

Kami menyarankan Anda menggunakan Google Chrome atau Mozilla Firefox untuk pengalaman yang lebih baik.

> **Note**
>
> UI IDT-Freertos tidak mendukung Microsoft Internet Explorer.

Unduh FreeRTOS

Anda dapat mengunduh rilis FreeRTOS GitHubdari dengan perintah berikut:

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

di mana <FREERTOS_RELEASE_VERSION>adalah versi FreeRTOS (misalnya, 202007.00) sesuai dengan versi IDT yang tercantum dalam. Versi yang didukung AWS IoT Device Tester Ini memastikan Anda memiliki kode sumber lengkap, termasuk submodul, dan menggunakan versi IDT yang benar untuk versi FreeRTOS Anda, dan sebaliknya.

Windows memiliki batasan panjang jalur 260 karakter. Struktur jalur FreeRTOS memiliki banyak level dalam, jadi jika Anda menggunakan Windows, pertahankan jalur file Anda di bawah batas 260 karakter. Misalnya, kloning `C:\FreeRTOS` FreeRTOS ke bukan. `C:\Users\username\programs\projects\myproj\FreeRTOS\`

Pertimbangan untuk kualifikasi LTS (kualifikasi untuk FreeRTOS yang menggunakan perpustakaan LTS)

- Agar mikrokontroler Anda ditetapkan sebagai mendukung versi FreeRTOS berbasis dukungan jangka panjang (LTS) di Katalog Perangkat AWS Mitra, Anda harus menyediakan file manifes. Untuk informasi lebih lanjut, lihat Daftar Periksa Kualifikasi FreerTOS di Panduan Kualifikasi FreerTOS.

- Untuk memvalidasi bahwa mikrokontroler Anda mendukung versi FreeRTOS berbasis LTS dan memenuhi syarat untuk diserahkan ke Katalog Perangkat AWS Mitra, Anda harus menggunakan (AWS IoT Device Tester IDT) dengan FreeRTOS Qualification (FRQ) test suite versi v1.4.x.
- Support untuk FreeRTOS versi berbasis LTS terbatas pada FreeRTOS versi 202012.xx.

Unduh IDT untuk FreeRTOS

Setiap versi FreeRTOS memiliki versi IDT yang sesuai untuk FreerTOS untuk melakukan tes kualifikasi. Unduh versi IDT yang sesuai untuk FreeRTOS dari. Versi yang didukung AWS IoT Device Tester

Ekstrak IDT untuk FreeRTOS ke lokasi pada sistem file tempat Anda telah membaca dan menulis izin. Karena Microsoft Windows memiliki batas karakter untuk panjang jalur, ekstrak IDT untuk FreeRTOS ke direktori root seperti atau. `C:\` `D:\`

> **Note**
>
> Kami menyarankan Anda mengekstrak paket IDT ke drive lokal. Mengizinkan beberapa pengguna menjalankan IDT dari lokasi bersama, seperti direktori NFS atau folder bersama jaringan Windows, dapat mengakibatkan sistem tidak merespons atau korupsi data.

Buat dan konfigurasikan AWS akun

Mendaftar untuk Akun AWS

Jika Anda tidak memiliki Akun AWS, selesaikan langkah-langkah berikut untuk membuatnya.

Untuk mendaftar untuk Akun AWS

1. Buka https://portal.aws.amazon.com/billing/pendaftaran.
2. Ikuti petunjuk online.

   Bagian dari prosedur pendaftaran melibatkan menerima panggilan telepon atau pesan teks dan memasukkan kode verifikasi pada keypad telepon.

   Saat Anda mendaftar untuk sebuah Akun AWS, sebuah Pengguna root akun AWSdibuat. Pengguna root memiliki akses ke semua Layanan AWS dan sumber daya di akun. Sebagai praktik keamanan terbaik, tetapkan akses administratif ke pengguna, dan gunakan hanya pengguna root untuk melakukan tugas yang memerlukan akses pengguna root.

AWS mengirimi Anda email konfirmasi setelah proses pendaftaran selesai. Kapan saja, Anda dapat melihat aktivitas akun Anda saat ini dan mengelola akun Anda dengan masuk https://aws.amazon.comke/ dan memilih Akun Saya.

Buat pengguna dengan akses administratif

Setelah Anda mendaftar Akun AWS, amankan Pengguna root akun AWS, aktifkan AWS IAM Identity Center, dan buat pengguna administratif sehingga Anda tidak menggunakan pengguna root untuk tugas sehari-hari.

Amankan Anda Pengguna root akun AWS

1. Masuk ke AWS Management Consolesebagai pemilik akun dengan memilih pengguna Root dan memasukkan alamat Akun AWS email Anda. Di laman berikutnya, masukkan kata sandi.

   Untuk bantuan masuk dengan menggunakan pengguna root, lihat Masuk sebagai pengguna root di AWS Sign-In Panduan Pengguna.
2. Mengaktifkan autentikasi multi-faktor (MFA) untuk pengguna root Anda.

   Untuk petunjuk, lihat Mengaktifkan perangkat MFA virtual untuk pengguna Akun AWS root (konsol) Anda di Panduan Pengguna IAM.

Buat pengguna dengan akses administratif

1. Aktifkan Pusat Identitas IAM.

   Untuk mendapatkan petunjuk, silakan lihat Mengaktifkan AWS IAM Identity Center di Panduan Pengguna AWS IAM Identity Center .
2. Di Pusat Identitas IAM, berikan akses administratif ke pengguna.

   Untuk tutorial tentang menggunakan Direktori Pusat Identitas IAM sebagai sumber identitas Anda, lihat Mengkonfigurasi akses pengguna dengan default Direktori Pusat Identitas IAM di Panduan AWS IAM Identity Center Pengguna.

Masuk sebagai pengguna dengan akses administratif

- Untuk masuk dengan pengguna Pusat Identitas IAM, gunakan URL masuk yang dikirim ke alamat email saat Anda membuat pengguna Pusat Identitas IAM.

Untuk bantuan masuk menggunakan pengguna Pusat Identitas IAM, lihat Masuk ke portal AWS akses di Panduan AWS Sign-In Pengguna.

Tetapkan akses ke pengguna tambahan

1. Di Pusat Identitas IAM, buat set izin yang mengikuti praktik terbaik menerapkan izin hak istimewa paling sedikit.

   Untuk petunjuknya, lihat Membuat set izin di Panduan AWS IAM Identity Center Pengguna.
2. Tetapkan pengguna ke grup, lalu tetapkan akses masuk tunggal ke grup.

   Untuk petunjuk, lihat Menambahkan grup di Panduan AWS IAM Identity Center Pengguna.

AWS IoT Device Tester kebijakan terkelola

Untuk mengaktifkan penguji perangkat menjalankan dan mengumpulkan metrik, kebijakan `AWSIoTDeviceTesterForFreeRTOSFullAccess` terkelola berisi izin berikut:

- `iot-device-tester:SupportedVersion`

  Memberikan izin untuk mendapatkan daftar versi FreeRTOS dan versi suite pengujian yang didukung oleh IDT, sehingga tersedia dari versi. AWS CLI
- `iot-device-tester:LatestIdt`

  Memberikan izin untuk mendapatkan AWS IoT Device Tester versi terbaru yang tersedia untuk diunduh.
- `iot-device-tester:CheckVersion`

  Memberikan izin untuk memeriksa apakah kombinasi produk, rangkaian pengujian, dan AWS IoT Device Tester versi kompatibel.
- `iot-device-tester:DownloadTestSuite`

  Memberikan izin AWS IoT Device Tester untuk mengunduh suite pengujian.
- `iot-device-tester:SendMetrics`

  Memberikan izin untuk mempublikasikan data metrik AWS IoT Device Tester penggunaan.

Memulai dengan IDT-Freertos UI

Bagian ini menunjukkan cara menggunakan UI IDT-Freertos untuk membuat atau memodifikasi konfigurasi Anda, dan kemudian menunjukkan cara menjalankan pengujian.

Topik

- Konfigurasikan AWS kredensional
- Buka UI IDT-Freertos
- Buat konfigurasi baru
- Memodifikasi konfigurasi yang ada
- Jalankan tes kualifikasi

Konfigurasikan AWS kredensional

Anda harus mengonfigurasi kredensi untuk AWS pengguna yang Anda buat. Buat dan konfigurasikan AWS akun Anda dapat menentukan kredensial Anda dengan salah satu dari dua cara berikut:

- Di file kredensial
- Sebagai variabel lingkungan

Konfigurasikan AWS kredensional dengan file kredensial

IDT menggunakan file kredensials yang sama sebagai AWS CLI. Untuk informasi selengkapnya, lihat File konfigurasi dan kredensial.

Lokasi file kredensial bervariasi, tergantung pada sistem operasi yang Anda gunakan:

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

Tambahkan AWS kredensi Anda ke `credentials` file dalam format berikut:

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

> **Note**
>
> Jika Anda tidak menggunakan `default` AWS profil, pastikan untuk menentukan nama profil di UI IDT-Freertos. Untuk informasi selengkapnya tentang profil, lihat Konfigurasi dan setelan file kredensi.

Konfigurasikan AWS kredensional dengan variabel lingkungan

Variabel lingkungan adalah variabel yang dikelola oleh sistem operasi dan digunakan oleh perintah sistem. Mereka tidak disimpan jika Anda menutup sesi SSH. UI IDT-Freertos menggunakan variabel `AWS_ACCESS_KEY_ID` dan `AWS_SECRET_ACCESS_KEY` lingkungan untuk menyimpan kredensil Anda. AWS

Untuk mengatur variabel ini di Linux, macOS, atau Unix, gunakan export:

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

Untuk menetapkan variabel ini di Windows, gunakan set:

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

Buka UI IDT-Freertos

Untuk membuka UI IDT-Freertos

1. Unduh versi IDT-Freertos yang didukung dan ekstrak arsip yang diunduh ke lokasi di sistem file tempat Anda telah membaca dan menulis izin.
2. Jalankan perintah berikut untuk menavigasi ke direktori instalasi IDT-Freertos:

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

3. Jalankan perintah berikut untuk membuka UI IDT-Freertos:

   Linux

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

Windows

```
./devicetestergui_win_x64-64
```

macOS

```
./devicetestergui_mac_x86-64
```

> **Note**
>
> Di Mac, untuk memungkinkan sistem Anda menjalankan UI, buka System Preferences -> Security & Privacy. Ketika Anda menjalankan tes, Anda mungkin perlu melakukan ini tiga kali lagi.

UI IDT-Freertos terbuka di browser default Anda. Untuk informasi tentang browser yang didukung, lihatGunakan browser web yang didukung.

Buat konfigurasi baru

Jika Anda adalah pengguna pertama kali, maka Anda harus membuat konfigurasi baru untuk menyiapkan file konfigurasi JSON yang diperlukan IDT-Freertos untuk menjalankan pengujian. Anda kemudian dapat menjalankan tes atau memodifikasi konfigurasi yang telah dibuat.

Untuk contoh`config.json`,`device.json`, dan `userdata.json` file, lihatTes pertama papan mikrokontroler Anda. Untuk contoh `resource.json` file yang hanya digunakan untuk menjalankan tes Bluetooth Low Energy (BLE), lihatJalankan tes Bluetooth Low Energy.

Untuk membuat konfigurasi baru

1. Di UI IDT-Freertos, buka menu navigasi, lalu pilih Buat konfigurasi baru.

> **Important**
>
> Anda harus mengonfigurasi AWS kredensialnya sebelum membuka UI. Jika Anda belum mengonfigurasi kredensialnya, tutup jendela browser IDT-Freertos UI, ikuti langkah-langkahnya, lalu buka kembali UI IDT-Freertos. Konfigurasikan AWS kredensional

2. Ikuti panduan konfigurasi untuk masuk ke pengaturan konfigurasi IDT yang digunakan untuk menjalankan tes kualifikasi. Wizard mengkonfigurasi pengaturan berikut dalam file konfigurasi JSON yang terletak di direktori. *devicetester-extract-location*`/config`

   - AWS pengaturan Akun AWS —Informasi yang digunakan IDT-Freertos untuk membuat AWS sumber daya selama pengujian dijalankan. Pengaturan ini dikonfigurasi dalam `config.json` file.
   - Repositori FreerTOS —Jalur absolut ke repositori FreerTOS dan kode porting, dan jenis kualifikasi yang ingin Anda lakukan. Pengaturan ini dikonfigurasi dalam `userdata.json` file.

     Anda harus mem-port FreeRTOS untuk perangkat Anda sebelum Anda dapat menjalankan tes kualifikasi. Untuk informasi lebih lanjut, lihat Panduan Porting FreerTOS
   - Build dan flash —Perintah build dan flash untuk perangkat keras Anda yang memungkinkan IDT untuk membangun dan mem-flash tes ke papan Anda secara otomatis. Pengaturan ini dikonfigurasi dalam `userdata.json` file.
   - Perangkat —Pengaturan kumpulan perangkat untuk perangkat yang akan diuji. Pengaturan ini dikonfigurasi di `id` dan `sku` bidang, dan `devices` blok untuk kumpulan perangkat dalam `device.json` file.
   - Jaringan —Pengaturan untuk menguji dukungan komunikasi jaringan untuk perangkat Anda. Pengaturan ini dikonfigurasi di `features` blok `device.json` file, dan di `clientWifiConfig` dan `testWifiConfig` blok dalam `userdata.json` file.
   - Echo server —Pengaturan konfigurasi server gema untuk tes soket aman. Pengaturan ini dikonfigurasi dalam `userdata.json` file.

     Untuk informasi selengkapnya tentang konfigurasi server gema, lihathttps://docs.aws.amazon.com/freertos/latest/portingguide/afr-echo-server.html.
   - CMake— (Opsional) Pengaturan untuk menjalankan pengujian fungsionalitas CMake build. Konfigurasi ini hanya diperlukan jika Anda menggunakannya CMake sebagai sistem build. Pengaturan ini dikonfigurasi dalam `userdata.json` file.
   - BLE —Pengaturan untuk menjalankan tes fungsionalitas Bluetooth Low Energy. Pengaturan ini dikonfigurasi di `features` blok `device.json` file dan dalam `resource.json` file.
   - OTA —Pengaturan untuk menjalankan tes fungsionalitas OTA. Pengaturan ini dikonfigurasi di `features` blok `device.json` file dan dalam `userdata.json` file.
3. Pada halaman Tinjauan, verifikasi informasi konfigurasi Anda.

Setelah Anda selesai meninjau konfigurasi Anda, untuk menjalankan tes kualifikasi Anda, pilih Jalankan tes.

Memodifikasi konfigurasi yang ada

Jika Anda telah menyiapkan file konfigurasi untuk IDT, maka Anda dapat menggunakan UI IDT-Freertos untuk memodifikasi konfigurasi yang ada. Pastikan bahwa file konfigurasi yang ada tersedia di *devicetester-extract-location*`/config` direktori.

Untuk memodifikasi konfigurasi baru

1. Di UI IDT-Freertos, buka menu navigasi, lalu pilih Edit konfigurasi yang ada.

   Dasbor konfigurasi menampilkan informasi tentang pengaturan konfigurasi yang ada. Jika konfigurasi salah atau tidak tersedia, status untuk konfigurasi tersebut adalah`Error validating configuration`.
2. Untuk mengubah pengaturan konfigurasi yang ada, selesaikan langkah-langkah berikut:

   a. Pilih nama pengaturan konfigurasi untuk membuka halaman pengaturannya.

   b. Ubah pengaturan, lalu pilih Simpan untuk membuat ulang file konfigurasi yang sesuai.

Setelah Anda selesai memodifikasi konfigurasi Anda, verifikasi bahwa semua pengaturan konfigurasi Anda lulus validasi. Jika status untuk setiap pengaturan konfigurasi adalah`Valid`, Anda dapat menjalankan tes kualifikasi menggunakan konfigurasi ini.

Jalankan tes kualifikasi

Setelah Anda membuat konfigurasi untuk IDT-Freertos, Anda dapat menjalankan tes kualifikasi Anda.

Untuk menjalankan tes kualifikasi

1. Validasi konfigurasi Anda.
2. Di menu navigasi, pilih Jalankan tes.
3. Untuk memulai uji coba, pilih Mulai tes.

IDT-Freertos menjalankan tes kualifikasi, dan menampilkan ringkasan uji coba dan kesalahan apa pun di konsol Test runner. Setelah uji coba selesai, Anda dapat melihat hasil pengujian dan log dari lokasi berikut:

- Hasil tes terletak di *devicetester-extract-location*/results/*execution-id* direktori.
- Log uji terletak di *devicetester-extract-location*/results/*execution-id*/logs direktori.

Untuk informasi selengkapnya tentang hasil pengujian dan log, lihat Lihat IDT untuk hasil FreeRTOS danLihat IDT untuk log FreeRTOS.

Jalankan tes Bluetooth Low Energy

Bagian ini menjelaskan cara mengatur dan menjalankan tes Bluetooth Low Energy menggunakan AWS IoT Device Tester FreeRTOS.

Tes Bluetooth tidak diperlukan untuk kualifikasi inti. Jika Anda tidak ingin menguji perangkat Anda dengan dukungan Bluetooth FreeRTOS, Anda dapat melewati pengaturan ini, pastikan untuk membiarkan fitur BLE di device.json disetel ke. No

Prasyarat

- Ikuti petunjuk dalam Tes pertama papan mikrokontroler Anda.
- Raspberry Pi 4B atau 3B +. (Diperlukan untuk menjalankan aplikasi pendamping Raspberry Pi BLE)
- Kartu micro SD dan adaptor kartu SD untuk perangkat lunak Raspberry Pi.

Pengaturan Raspberry Pi

Untuk menguji kemampuan BLE perangkat yang diuji (DUT), Anda harus memiliki Raspberry Pi Model 4B atau 3B +.

Untuk mengatur Raspberry Pi Anda untuk menjalankan tes BLE

1. Unduh salah satu gambar Yocto kustom yang berisi perangkat lunak yang diperlukan untuk melakukan tes.
   - Gambar untuk Raspberry Pi 4B
   - Gambar untuk Raspberry Pi 3B+

> **Note**
>
> Gambar Yocto hanya boleh digunakan untuk pengujian dengan AWS IoT Device Tester FreeRTOS dan bukan untuk tujuan lain.

2. Flash gambar yocto ke kartu SD untuk Raspberry Pi.

   - Menggunakan alat penulisan kartu SD seperti Etcher, flash *image-name*`.rpi-sd.img` file yang diunduh ke kartu SD. Karena citra sistem operasi berukuran besar, langkah ini mungkin memerlukan waktu. Kemudian keluarkan kartu SD Anda dari komputer Anda dan masukkan kartu microSD ke Raspberry Pi Anda.

3. Konfigurasikan Raspberry Pi Anda.

   a. Untuk boot pertama, kami sarankan Anda menghubungkan Raspberry Pi ke monitor, keyboard, dan mouse.

   b. Hubungkan Raspberry Pi Anda ke sumber daya micro USB.

   c. Masuk menggunakan kredensyal default. Untuk ID pengguna, masukkan**`root`**. Untuk kata sandi, masukkan**`idtafr`**.

   d. Menggunakan koneksi Ethernet atau Wi-Fi, sambungkan Raspberry Pi ke jaringan Anda.

      i. Untuk menghubungkan Raspberry Pi Anda melalui Wi-Fi, buka `/etc/wpa_supplicant.conf` di Raspberry Pi dan tambahkan kredensi Wi-Fi Anda ke konfigurasi. `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. Jalankan `ifup wlan0` untuk memulai koneksi Wi-Fi. Mungkin perlu satu menit untuk terhubung ke jaringan Wi-Fi Anda.

e. Untuk koneksi Ethernet, jalankan`ifconfig eth0`. Untuk koneksi Wi-Fi, jalankan`ifconfig wlan0`. Catat alamat IP, yang muncul seperti `inet addr` pada output perintah. Anda memerlukan alamat IP nanti dalam prosedur ini.

f. (Opsional) Tes menjalankan perintah pada Raspberry Pi melalui SSH menggunakan kredensyal default untuk gambar yocto. Untuk keamanan tambahan, kami sarankan Anda mengatur otentikasi kunci publik untuk SSH dan menonaktifkan SSH berbasis kata sandi.

   i. Buat kunci SSH menggunakan perintah OpenSSL`ssh-keygen`. Jika Anda sudah memiliki key pair SSK di komputer host Anda, itu adalah praktik terbaik untuk membuat yang baru AWS IoT Device Tester untuk memungkinkan FreeRTOS masuk ke Raspberry Pi Anda.

   > **Note**
   >
   > Windows tidak datang dengan klien SSH yang diinstal. Untuk informasi tentang cara menginstal klien SSH di Windows, lihat Mengunduh Perangkat Lunak SSH.

   ii. Perintah `ssh-keygen` meminta Anda untuk memberikan nama dan path untuk menyimpan pasangan kunci tersebut. Secara default, file pasangan kunci diberi nama `id_rsa` (kunci privat) dan `id_rsa.pub` (kunci publik). Di macOS dan Linux, lokasi default file ini adalah `~/.ssh/`. Di Windows, lokasi default untuk file ini adalah `C:\Users\`*`user-name`*.

   iii. Ketika Anda diminta untuk frase kunci, cukup tekan ENTER untuk melanjutkan.

   iv. Untuk menambahkan kunci SSH Anda ke Raspberry Pi Anda sehingga AWS IoT Device Tester FreeRTOS dapat masuk ke perangkat, gunakan `ssh-copy-id` perintah dari komputer host Anda. Perintah ini menambahkan kunci publik Anda ke dalam `~/.ssh/authorized_keys` file di Raspberry Pi Anda.

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

   v. Saat diminta kata sandi, masukkan**`idtafr`**. Ini adalah kata sandi default untuk gambar yocto.

   > **Note**
   >
   > `ssh-copy-id`Perintah mengasumsikan kunci publik diberi nama`id_rsa.pub`. Di macOS dan Linux, lokasi defaultnya adalah. `~/.ssh/` Di Windows, lokasi default untuk file ini adalah `C:\Users\`*`user-name`*`\.ssh`. Jika Anda

memberikan kunci publik nama yang berbeda atau menyimpannya di lokasi yang berbeda, Anda harus menentukan path yang memenuhi syarat untuk kunci publik SSH Anda dengan menggunakan `-i` untuk `ssh-copy-id` (misalnya, `ssh-copy-id -i ~/my/path/myKey.pub`). Untuk informasi lebih lanjut tentang cara membuat kunci SSH dan menyalin kunci publik, lihat SSH-COPY-ID.

vi. Untuk menguji apakah otentikasi kunci publik berfungsi, jalankan`ssh -i` */my/path/myKey* `root@`*raspberry-pi-device-ip*.

Jika Anda tidak diminta untuk kata sandi, otentikasi kunci publik Anda berfungsi.

vii. Verifikasi bahwa Anda dapat masuk ke Raspberry Pi Anda menggunakan kunci publik, lalu nonaktifkan SSH berbasis kata sandi.

A. Pada Raspberry Pi, edit `/etc/ssh/sshd_config` file.

B. Atur atribut `PasswordAuthentication` ke no.

C. Simpan dan tutup file `sshd_config`.

D. Muat ulang server SSH dengan menjalankan. `/etc/init.d/sshd reload`

g. Buat `resource.json` file.

i. Di direktori tempat Anda mengekstrak AWS IoT Device Tester, buat file bernama. `resource.json`

ii. Tambahkan informasi berikut tentang Raspberry Pi Anda ke file, ganti *rasp-pi-ip-address* dengan alamat IP Raspberry Pi Anda.

```
[
    {
        "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. Jika Anda tidak memilih untuk menggunakan otentikasi kunci publik untuk SSH, tambahkan berikut ini ke `connectivity` bagian `resource.json` file.

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

iv. (Opsional) Jika Anda memilih untuk menggunakan otentikasi kunci publik untuk SSH, tambahkan berikut ini ke `connectivity` bagian `resource.json` file.

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

Pengaturan perangkat FreeRTOS

Dalam `device.json` file Anda, atur BLE fitur ke`Yes`. Jika Anda memulai dengan `device.json` file dari sebelum tes Bluetooth tersedia, Anda perlu menambahkan fitur untuk BLE ke `features` array:

```
{
    ...
```

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

Jalankan tes BLE

Setelah Anda mengaktifkan fitur BLE`device.json`, pengujian BLE berjalan saat Anda menjalankan `devicetester_`*`[linux | mac | win_x86-64]`* `run-suite` tanpa menentukan id grup.

Jika Anda ingin menjalankan tes BLE secara terpisah, Anda dapat menentukan ID grup untuk BLE:`devicetester_`*`[linux | mac | win_x86-64]`* `run-suite --userdata` *`path-to-userdata`*`/userdata.json --group-id FullBLE`.

Untuk kinerja yang paling andal, letakkan Raspberry Pi Anda di dekat perangkat yang sedang diuji (DUT).

Memecahkan masalah tes BLE

Pastikan Anda telah mengikuti langkah-langkahnyaTes pertama papan mikrokontroler Anda. Jika tes selain BLE gagal, maka masalahnya kemungkinan besar bukan karena konfigurasi Bluetooth.

Jalankan suite kualifikasi FreeRTOS

Anda menggunakan AWS IoT Device Tester untuk FreeRTOS executable untuk berinteraksi dengan IDT untuk FreeRTOS. Contoh baris perintah berikut menunjukkan cara menjalankan tes kualifikasi untuk kumpulan perangkat (satu set perangkat yang identik).

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
```

Menjalankan serangkaian tes pada kolam perangkat. `userdata.json`File harus berada di *devicetester_extract_location*`/devicetester_afreertos_`*[win|mac|linux]*`/configs/` direktori.

> **Note**
>
> Jika Anda menjalankan IDT untuk FreeRTOS di Windows, gunakan garis miring maju (/) untuk menentukan jalur ke file. `userdata.json`

Gunakan perintah berikut untuk menjalankan grup pengujian tertentu:

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

`pool-id`Parameter `suite-id` dan bersifat opsional jika Anda menjalankan rangkaian pengujian tunggal pada satu kumpulan perangkat (yaitu, Anda hanya memiliki satu kumpulan perangkat yang ditentukan dalam `device.json` file Anda).

Gunakan perintah berikut untuk menjalankan kasus uji tertentu dalam kelompok uji:

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

Anda dapat menggunakan `list-test-cases` perintah untuk daftar kasus uji dalam kelompok uji.

IDT untuk opsi baris perintah FreeRTOS

grup-id

(Opsional) Kelompok uji yang akan dijalankan, sebagai daftar yang dipisahkan koma. Jika tidak ditentukan, IDT akan menjalankan semua grup uji di rangkaian tes.

pool-id

(Opsional) Kumpulan perangkat untuk diuji. Ini diperlukan jika Anda menentukan beberapa kumpulan perangkat di`device.json`. Jika Anda hanya memiliki satu kumpulan perangkat, Anda dapat menghilangkan opsi ini.

suite-id

(Opsional) Versi test suite untuk dijalankan. Jika tidak ditentukan, IDT menggunakan versi terbaru dalam direktori tes pada sistem Anda.

> **Note**
>
> Mulai IDT v3.0.0, IDT memeriksa secara online untuk rangkaian pengujian yang lebih baru. Untuk informasi selengkapnya, lihat Versi rangkaian tes.

tes-id

(Opsional) Tes yang akan dijalankan, sebagai daftar yang dipisahkan koma. Jika ditentukan, `group-id` harus menentukan satu grup.

Example

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

update-idt

(Opsional) Jika parameter ini tidak disetel dan versi IDT yang lebih baru tersedia, Anda akan diminta untuk memperbarui IDT. Jika parameter ini disetel keY, IDT akan menghentikan eksekusi pengujian jika mendeteksi bahwa versi yang lebih baru tersedia. Jika parameter ini diatur keN, IDT akan melanjutkan eksekusi pengujian.

update-managed-policy

(Opsional) Jika parameter ini tidak digunakan dan IDT mendeteksi bahwa kebijakan terkelola Anda tidak up-to-date, Anda akan diminta untuk memperbarui kebijakan terkelola. Jika parameter ini disetel keY, IDT akan menghentikan eksekusi pengujian jika mendeteksi bahwa

kebijakan terkelola Anda tidak. up-to-date Jika parameter ini diatur keN, IDT akan melanjutkan eksekusi pengujian.

upgrade-test-suite

(Opsional) Jika tidak digunakan, dan versi test suite yang lebih baru tersedia, Anda diminta untuk mengunduhnya. Untuk menyembunyikan prompt, tentukan y untuk selalu mengunduh rangkaian pengujian terbaru, atau n menggunakan rangkaian pengujian yang ditentukan atau versi terbaru di sistem Anda.

Example

Contoh

Untuk selalu mengunduh dan menggunakan test suite terbaru, gunakan perintah berikut.

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

Untuk menggunakan test suite terbaru pada sistem Anda, gunakan perintah berikut.

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

-h

Gunakan opsi bantuan untuk mempelajari lebih lanjut tentang `run-suite` opsi.

Example

Contoh

```
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
```

`userdata.json`File harus ditempatkan di *`devicetester_extract_location`*`/devicetester_afreertos_`*`[win|mac|linux]`*`/configs/` direktori.

> **Note**
>
> Jika Anda menjalankan IDT untuk FreeRTOS di Windows, gunakan garis miring maju (/) untuk menentukan jalur ke file. `userdata.json`

Gunakan perintah berikut untuk menjalankan kelompok uji tertentu.

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

`suite-id`dan `pool-id` bersifat opsional jika Anda menjalankan rangkaian pengujian tunggal pada satu kumpulan perangkat (yaitu, Anda hanya memiliki satu kumpulan perangkat yang ditentukan dalam `device.json` file Anda).

IDT untuk opsi baris perintah FreeRTOS

grup-id

(Opsional) Menentukan kelompok uji.

pool-id

Menentukan kolam perangkat untuk menguji. Jika Anda hanya memiliki satu kumpulan perangkat, Anda dapat menghilangkan opsi ini.

suite-id

(Opsional) Menentukan rangkaian pengujian untuk dijalankan.

IDT untuk perintah FreeRTOS

Perintah IDT untuk FreeRTOS mendukung operasi berikut:

IDT v3.0.0 and later

`help`

Mencantumkan informasi tentang perintah yang ditentukan.

`list-groups`

Daftar grup dalam suite tertentu.

`list-suites`

Daftar suite yang tersedia.

`list-supported-products`

Daftar produk yang didukung dan versi rangkaian pengujian.

`list-supported-versions`

Daftar FreeRTOS dan versi test suite yang didukung oleh versi IDT saat ini.

`list-test-cases`

Daftar kasus uji dalam grup tertentu.

`run-suite`

Menjalankan serangkaian tes pada kolam perangkat.

Gunakan `--suite-id` opsi untuk menentukan versi suite pengujian, atau hilangkan untuk menggunakan versi terbaru di sistem Anda.

Gunakan `--test-id` untuk menjalankan kasus uji individual.

Example

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

Untuk daftar lengkap opsi lihatJalankan suite kualifikasi FreeRTOS.

> **Note**
>
> Mulai IDT v3.0.0, IDT memeriksa secara online untuk rangkaian pengujian yang lebih baru. Untuk informasi selengkapnya, lihat Versi rangkaian tes.

IDT v1.7.0 and earlier

`help`

Mencantumkan informasi tentang perintah yang ditentukan.

`list-groups`

Daftar grup dalam suite tertentu.

`list-suites`

Daftar suite yang tersedia.

`run-suite`

Menjalankan serangkaian tes pada kolam perangkat.

Tes untuk kualifikasi ulang

Saat versi baru IDT untuk tes kualifikasi FreeRTOS dirilis, atau saat Anda memperbarui paket atau driver perangkat khusus papan Anda, Anda dapat menggunakan IDT untuk FreeRTOS untuk menguji papan mikrokontroler Anda. Untuk kualifikasi berikutnya, pastikan Anda memiliki versi terbaru FreeRTOS dan IDT untuk FreeRTOS dan jalankan tes kualifikasi lagi.

Lihat IDT untuk hasil FreeRTOS

Saat berjalan, IDT menuliskan kesalahan ke konsol, file log, dan laporan tes. Setelah IDT menyelesaikan rangkaian pengujian kualifikasi, IDT menulis ringkasan uji coba ke konsol dan menghasilkan dua laporan pengujian. Laporan-laporan ini dapat ditemukan di *`devicetester-extract-location`*`/results/`*`execution-id`*`/`. Kedua laporan menangkap hasil dari pelaksanaan kualifikasi tes suite.

`awsiotdevicetester_report.xml`Ini adalah laporan pengujian kualifikasi yang Anda kirimkan AWS untuk mencantumkan perangkat Anda di Katalog Perangkat AWS Mitra. Laporan tersebut berisi elemen berikut:

- IDT untuk versi FreeRTOS.
- Versi FreeRTOS yang diuji.
- Fitur FreeRTOS yang didukung oleh perangkat berdasarkan tes yang dilewati.
- SKU dan nama perangkat yang ditentukan dalam `device.json` file.

- Fitur perangkat yang ditentukan dalam `device.json` file.
- Ringkasan agregat hasil uji kasus.
- Rincian hasil kasus uji oleh pustaka yang diuji berdasarkan fitur perangkat (misalnya, FullMQTT FullWiFi, dan sebagainya).
- Apakah kualifikasi FreeRTOS ini untuk versi 202012.00 yang menggunakan perpustakaan LTS.

`FRQ_Report.xml`Ini adalah laporan dalam format JUnit XHTML standar. Anda dapat mengintegrasikannya ke dalam CI/CD platform seperti Jenkins, Bamboo, dan sebagainya. Laporan tersebut berisi elemen berikut:

- Ringkasan agregat hasil kasus uji.
- Rincian hasil kasus uji oleh pustaka yang diuji berdasarkan fitur perangkat.

Menafsirkan IDT untuk hasil FreeRTOS

Bagian laporan di `awsiotdevicetester_report.xml` atau `FRQ_Report.xml` mencantumkan hasil tes yang dijalankan.

Tag XHTML pertama `<testsuites>` berisi ringkasan keseluruhan eksekusi pengujian. Misalnya:

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

Atribut yang digunakan dalam **`<testsuites>`** tag

`name`

Nama dari tes suite.

`time`

Waktu, dalam hitungan detik, yang dibutuhkan untuk menjalankan suite kualifikasi.

`tests`

Jumlah kasus uji yang dieksekusi.

`failures`

Jumlah kasus uji yang dijalankan, tetapi tidak lulus.

`errors`

Jumlah kasus uji yang tidak dapat dijalankan oleh IDT untuk FreeRTOS.

`disabled`

Atribut ini tidak digunakan dan bisa diabaikan.

Jika tidak ada kegagalan atau kesalahan kasus uji, perangkat Anda memenuhi persyaratan teknis untuk menjalankan FreeRTOS dan dapat beroperasi dengan layanan. AWS IoT Jika Anda memilih untuk mencantumkan perangkat Anda di Katalog Perangkat AWS Mitra, Anda dapat menggunakan laporan ini sebagai bukti kualifikasi.

Jika terjadi kegagalan atau kesalahan kasus uji, Anda dapat mengidentifikasi kasus uji yang gagal dengan meninjau tag `<testsuites>` XHTML. Tag `<testsuite>` XHTML di dalam `<testsuites>` tag menunjukkan ringkasan hasil kasus uji untuk grup uji.

```
<testsuite name="FullMQTT" package="" tests="16" failures="0" time="76"
disabled="0" errors="0" skipped="0">
```

Formatnya mirip dengan `<testsuites>` tag, tetapi dengan atribut `skipped` yang disebut yang tidak digunakan dan dapat diabaikan. Di dalam setiap tag `<testsuite>` XHTML, ada `<testcase>` tag untuk setiap kasus uji yang dieksekusi untuk kelompok uji. Misalnya:

```
<testcase classname="mcu.Full_MQTT" name="AFQP_MQTT_Connect_HappyCase"
attempts="1"></testcase>
```

Atribut yang digunakan dalam **`<awsproduct>`** tag

`name`

Nama produk yang sedang diuji.

`version`

Versi produk yang sedang diuji.

`sdk`

Jika Anda menjalankan IDT dengan SDK, blok ini berisi nama dan versi SDK Anda. Jika Anda tidak menjalankan IDT dengan SDK, maka blok ini berisi:

```
<sdk>
```

```
    <name>N/A</vame>
    <version>N/A</version>
</sdk>
```

`features`

Fitur divalidasi. Fitur yang ditandai sebagai `required` wajib mengirimkan forum Anda untuk kualifikasi. Cuplikan berikut menunjukkan bagaimana ini muncul dalam file. `awsiotdevicetester_report.xml`

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

Fitur yang ditandai sebagai `optional` tidak diperlukan untuk kualifikasi. Potongan berikut menunjukkan fitur opsional.

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

Jika tidak ada kegagalan pengujian atau kesalahan untuk fitur yang diperlukan, perangkat Anda memenuhi persyaratan teknis untuk menjalankan FreeRTOS dan dapat beroperasi dengan layanan. AWS IoT Jika ingin mencantumkan perangkat di Katalog Perangkat AWS Mitra, Anda dapat menggunakan laporan ini sebagai bukti kualifikasi.

Jika terjadi kegagalan atau kesalahan uji, Anda dapat mengidentifikasi pengujian yang gagal tersebut dengan meninjau tanda XML `<testsuites>`. Tag XML `<testsuite>` di dalam tag `<testsuites>` menunjukkan ringkasan hasil tes untuk grup uji. Misalnya:

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

Formatnya mirip dengan `<testsuites>` tag, tetapi memiliki `skipped` atribut yang tidak digunakan dan dapat diabaikan. Di dalam masing-masing tanda XML `<testsuite>`, terdapat tanda `<testcase>` untuk setiap tes yang dieksekusi untuk suatu grup uji. Misalnya:

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

`lts`

Benar jika Anda memenuhi syarat untuk versi FreeRTOS yang menggunakan pustaka LTS, false sebaliknya.

Atribut yang digunakan dalam **`<testcase>`** tag

`name`

Nama kasus uji.

`attempts`

Berapa kali IDT untuk FreeRTOS mengeksekusi kasus uji.

Ketika tes gagal atau kesalahan terjadi, tanda `<failure>` atau `<error>` akan ditambahkan ke tanda `<testcase>` dengan informasi untuk pemecahan masalah. Misalnya:

```
<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>
```

Untuk informasi selengkapnya, lihat Memecahkan masalah kesalahan.

Lihat IDT untuk log FreeRTOS

Anda dapat menemukan log yang dihasilkan IDT untuk FreeRTOS dari eksekusi pengujian. *devicetester-extract-location*`/results/`*execution-id*`/logs` Dua rangkaian log yang dihasilkan:

`test_manager.log`

Berisi log yang dihasilkan dari IDT untuk FreeRTOS (misalnya, konfigurasi terkait log dan pembuatan laporan).

***`test_group_id`*`__`*`test_case_id`*`.log`**(misalnya,**`FullMQTT__Full_MQTT.log`**)

File log untuk kasus uji, termasuk output dari perangkat yang diuji. File log diberi nama sesuai dengan kelompok uji dan kasus uji yang dijalankan.

Kembangkan dan jalankan rangkaian pengujian IDT Anda sendiri

Mulai IDT v4.0.0, IDT untuk FreeRTOS menggabungkan pengaturan konfigurasi standar dan format hasil dengan lingkungan rangkaian pengujian yang memungkinkan Anda mengembangkan rangkaian pengujian khusus untuk perangkat dan perangkat lunak perangkat Anda. Anda dapat menambahkan

tes khusus untuk validasi internal Anda sendiri atau memberikannya kepada pelanggan Anda untuk verifikasi perangkat.

Gunakan IDT untuk mengembangkan dan menjalankan rangkaian tes kustom, sebagai berikut:

Untuk mengembangkan suite uji khusus

- Buat rangkaian pengujian dengan logika pengujian khusus untuk perangkat yang ingin Anda uji.
- Sediakan IDT dengan rangkaian tes kustom Anda untuk menguji runner. Sertakan informasi tentang konfigurasi pengaturan khusus untuk rangkaian pengujian Anda.

Untuk menjalankan suite pengujian khusus

- Atur perangkat yang ingin Anda uji.
- Terapkan konfigurasi pengaturan yang diperlukan oleh rangkaian tes yang ingin Anda gunakan.
- Gunakan IDT untuk menjalankan rangkaian tes kustom Anda.
- Lihat hasil tes dan log eksekusi untuk tes yang dijalankan oleh IDT.

Unduh versi terbaru AWS IoT Device Tester for FreeRTOS

Unduh versi terbaru IDT dan mengekstraksi perangkat lunak ke lokasi pada sistem file Anda di mana Anda telah membaca dan menulis izin.

> **Note**
>
> IDT tidak mendukung yang sedang dijalankan oleh beberapa pengguna dari lokasi bersama, seperti direktori NFS atau folder bersama jaringan Windows. Kami sarankan Anda mengekstraksi paket IDT ke drive lokal dan menjalankan biner IDT pada workstation lokal Anda.
>
> Windows memiliki batasan panjang jalur 260 karakter. Jika Anda menggunakan Windows, ekstraksi IDT ke direktori root seperti `C:\` atau `D:\` agar jalur Anda tetap di bawah batas 260 karakter.

Alur kerja test suite

Rangkaian uji terdiri dari tiga jenis file:

- File konfigurasi yang menyediakan IDT informasi tentang cara menjalankan rangkaian pengujian.
- File yang dapat dieksekusi yang digunakan oleh IDT untuk menjalankan uji kasus.

- File tambahan diperlukan untuk menjalankan tes.

Selesaikan langkah-langkah dasar berikut untuk membuat tes IDT kustom:

1. Buat file konfigurasi untuk rangkaian pengujian Anda.
2. Buat uji kasus yang dapat dieksekusi yang berisi logika ujian untuk rangkaian tes anda.
3. Verifikasi dan dokumentasi informasi konfigurasi yang diperlukan untuk menguji runner untuk menjalankan rangkaian tes.
4. Verifikasi bahwa IDT dapat menjalankan rangkain tes Anda dan buat hasil pengujian seperti yang diharapkan.

Untuk cepat membangun rangkaian kustom sampel dan menjalankannya, ikuti petunjuk di Tutorial: Bangun dan jalankan sampel rangkaian tes IDT.

Untuk memulai membuat rangkaian tes kustom di Python, lihat Tutorial: Kembangkan rangkaian tes IDT sederhana.

Tutorial: Bangun dan jalankan sampel rangkaian tes IDT

AWS IoT Device Tester Unduhan menyertakan kode sumber untuk rangkaian pengujian sampel. Anda dapat menyelesaikan tutorial ini untuk membangun dan menjalankan rangkaian pengujian sampel untuk memahami bagaimana Anda dapat menggunakan FreeRTOS AWS IoT Device Tester untuk menjalankan suite pengujian khusus. Meskipun tutorial ini menggunakan SSH, berguna untuk mempelajari cara menggunakan AWS IoT Device Tester dengan perangkat FreeRTOS.

Dalam tutorial ini, Anda akan melakukan langkah-langkah berikut:

1. Bangun rangkaian uji sampel
2. Gunakan IDT untuk menjalankan rangkaian uji sampel

Topik

- Siapkan prasyarat untuk rangkaian pengujian sampel
- Konfigurasi informasi perangkat untuk IDT
- Bangun rangkaian uji sampel
- Gunakan IDT untuk menjalankan rangkaian uji sampel
- Memecahkan masalah kesalahan

Siapkan prasyarat untuk rangkaian pengujian sampel

Untuk menyelesaikan tutorial ini, Anda memerlukan hal berikut ini:

- Persyaratan komputer host
  - Versi terbaru dari AWS IoT Device Tester
  - Python 3.7 atau yang lebih baru

    Untuk memeriksa versi Python yang diinstal pada komputer Anda, jalankan perintah berikut:

    ```
    python3 --version
    ```

    Pada Windows, jika penggunaan perintah ini menghasilkan kesalahan, gunakan `python --version` sebagai gantinya. Jika nomor versi yang dikembalikan adalah 3,7 atau lebih besar, jalankan perintah berikut di terminal Powershell untuk mengatur `python3` sebagai alias untuk perintah `python`.

    ```
    Set-Alias -Name "python3" -Value "python"
    ```

    Jika tidak ada informasi versi yang dikembalikan atau jika nomor versi kurang dari 3,7, ikuti petunjuk di Mengunduh Py untuk menginstal Python 3.7+. Untuk informasi selengkapnya, lihat dokumentasi Python.
  - urllib3

    Untuk memverifikasi bahwa `urllib3` diinstal dengan benar, jalankan perintah berikut:

    ```
    python3 -c 'import urllib3'
    ```

    Jika `urllib3` belum terinstal, gunakan perintah berikut untuk menginstalnya:

    ```
    python3 -m pip install urllib3
    ```
- Persyaratan perangkat
  - Perangkat dengan sistem operasi Linux dan koneksi jaringan ke jaringan yang sama dengan komputer host Anda.

    Kami menyarankan agar Anda menggunakan Raspberry Pi dengan OS Raspberry Pi. Pastikan Anda mengatur SSH pada Raspberry Pi Anda untuk terhubung secara jarak jauh ke sana.

Konfigurasi informasi perangkat untuk IDT

Konfigurasi informasi perangkat Anda untuk IDT untuk menjalankan tes. Anda harus memperbarui templat `device.json` yang terletak di folder *<device-tester-extract-location>*`/configs` dengan informasi berikut.

```
[
  {
    "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>"
            }
          }
        }
      }
    ]
  }
]
```

Di objek `devices`, berikan informasi berikut:

`id`

Pengenal unik yang ditetapkan pengguna untuk perangkat Anda.

`connectivity.ip`

Alamat IP perangkat Anda.

`connectivity.port`

Opsional. Nomor port yang digunakan untuk koneksi SSH ke perangkat Anda.

`connectivity.auth`

Informasi autentikasi untuk koneksi tersebut.

Properti ini hanya berlaku jika `connectivity.protocol` diatur ke `ssh`.

`connectivity.auth.method`

Metode autentikasi yang digunakan untuk mengakses perangkat melalui protokol konektivitas yang diberikan.

Nilai yang didukung adalah:

- `pki`
- `password`

`connectivity.auth.credentials`

Kredensial yang digunakan untuk autentikasi.

`connectivity.auth.credentials.user`

Nama pengguna yang digunakan untuk masuk ke perangkat Anda.

`connectivity.auth.credentials.privKeyPath`

Jalur lengkap ke kunci pribadi yang digunakan untuk masuk ke perangkat Anda.

Nilai ini hanya berlaku jika `connectivity.auth.method` diatur ke `pki`.

`devices.connectivity.auth.credentials.password`

Kata sandi yang digunakan untuk masuk ke perangkat Anda.

Nilai ini hanya berlaku jika `connectivity.auth.method` diatur ke `password`.

> **Note**
>
> Tentukan `privKeyPath` hanya jika `method` diatur ke `pki`.
> Tentukan `password` hanya jika `method` diatur ke `password`.

Bangun rangkaian uji sampel

Folder *<device-tester-extract-location>*/samples/python berisi contoh file konfigurasi, kode sumber, dan SDK Klien IDT yang dapat Anda gabungkan ke dalam rangkaian tes dengan menggunakan skrip build yang disediakan. Pohon direktori berikut menunjukkan lokasi file contoh ini:

```
<device-tester-extract-location>
### ...
### tests
### samples
#   ### ...
#   ### python
#       ### configuration
#       ### src
#       ### build-scripts
#           ### build.sh
#           ### build.ps1
### sdks
    ### ...
    ### python
        ### idt_client
```

Untuk membangun rangkaian tes, jalankan perintah berikut pada komputer host Anda:

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
```

Hal ini akan menciptakan rangkaian uji sampel di folder IDTSampleSuitePython_1.0.0 dalam folder *<device-tester-extract-location>*/tests. Tinjau file dalam IDTSampleSuitePython_1.0.0 folder untuk memahami bagaimana rangkaian pengujian sampel terstruktur dan untuk melihat berbagai contoh executable test case dan file konfigurasi pengujian.

> **Note**
>
> Rangkaian pengujian sampel menyertakan kode sumber python. Jangan sertakan informasi sensitif dalam kode rangkaian pengujian Anda.

Langkah berikutnya: Gunakan IDT untuk menjalankan rangkaian tes sampel yang Anda buat.

Gunakan IDT untuk menjalankan rangkaian uji sampel

Untuk membangun rangkaian tes, jalankan perintah berikut pada komputer host Anda:

```
cd <device-tester-extract-location>/bin
./devicetester_[linux | mac | win_x86-64] run-suite --suite-id IDTSampleSuitePython
```

IDT menjalankan sampel rangkaian tes dan mengalirkan hasil ke konsol. Ketika tes telah selesai berjalan, Anda akan melihat informasi berikut:

```
========== 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
```

Memecahkan masalah kesalahan

Gunakan informasi berikut untuk membantu menyelesaikan masalah dengan menyelesaikan tutorial.

Kasus uji tidak berjalan dengan sukses

- Jika tes tidak berjalan sukses, IDT akan mengalirkan log kesalahan ke konsol yang dapat membantu Anda memecahkan masalah uji coba. Pastikan Anda memenuhi semua prasyarat untuk tutorial ini.

Tidak dapat terhubung ke perangkat yang sedang diuji

Verifikasi hal berikut:

- File `device.json` Anda berisi alamat IP, port, dan informasi autentikasi yang benar.
- Anda dapat terhubung ke perangkat Anda melalui SSH dari komputer host Anda.

Tutorial: Kembangkan rangkaian tes IDT sederhana

Rangkaian tes menggabungkan hal berikut:

- Uji executable yang berisi logika uji
- File konfigurasi yang menjelaskan rangkaian pengujian

Tutorial ini menunjukkan cara menggunakan IDT untuk FreeRTOS untuk mengembangkan rangkaian uji Python yang berisi satu kasus uji. Meskipun tutorial ini menggunakan SSH, berguna untuk mempelajari cara menggunakan AWS IoT Device Tester dengan perangkat FreeRTOS.

Dalam tutorial ini, Anda akan melakukan langkah-langkah berikut:

1. Buat direktori rangkaian tes
2. Buat file konfigurasi
3. Buat executable uji kasus
4. Jalankan rangkaian tes

Ikuti langkah-langkah di bawah ini untuk menyelesaikan tutorial untuk mengembangkan rangkaian tes IDT sederhana.

Topik

- Siapkan prasyarat untuk rangkaian pengujian IDT sederhana
- Buat direktori rangkaian tes
- Buat file konfigurasi

- Dapatkan SDK klien IDT
- Buat executable uji kasus
- Konfigurasi informasi perangkat untuk IDT
- Jalankan rangkaian tes
- Memecahkan masalah kesalahan
- Buat file konfigurasi rangkaian tes IDT
- Konfigurasikan orkestrator uji IDT
- Konfigurasikan state machine IDT
- Buat kasus uji IDT yang dapat dieksekusi
- Gunakan konteks IDT
- Mengonfigurasi pengaturan untuk test runner
- Debug dan jalankan rangkaian tes kustom
- Tinjau hasil tes dan log IDT
- Kirim metrik penggunaan IDT

Siapkan prasyarat untuk rangkaian pengujian IDT sederhana

Untuk menyelesaikan tutorial ini, Anda memerlukan hal berikut ini:

- Persyaratan komputer host
  - Versi terbaru dari AWS IoT Device Tester
  - Python 3.7 atau yang lebih baru

    Untuk memeriksa versi Python yang diinstal pada komputer Anda, jalankan perintah berikut:

    ```
    python3 --version
    ```

    Pada Windows, jika penggunaan perintah ini menghasilkan kesalahan, gunakan `python --version` sebagai gantinya. Jika nomor versi yang dikembalikan adalah 3,7 atau lebih besar, jalankan perintah berikut di terminal Powershell untuk mengatur `python3` sebagai alias untuk perintah `python`.

    ```
    Set-Alias -Name "python3" -Value "python"
    ```

Jika tidak ada informasi versi yang dikembalikan atau jika nomor versi kurang dari 3,7, ikuti petunjuk di Mengunduh Py untuk menginstal Python 3.7+. Untuk informasi selengkapnya, lihat dokumentasi Python.

- urllib3

  Untuk memverifikasi bahwa `urllib3` diinstal dengan benar, jalankan perintah berikut:

  ```
  python3 -c 'import urllib3'
  ```

  Jika `urllib3` belum terinstal, gunakan perintah berikut untuk menginstalnya:

  ```
  python3 -m pip install urllib3
  ```

- Persyaratan perangkat
  - Perangkat dengan sistem operasi Linux dan koneksi jaringan ke jaringan yang sama dengan komputer host Anda.

    Kami menyarankan agar Anda menggunakan Raspberry Pi dengan OS Raspberry Pi. Pastikan Anda mengatur SSH pada Raspberry Pi Anda untuk terhubung secara jarak jauh ke sana.

Buat direktori rangkaian tes

IDT secara logis memisahkan uji kasus ke dalam grup uji dalam setiap rangkaian tes. Setiap uji kasus harus berada di dalam grup uji. Untuk tutorial ini, buat folder bernama `MyTestSuite_1.0.0` dan buat pohon direktori berikut dalam folder ini:

```
MyTestSuite_1.0.0
### suite
    ### myTestGroup
        ### myTestCase
```

Buat file konfigurasi

Test suite Anda harus berisi file konfigurasi wajib berikut:

File yang diperlukan

`suite.json`

Berisi informasi tentang rangkaian pengujian. Lihat Konfigurasikan suite.json.

`group.json`

Berisi informasi tentang grup uji. Anda harus membuat file `group.json` untuk setiap grup uji di rangkaian tes Anda. Lihat Konfigurasikan group.json.

`test.json`

Berisi informasi tentang grup uji. Anda harus membuat file `test.json` untuk setiap grup uji di rangkaian tes Anda. Lihat Konfigurasikan test.json.

1. Di folder `MyTestSuite_1.0.0/suite`, buat file `suite.json` dengan struktur berikut:

```
{
    "id": "MyTestSuite_1.0.0",
    "title": "My Test Suite",
    "details": "This is my test suite.",
    "userDataRequired": false
}
```

2. Di folder `MyTestSuite_1.0.0/myTestGroup`, buat file `group.json` dengan struktur berikut:

```
{
    "id": "MyTestGroup",
    "title": "My Test Group",
    "details": "This is my test group.",
    "optional": false
}
```

3. Di folder `MyTestSuite_1.0.0/myTestGroup/myTestCase`, buat file `test.json` dengan struktur berikut:

```
{
    "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"
            ]
        }
    }
}
```

Pohon direktori untuk folder `MyTestSuite_1.0.0` Anda sekarang akan terlihat seperti berikut ini:

```
MyTestSuite_1.0.0
### suite
    ### suite.json
    ### myTestGroup
        ### group.json
        ### myTestCase
            ### test.json
```

Dapatkan SDK klien IDT

Anda menggunakan SDK Klien IDT untuk memungkinkan IDT berinteraksi dengan perangkat yang sedang diuji dan melaporkan hasil pengujian. Untuk tutorial ini, Anda akan menggunakan versi Python dari SDK.

Dari folder *<device-tester-extract-location>*`/sdks/python/`, salin folder `idt_client` ke folder `MyTestSuite_1.0.0/suite/myTestGroup/myTestCase` Anda.

Untuk memverifikasi bahwa SDK berhasil disalin, jalankan perintah berikut.

```
cd MyTestSuite_1.0.0/suite/myTestGroup/myTestCase
```

```
python3 -c 'import idt_client'
```

Buat executable uji kasus

Executable uji kasus berisi logika tes yang ingin Anda jalankan. Sebuah rangkaian tes dapat berisi beberapa executable uji kasus. Untuk tutorial ini, Anda hanya akan membuat satu executable uji kasus.

1. Buat file rangkaian test.

   Di folder `MyTestSuite_1.0.0/suite/myTestGroup/myTestCase`, buat file `myTestCase.py` dengan konten berikut:

   ```
   from idt_client import *

   def main():
       # Use the client SDK to communicate with IDT
       client = Client()

   if __name__ == "__main__":
       main()
   ```

2. Gunakan fungsi SDK klien untuk menambahkan logika uji berikut ke file `myTestCase.py` Anda:

   a. Jalankan perintah SSH pada perangkat yang diuji.

      ```
      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. Kirim hasil tes ke 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()
```

Konfigurasi informasi perangkat untuk IDT

Konfigurasi informasi perangkat Anda untuk IDT untuk menjalankan tes. Anda harus memperbarui templat `device.json` yang terletak di folder *<device-tester-extract-location>*`/configs` dengan informasi berikut.

```
[
  {
    "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>"
            }
          }
        }
      }
    ]
  }
]
```

Di objek `devices`, berikan informasi berikut:

`id`

Pengenal unik yang ditetapkan pengguna untuk perangkat Anda.

`connectivity.ip`

Alamat IP perangkat Anda.

`connectivity.port`

Opsional. Nomor port yang digunakan untuk koneksi SSH ke perangkat Anda.

`connectivity.auth`

Informasi autentikasi untuk koneksi tersebut.

Properti ini hanya berlaku jika `connectivity.protocol` diatur ke `ssh`.

`connectivity.auth.method`

Metode autentikasi yang digunakan untuk mengakses perangkat melalui protokol konektivitas yang diberikan.

Nilai yang didukung adalah:

- `pki`
- `password`

`connectivity.auth.credentials`

Kredensial yang digunakan untuk autentikasi.

`connectivity.auth.credentials.user`

Nama pengguna yang digunakan untuk masuk ke perangkat Anda.

`connectivity.auth.credentials.privKeyPath`

Jalur lengkap ke kunci pribadi yang digunakan untuk masuk ke perangkat Anda.

Nilai ini hanya berlaku jika `connectivity.auth.method` diatur ke `pki`.

`devices.connectivity.auth.credentials.password`

Kata sandi yang digunakan untuk masuk ke perangkat Anda.

Nilai ini hanya berlaku jika `connectivity.auth.method` diatur ke `password`.

> **Note**
>
> Tentukan `privKeyPath` hanya jika `method` diatur ke `pki`.
> Tentukan `password` hanya jika `method` diatur ke `password`.

Jalankan rangkaian tes

Setelah Anda membuat rangkaian tes Anda, Anda ingin memastikan bahwa rangkaian tes itu berfungsi seperti yang diharapkan. Selesaikan langkah-langkah berikut untuk menjalankan rangkaian pengujian dengan kolam perangkat yang sudah ada untuk melakukannya.

1. Salin folder `MyTestSuite_1.0.0` Anda ke dalam *`<device-tester-extract-location>`*`/tests`.
2. Jalankan perintah berikut:

```
cd <device-tester-extract-location>/bin
./devicetester_[linux | mac | win_x86-64] run-suite --suite-id MyTestSuite
```

IDT menjalankan rangkaian tes Anda dan mengalirkan hasilnya ke konsol. Ketika tes telah selesai berjalan, Anda akan melihat informasi berikut:

```
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
```

Memecahkan masalah kesalahan

Gunakan informasi berikut untuk membantu menyelesaikan masalah dengan menyelesaikan tutorial.

Kasus uji tidak berjalan dengan sukses

Jika tes tidak berjalan sukses, IDT akan mengalirkan log kesalahan ke konsol yang dapat membantu Anda memecahkan masalah uji coba. Sebelum Anda memeriksa log kesalahan, verifikasi hal berikut:

- SDK klien IDT berada di folder yang benar seperti yang dijelaskan dalam. Dapatkan SDK klien IDT
- Pastikan Anda memenuhi semua prasyarat untuk tutorial ini. Untuk informasi selengkapnya, lihat Siapkan prasyarat untuk rangkaian pengujian IDT sederhana.

Tidak dapat terhubung ke perangkat yang sedang diuji

Verifikasi hal berikut:

- File `device.json` Anda berisi alamat IP, port, dan informasi autentikasi yang benar.
- Anda dapat terhubung ke perangkat Anda melalui SSH dari komputer host Anda.

Buat file konfigurasi rangkaian tes IDT

Bagian ini menjelaskan format di mana Anda membuat file konfigurasi yang Anda sertakan saat Anda menulis rangkaian pengujian kustom.

File konfigurasi yang diperlukan

`suite.json`

Berisi informasi tentang rangkaian pengujian. Lihat Konfigurasikan suite.json.

`group.json`

Berisi informasi tentang grup uji. Anda harus membuat file `group.json` untuk setiap grup uji di rangkaian tes Anda. Lihat Konfigurasikan group.json.

`test.json`

Berisi informasi tentang grup uji. Anda harus membuat file `test.json` untuk setiap grup uji di rangkaian tes Anda. Lihat Konfigurasikan test.json.

File konfigurasi opsional

`test_orchestrator.yaml` atau **`state_machine.json`**

Mendefinisikan bagaimana pengujian dijalankan saat IDT menjalankan rangkaian pengujian. SSe Konfigurasikan test_orchestrator.yaml.

> **Note**
>
> Mulai IDT v4.5.2, Anda menggunakan `test_orchestrator.yaml` file untuk menentukan alur kerja pengujian. Di versi IDT sebelumnya, Anda menggunakan `state_machine.json` file tersebut. Untuk informasi tentang mesin negara, lihatKonfigurasikan state machine IDT.

`userdata_schema.json`

Menentukan skema untuk file `userdata.json` yang dapat disertakan oleh test runner dalam konfigurasi pengaturannya. File `userdata.json` digunakan untuk konfigurasi informasi tambahan apa pun yang diperlukan untuk menjalankan tes tetapi tidak terdapat dalam file `device.json`. Lihat Konfigurasikan userdata_schema.json.

File konfigurasi ditempatkan di Anda *<custom-test-suite-folder>* seperti yang ditunjukkan di sini.

```
<custom-test-suite-folder>
### suite
    ### suite.json
    ### test_orchestrator.yaml
    ### userdata_schema.json
    ### <test-group-folder>
        ### group.json
        ### <test-case-folder>
            ### test.json
```

Konfigurasikan suite.json

File `suite.json` menetapkan variabel lingkungan dan menentukan apakah data pengguna diperlukan untuk menjalankan rangkaian tes. Gunakan templat berikut untuk mengonfigurasi file *<custom-test-suite-folder>*`/suite/suite.json` Anda:

```
{
    "id": "<suite-name>_<suite-version>",
    "title": "<suite-title>",
    "details": "<suite-details>",
    "userDataRequired": true | false,
    "environmentVariables": [
        {
            "key": "<name>",
            "value": "<value>",
        },
        ...
        {
            "key": "<name>",
            "value": "<value>",
        }
```

```
    ]
}
```

Semua kolom yang berisi nilai wajib diisi seperti yang dijelaskan di sini:

`id`

ID unik yang ditetapkan pengguna untuk rangkaian uji. Nilai dari `id` harus cocok dengan nama folder rangkaian uji tempat file `suite.json` berada. Nama rangkaian dan versi rangkaian juga harus memenuhi persyaratan berikut:

- *`<suite-name>`* tidak dapat berisi garis bawah.
- *`<suite-version>`* dilambangkan sebagai *`x.x.x`*, di mana `x` adalah angka.

ID ditampilkan dalam laporan uji yang dihasilkan IDT.

`title`

Nama yang ditetapkan pengguna untuk produk atau fitur yang diuji oleh rangkaian tes ini. Nama ditampilkan dalam IDT CLI untuk test runner.

`details`

Deskripsi singkat tentang tujuan dari rangkaian tes.

`userDataRequired`

Menentukan apakah test runner perlu menyertakan informasi kustom dalam file `userdata.json`. Jika Anda menetapkan nilai ini ke `true`, Anda juga harus menyertakan file `userdata_schema.json` dalam folder rangkaian uji Anda.

`environmentVariables`

Opsional. Serangkaian variabel lingkungan yang akan ditetapkan untuk rangkaian tes ini.

`environmentVariables.key`

Nama variabel lingkungan.

`environmentVariables.value`

Nilai variabel lingkungan.

Konfigurasikan group.json

File `group.json` menentukan apakah grup uji itu wajib atau opsional. Gunakan templat berikut untuk mengonfigurasi file *`<custom-test-suite-folder>`*`/suite/`*`<test-group>`*`/group.json` Anda:

```
{
    "id": "<group-id>",
    "title": "<group-title>",
    "details": "<group-details>",
    "optional": true | false,
}
```

Semua kolom yang berisi nilai wajib diisi seperti yang dijelaskan di sini:

`id`

ID unik yang ditetapkan pengguna untuk grup uji. Nilai `id` harus sesuai dengan nama folder grup uji tempat `group.json` file berada dan tidak boleh memiliki garis bawah (_). ID digunakan dalam laporan uji yang dihasilkan IDT.

`title`

Nama deskriptif untuk grup uji. Nama tersebut ditampilkan dalam IDT CLI untuk test runner.

`details`

Deskripsi singkat tentang tujuan dari grup tes.

`optional`

Opsional. Atur ke `true` untuk menampilkan grup tes ini sebagai grup opsional setelah IDT selesai menjalankan tes yang diperlukan. Nilai defaultnya adalah `false`.

Konfigurasikan test.json

File `test.json` menentukan executable uji kasus dan variabel lingkungan yang digunakan oleh uji kasus. Untuk informasi selengkapnya tentang cara membuat executable uji kasus, lihat Buat kasus uji IDT yang dapat dieksekusi.

Gunakan templat berikut untuk mengonfigurasi file *`<custom-test-suite-folder>`*`/suite/`*`<test-group>`*`/`*`<test-case>`*`/test.json` Anda:

```
{
    "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>",
        }
```

```
    ]
}
```

Semua kolom yang berisi nilai wajib diisi seperti yang dijelaskan di sini:

`id`

ID unik yang ditetapkan pengguna untuk grup uji. Nilai `id` harus sesuai dengan nama folder kasus uji tempat `test.json` file berada dan tidak boleh memiliki garis bawah (_). ID digunakan dalam laporan uji yang dihasilkan IDT.

`title`

Nama deskriptif untuk uji kasus. Nama tersebut ditampilkan dalam IDT CLI untuk test runner.

`details`

Deskripsi singkat tentang tujuan dari uji kasus.

`requireDUT`

Opsional. Atur ke `true` jika perangkat diperlukan untuk menjalankan tes ini, jika tidak atur ke `false`. Nilai defaultnya adalah `true`. Test runner akan mengonfigurasi perangkat yang akan digunakannya untuk menjalankan pengujian di file `device.json`.

`requiredResources`

Opsional. Serangkaian hal yang menyediakan informasi tentang perangkat sumber daya yang diperlukan untuk menjalankan tes ini.

`requiredResources.name`

Nama unik yang akan diberikan kepada sumber daya perangkat ketika tes ini berjalan.

`requiredResources.features`

Serangkaian fitur perangkat sumber daya yang ditetapkan pengguna.

`requiredResources.features.name`

Nama fitur. Fitur perangkat yang ingin Anda gunakan untuk perangkat ini. Nama ini dicocokkan dengan nama fitur yang disediakan oleh test runner di file `resource.json`.

`requiredResources.features.version`

Opsional. Versi fitur. Nama ini dicocokkan dengan versi fitur yang disediakan oleh test runner di file `resource.json`. Jika versi tidak tersedia, maka fitur tersebut tidak dicentang. Jika nomor versi tidak diperlukan untuk fitur tersebut, biarkan kolom ini kosong.

`requiredResources.features.jobSlots`

Opsional. Jumlah tes simultan yang dapat didukung fitur ini. Nilai default-nya adalah 1. Jika Anda ingin IDT menggunakan perangkat yang berbeda untuk masing-masing fitur, kami sarankan Anda menetapkan nilai ini ke 1.

`execution.timeout`

Jumlah waktu (dalam milidetik) yang ditunggu oleh IDT hingga tes tersebut selesai dijalankan. Untuk informasi selengkapnya tentang pengaturan parameter ini, lihat Buat kasus uji IDT yang dapat dieksekusi.

`execution.os`

Executable uji kasus yang akan dijalankan berdasarkan sistem operasi komputer host yang menjalankan IDT. Nilai yang didukung adalah `linux`, `mac`, dan `win`.

`execution.os.cmd`

Jalur ke executable uji kasus yang ingin Anda jalankan untuk sistem operasi tertentu. Lokasi ini harus berada di jalur sistem.

`execution.os.args`

Opsional. Argumen yang akan disediakan untuk menjalankan executable uji kasus.

`environmentVariables`

Opsional. Serangkaian variabel lingkungan yang akan ditetapkan untuk uji kasus ini.

`environmentVariables.key`

Nama variabel lingkungan.

`environmentVariables.value`

Nilai variabel lingkungan.

> **Note**
>
> Jika Anda menentukan variabel lingkungan yang sama di file `test.json` dan di file `suite.json`, nilai dalam file `test.json` akan diutamakan.

Konfigurasikan test_orchestrator.yaml

Orkestrator pengujian adalah konstruksi yang mengontrol alur eksekusi test suite. Ia menentukan keadaan awal dari rangkaian tes, mengelola transisi keadaan berdasarkan aturan yang ditetapkan pengguna, dan terus melakukan transisi melalui keadaan-keadaan tersebut sampai mencapai keadaan akhir.

Jika rangkaian pengujian Anda tidak menyertakan orkestrator pengujian yang ditentukan pengguna, IDT akan menghasilkan orkestrator pengujian untuk Anda.

Orkestrator uji default melakukan fungsi-fungsi berikut:

- Menyediakan test runner dengan kemampuan untuk memilih dan menjalankan grup uji tertentu, dan bukan seluruh rangkaian uji.
- Jika grup uji tertentu tidak dipilih, ia menjalankan setiap grup uji di rangkaian uji dengan urutan acak.
- Membuat laporan dan mencetak ringkasan konsol yang menunjukkan hasil tes untuk setiap grup uji dan uji kasus.

Untuk informasi selengkapnya tentang bagaimana fungsi orkestrator pengujian IDT, lihat. Konfigurasikan orkestrator uji IDT

Konfigurasikan userdata_schema.json

File `userdata_schema.json` menentukan skema di mana test runner menyediakan data pengguna. Data pengguna diperlukan jika rangkaian uji Anda memerlukan informasi yang tidak ada di file `device.json`. Misalnya, pengujian Anda mungkin memerlukan kredensial jaringan Wi-Fi, port terbuka tertentu, atau sertifikat yang harus diberikan pengguna. Informasi ini dapat diberikan kepada IDT sebagai parameter input yang disebut `userdata`, nilai yang merupakan file `userdata.json`, yang dibuat oleh para pengguna dalam *`<device-tester-extract-location>`*`/config` mereka. Format file `userdata.json` didasarkan pada `userdata_schema.json` yang Anda sertakan dalam rangkaian tes.

Untuk menunjukkan hal tersebut test runner harus menyediakan file `userdata.json`:

1. Di file `suite.json`, atur `userDataRequired` ke `true`.
2. Di *`<custom-test-suite-folder>`* Anda, buat file `userdata_schema.json`.
3. Edit file `userdata_schema.json` untuk membuat Skema IETF Draft v4 JSON yang valid.

Ketika IDT menjalankan rangkaian uji Anda, secara otomatis ia membaca skema dan menggunakannya untuk memvalidasi file `userdata.json` yang disediakan oleh test runner. Jika valid, isi `userdata.json` file tersedia dalam konteks IDT dan dalam konteks orkestrator pengujian.

Konfigurasikan orkestrator uji IDT

Mulai IDT v4.5.2, IDT menyertakan komponen orkestrator pengujian baru. Orkestrator pengujian adalah komponen IDT yang mengontrol alur eksekusi rangkaian pengujian, dan menghasilkan laporan pengujian setelah IDT selesai menjalankan semua pengujian. Orkestrator pengujian menentukan pemilihan pengujian dan urutan pengujian dijalankan berdasarkan aturan yang ditentukan pengguna.

Jika rangkaian pengujian Anda tidak menyertakan orkestrator pengujian yang ditentukan pengguna, IDT akan menghasilkan orkestrator pengujian untuk Anda.

Orkestrator uji default melakukan fungsi-fungsi berikut:

- Menyediakan test runner dengan kemampuan untuk memilih dan menjalankan grup uji tertentu, dan bukan seluruh rangkaian uji.
- Jika grup uji tertentu tidak dipilih, ia menjalankan setiap grup uji di rangkaian uji dengan urutan acak.
- Membuat laporan dan mencetak ringkasan konsol yang menunjukkan hasil tes untuk setiap grup uji dan uji kasus.

Orkestrator uji menggantikan mesin status IDT. Kami sangat menyarankan Anda menggunakan orkestrator pengujian untuk mengembangkan rangkaian pengujian Anda alih-alih mesin status IDT. Orkestrator uji menyediakan fitur-fitur yang disempurnakan berikut:

- Menggunakan format deklaratif dibandingkan dengan format imperatif yang digunakan mesin status IDT. Ini memungkinkan Anda menentukan tes mana yang ingin Anda jalankan dan kapan Anda ingin menjalankannya.
- Mengelola penanganan grup tertentu, pembuatan laporan, penanganan kesalahan, dan pelacakan hasil sehingga Anda tidak diharuskan mengelola tindakan ini secara manual.
- Menggunakan format YAMAL, yang mendukung komentar secara default.
- Membutuhkan 80 persen lebih sedikit ruang disk daripada orkestrator pengujian untuk menentukan alur kerja yang sama.

- Menambahkan validasi pra-pengujian untuk memverifikasi bahwa definisi alur kerja Anda tidak mengandung pengujian yang salah IDs atau dependensi melingkar.

Format orkestrator uji

Anda dapat menggunakan templat berikut untuk mengonfigurasi file *custom-test-suite-folder*/suite/test_orchestrator.yaml Anda:

```
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
```

Semua kolom yang berisi nilai wajib diisi seperti yang dijelaskan di sini:

`Aliases`

Opsional. String yang ditentukan pengguna yang memetakan ke ekspresi konteks. Alias memungkinkan Anda menghasilkan nama ramah untuk mengidentifikasi ekspresi konteks dalam konfigurasi orkestrator pengujian Anda. Ini sangat berguna jika Anda membuat ekspresi konteks kompleks atau ekspresi yang Anda gunakan di banyak tempat.

Anda dapat menggunakan ekspresi konteks untuk menyimpan kueri konteks yang memungkinkan Anda mengakses data dari konfigurasi IDT lainnya. Untuk informasi selengkapnya, lihat Akses data dalam konteks.

Example

Contoh

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

`ConditionalTests`

Opsional. Daftar kondisi, dan kasus uji terkait yang dijalankan ketika setiap kondisi terpenuhi. Setiap kondisi dapat memiliki beberapa kasus uji; Namun, Anda dapat menetapkan kasus uji yang diberikan hanya untuk satu kondisi.

Secara default, IDT menjalankan kasus uji apa pun yang tidak ditetapkan ke kondisi dalam daftar ini. Jika Anda tidak menentukan bagian ini, IDT menjalankan semua grup pengujian dalam rangkaian pengujian.

Setiap item dalam `ConditionalTests` daftar mencakup parameter berikut:

`Condition`

Ekspresi konteks yang mengevaluasi nilai Boolean. Jika nilai yang dievaluasi benar, IDT menjalankan kasus uji yang ditentukan dalam parameter. `Tests`

`Tests`

Daftar deskriptor tes.

Setiap deskriptor tes menggunakan ID kelompok uji dan satu atau lebih kasus uji IDs untuk mengidentifikasi pengujian individu untuk dijalankan dari kelompok uji tertentu. Deskriptor tes menggunakan format berikut:

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

Example

Contoh

Contoh berikut menggunakan ekspresi konteks generik yang dapat Anda definisikan sebagai`Aliases`.

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

Berdasarkan kondisi yang ditentukan, IDT memilih kelompok uji sebagai berikut:

- Jika `Condition1` benar, IDT menjalankan tes dalam kelompok uji A, B, dan C.
- Jika `Condition2` benar, IDT menjalankan tes dalam kelompok uji C dan D.

`Order`

Opsional. Urutan untuk menjalankan tes. Anda menentukan urutan pengujian di tingkat kelompok uji. Jika Anda tidak menentukan bagian ini, IDT menjalankan semua grup pengujian yang berlaku dalam urutan acak. Nilai `Order` adalah daftar daftar deskriptor grup. Setiap grup pengujian yang tidak Anda cantumkan`Order`, dapat dijalankan secara paralel dengan grup pengujian terdaftar lainnya.

Setiap daftar deskriptor grup berisi salah satu deskriptor grup lainnya, dan mengidentifikasi urutan untuk menjalankan grup yang ditentukan dalam setiap deskriptor. Anda dapat menggunakan format berikut untuk menentukan deskriptor grup individual:

- *`group-id`*—ID grup dari grup uji yang ada.
- [*`group-id`*, *`group-id`*]—Daftar kelompok uji yang dapat dijalankan dalam urutan apa pun relatif satu sama lain.
- `"*"`—Wildcard. Ini setara dengan daftar semua kelompok uji yang belum ditentukan dalam daftar deskriptor grup saat ini.

Nilai untuk juga `Order` harus memenuhi persyaratan berikut:

- Grup pengujian IDs yang Anda tentukan dalam deskriptor grup harus ada di rangkaian pengujian Anda.
- Setiap daftar deskriptor kelompok harus menyertakan setidaknya satu kelompok uji.
- Setiap daftar deskriptor grup harus berisi grup IDs unik. Anda tidak dapat mengulang ID grup uji dalam deskriptor grup individual.
- Daftar deskriptor grup dapat memiliki paling banyak satu deskriptor grup wildcard. Deskriptor grup wildcard harus menjadi item pertama atau terakhir dalam daftar.

Example

Contoh

Untuk rangkaian pengujian yang berisi grup pengujian A, B, C, D, dan E, daftar contoh berikut menunjukkan cara berbeda untuk menentukan bahwa IDT pertama-tama harus menjalankan grup uji A, kemudian menjalankan grup uji B, dan kemudian menjalankan grup uji C, D, dan E dalam urutan apa pun.

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

      - - B
        - C

      - - B
        - D

      - - B
        - E
  ```
  ```

`Features`

Opsional. Daftar fitur produk yang Anda ingin IDT tambahkan ke `awsiotdevicetester_report.xml` file. Jika Anda tidak menentukan bagian ini, IDT tidak akan menambahkan fitur produk apa pun ke laporan.

Fitur produk adalah informasi yang ditetapkan pengguna tentang kriteria spesifik yang mungkin dipenuhi oleh perangkat. Misalnya, fitur produk MQTT dapat menunjukkan bahwa perangkat menerbitkan pesan MQTT dengan benar. Dalam`awsiotdevicetester_report.xml`, fitur produk ditetapkan sebagai`supported`,`not-supported`, atau nilai yang ditentukan pengguna khusus, berdasarkan apakah tes tertentu lulus.

Setiap item dalam `Features` daftar terdiri dari parameter berikut:

`Name`

Nama fitur.

`Value`

Opsional. Nilai kustom yang ingin Anda gunakan dalam laporan, bukan`supported`. Jika nilai ini tidak ditentukan, maka IDT berbasis menetapkan nilai fitur ke `supported` atau `not-supported` berdasarkan hasil pengujian. Jika Anda menguji fitur yang sama dengan kondisi yang berbeda, Anda dapat menggunakan nilai kustom untuk setiap instance fitur tersebut dalam `Features` daftar, dan IDT menggabungkan nilai fitur untuk kondisi yang didukung. Untuk informasi selengkapnya, silakan lihat

`Condition`

Ekspresi konteks yang mengevaluasi nilai Boolean. Jika nilai yang dievaluasi benar, IDT menambahkan fitur ke laporan pengujian setelah selesai menjalankan rangkaian pengujian. Jika nilai yang dievaluasi salah, tes tidak termasuk dalam laporan.

`Tests`

Opsional. Daftar deskriptor tes. Semua tes yang ditentukan dalam daftar ini harus lulus agar fitur didukung.

Setiap deskriptor pengujian dalam daftar ini menggunakan ID kelompok uji dan satu atau lebih kasus uji IDs untuk mengidentifikasi pengujian individual yang akan dijalankan dari kelompok uji tertentu. Deskriptor tes menggunakan format berikut:

```
GroupId: group-id
```

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

Anda harus menentukan salah satu `Tests` atau `OneOfTests` untuk setiap fitur dalam `Features` daftar.

`OneOfTests`

Opsional. Daftar deskriptor tes. Setidaknya salah satu tes yang ditentukan dalam daftar ini harus lulus agar fitur didukung.

Setiap deskriptor pengujian dalam daftar ini menggunakan ID kelompok uji dan satu atau lebih kasus uji IDs untuk mengidentifikasi pengujian individual yang akan dijalankan dari kelompok uji tertentu. Deskriptor tes menggunakan format berikut:

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

Anda harus menentukan salah satu `Tests` atau `OneOfTests` untuk setiap fitur dalam `Features` daftar.

`IsRequired`

Nilai boolean yang menentukan apakah fitur tersebut diperlukan dalam laporan pengujian. Nilai default-nya adalah `false`.

Konteks orkestrator uji

Konteks orkestrator pengujian adalah dokumen JSON read-only yang berisi data yang tersedia untuk orkestrator pengujian selama eksekusi. Konteks orkestrator pengujian hanya dapat diakses dari orkestrator pengujian, dan berisi informasi yang menentukan alur pengujian. Misalnya, Anda dapat menggunakan informasi yang dikonfigurasi oleh test runner di file `userdata.json` untuk menentukan apakah pengujian tertentu wajib dijalankan.

Konteks orkestrator pengujian menggunakan format berikut:

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

```
    },
    "config": {
        <config-json-content>
    }
}
```

`pool`

Informasi tentang kolam perangkat yang dipilih untuk uji coba. Untuk kolam perangkat yang dipilih, informasi ini diambil dari elemen rangkaian perangkat tingkat atas yang sesuai yang ditentukan dalam file `device.json`.

`userData`

Informasi di file `userdata.json`.

`config`

Informasi di file `config.json`.

Anda dapat menanyakan konteks menggunakan JSONPath notasi. Sintaks untuk JSONPath kueri dalam definisi negara adalah. `{{query}}` Saat Anda mengakses data dari konteks orkestrator pengujian, pastikan setiap nilai mengevaluasi string, angka, atau Boolean.

Untuk informasi selengkapnya tentang penggunaan JSONPath notasi untuk mengakses data dari konteks, lihatGunakan konteks IDT.

Konfigurasikan state machine IDT

> **Important**
>
> Mulai IDT v4.5.2, mesin status ini tidak digunakan lagi. Kami sangat menyarankan Anda menggunakan orkestrator uji baru. Untuk informasi selengkapnya, lihat Konfigurasikan orkestrator uji IDT.

State machine adalah suatu konstruksi yang mengendalikan aliran eksekusi rangkaian uji. Ia menentukan keadaan awal dari rangkaian tes, mengelola transisi keadaan berdasarkan aturan yang ditetapkan pengguna, dan terus melakukan transisi melalui keadaan-keadaan tersebut sampai mencapai keadaan akhir.

Jika rangkaian tes Anda tidak menyertakan state machine yang ditetapkan pengguna, IDT akan membuat state machine untuk Anda. State machine default melakukan fungsi-fungsi berikut:

- Menyediakan test runner dengan kemampuan untuk memilih dan menjalankan grup uji tertentu, dan bukan seluruh rangkaian uji.
- Jika grup uji tertentu tidak dipilih, ia menjalankan setiap grup uji di rangkaian uji dengan urutan acak.
- Membuat laporan dan mencetak ringkasan konsol yang menunjukkan hasil tes untuk setiap grup uji dan uji kasus.

State machine untuk rangkaian uji IDT harus memenuhi kriteria berikut:

- Setiap keadaan sesuai dengan tindakan yang harus ddilakukan oleh IDT, seperti menjalankan grup uji atau produk file laporan.
- Transisi ke suatu keadaan akan menghasilkan tindakan yang terkait dengan keadaan tersebut.
- Setiap keadaan menentukan aturan transisi untuk keadaan berikutnya.
- Keadaan akhir harus berupa `Succeed` atau `Fail`.

Format state machine

Anda dapat menggunakan templat berikut untuk mengonfigurasi file *<custom-test-suite-folder>*`/suite/state_machine.json` Anda:

```
{
  "Comment": "<description>",
  "StartAt": "<state-name>",
  "States": {
    "<state-name>": {
      "Type": "<state-type>",
      // Additional state configuration
    }

    // Required states
    "Succeed": {
      "Type": "Succeed"
    },
    "Fail": {
      "Type": "Fail"
    }
```

```
  }
}
```

Semua kolom yang berisi nilai wajib diisi seperti yang dijelaskan di sini:

`Comment`

Sebuah deskripsi tentang state machine.

`StartAt`

Nama keadaan di mana IDT mulai menjalankan rangkaian tes. Nilai dari `StartAt` harus diatur ke salah satu keadaan yang tercantum di objek `States`.

`States`

Sebuah objek yang memetakan nama keadaan yang ditetapkan pengguna ke keadaan IDT yang valid. Setiap Negara. *`state-name`*objek berisi definisi status valid yang dipetakan ke. *`state-name`*

Objek `States` harus mencakup keadaan `Succeed` dan `Fail`. Untuk informasi lebih lanjut tentang keadaan yang valid, lihat Keadaan yang valid dan definisi keadaan.

Keadaan yang valid dan definisi keadaan

Bagian ini menjelaskan definisi keadaan pada semua keadaan yang valid yang dapat digunakan dalam state machine IDT. Beberapa keadaan berikut mendukung konfigurasi pada tingkat uji kasus. Namun, kami sarankan Anda untuk mengonfigurasi aturan transisi keadaan pada tingkat grup uji dan bukan pada tingkat uji kasus kecuali jika benar-benar diperlukan.

Definisi keadaan

- RunTask
- Pilihan
- Paralel
- AddProductFeatures
- Laporan
- LogMessage
- SelectGroup

- Gagal
- Berhasil

RunTask

Keadaan `RunTask` menjalankan uji kasus dari grup uji yang ditentukan dalam rangkaian tes.

```
{
    "Type": "RunTask",
    "Next": "<state-name>",
    "TestGroup": "<group-id>",
    "TestCases": [
        "<test-id>"
    ],
    "ResultVar": "<result-name>"
}
```

Semua kolom yang berisi nilai wajib diisi seperti yang dijelaskan di sini:

`Next`

Nama keadaan yang akan ditransisi setelah melaksanakan tindakan dalam keadaan saat ini.

`TestGroup`

Opsional. ID grup uji yang akan dijalankan. Jika nilai ini tidak ditentukan, IDT akan menjalankan grup uji yang dipilih oleh test runner.

`TestCases`

Opsional. Array kasus uji IDs dari grup yang ditentukan dalam`TestGroup`. Berdasarkan nilai-nilai `TestGroup` dan `TestCases`, IDT menentukan perilaku eksekusi tes sebagai berikut:

- Ketika `TestGroup` dan `TestCases` ditentukan, IDT akan menjalankan uji kasus tertentu dari grup uji.
- Ketika `TestCases` ditentukan tetapi `TestGroup` tidak ditentukan, IDT akan menjalankan uji kasus yang ditentukan.
- Ketika `TestGroup` ditentukan tetapi `TestCases` tidak ditentukan, IDT akan menjalankan semua uji kasus di grup uji yang ditentukan.
- Ketika `TestGroup` ataupun `TestCases` tidak ditentukan, IDT akan menjalankan semua uji kasus dari grup uji yang dipilih oleh test runner dari IDT CLI. Untuk mengaktifkan pilihan

grup untuk test runner, Anda harus menyertakan keadaan `RunTask` dan `Choice` dalam file `statemachine.json` Anda. Untuk contoh cara kerjanya, lihat Contoh state machine: Jalankan grup uji yang dipilih pengguna.

Untuk informasi selengkapnya tentang mengaktifkan perintah IDT CLI untuk test runner, lihat the section called “Aktifkan perintah IDT CLI”.

`ResultVar`

Nama variabel konteks yang akan diatur dengan hasil uji yang dijalankan. Jangan tentukan nilai ini jika Anda tidak menentukan nilai untuk `TestGroup`. IDT menetapkan nilai variabel yang Anda tentukan di `ResultVar` hingga `true` atau `false` berdasarkan berikut ini:

- Jika nama variabel adalah dari bentuk *`text`*`_`*`text`*`_passed`, maka nilainya akan diatur ke apakah semua tes dalam grup uji pertama akan dilalui atau dilompati.
- Dalam semua kasus lainnya, nilai akan diatur ke apakah semua tes di semua grup uji akan dilalui atau dilompati.

Biasanya, Anda akan menggunakan `RunTask` status untuk menentukan ID grup uji tanpa menentukan kasus uji individual IDs, sehingga IDT akan menjalankan semua kasus uji dalam kelompok uji yang ditentukan. Semua uji kasus yang dijalankan oleh keadaan ini berjalan secara paralel, dengan urutan acak. Namun, jika semua uji kasus memerlukan perangkat untuk dijalankan, dan hanya satu perangkat yang tersedia, maka uji kasus akan berjalan secara berurutan sebagai gantinya.

Penanganan kesalahan

Jika salah satu grup uji atau kasus uji IDs yang ditentukan tidak valid, maka status ini mengeluarkan kesalahan `RunTaskError` eksekusi. Jika keadaan ini menemukan kesalahan eksekusi, ia juga akan menetapkan variabel `hasExecutionError` dalam konteks state machine ke `true`.

Pilihan

Keadaan `Choice` memungkinkan Anda secara dinamis mengatur keadaan berikutnya yang akan ditransisikan berdasarkan keadaan yang ditetapkan pengguna.

```
{
    "Type": "Choice",
    "Default": "<state-name>",
    "FallthroughOnError": true | false,
```

```
    "Choices": [
        {
            "Expression": "<expression>",
            "Next": "<state-name>"
        }
    ]
}
```

Semua kolom yang berisi nilai wajib diisi seperti yang dijelaskan di sini:

`Default`

Keadaan default yang akan ditransisikan jika tidak terdapat ekspresi yang ditentukan di `Choices` dapat dievaluasi pada `true`.

`FallthroughOnError`

Opsional. Menentukan perilaku ketika keadaan tersebut bertemu kesalahan dalam mengevaluasi ekspresi. Atur ke `true` jika Anda ingin melompati ekspresi jika hasil evaluasi menghasilkan kesalahan. Jika tidak ada ekspresi yang cocok, state machine akan bertransisi ke keadaan `Default`. Jika nilai `FallthroughOnError` tidak ditentukan, default-nya adalah `false`.

`Choices`

Serangkaian ekspresi dan keadaan untuk menentukan keadaan mana yang akan ditransisikan setelah mengeksekusi tindakan dalam keadaan saat ini.

`Choices.Expression`

Ekspresi yang harus dievaluasi pada nilai boolean. Jika ekspresi mengevaluasi `true`, maka state machine akan bertransisi ke keadaan yang ditentukan dalam `Choices.Next`. String ekspresi mengambil nilai-nilai dari konteks state machine dan kemudian melakukan operasi padanya untuk sampai pada nilai boolean. Untuk informasi tentang mengakses konteks mesin status, lihatKonteks mesin keadaan.

`Choices.Next`

Nama keadaan yang akan ditransisikan jika ekspresi yang ditentukan dalam `Choices.Expression` dievaluasi pada `true`.

Penanganan kesalahan

Keadaan `Choice` dapat memerlukan penanganan kesalahan dalam kasus berikut:

- Beberapa variabel dalam ekspresi pilihan tidak ada dalam konteks state machine.
- Hasil ekspresi bukan merupakan nilai boolean.
- Hasil pencarian JSON bukanlah string, nomor, atau boolean.

Anda tidak dapat menggunakan blok `Catch` untuk menangani kesalahan dalam keadaan ini. Jika Anda ingin berhenti mengeksekusi state machine ketika menemukan kesalahan, Anda harus mengatur `FallthroughOnError` ke `false`. Namun, kami menyarankan agar Anda mengatur `FallthroughOnError` ke `true` Anda, dan tergantung pada kasus penggunaan Anda, lakukan salah satu langkah berikut:

- Jika variabel yang Anda akses diharapkan untuk tidak ada dalam beberapa kasus, gunakan nilai `Default` dan blok `Choices` tambahan untuk menentukan keadaan berikutnya.
- Jika variabel yang Anda akses harus selalu ada, atur keadaan `Default`ke `Fail`.

Paralel

Keadaan `Parallel` memungkinkan Anda untuk menentukan dan menjalankan state machine baru secara paralel satu sama lain.

```
{
    "Type": "Parallel",
    "Next": "<state-name>",
    "Branches": [
        <state-machine-definition>
    ]
}
```

Semua kolom yang berisi nilai wajib diisi seperti yang dijelaskan di sini:

`Next`

Nama keadaan yang akan ditransisi setelah melaksanakan tindakan dalam keadaan saat ini.

`Branches`

Serangkaian definisi state machine yang akan dijalankan. Setiap definisi state machine harus berisi keadaan `StartAt`, `Succeed`, dan `Fail` miliknya sendiri. Definisi state machine dalam rangkaian ini tidak dapat mengacu pada keadaan di luar definisinya sendiri.

> **Note**
>
> Karena setiap cabang state machine memiliki konteks state machine yang sama, pengaturan variabel dalam satu cabang dan kemudian pembacaan variabel-variabel dari cabang lain dapat mengakibatkan perilaku yang tidak terduga.

Keadaan `Parallel` bergerak ke keadaan berikutnya hanya setelah keadaan tersebut menjalankan semua cabang state machine. Setiap keadaan yang memerlukan perangkat akan menunggu untuk berjalan hingga perangkat tersebut tersedia. Jika beberapa perangkat tersedia, keadaan ini akan menjalankan uji kasus dari beberapa grup secara paralel. Jika tidak tersedia perangkat yang memadai, uji kasus akan berjalan secara berurutan. Karena uji kasus dijalankan dalam urutan acak ketika berjalan secara paralel, perangkat yang berbeda mungkin digunakan untuk menjalankan tes dari grup tes yang sama.

Penanganan kesalahan

Pastikan bahwa baik state machine cabang dan state machine induk bertransisi ke keadaan `Fail` untuk menangani kesalahan eksekusi.

Karena state machine cabang tidak mengirimkan kesalahan eksekusi ke state machine induk, Anda tidak dapat menggunakan blok `Catch` untuk menangani kesalahan eksekusi di state machine cabang. Sebagai gantinya, gunakan nilai `hasExecutionErrors` dalam konteks state machine bersama. Untuk contoh cara bekerjanya, lihat Contoh state machine: Jalankan dua grup uji secara paralel.

AddProductFeatures

Keadaan `AddProductFeatures` memungkinkan Anda menambahkan fitur produk ke file `awsiotdevicetester_report.xml` yang dihasilkan oleh IDT.

Fitur produk adalah informasi yang ditetapkan pengguna tentang kriteria spesifik yang mungkin dipenuhi oleh perangkat. Misalnya, fitur produk `MQTT` dapat menetapkan bahwa perangkat akan menerbitkan pesan MQTT dengan benar. Dalam laporan tersebut, fitur produk ditetapkan sebagai `supported`, `not-supported`, atau nilai kustom, berdasarkan apakah tes yang ditentukan berhasil dilalui.

> **Note**
>
> Keadaan `AddProductFeatures` tidak menghasilkan laporan dengan sendirinya. Keadaan ini harus bertransisi ke keadaan `Report` untuk menghasilkan laporan.

```
{
    "Type": "Parallel",
    "Next": "<state-name>",
    "Features": [
        {
            "Feature": "<feature-name>",
            "Groups": [
                "<group-id>"
            ],
            "OneOfGroups": [
                "<group-id>"
            ],
            "TestCases": [
                "<test-id>"
            ],
            "IsRequired": true | false,
            "ExecutionMethods": [
                "<execution-method>"
            ]
        }
    ]
}
```

Semua kolom yang berisi nilai wajib diisi seperti yang dijelaskan di sini:

`Next`

Nama keadaan yang akan ditransisi setelah melaksanakan tindakan dalam keadaan saat ini.

`Features`

Serangkaian fitur produk yang akan ditampilkan di file `awsiotdevicetester_report.xml`.

`Feature`

Nama fitur

`FeatureValue`

Opsional. Nilai kustom yang akan digunakan dalam laporan, dan bukan `supported`. Jika nilai ini tidak ditentukan, maka berdasarkan hasil tes, nilai fitur akan diatur ke `supported` atau `not-supported`.

Jika Anda menggunakan nilai kustom untuk `FeatureValue`, Anda dapat menguji fitur yang sama dengan kondisi yang berbeda, dan IDT akan menggabungkan nilai fitur untuk kondisi yang didukung. Misalnya, petikan berikut menunjukkan fitur `MyFeature` dengan dua nilai fitur yang terpisah:

```
...
{
    "Feature": "MyFeature",
    "FeatureValue": "first-feature-supported",
    "Groups": ["first-feature-group"]
},
{
    "Feature": "MyFeature",
    "FeatureValue": "second-feature-supported",
    "Groups": ["second-feature-group"]
},
...
```

Jika kedua grup uji itu lulus, nilai fitur akan diatur ke `first-feature-supported, second-feature-supported`.

`Groups`

Opsional. Array kelompok uji IDs. Semua tes dalam setiap kelompok uji yang ditentukan harus lulus pada fitur yang akan didukung.

`OneOfGroups`

Opsional. Array kelompok uji IDs. Semua tes dalam setidaknya satu kelompok uji yang ditentukan harus lulus pada fitur yang akan didukung.

`TestCases`

Opsional. Sebuah array kasus uji IDs. Jika Anda menentukan nilai ini, maka hal berikut ini akan berlaku:

- Semua uji kasus yang ditentukan harus lulus pada fitur yang akan didukung.

- `Groups` harus berisi hanya satu ID grup uji.
- `OneOfGroups` tidak boleh ditentukan.

`IsRequired`

Opsional. Atur ke `false` untuk menandai fitur ini sebagai fitur opsional dalam laporan. Nilai default adalah `true`.

`ExecutionMethods`

Opsional. Serangkaian metode eksekusi yang sesuai dengan nilai `protocol` yang ditentukan dalam file `device.json`. Jika nilai ini ditentukan, test runner harus menentukan nilai `protocol` yang cocok dengan salah satu nilai dalam rangkaian ini untuk menyertakan fitur tersebut dalam laporan. Jika nilai ini tidak ditentukan, maka fitur itu akan selalu disertakan dalam laporan.

Untuk menggunakan keadaan `AddProductFeatures`, Anda harus menetapkan nilai `ResultVar` di keadaan `RunTask` ke salah satu nilai berikut:

- Jika Anda menentukan kasus uji individual IDs, maka atur `ResultVar` ke*`group-id_test-id`*`_passed`.
- Jika Anda tidak menentukan kasus uji individual IDs, maka setel `ResultVar` ke*`group-id`*`_passed`.

Keadaan `AddProductFeatures` akan mengecek hasil tes dengan cara berikut:

- Jika Anda tidak menentukan kasus uji apa pun IDs, maka hasil untuk setiap kelompok uji ditentukan dari nilai *`group-id`*`_passed` variabel dalam konteks mesin negara.
- Jika Anda memang menentukan kasus uji IDs, maka hasil untuk masing-masing pengujian ditentukan dari nilai *`group-id_test-id`*`_passed` variabel dalam konteks mesin negara.

Penanganan kesalahan

Jika ID grup yang disediakan dalam keadaan ini bukan ID grup yang valid, maka keadaan ini akan menghasilkan kesalahan eksekusi `AddProductFeaturesError`. Jika keadaan ini menemukan kesalahan eksekusi, ia juga akan menetapkan variabel `hasExecutionErrors` dalam konteks state machine ke `true`.

Laporan

Keadaan `Report` menghasilkan file *`suite-name`*`_Report.xml` dan `awsiotdevicetester_report.xml`. Keadaan ini juga mengalirkan laporan ke konsol.

```
{
    "Type": "Report",
    "Next": "<state-name>"
}
```

Semua kolom yang berisi nilai wajib diisi seperti yang dijelaskan di sini:

`Next`

Nama keadaan yang akan ditransisi setelah melaksanakan tindakan dalam keadaan saat ini.

Anda harus selalu beralih ke keadaan `Report` menjelang akhir aliran eksekusi tes agar test runner dapat melihat hasil tes. Biasanya, keadaan berikutnya setelah keadaan ini adalah `Succeed`.

Penanganan kesalahan

Jika keadaan ini mengalami masalah dalam menghasilkan laporan, keadaan tersebut akan mengeluarkan kesalahan eksekusi `ReportError`.

LogMessage

Keadaan `LogMessage` akan menghasilkan file `test_manager.log` dan mengalirkan pesan log ke konsol.

```
{
    "Type": "LogMessage",
    "Next": "<state-name>"
    "Level": "info | warn | error"
    "Message": "<message>"
}
```

Semua kolom yang berisi nilai wajib diisi seperti yang dijelaskan di sini:

`Next`

Nama keadaan yang akan ditransisi setelah melaksanakan tindakan dalam keadaan saat ini.

`Level`

Tingkat kesalahan tempat membuat pesan log. Jika Anda menentukan tingkat yang tidak valid, keadaan ini akan menghasilkan pesan kesalahan dan membuangnya.

`Message`

Pesan yang akan dicatat.

SelectGroup

Keadaan `SelectGroup` memperbarui konteks state machine untuk menunjukkan grup mana yang dipilih. Nilai-nilai yang ditetapkan oleh keadaan ini digunakan oleh setiap kondisi `Choice` berikutnya.

```
{
    "Type": "SelectGroup",
    "Next": "<state-name>"
    "TestGroups": [
        <group-id>"
    ]
}
```

Semua kolom yang berisi nilai wajib diisi seperti yang dijelaskan di sini:

`Next`

Nama keadaan yang akan ditransisi setelah melaksanakan tindakan dalam keadaan saat ini.

`TestGroups`

Serangkaian grup uji yang akan ditandai sudah dipilih. Untuk setiap ID grup uji dalam rangkaian ini, variabel *`group-id`*`_selected` akan diatur ke `true` dalam konteks. Pastikan Anda menyediakan grup pengujian yang valid IDs karena IDT tidak memvalidasi apakah grup yang ditentukan ada.

Gagal

Keadaan `Fail` menunjukkan bahwa state machine tidak mengeksekusi dengan benar. Ini adalah keadaan akhir untuk state machine, dan setiap definisi state machine harus mencakup keadaan ini.

```
{
```

```
    "Type": "Fail"
}
```

Berhasil

Keadaan `Succeed` menunjukkan bahwa state machine mengeksekusi dengan benar. Ini adalah keadaan akhir untuk state machine, dan setiap definisi state machine harus mencakup keadaan ini.

```
{
    "Type": "Succeed"
}
```

Konteks mesin keadaan

Konteks state machine adalah dokumen JSON baca-saja yang berisi data yang tersedia untuk state machine selama eksekusi. Konteks state machine hanya dapat diakses dari state machine, dan berisi informasi yang menentukan aliran uji. Misalnya, Anda dapat menggunakan informasi yang dikonfigurasi oleh test runner di file `userdata.json` untuk menentukan apakah pengujian tertentu wajib dijalankan.

Konteks state machine menggunakan format berikut:

```
{
    "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`

Informasi tentang kolam perangkat yang dipilih untuk uji coba. Untuk kolam perangkat yang dipilih, informasi ini diambil dari elemen rangkaian perangkat tingkat atas yang sesuai yang ditentukan dalam file `device.json`.

`userData`

Informasi di file `userdata.json`.

`config`

Informasi menyematkan file `config.json`.

`suiteFailed`

Nilai diatur ke `false` ketika state machine dimulai. Jika grup uji gagal dalam keadaan `RunTask`, maka nilai ini akan ditetapkan ke `true` untuk durasi sisa eksekusi state machine.

`specificTestGroups`

Jika test runner memilih grup pengujian tertentu untuk dijalankan, bukan seluruh rangkaian pengujian, kunci ini dibuat dan berisi daftar grup pengujian tertentu. IDs

`specificTestCases`

Jika test runner memilih kasus pengujian tertentu untuk dijalankan, bukan seluruh rangkaian pengujian, kunci ini dibuat dan berisi daftar kasus uji tertentu. IDs

`hasExecutionErrors`

Tidak keluar saat state machine dimulai. Jika keadaan apa pun menemukan kesalahan eksekusi, variabel ini akan dibuat dan diatur ke `true` selama durasi sisa eksekusi state machine.

Anda dapat menanyakan konteks menggunakan JSONPath notasi. Sintaks untuk JSONPath kueri dalam definisi negara adalah. `{{$.query}}` Anda dapat menggunakan JSONPath kueri sebagai string placeholder di beberapa negara bagian. IDT menggantikan string placeholder dengan nilai kueri yang dievaluasi JSONPath dari konteks. Anda dapat menggunakan placeholder untuk nilai-nilai berikut:

- Nilai `TestCases` dalam keadaan `RunTask`.
- Nilai `Expression` keadaan `Choice`.

Ketika Anda mengakses data dari konteks state machine, pastikan keadaan berikut dipenuhi:

- Jalur JSON Anda harus dimulai dengan `$.`
- Setiap nilai harus dievaluasi pada string, angka, atau boolean.

Untuk informasi selengkapnya tentang penggunaan JSONPath notasi untuk mengakses data dari konteks, lihatGunakan konteks IDT.

Kesalahan eksekusi

Kesalahan eksekusi adalah kesalahan dalam definisi state machine yang ditemui oleh state machine mesin ketika mengeksekusi keadaan. IDT mencatat informasi tentang setiap kesalahan dalam file `test_manager.log` dan mengalirkan pesan log ke konsol.

Anda dapat menggunakan metode berikut untuk menangani kesalahan eksekusi:

- Tambahkan blok `Catch` dalam definisi keadaan.
- Periksa nilai dari nilai `hasExecutionErrors` dalam konteks state machine.

Tangkap

Untuk menggunakan `Catch`, tambahkan hal berikut ini ke definisi keadaan Anda:

```
"Catch": [
    {
        "ErrorEquals": [
            "<error-type>"
        ]
        "Next": "<state-name>"
    }
]
```

Semua kolom yang berisi nilai wajib diisi seperti yang dijelaskan di sini:

`Catch.ErrorEquals`

Serangkaian jenis kesalahan yang akan ditangkap. Jika kesalahan eksekusi cocok dengan salah satu nilai yang ditentukan, maka state machine akan bertransisi ke keadaan yang ditentukan dalam `Catch.Next`. Lihat setiap definisi keadaan untuk informasi tentang jenis kesalahan yang dihasilkannya.

`Catch.Next`

Keadaan berikutnya yang akan ditransisikan jika keadaan saat ini menemukan kesalahan eksekusi yang cocok dengan salah satu nilai yang ditentukan dalam `Catch.ErrorEquals`.

Blok tangkapan ditangani secara berurutan hingga salah satunya cocok. Jika tidak ada kesalahan yang cocok dengan yang tercantum dalam blok Tangkapan, maka state machine akan terus mengeksekusi. Karena kesalahan eksekusi adalah akibat dari definisi keadaan yang salah, kami sarankan Anda beralih ke keadaan gagal ketika suatu keadaan mengalami kesalahan eksekusi.

hasExecutionError

Ketika beberapa keadaan mengalami kesalahan eksekusi, selain mengeluarkan kesalahan, keaadaan itu juga mengatur nilai `hasExecutionError` ke `true` dalam konteks state machine. Anda dapat menggunakan nilai ini untuk mendeteksi ketika terjadi kesalahan, dan kemudian menggunakan keadaan `Choice` untuk mengalihkan state machine ke keadaan `Fail`.

Metode ini memiliki karakteristik sebagai berikut.

- State machine tidak dimulai dengan nilai yang ditugaskan pada `hasExecutionError`, dan nilai ini tidak tersedia sampai keadaan tertentu menetapkannya. Ini berarti bahwa Anda harus secara tegas mengatur `FallthroughOnError` ke `false` untuk keadaan `Choice` yang mengakses nilai ini untuk mencegah state machine berhenti jika tidak ada kesalahan eksekusi yang terjadi.
- Setelah ditetapkan ke `true`, `hasExecutionError` tidak pernah ditetapkan menjadi salah atau dihapus dari konteks. Ini berarti bahwa nilai ini berguna hanya pertama kalinya ketika nilai tersebut ditetapkan ke `true`, dan untuk semua keadaan berikutnya, nilai itu tidak memberikan nilai yang berarti.
- Nilai `hasExecutionError` dibagi dengan semua cabang state machine pada keadaan `Parallel`, yang dapat mengakibatkan hasil yang tidak diharapkan tergantung pada urutan yang diakses.

Karena karakteristik ini, kami tidak menyarankan Anda menggunakan metode ini jika Anda dapat menggunakan blok Catch sebagai gantinya.

Contoh mesin keadaan

Bagian ini menyediakan beberapa contoh konfigurasi state machine.

Contoh

- Contoh state machine: Jalankan grup uji tunggal
- State machine contoh: Jalankan grup uji yang dipilih pengguna
- Contoh state machine: Jalankan grup uji tunggal dengan fitur produk
- Contoh state machine: Jalankan dua grup uji secara paralel

Contoh state machine: Jalankan grup uji tunggal

State machine ini:

- Menjalankan grup uji dengan id `GroupA`, yang harus ada dalam rangkaian pada file `group.json`.
- Memeriksa kesalahan eksekusi dan bertransisi ke `Fail` jika ada yang ditemukan.
- Menghasilkan laporan dan bertransisi ke `Succeed` jika tidak ada kesalahan, dan `Fail` bila sebaliknya.

```
{
    "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"
        }
    }
}
```

State machine contoh: Jalankan grup uji yang dipilih pengguna

State machine ini:

- Memeriksa apakah test runner telah memilih grup uji tertentu. State machine tidak memeriksa uji kasus tertentu karena test runner tidak dapat memilih uji kasus tanpa sekaligus memilih grup uji.
- Jika grup uji sudah dipilih:
  - Jalankan uji kasus dalam grup uji yang dipilih. Untuk melakukannya, state machine tidak secara tegas menentukan grup uji atau uji kasus di keadaan `RunTask`
  - Buat laporan setelah menjalankan semua tes dan keluar.
- Jika grup uji tidak dipilih:
  - Jalankan tes dalam grup uji `GroupA`.
  - Buat laporan dan keluar.

```
{
    "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"
        }
    }
}
```

Contoh state machine: Jalankan grup uji tunggal dengan fitur produk

State machine ini:

- Menjalankan grup uji `GroupA`.
- Memeriksa kesalahan eksekusi dan bertransisi ke `Fail` jika ada yang ditemukan.
- Menambahkan fitur `FeatureThatDependsOnGroupA` pada file `awsiotdevicetester_report.xml`:
  - Jika `GroupA` lulus, fitur tersebut diatur ke `supported`.
  - Fitur ini tidak ditandai opsional dalam laporan.
- Menghasilkan laporan dan bertransisi ke `Succeed` jika tidak ada kesalahan, dan `Fail` bila sebaliknya.

```
{
    "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"
        }
    }
}
```

Contoh state machine: Jalankan dua grup uji secara paralel

State machine ini:

- Menjalankan grup tes `GroupA` dan `GroupB` secara paralel. Variabel `ResultVar` yang disimpan dalam konteks tersebut oleh keadaan `RunTask` dalam state machine cabang yang tersedia pada keadaan `AddProductFeatures`
- Memeriksa kesalahan eksekusi dan bertransisi ke `Fail` jika ada yang ditemukan. State machine ini tidak menggunakan blok `Catch` karena metode itu tidak mendeteksi kesalahan eksekusi di state machine cabang.

- Menambahkan fitur ke file `awsiotdevicetester_report.xml` berdasarkan grup-grup yang lulus
  - Jika `GroupA` lulus, fitur tersebut diatur ke `supported`.
  - Fitur ini tidak ditandai opsional dalam laporan.
- Menghasilkan laporan dan bertransisi ke `Succeed` jika tidak ada kesalahan, dan `Fail` bila sebaliknya.

Jika dua perangkat dikonfigurasi di kolam perangkat, baik `GroupA` maupun `GroupB` dapat berjalan pada saat yang sama. Namun, jika `GroupA` atau `GroupB` memiliki beberapa tes di dalamnya, maka kedua perangkat dapat dialokasikan pada tes tersebut. Jika hanya satu perangkat yang dikonfigurasi, grup uji akan berjalan secara berurutan.

```
{
    "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"
        }
    }
}
```

Buat kasus uji IDT yang dapat dieksekusi

Anda dapat membuat dan menempatkan test case yang dapat dieksekusi di folder test suite dengan cara berikut:

- Untuk rangkaian tes yang menggunakan argumen atau variabel lingkungan dari file `test.json` untuk menentukan tes mana yang akan dijalankan, Anda dapat membuat uji kasus tunggal yang dapat dieksekusi untuk seluruh rangkaian tes, atau tes yang dapat dijalankan untuk setiap grup uji di rangkaian tes.
- Untuk rangkaian tes di mana Anda ingin menjalankan tes tertentu berdasarkan perintah tertentu, Anda membuat satu executable uji kasus untuk setiap uji kasus di rangkaian tes.

Sebagai penyusun tes, Anda dapat menentukan pendekatan yang sesuai untuk kasus penggunaan Anda dan menyusun executable uji kasus yang sesuai. Pastikan bahwa Anda menyediakan jalur eksekusi uji kasus yang benar di setiap file `test.json`, dan bahwa executable yang ditentukan berjalan dengan benar.

Ketika semua perangkat siap untuk dijalankan oleh uji kasus, IDT akan membaca file-file berikut:

- `test.json` untuk uji kasus yang dipilih menentukan proses yang akan dimulai dan variabel lingkungan yang akan diatur.
- `suite.json` untuk rangkaian uji tersebut menentukan variabel lingkungan yang akan diatur.

IDT memulai proses eksekusi pengujian yang diperlukan berdasarkan perintah dan argumen yang ditentukan dalam `test.json` file, dan meneruskan variabel lingkungan yang diperlukan ke proses.

Gunakan IDT Client SDK

Klien IDT SDKs memungkinkan Anda menyederhanakan cara Anda menulis logika pengujian dalam pengujian yang dapat dieksekusi dengan perintah API yang dapat Anda gunakan berinteraksi dengan IDT dan perangkat yang sedang diuji. IDT saat ini menyediakan yang berikut: SDKs

- IDT Client SDK for Python
- IDT Client SDK for Go
- SDK Klien IDT untuk Java

Ini SDKs terletak di *<device-tester-extract-location>*/sdks folder. Ketika Anda membuat executable uji kasus yang baru, Anda harus menyalin SDK yang ingin Anda gunakan ke folder yang berisi executable uji kasus dan mengacu pada SDK dalam kode Anda. Bagian ini memberikan penjelasan singkat tentang perintah API yang tersedia yang dapat Anda gunakan dalam executable uji kasus Anda.

Dalam Bagian Ini

- Interaksi perangkat
- Interaksi IDT
- Interaksi host

Interaksi perangkat

Perintah berikut memungkinkan Anda untuk berkomunikasi dengan perangkat yang diuji tanpa harus menerapkan interaksi perangkat tambahan dan fungsi manajemen konektivitas apa pun.

`ExecuteOnDevice`

Memungkinkan rangkaian tes untuk menjalankan perintah shell pada perangkat yang mendukung SSH atau koneksi Docker shell.

`CopyToDevice`

Memungkinkan rangkaian tes untuk menyalin file lokal dari mesin host yang menjalankan IDT ke lokasi yang ditentukan pada perangkat yang mendukung SSH atau koneksi Docker shell.

`ReadFromDevice`

Memungkinkan rangkaian tes untuk membaca dari port serial perangkat yang mendukung koneksi UART.

> **Note**
>
> Karena IDT tidak mengelola koneksi langsung ke perangkat yang dibuat menggunakan informasi akses perangkat dari konteks, sebaiknya gunakan perintah API interaksi perangkat ini di executable uji kasus. Namun, jika perintah ini tidak memenuhi persyaratan uji kasus Anda, maka Anda dapat mengambil informasi akses perangkat dari konteks IDT dan menggunakannya untuk membuat koneksi langsung ke perangkat dari rangkaian tes.

Untuk membuat sambungan langsung, ambil informasi di `device.connectivity` dan `resource.devices.connectivity` masing-masing untuk perangkat Anda yang sedang diuji dan untuk perangkat sumber daya. Untuk informasi lebih lanjut mengenai penggunaan konteks IDT, lihat Gunakan konteks IDT.

Interaksi IDT

Perintah berikut memungkinkan rangkaian tes Anda untuk berkomunikasi dengan IDT.

`PollForNotifications`

Memungkinkan rangkaian tes untuk memeriksa notifikasi dari IDT.

`GetContextValue` dan **`GetContextString`**

Memungkinkan rangkaian tes untuk mengambil nilai-nilai dari konteks IDT. Untuk informasi selengkapnya, lihat Gunakan konteks IDT.

`SendResult`

Memungkinkan rangkaian tes untuk melaporkan hasil uji kasus ke IDT. Perintah ini harus dipanggil pada akhir setiap uji kasus di rangkaian tes.

Interaksi host

Perintah berikut memungkinkan rangkaian tes Anda untuk berkomunikasi dengan mesin host.

`PollForNotifications`

Memungkinkan rangkaian tes untuk memeriksa notifikasi dari IDT.

`GetContextValue` dan **`GetContextString`**

Memungkinkan rangkaian tes untuk mengambil nilai-nilai dari konteks IDT. Untuk informasi selengkapnya, lihat Gunakan konteks IDT.

`ExecuteOnHost`

Memungkinkan rangkaian tes untuk menjalankan perintah pada mesin lokal dan memungkinkan IDT untuk mengelola siklus hidup executable uji kasus.

Aktifkan perintah IDT CLI

Perintah `run-suite` IDT CLI menyediakan beberapa pilihan yang membiarkan test runner untuk mengkustomisasi pelaksanaan tes. Untuk memungkinkan test runner menggunakan opsi ini untuk menjalankan rangkaian tes kustom Anda, Anda menerapkan dukungan untuk IDT CLI. Jika Anda tidak menerapkan dukungan, test runner masih akan dapat menjalankan tes, tetapi beberapa opsi CLI tidak akan berfungsi dengan benar. Untuk memberikan pengalaman pelanggan yang ideal, kami merekomendasikan agar Anda menerapkan dukungan untuk argumen berikut untuk perintah `run-suite` dalam IDT CLI:

`timeout-multiplier`

Menentukan nilai yang lebih besar dari 1,0 yang akan diterapkan pada semua batas waktu saat menjalankan tes.

Test runner dapat menggunakan argumen ini untuk meningkatkan batas waktu untuk uji kasus yang ingin dijalankannya. Ketika test runner menentukan argumen ini pada perintah `run-suite`, IDT akan menggunakannya untuk menghitung nilai variabel lingkungan IDT_TEST_TIMEOUT dan menetapkan kolom `config.timeoutMultiplier` dalam konteks IDT. Untuk mendukung argumen ini, Anda harus melakukan hal berikut:

- Alih-alih langsung menggunakan nilai batas waktu dari file `test.json`, baca variabel lingkungan IDT_TEST_TIMEOUT untuk mendapatkan nilai batas waktu yang dihitung dengan benar.
- Ambil nilai `config.timeoutMultiplier` dari konteks IDT dan terapkan ia pada batas waktu yang panjang.

Untuk informasi selengkapnya tentang keluar lebih awal karena peristiwa habis waktu, lihat Tentukan perilaku keluar.

`stop-on-first-failure`

Tentukan bahwa IDT harus berhenti menjalankan semua tes jika menemui kegagalan.

Ketika test runner menentukan argumen ini pada perintah `run-suite`, IDT akan berhenti menjalankan pengujian tersebut segera setelah menemui kegagalan. Namun, jika uji kasus berjalan secara paralel, hal ini dapat menyebabkan hasil yang tidak terduga. Untuk menerapkan dukungan, pastikan bahwa jika IDT menemui peristiwa ini, logika pengujian Anda akan menginstruksikan semua uji kasus yang sedang berjalan untuk berhenti, membersihkan sumber daya sementara, dan melaporkan hasil tes ke IDT. Untuk informasi selengkapnya tentang keluar lebih awal karena menemui kegagalan, lihat Tentukan perilaku keluar.

`group-id` dan **`test-id`**

Menentukan bahwa IDT harus menjalankan hanya grup uji atau uji kasus yang dipilih.

Test runner dapat menggunakan argumen ini dengan perintah `run-suite` untuk menentukan perilaku eksekusi tes berikut:

- Jalankan semua tes di dalam grup uji yang ditentukan.
- Jalankan pilihan tes dari dalam grup uji tertentu.

Untuk mendukung argumen ini, state machine untuk rangkaian tes Anda harus menyertakan serangkaian keadaan `RunTask` dan `Choice` tertentu pada state machine Anda. Jika Anda tidak menggunakan state machine kustom, maka state machine IDT default akan meliputi keadaan yang diperlukan untuk Anda dan Anda tidak perlu melakukan tindakan tambahan. Namun, jika Anda menggunakan state machine kustom, gunakan State machine contoh: Jalankan grup uji yang dipilih pengguna sebagai contoh untuk menambahkan keadaan yang diperlukan dalam state machine Anda.

Untuk informasi selengkapnya tentang perintah IDT CLI, lihat Debug dan jalankan rangkaian tes kustom.

Menulis log peristiwa

Saat tes berjalan, Anda mengirim data ke `stdout` dan `stderr` untuk menuliskan log peristiwa dan pesan kesalahan pada konsol. Untuk informasi lebih lanjut tentang format pesan konsol, lihat Format pesan konsol.

Ketika IDT selesai menjalankan rangkaian tes tersebut, informasi ini juga tersedia di file `test_manager.log` yang terletak di *`<devicetester-extract-location>`*`/results/`*`<execution-id>`*`/logs`.

Anda dapat mengonfigurasi setiap uji kasus untuk menuliskan log dari pengujiannya yang dijalankan, termasuk log dari perangkat yang diuji, ke file *`<group-id>_<test-id>`* yang terletak di *`<device-tester-extract-location>`*`/results/`*`execution-id`*`/logs`. Untuk melakukan ini, ambil path ke berkas log dari konteks IDT dengan kueri `testData.logFilePath`, buat file di path itu, dan tuliskan konten yang Anda inginkan padanya. IDT secara otomatis memperbarui jalur berdasarkan uji kasus yang berjalan. Jika Anda memilih untuk tidak membuat file log untuk uji kasus, maka tidak ada file yang akan dibuat untuk uji kasus itu.

Anda juga dapat mengatur executable teks Anda untuk membuat berkas log tambahan yang diperlukan dalam folder *<device-tester-extract-location>*/logs. Kami menyarankan Anda untuk menentukan prefiks yang unik untuk nama file log sehingga file Anda tidak akan ditimpa.

Laporkan hasil ke IDT

IDT menuliskan hasil tes ke file awsiotdevicetester_report.xml dan *suite-name*_report.xml. File laporan ini terletak di *<device-tester-extract-location>*/results/*<execution-id>*/. Kedua laporan tersebut menangkap hasil dari eksekusi rangkaian tes. Untuk informasi selengkapnya tentang skema yang menggunakan IDT untuk laporan ini, lihat Tinjau hasil tes dan log IDT

Untuk mengisi konten file *suite-name*_report.xml, Anda harus menggunakan perintah SendResult untuk melaporkan hasil tes ke IDT sebelum eksekusi tes itu selesai. Jika IDT tidak dapat menemukan hasil tes, ia akan mengeluarkan kesalahan untuk uji kasus tersebut. Kutipan Python berikut menunjukkan perintah yang akan mengirimkan hasil tes ke IDT:

```
request-variable = SendResultRequest(TestResult(result))
client.send_result(request-variable)
```

Jika Anda tidak melaporkan hasil melalui API, IDT akan mencari hasil tes di folder artefak tes. Path ke folder ini disimpan dalam testData.testArtifactsPath yang disimpan dalam konteks IDT. Dalam folder ini, IDT menggunakan file XML yang diurutkan menurut abjad pertama yang ditempatkannya sebagai hasil tes.

Jika logika pengujian Anda menghasilkan hasil JUnit XHTML, Anda dapat menulis hasil pengujian ke file XHTML di folder artefak untuk langsung memberikan hasil ke IDT alih-alih mengurai hasil dan kemudian menggunakan API untuk mengirimkannya ke IDT.

Jika Anda menggunakan metode ini, pastikan bahwa logika pengujian Anda secara akurat merangkum hasil pengujian dan memformat file hasil Anda dalam format yang sama seperti file *suite-name*_report.xml. IDT tidak melakukan validasi data yang Anda berikan, dengan pengecualian berikut:

- IDT mengabaikan semua properti dari tanda testsuites. Sebaliknya, IDT menghitung properti tag dari hasil grup pengujian lainnya yang dilaporkan.
- Setidaknya satu testsuite tag harus ada dalam testsuites.

Karena IDT menggunakan folder artefak yang sama untuk semua uji kasus dan tidak menghapus file hasil antara pengujian yang berjalan, metode ini mungkin juga akan menyebabkan pelaporan yang salah jika IDT membaca file yang salah. Kami menyarankan Anda menggunakan nama yang sama untuk file hasil XML yang dihasilkan di semua uji kasus untuk menimpa hasil untuk setiap uji kasus dan pastikan bahwa hasil yang benar tersedia untuk digunakan oleh IDT. Meskipun Anda dapat menggunakan pendekatan campuran untuk pelaporan di rangkaian pengujian Anda, yaitu menggunakan file hasil XML untuk beberapa uji kasus dan mengirimkan hasil melalui API untuk uji kasus lainnya, kami tidak merekomendasikan pendekatan ini.

Tentukan perilaku keluar

Konfigurasikan executable teks Anda agar selalu keluar dengan kode keluar 0, meskipun uji kasus melaporkan kegagalan atau hasil kesalahan. Gunakan kode keluar bukan nol hanya untuk menunjukkan bahwa suatu uji kasus tidak berjalan atau jika executable uji kasus tidak dapat menyampaikan hasil apapun ke IDT. Ketika IDT menerima kode keluar bukan nol, IDT akan menandai uji kasus tersebut telah mengalami kesalahan yang mencegahnya berjalan.

IDT mungkin meminta atau mengharapkan uji kasus untuk berhenti berjalan sebelum selesai dalam peristiwa berikut. Gunakan informasi ini untuk mengonfigurasi executable uji kasus untuk mendeteksi setiap peristiwa ini dari uji kasus:

Batas waktu

Terjadi ketika uji kasus berjalan lebih lama daripada nilai batas waktu yang ditentukan dalam file `test.json`. Jika test runner menggunakan argumen `timeout-multiplier` untuk menentukan pengali batas waktu, IDT akan menghitung nilai batas waktu dengan pengali tersebut.

Untuk mendeteksi peristiwa ini, gunakan variabel lingkungan IDT_TEST_TIMEOUT. Ketika test runner meluncurkan tes, IDT akan menetapkan nilai variabel lingkungan IDT_TEST_TIMEOUT pada nilai batas waktu yang dihitung (dalam detik) dan melewati variabel pada executable uji kasus. Anda dapat membaca nilai variabel untuk menetapkan penghitung waktu yang sesuai.

Mengganggu

Terjadi ketika test runner menginterupsi IDT. Misalnya, dengan menekan Ctrl+C.

Karena terminal menyebarkan sinyal ke semua proses anak, Anda cukup mengonfigurasi bagian yang menangani sinyal dalam uji kasus Anda untuk mendeteksi sinyal yang terinterupsi.

Atau, Anda dapat secara berkala mengumpulkan API untuk memeriksa nilai boolean `CancellationRequested` di respons API `PollForNotifications`. Ketika IDT menerima sinyal terinterupsi, ia akan menetapkan nilai boolean `CancellationRequested` untuk `true`.

Berhenti pada kegagalan pertama

Terjadi ketika uji kasus yang berjalan secara paralel dengan uji kasus gagal dan test runner menggunakan argumen `stop-on-first-failure` untuk menentukan bahwa IDT harus berhenti ketika menemui kegagalan apa pun.

Untuk mendeteksi peristiwa ini, Anda dapat secara berkala mengumpulkan API untuk memeriksa nilai boolean `CancellationRequested` di respons API `PollForNotifications`. Ketika IDT menemui kegagalan dan dikonfigurasi untuk berhenti pada kegagalan pertama, tetapkan nilai boolean `CancellationRequested` untuk `true`.

Ketika salah satu peristiwa ini terjadi, IDT akan menunggu selama 5 menit untuk setiap uji kasus yang sedang berjalan saat ini untuk menyelesaikan prosesnya. Jika semua uji kasus yang berjalan tidak keluar dalam waktu 5 menit, IDT akan memaksa masing-masing proses untuk berhenti. Jika IDT belum menerima hasil tes sebelum proses berakhir, ia akan menandai uji kasus telah habis waktu. Sebagai praktik terbaik, Anda harus memastikan bahwa uji kasus Anda melakukan tindakan berikut ketika menghadapi salah satu peristiwa berikut:

1. Berhenti menjalankan logika uji normal.
2. Bersihkan sumber daya sementara apa pun, seperti uji artefak pada perangkat yang sedang diuji.
3. Laporkan hasil tes ke IDT, seperti kegagalan uji atau kesalahan.
4. Keluar

Gunakan konteks IDT

Ketika IDT menjalankan rangkaian tes, rangkaian tes tersebut dapat mengakses serangkaian data yang dapat digunakan untuk menentukan bagaimana setiap tes akan berjalan. Data ini disebut konteks IDT. Sebagai contoh, konfigurasi data pengguna yang disediakan oleh test runner di file `userdata.json` tersedia pada rangkaian tes dalam konteks IDT.

Konteks IDT dapat dianggap sebagai dokumen JSON hanya-baca. Rangkaian uji dapat mengambil data dari dan menuliskan data ke konteks tersebut dengan menggunakan jenis data JSON standar seperti objek, rangkaian, angka, dan sebagainya.

Skema konteks

Konteks state machine menggunakan format berikut:

```
{
    "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`

Informasi dari `config.json` file. `config`Bidang ini juga berisi bidang tambahan berikut:

`config.timeoutMultiplier`

Pengganda untuk setiap nilai batas waktu yang digunakan oleh rangkaian tes. Nilai ini ditentukan oleh test runner dari IDT CLI. Nilai default-nya adalah 1.

`config.idRootPath`

Nilai ini adalah placeholder untuk nilai jalur absolut IDT saat mengonfigurasi file. `userdata.json` Ini digunakan oleh perintah build dan flash.

`device`

Informasi tentang kolam perangkat yang dipilih untuk uji coba. Informasi ini setara dengan elemen rangkaian `devices` dalam file `device.json` untuk perangkat yang dipilih.

`devicePool`

Informasi tentang kolam perangkat yang dipilih untuk uji coba. Informasi ini setara dengan elemen rangkaian kolam perangkat tingkat ata yang ditentukan di file `device.json` untuk kolam perangkat yang dipilih.

`resource`

Informasi tentang perangkat sumber daya dari file `resource.json`.

`resource.devices`

Informasi ini setara dengan rangkaian `devices` yang ditentukan dalam file `resource.json`. Setiap elemen `devices` mencakup kolom tambahan berikut:

`resource.device.name`

Nama sumber daya. Nilai ini diatur ke nilai `requiredResource.name` pada file `test.json`.

`testData.awsCredentials`

AWS Kredensi yang digunakan oleh tes untuk terhubung ke cloud. AWS Informasi ini diperoleh dari file `config.json`.

`testData.logFilePath`

Path ke file log di mana uji kasus menuliskan pesan log. Rangkaian tes membuat file ini jika tidak ada.

`userData`

Informasi yang diberikan oleh test runner di file `userdata.json`.

Akses data dalam konteks

Anda dapat menanyakan konteks menggunakan JSONPath notasi dari file konfigurasi Anda dan dari teks Anda yang dapat dieksekusi dengan dan. `GetContextValue` `GetContextString` APIs Sintaks untuk JSONPath string untuk mengakses konteks IDT bervariasi sebagai berikut:

- Pada `suite.json` dan `test.json`, Anda menggunakan `{{`*`query`*`}}`. Artinya, jangan gunakan elemen root `$.` untuk memulai ekspresi Anda.
- Pada `statemachine.json`, Anda menggunakan `{{$.`*`query`*`}}`.
- Dalam perintah API, Anda menggunakan *`query`* atau `{{$.`*`query`*`}}`, tergantung pada perintahnya. Untuk informasi selengkapnya, lihat dokumentasi sebaris di. SDKs

Tabel berikut menjelaskan operator dalam JSONPath ekspresi foobar khas:

Operator	Deskripsi
`$`	Elemen root. Karena nilai konteks tingkat atas untuk IDT adalah objek, Anda biasanya akan menggunakannya `$.` untuk memulai kueri Anda.
`.childName`	Mengakses elemen anak dengan nama `childName` dari objek. Jika diterapkan ke array, menghasilkan array baru dengan operator ini diterapkan ke setiap elemen. Nama elemen peka huruf besar/kecil. Misalnya, kueri untuk mengakses `awsRegion` nilai dalam `config` objek adalah`$.config.awsRegion` .
`[start:end]`	Memfilter elemen dari array, mengambil item yang dimulai dari `start` indeks dan naik ke `end` indeks, keduanya inklusif.

Operator	Deskripsi
`[index1, index2, ... , indexN]`	Memfilter elemen dari array, mengambil item dari hanya indeks yang ditentukan.
`[?(expr)]`	Menyaring elemen dari array menggunakan `expr` ekspresi. Ekspresi ini harus mengevalu asi nilai boolean.

Untuk membuat ekspresi filter, gunakan sintaks berikut:

```
<jsonpath> | <value> operator <jsonpath> | <value>
```

Dalam sintaks ini:

- `jsonpath`adalah JSONPath yang menggunakan sintaks JSON standar.
- `value` adalah setiap nilai kustom yang menggunakan sintaks JSON standar.
- `operator` adalah salah satu dari operator berikut ini:
  - < (Kurang dari)
  - <= (Kurang dari atau sama dengan)
  - == (Sama dengan)

    Jika nilai JSONPath atau dalam ekspresi Anda adalah nilai array, boolean, atau objek, maka ini adalah satu-satunya operator biner yang didukung yang dapat Anda gunakan.
  - >= (Lebih besar dari atau sama dengan)
  - > (Lebih besar dari)
  - =~ (Kecocokan ekspresi reguler). Untuk menggunakan operator ini dalam ekspresi filter, nilai JSONPath or di sisi kiri ekspresi Anda harus mengevaluasi ke string dan sisi kanan harus berupa nilai pola yang mengikuti RE2sintaks.

Anda dapat menggunakan JSONPath kueri dalam formulir {{*query*}} sebagai string placeholder di dalam dan bidang dalam file `args` dan dalam `environmentVariables` bidang dalam `test.json` file. `environmentVariables` `suite.json` IDT melakukan pencarian konteks dan mengisi kolom dengan nilai kueri yang dievaluasi. Misalnya, di file `suite.json`, Anda dapat menggunakan string placeholder untuk menentukan nilai variabel lingkungan yang berubah dengan setiap uji kasus dan

IDT akan mengisi variabel lingkungan dengan nilai yang benar untuk setiap uji kasus. Namun, ketika Anda menggunakan string placeholder di file `test.json` dan `suite.json`, pertimbangan berikut berlaku untuk kueri Anda:

- Anda harus menuliskan setiap kejadian kunci `devicePool` dalam kueri Anda semua dengan huruf kecil. Artinya, gunakan `devicepool` sebagai gantinya
- Untuk rangkaian, Anda hanya dapat menggunakan rangkaian string. Selain itu, rangkaian menggunakan format `item1, item2,...,itemN` non-standar. Jika rangkaian tersebut hanya berisi satu elemen, maka rangkaian itu akan diserialkan sebagai `item`, sehingga menjadikannya tidak dapat dibedakan dari kolom string.
- Anda tidak dapat menggunakan placeholder untuk mengambil objek dari konteks.

Karena pertimbangan ini, kami merekomendasikan bahwa bila memungkinkan, Anda menggunakan API untuk mengakses konteks dalam logika pengujian Anda dan bukan string placeholder di file `test.json` dan `suite.json`. Namun, dalam beberapa kasus mungkin lebih mudah menggunakan JSONPath placeholder untuk mengambil string tunggal untuk ditetapkan sebagai variabel lingkungan.

Mengonfigurasi pengaturan untuk test runner

Untuk menjalankan rangkaian tes kustom, test runner harus mengonfigurasi pengaturannya berdasarkan rangkaia tes yang ingin dijalankannya. Pengaturan ditentukan berdasarkan templat file konfigurasi yang terletak di *`<device-tester-extract-location>`*`/configs/` folder. Jika diperlukan, pelari uji juga harus menyiapkan AWS kredensil yang akan digunakan IDT untuk terhubung ke cloud. AWS

Sebagai penyusun tes, Anda perlu mengonfigurasi file-file ini untuk men-debug rangkaian tes. Anda harus memberikan petunjuk kepada test runner agar dapat mengonfigurasi pengaturan berikut yang diperlukan untuk menjalankan rangkaian tes Anda.

Konfigurasikan device.json

File `device.json` berisi informasi tentang perangkat tempat uji dijalankan (misalnya, alamat IP, informasi login, sistem operasi, dan arsitektur CPU).

Test runner dapat memberikan informasi ini dengan menggunakan file `device.json` templat berikut yang terletak di folder *`<device-tester-extract-location>`*`/configs/`.

```
[
    {
```

```
        "id": "<pool-id>",
        "sku": "<pool-sku>",
        "features": [
            {
                "name": "<feature-name>",
                "value": "<feature-value>",
                "configs": [
                    {
                        "name": "<config-name>",
                        "value": "<config-value>"
                    }
                ],
            }
        ],
        "devices": [
            {
                "id": "<device-id>",
                "pairedResource": "<device-id>", //used for no-op protocol
                "connectivity": {
                    "protocol": "ssh | uart | docker | no-op",
                    // ssh
                    "ip": "<ip-address>",
                    "port": <port-number>,
                    "publicKeyPath": "<public-key-path>",
                    "auth": {
                        "method": "pki | password",
                        "credentials": {
                            "user": "<user-name>",
                            // pki
                            "privKeyPath": "/path/to/private/key",

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

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

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

```
        ]
    }
]
```

Semua kolom yang berisi nilai wajib diisi seperti yang dijelaskan di sini:

`id`

ID alfanumerik yang ditetapkan pengguna secara unik mengidentifikasi kumpulan perangkat yang disebut kolam perangkat. Perangkat yang termasuk dalam suatu kolam harus memiliki perangkat keras yang identik. Ketika Anda menjalankan serangkaian tes, perangkat di kolam tersebut digunakan untuk memparalelkan beban kerja. Beberapa perangkat digunakan untuk menjalankan tes yang berbeda.

`sku`

Nilai alfanumerik secara unik mengidentifikasi perangkat yang sedang diuji. SKU digunakan untuk melacak perangkat yang berkualitas.

> **Note**
>
> Jika Anda ingin mencantumkan papan Anda di Katalog Perangkat AWS Mitra, SKU yang Anda tentukan di sini harus cocok dengan SKU yang Anda gunakan dalam proses pencatatan.

`features`

Opsional. Rangkaian yang berisi fitur perangkat yang didukung. Fitur perangkat adalah nilai-nilai yang ditetapkan pengguna yang Anda konfigurasikan di rangkaian tes Anda. Anda harus memberikan informasi kepada test runner tentang nama fitur dan nilai-nilai yang akan disertakan dalam file `device.json`. Misalnya, jika Anda ingin menguji perangkat yang berfungsi sebagai server MQTT untuk perangkat lain, maka Anda dapat mengonfigurasi logika uji Anda untuk memvalidasi tingkat tertentu yang didukung untuk fitur bernama `MQTT_QoS`. Pelari uji memberikan nama fitur ini dan menetapkan nilai fitur ke level QoS yang didukung oleh perangkat mereka. Anda dapat mengambil informasi yang diberikan dari konteks IDT dengan kueri `devicePool.features`, atau dari konteks state machine dengan kueri `pool.features`.

`features.name`

Nama fitur.

`features.value`

Nilai fitur yang didukung.

`features.configs`

Pengaturan konfigurasi, jika diperlukan, untuk fitur.

`features.config.name`

Nama pengaturan konfigurasi.

`features.config.value`

Nilai pengaturan yang didukung.

`devices`

Rangkaian perangkat di kolam yang akan diuji. Setidaknya diperlukan satu perangkat.

`devices.id`

Pengenal unik yang ditetapkan pengguna untuk perangkat yang sedang diuji.

`devices.pairedResource`

Pengidentifikasi unik yang ditentukan pengguna untuk perangkat sumber daya. Nilai ini diperlukan saat Anda menguji perangkat menggunakan protokol `no-op` konektivitas.

`connectivity.protocol`

Protokol komunikasi yang digunakan untuk berkomunikasi dengan perangkat ini. Setiap perangkat di kolam harus menggunakan protokol yang sama.

Saat ini, satu-satunya nilai yang didukung adalah `ssh` dan `uart` untuk perangkat fisik, `docker` untuk wadah Docker, dan `no-op` untuk perangkat yang tidak memiliki koneksi langsung dengan mesin host IDT tetapi memerlukan perangkat sumber daya sebagai middleware fisik untuk berkomunikasi dengan mesin host.

Untuk perangkat tanpa operasi, Anda mengonfigurasi ID perangkat sumber daya di`devices.pairedResource`. Anda juga harus menentukan ID ini dalam `resource.json` file. Perangkat yang dipasangkan harus berupa perangkat yang dipasangkan secara fisik dengan perangkat yang sedang diuji. Setelah IDT mengidentifikasi dan menghubungkan ke perangkat sumber daya yang dipasangkan, IDT tidak akan terhubung ke perangkat sumber daya lain sesuai dengan fitur yang dijelaskan dalam file. `test.json`

`connectivity.ip`

Alamat IP perangkat yang sedang diuji.

Properti ini hanya berlaku jika `connectivity.protocol` diatur ke `ssh`.

`connectivity.port`

Opsional. Jumlah port yang akan digunakan untuk koneksi SSH.

Nilai default-nya adalah 22.

Properti ini hanya berlaku jika `connectivity.protocol` diatur ke `ssh`.

`connectivity.publicKeyPath`

Opsional. Jalur lengkap ke kunci publik digunakan untuk mengautentikasi koneksi ke perangkat yang sedang diuji. Saat Anda menentukan`publicKeyPath`, IDT memvalidasi kunci publik perangkat saat membuat koneksi SSH ke perangkat yang sedang diuji. Jika nilai ini tidak ditentukan, IDT membuat koneksi SSH, tetapi tidak memvalidasi kunci publik perangkat.

Kami sangat menyarankan Anda menentukan jalur ke kunci publik, dan Anda menggunakan metode aman untuk mengambil kunci publik ini. Untuk klien SSH berbasis baris perintah standar, kunci publik disediakan dalam file. `known_hosts` Jika Anda menentukan file kunci publik terpisah, file ini harus menggunakan format yang sama dengan `known_hosts` file, yaitu,`ip-address key-type public-key`.

`connectivity.auth`

Informasi autentikasi untuk koneksi tersebut.

Properti ini hanya berlaku jika `connectivity.protocol` diatur ke `ssh`.

`connectivity.auth.method`

Metode autentikasi yang digunakan untuk mengakses perangkat melalui protokol konektivitas yang diberikan.

Nilai yang didukung adalah:

- `pki`
- `password`

`connectivity.auth.credentials`

Kredensial yang digunakan untuk autentikasi.

`connectivity.auth.credentials.password`

Kata sandi yang digunakan untuk masuk ke perangkat yang sedang diuji.

Nilai ini hanya berlaku jika `connectivity.auth.method` diatur ke `password`.

`connectivity.auth.credentials.privKeyPath`

Jalur lengkap ke kunci privat yang digunakan untuk masuk ke perangkat yang sedang diuji.

Nilai ini hanya berlaku jika `connectivity.auth.method` diatur ke `pki`.

`connectivity.auth.credentials.user`

Nama pengguna untuk masuk ke perangkat yang sedang diuji.

`connectivity.serialPort`

Opsional. Port serial tempat perangkat itu terhubung.

Properti ini hanya berlaku jika `connectivity.protocol` diatur ke `uart`.

`connectivity.containerId`

ID kontainer atau nama kontainer Docker yang sedang diuji.

Properti ini hanya berlaku jika `connectivity.protocol` diatur ke `docker`.

`connectivity.containerUser`

Opsional. Nama pengguna untuk pengguna di dalam kontainer. Nilai default adalah pengguna yang disediakan di Dockerfile.

Nilai default-nya adalah 22.

Properti ini hanya berlaku jika `connectivity.protocol` diatur ke `docker`.

> **Note**
>
> Untuk memeriksa apakah test runner mengonfigurasi koneksi perangkat yang salah untuk suatu pengujian, Anda dapat mengambil `pool.Devices[0].Connectivity.Protocol` dari konteks state machine dan membandingkannya dengan nilai yang diharapkan pada keadaan `Choice`. Jika protokol yang salah digunakan, cetak pesan dengan menggunakan keadaan `LogMessage` dan beralihlah ke keadaan `Fail`.

Atau, Anda dapat menggunakan kode penanganan kesalahan untuk melaporkan kegagalan pengujian untuk jenis perangkat yang salah.

(Opsional) Konfigurasikan userdata.json

File `userdata.json` berisi informasi tambahan yang diperlukan oleh rangkaian tes tetapi tidak ditentukan dalam file `device.json`. Format file ini tergantung pada file `userdata_scheme.json` yang ditentukan dalam rangkaian uji tersebut. Jika Anda seorang penyusun tes, pastikan Anda memberikan informasi ini kepada pengguna yang akan menjalankan rangkaian tes yang Anda susun.

(Opsional) Konfigurasikan resource.json

File `resource.json` berisi informasi tentang perangkat apa pun yang akan digunakan sebagai perangkat sumber daya. Perangkat sumber daya adalah perangkat yang diperlukan untuk menguji kemampuan tertentu dari perangkat yang diuji. Misalnya, untuk menguji kemampuan Bluetooth perangkat, Anda mungkin menggunakan perangkat sumber daya untuk menguji apakah perangkat Anda dapat berhasil tersambung. Perangkat sumber daya bersifat opsional, dan Anda dapat memerlukan perangkat sumber daya sebanyak yang Anda butuhkan. Sebagai penyusun tes, Anda menggunakan file test.json untuk menentukan fitur perangkat sumber daya yang diperlukan untuk tes. Test runner kemudian akan menggunakan file `resource.json` untuk menyediakan kolam perangkat sumber daya yang memiliki fitur yang diperlukan. Pastikan Anda memberikan informasi ini kepada pengguna yang akan menjalankan rangkaian tes yang Anda tulis.

Test runner dapat memberikan informasi ini dengan menggunakan file `resource.json` templat berikut yang terletak di folder *`<device-tester-extract-location>`*`/configs/`.

```
[
    {
        "id": "<pool-id>",
        "features": [
            {
                "name": "<feature-name>",
                "version": "<feature-value>",
                "jobSlots": <job-slots>
            }
        ],
        "devices": [
            {
                "id": "<device-id>",
                "connectivity": {
```

```
                    "protocol": "ssh | uart | docker",
                    // ssh
                    "ip": "<ip-address>",
                    "port": <port-number>,
                    "publicKeyPath": "<public-key-path>",
                    "auth": {
                        "method": "pki | password",
                        "credentials": {
                            "user": "<user-name>",
                            // pki
                            "privKeyPath": "/path/to/private/key",

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

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

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

Semua kolom yang berisi nilai wajib diisi seperti yang dijelaskan di sini:

`id`

ID alfanumerik yang ditetapkan pengguna secara unik mengidentifikasi kumpulan perangkat yang disebut kolam perangkat. Perangkat yang termasuk dalam suatu kolam harus memiliki perangkat keras yang identik. Ketika Anda menjalankan serangkaian tes, perangkat di kolam tersebut digunakan untuk memparalelkan beban kerja. Beberapa perangkat digunakan untuk menjalankan tes yang berbeda.

`features`

Opsional. Rangkaian yang berisi fitur perangkat yang didukung. Informasi yang diperlukan dalam kolom ini ditentukan dalam file test.json di rangkaian tes dan menentukan tes mana yang akan

dijalankan dan bagaimana menjalankan tes tersebut. Jika rangkaian tes tidak memerlukan fitur apa pun, kolom ini tidak wajib diisi.

`features.name`

Nama fitur.

`features.version`

Versi fitur.

`features.jobSlots`

Pengaturan untuk menunjukkan berapa banyak tes yang dapat secara bersamaan menggunakan perangkat. Nilai default-nya adalah 1.

`devices`

Rangkaian perangkat di kolam yang akan diuji. Setidaknya diperlukan satu perangkat.

`devices.id`

Pengenal unik yang ditetapkan pengguna untuk perangkat yang sedang diuji.

`connectivity.protocol`

Protokol komunikasi yang digunakan untuk berkomunikasi dengan perangkat ini. Setiap perangkat di kolam harus menggunakan protokol yang sama.

Saat ini, satu-satunya nilai yang didukung adalah `ssh` dan `uart` untuk perangkat fisik, dan `docker` untuk kontainer Docker.

`connectivity.ip`

Alamat IP perangkat yang sedang diuji.

Properti ini hanya berlaku jika `connectivity.protocol` diatur ke `ssh`.

`connectivity.port`

Opsional. Jumlah port yang akan digunakan untuk koneksi SSH.

Nilai default-nya adalah 22.

Properti ini hanya berlaku jika `connectivity.protocol` diatur ke `ssh`.

`connectivity.publicKeyPath`

Opsional. Jalur lengkap ke kunci publik digunakan untuk mengautentikasi koneksi ke perangkat yang sedang diuji. Saat Anda menentukan`publicKeyPath`, IDT memvalidasi kunci

publik perangkat saat membuat koneksi SSH ke perangkat yang sedang diuji. Jika nilai ini tidak ditentukan, IDT membuat koneksi SSH, tetapi tidak memvalidasi kunci publik perangkat.

Kami sangat menyarankan Anda menentukan jalur ke kunci publik, dan Anda menggunakan metode aman untuk mengambil kunci publik ini. Untuk klien SSH berbasis baris perintah standar, kunci publik disediakan dalam file. `known_hosts` Jika Anda menentukan file kunci publik terpisah, file ini harus menggunakan format yang sama dengan `known_hosts` file, yaitu,`ip-address key-type public-key`.

`connectivity.auth`

Informasi autentikasi untuk koneksi tersebut.

Properti ini hanya berlaku jika `connectivity.protocol` diatur ke `ssh`.

`connectivity.auth.method`

Metode autentikasi yang digunakan untuk mengakses perangkat melalui protokol konektivitas yang diberikan.

Nilai yang didukung adalah:

- `pki`
- `password`

`connectivity.auth.credentials`

Kredensial yang digunakan untuk autentikasi.

`connectivity.auth.credentials.password`

Kata sandi yang digunakan untuk masuk ke perangkat yang sedang diuji.

Nilai ini hanya berlaku jika `connectivity.auth.method` diatur ke `password`.

`connectivity.auth.credentials.privKeyPath`

Jalur lengkap ke kunci privat yang digunakan untuk masuk ke perangkat yang sedang diuji.

Nilai ini hanya berlaku jika `connectivity.auth.method` diatur ke `pki`.

`connectivity.auth.credentials.user`

Nama pengguna untuk masuk ke perangkat yang sedang diuji.

`connectivity.serialPort`

Opsional. Port serial tempat perangkat itu terhubung.

Properti ini hanya berlaku jika `connectivity.protocol` diatur ke `uart`.

`connectivity.containerId`

ID kontainer atau nama kontainer Docker yang sedang diuji.

Properti ini hanya berlaku jika `connectivity.protocol` diatur ke `docker`.

`connectivity.containerUser`

Opsional. Nama pengguna untuk pengguna di dalam kontainer. Nilai default adalah pengguna yang disediakan di Dockerfile.

Nilai default-nya adalah 22.

Properti ini hanya berlaku jika `connectivity.protocol` diatur ke `docker`.

(Opsional) Konfigurasikan config.json

File `config.json` berisi informasi konfigurasi untuk IDT. Biasanya, pelari uji tidak perlu memodifikasi file ini kecuali untuk memberikan kredensi AWS pengguna mereka untuk IDT, dan secara opsional, suatu wilayah. AWS Jika AWS kredensil dengan izin yang diperlukan disediakan, AWS IoT Device Tester kumpulkan dan kirimkan metrik penggunaan ke. AWS Ini adalah fitur opt-in dan digunakan untuk meningkatkan fungsi IDT. Untuk informasi selengkapnya, lihat Kirim metrik penggunaan IDT.

Pelari uji dapat mengonfigurasi AWS kredensialnya dengan salah satu cara berikut:

- File kredensil

  IDT menggunakan file kredensial yang sama sebagai AWS CLI. Untuk informasi selengkapnya, lihat File konfigurasi dan kredensial.

  Lokasi file kredensial itu bervariasi, tergantung pada sistem operasi yang Anda gunakan:

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

Variabel lingkungan adalah variabel yang dikelola oleh sistem operasi dan digunakan oleh perintah sistem. Variabel yang ditentukan selama sesi SSH tidak akan tersedia setelah sesi itu ditutup. IDT dapat menggunakan variabel lingkungan `AWS_ACCESS_KEY_ID` dan `AWS_SECRET_ACCESS_KEY` untuk menyimpan kredensial AWS

Untuk mengatur variabel ini di Linux, macOS, atau Unix, gunakan export:

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

Untuk menetapkan variabel ini di Windows, gunakan set:

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

Untuk mengonfigurasi AWS kredensi IDT, pelari uji mengedit `auth` bagian dalam `config.json` file yang terletak di folder. *<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>"
        }
    }
}
]
```

Semua kolom yang berisi nilai wajib diisi seperti yang dijelaskan di sini:

> **Note**
>
> Semua jalur dalam file ini didefinisikan relatif terhadap file*<device-tester-extract-location>*.

`log.location`

Jalur ke folder log di file*<device-tester-extract-location>*.

`configFiles.root`

Path ke folder yang berisi file konfigurasi.

`configFiles.device`

Jalur ke file `device.json`.

`testPath`

Path ke folder yang berisi rangkaian tes.

`reportPath`

Path ke folder yang akan berisi hasil tes setelah IDT menjalankan rangkaian tes.

`awsRegion`

Opsional. AWS Wilayah yang akan digunakan test suite. Jika tidak ditetapkan, rangkaian tes akan menggunakan wilayah default yang ditentukan dalam setiap rangkaian tes.

`auth.method`

Metode yang digunakan IDT untuk mengambil kredensil AWS . Nilai yang didukung adalah `file` untuk mengambil kredensial dari file kredensial, dan `environment` untuk mengambil klredensial dengan menggunakan variabel lingkungan.

`auth.credentials.profile`

Profil kredensial yang akan digunakan dari file kredensial. Properti ini hanya berlaku jika `auth.method` diatur ke `file`.

Debug dan jalankan rangkaian tes kustom

Setelah konfigurasi yang diperlukan diatur, IDT dapat menjalankan rangkaian tes Anda. Waktu aktif dari rangkaian tes penuh akan tergantung pada perangkat keras dan komposisi rangkaian tes. Untuk

referensi, dibutuhkan sekitar 30 menit untuk menyelesaikan rangkaian tes kualifikasi FreeRTOS lengkap pada Raspberry Pi 3B.

Ketika Anda menyusun rangkaian tes Anda, Anda dapat menggunakan IDT untuk menjalankan rangkaian tes dalam mode debug untuk memeriksa kode Anda sebelum Anda menjalankannya atau memberikannya kepada test runner.

Jalankan IDT dalam mode debug

Karena rangkaian tes tergantung pada IDT untuk berinteraksi dengan perangkat, menyediakan konteks, dan menerima hasil, Anda tidak bisa hanya men-debug rangkaian tes Anda di IDE tanpa berinteraksi dengan IDT. Untuk melakukannya, IDT CLI menyediakan perintah `debug-test-suite` yang memungkinkan Anda menjalankan IDT dalam mode debug. Jalankan perintah berikut untuk menampilkan opsi yang tersedia untuk `debug-test-suite`:

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

Saat Anda menjalankan IDT dalam mode debug, IDT sebenarnya tidak meluncurkan rangkaian pengujian atau menjalankan orkestrator pengujian; sebaliknya, IDT berinteraksi dengan IDE Anda untuk merespons permintaan yang dibuat dari rangkaian pengujian yang berjalan di IDE dan mencetak log ke konsol. IDT tidak melakukan timeout dan menunggu untuk keluar hingga secara manual terinterupsi. Dalam mode debug, IDT juga tidak menjalankan orkestrator pengujian dan tidak akan menghasilkan file laporan apa pun. Untuk men-debug rangkaian pengujian Anda, Anda harus menggunakan IDE Anda untuk memberikan beberapa informasi yang biasanya diperoleh IDT dari file konfigurasi. Pastikan Anda memberikan informasi berikut:

- Variabel lingkungan dan argumen untuk setiap tes. IDT tidak akan membaca informasi ini dari `test.json` atau `suite.json`.
- Argumen untuk memilih perangkat sumber daya. IDT tidak akan membaca informasi ini dari `test.json`.

Untuk men-debug rangkaian tes Anda, selesaikan langkah berikut:

1. Buat file konfigurasi pengaturan yang diperlukan untuk menjalankan rangkaian tes. Misalnya, jika rangkaian tes Anda memerlukan `device.json`, `resource.json`, dan `user data.json`, pastikan Anda mengonfigurasi semuanya sesuai kebutuhan.
2. Jalankan perintah berikut untuk menempatkan IDT dalam mode debug dan pilih perangkat yang diperlukan untuk menjalankan tes.

```
devicetester_[linux | mac | win_x86-64] debug-test-suite [options]
```

Setelah Anda menjalankan perintah ini, IDT akan menunggu permintaan dari rangkaian tes dan kemudian menanggapinya. IDT juga akan menghasilkan variabel lingkungan yang diperlukan untuk proses kasus untuk IDT Client SDK.

3. Dalam IDE Anda, gunakan konfigurasi `run` atau `debug` untuk melakukan hal berikut:
   a. Menetapkan nilai-nilai variabel lingkungan yang dihasilkan IDT.
   b. Tetapkan nilai dari setiap variabel lingkungan atau argumen yang Anda tentukan dalam file `test.json` dan `suite.json` Anda.
   c. Menetapkan breakpoint sesuai kebutuhan.
4. Menjalankan rangkaian tes di IDE Anda.

   Anda dapat men-debug dan kembali menjalankan rangkaain tes sebanyak mungkin yang diperlukan. IDT tidak melakukan timeout dalam mode debug.
5. Setelah Anda menyelesaikan debugging, interupsi IDT untuk keluar dari mode debug.

Perintah IDT CLI untuk menjalankan percobaan

Bagian berikut menjelaskan perintah IDT CLI:

IDT v4.0.0

`help`

Mendaftar informasi tentang perintah yang ditentukan.

`list-groups`

Mendaftar grup dalam rangkaian tes yang diberikan.

`list-suites`

Mencantumkan rangkaian tes yang tersedia.

`list-supported-products`

Daftar produk yang didukung untuk versi IDT Anda, dalam hal ini versi FreeRTOS, dan versi rangkaian pengujian kualifikasi FreeRTOS yang tersedia untuk versi IDT saat ini.

`list-test-cases`

Mencantumkan uji kasus dalam grup uji yang diberikan. Opsi berikut didukung:

- `group-id`. Grup uji yang harus dicari. Opsi ini diperlukan dan harus menentukan satu grup.

`run-suite`

Menjalankan serangkaian tes pada kolam perangkat. Berikut ini adalah beberapa opsi yang umum digunakan:

- `suite-id`. Versi rangkaian tes yang akan dijalankan. Jika tidak ditentukan, IDT akan menggunakan versi terbaru dalam folder `tests`.
- `group-id`. Grup uji yang akan jalankan, sebagai daftar yang dipisahkan koma. Jika tidak ditentukan, IDT akan menjalankan semua grup uji di rangkaian tes.
- `test-id`. Uji kasus yang akan dijalankan, sebagai daftar yang dipisahkan koma. Ketika ditentukan, `group-id` harus menentukan satu grup.
- `pool-id`. Kolam perangkat yang akan diuji. Test runner harus menentukan kolam jika memiliki beberapa perangkat kolam yang ditentukan dalam file `device.json`.
- `timeout-multiplier`. Mengkonfigurasi IDT untuk mengubah batas waktu eksekusi tes yang ditentukan dalam file `test.json` untuk tes dengan pengganda yang ditetapkan pengguna.
- `stop-on-first-failure`. Mengkonfigurasi IDT untuk menghentikan eksekusi pada kegagalan pertama. Pilihan ini harus digunakan dengan `group-id` untuk men-debug grup uji yang ditentukan.
- `userdata`. Menetapkan file yang berisi informasi data pengguna yang diperlukan untuk menjalankan rangkaian tes. Hal ini hanya diperlukan jika `userdataRequired` diatur ke BETUL di file `suite.json`untuk rangkaian uji itu.

Untuk informasi lebih lanjut tentang opsi `run-suite`, gunakan opsi `help`:

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

`debug-test-suite`

Jalankan rangkaian tes dalam mode debug. Untuk informasi selengkapnya, lihat Jalankan IDT dalam mode debug.

Tinjau hasil tes dan log IDT

Bagian ini menjelaskan format di mana IDT menghasilkan log konsol dan laporan uji.

Format pesan konsol

AWS IoT Device Tester menggunakan format standar untuk mencetak pesan ke konsol saat memulai rangkaian pengujian. Kutipan berikut menunjukkan contoh pesan konsol yang dihasilkan oleh IDT.

```
[INFO] [2000-01-02 03:04:05]: Using suite: MyTestSuite_1.0.0
 executionId=9a52f362-1227-11eb-86c9-8c8590419f30
```

Sebagian besar pesan konsol terdiri dari kolom berikut:

`time`

Sebuah cap waktu ISO 8601 penuh untuk peristiwa yang tercatat.

`level`

Tingkat pesan untuk peristiwa yang tercatat. Biasanya, tingkat pesan yang tercatat adalah salah satu dari `info`, `warn`, atau `error`. IDT mengeluarkan pesan `fatal` atau `panic` jika bertemu dengan peristiwa yang diharapkan yang menyebabkannya keluar lebih awal.

`msg`

Pesan yang dicatat.

`executionId`

Sebuah string ID yang unik untuk proses IDT saat ini. ID ini digunakan untuk membedakan antara IDT individual yang berjalan.

Pesan konsol yang dihasilkan dari rangkaian tes memberikan informasi tambahan tentang perangkat yang diuji berikut rangkaian uji, grup uji, dan uji kasus yang dijalankan oleh IDT. Kutipan berikut menunjukkan contoh pesan konsol yang dihasilkan dari rangkaian tes.

```
[INFO] [2000-01-02 03:04:05]: Hello world! suiteId=MyTestSuitegroupId=myTestGroup
 testCaseId=myTestCase deviceId=my-
deviceexecutionId=9a52f362-1227-11eb-86c9-8c8590419f30
```

Bagian tertentu dari rangkaian tes pada pesan konsol berisi kolom-kolom berikut:

`suiteId`

Nama rangkaian tes yang sedang berjalan saat ini.

`groupId`

ID grup uji yang sedang berjalan saat ini.

`testCaseId`

ID uji kasus yang berjalan saat ini.

`deviceId`

ID dari perangkat yang diuji yang sedang digunakan oleh uji kasus saat ini.

Ringkasan uji berisi informasi tentang rangkaian uji, hasil uji untuk setiap grup yang dijalankan, dan lokasi log dan file laporan yang dihasilkan. Contoh berikut menunjukkan pesan ringkasan tes.

```
========== Test Summary ==========
Execution Time:     5m00s
Tests Completed:    4
Tests Passed:       3
Tests Failed:       1
Tests Skipped:      0
---------------------------------
Test Groups:
    GroupA:         PASSED
    GroupB:         FAILED
---------------------------------
Failed Tests:
    Group Name: GroupB
        Test Name: TestB1
            Reason: Something bad happened
---------------------------------
Path to AWS IoT Device Tester Report: /path/to/awsiotdevicetester_report.xml
Path to Test Execution Logs: /path/to/logs
Path to Aggregated JUnit Report: /path/to/MyTestSuite_Report.xml
```

AWS IoT Device Tester skema laporan

`awsiotdevicetester_report.xml` adalah laporan bertanda tangan yang berisi informasi berikut:

- Versi IDT.

- Versi rangkaian tes.
- Tanda tangan laporan dan kunci yang digunakan untuk menandatangani laporan.
- SKU Perangkat dan nama kolam perangkat yang ditentukan dalam file `device.json`.
- Versi produk dan fitur perangkat yang diuji.
- Ringkasan agregat hasil tes. Informasi ini sama dengan yang terkandung dalam file *suite-name*`_report.xml`.

```
<apnreport>
    <awsiotdevicetesterversion>idt-version</awsiotdevicetesterversion>
    <testsuiteversion>test-suite-version</testsuiteversion>
    <signature>signature</signature>
    <keyname>keyname</keyname>
    <session>
        <testsession>execution-id</testsession>
        <starttime>start-time</starttime>
        <endtime>end-time</endtime>
    </session>
    <awsproduct>
        <name>product-name</name>
        <version>product-version</version>
        <features>
            <feature name="<feature-name>" value="supported | not-supported | <feature-
value>" type="optional | required"/>
        </features>
    </awsproduct>
    <device>
        <sku>device-sku</sku>
        <name>device-name</name>
        <features>
            <feature name="<feature-name>" value="<feature-value>"/>
        </features>
        <executionMethod>ssh | uart | docker</executionMethod>
    </device>
    <devenvironment>
        <os name="<os-name>"/>
    </devenvironment>
    <report>
        <suite-name-report-contents>
    </report>
</apnreport>
```

File `awsiotdevicetester_report.xml` berisi tanda `<awsproduct>` yang berisi informasi tentang produk yang sedang diuji dan fitur produk yang divalidasi setelah menjalankan serangkaian pengujian.

Atribut yang digunakan dalam **`<awsproduct>`** tag

`name`

Nama produk yang sedang diuji.

`version`

Versi produk yang sedang diuji.

`features`

Fitur divalidasi. Fitur yang ditandai sebagai `required` diperlukan bagi rangkaian tes untuk memvalidasi perangkat. Potongan berikut menunjukkan bagaimana informasi ini muncul di file `awsiotdevicetester_report.xml`.

```
<feature name="ssh" value="supported" type="required"></feature>
```

Fitur yang ditandai sebagai `optional` tidak diperlukan untuk validasi. Potongan berikut menunjukkan fitur opsional.

```
<feature name="hsi" value="supported" type="optional"></feature>
<feature name="mqtt" value="not-supported" type="optional"></feature>
```

Skema laporan rangkaian uji

`suite-name``_Result.xml`Laporan ini dalam format JUnit XHTML. Anda dapat mengintegrasikannya ke dalam platform integrasi dan deployment berkelanjutan seperti Jenkins, Bamboo, dan sebagainya. Laporan ini berisi ringkasan agregat hasil tes.

```
<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>
```

Bagian laporan baik di `awsiotdevicetester_report.xml` maupun *`suite-name`*`_report.xml` mendaftar tes yang dijalankan dan hasilnya.

Tag XML pertama `<testsuites>` berisi ringkasan pelaksanaan tes. Misalnya:

```
<testsuites name="MyTestSuite results" time="2299" tests="28" failures="0" errors="0"
 disabled="0">
```

Atribut yang digunakan dalam **`<testsuites>`** tag

`name`

Nama rangkaian tes.

`time`

Waktu, dalam hitungan detik, yang dibutuhkannya untuk menjalankan rangkaian tes.

`tests`

Jumlah tes yang dilaksanakan.

`failures`

Jumlah tes yang dijalankan, tetapi tidak lulus.

`errors`

Jumlah tes yang tidak dapat dilaksanakan oelh IDT.

`disabled`

Atribut ini tidak digunakan dan bisa diabaikan.

Jika pengujian gagal atau salah, Anda dapat mengidentifikasi pengujian yang gagal dengan meninjau tanda XML `<testsuites>`. Tag XML `<testsuite>` di dalam tag `<testsuites>` menunjukkan ringkasan hasil tes untuk grup uji. Misalnya:

```
<testsuite name="combination" package="" tests="1" failures="0" time="161" disabled="0"
 errors="0" skipped="0">
```

Format ini serupa dengan tanda `<testsuites>`, tetapi dengan atribut `skipped` yang tidak digunakan dan dapat diabaikan. Di dalam masing-masing tanda XML `<testsuite>`, terdapat tanda `<testcase>` untuk setiap tes yang dieksekusi untuk suatu grup uji. Misalnya:

```
<testcase classname="Security Test" name="IP Change Tests" attempts="1"></testcase>
```

Atribut yang digunakan dalam **`<testcase>`** tag

`name`

Nama tes.

`attempts`

Berapa kali IDT mengeksekusi uji kasus.

Ketika tes gagal atau kesalahan terjadi, tag `<failure>` atau `<error>` akan ditambahkan ke tag `<testcase>` dengan informasi untuk pemecahan masalah. Misalnya:

```
<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>
```

Kirim metrik penggunaan IDT

Jika Anda memberikan AWS kredensil dengan izin yang diperlukan, AWS IoT Device Tester kumpulkan dan kirimkan metrik penggunaan ke. AWS Ini adalah fitur opt-in dan digunakan untuk meningkatkan fungsi IDT. IDT mengumpulkan informasi seperti berikut:

- ID AWS akun yang digunakan untuk menjalankan IDT
- Perintah IDT CLI yang digunakan untuk menjalankan tes
- Rangkaian tes yang dijalankan
- Suite pengujian di *<device-tester-extract-location>* folder
- Jumlah perangkat yang dikonfigurasi dalam kolam perangkat
- Nama uji kasus dan waktu aktif
- Informasi hasil tes, seperti apakah tes berhasil dilalui, gagal, mengalami kesalahan, atau dilewati
- Fitur produk yang diuji
- Perilaku keluar IDT, seperti keluar tak terduga atau lebih awal

Semua informasi yang dikirimkan IDT juga dicatat pada file `metrics.log` di folder *<device-tester-extract-location>*`/results/`*<execution-id>*`/`. Anda dapat melihat file log untuk melihat informasi yang dikumpulkan ketika tes dijalankan. File ini dibuat hanya jika Anda memilih untuk mengumpulkan metrik penggunaan.

Untuk menonaktifkan pengumpulam metrik, Anda tidak perlu melakukan tindakan tambahan. Cukup jangan menyimpan AWS kredensil Anda, dan jika Anda memiliki AWS kredensil yang disimpan, jangan konfigurasikan `config.json` file untuk mengaksesnya.

Mendaftar untuk Akun AWS

Jika Anda tidak memiliki Akun AWS, selesaikan langkah-langkah berikut untuk membuatnya.

Untuk mendaftar untuk Akun AWS

1. Buka https://portal.aws.amazon.com/billing/pendaftaran.
2. Ikuti petunjuk online.

   Bagian dari prosedur pendaftaran melibatkan menerima panggilan telepon atau pesan teks dan memasukkan kode verifikasi pada keypad telepon.

Saat Anda mendaftar untuk sebuah Akun AWS, sebuah Pengguna root akun AWSdibuat. Pengguna root memiliki akses ke semua Layanan AWS dan sumber daya di akun. Sebagai praktik keamanan terbaik, tetapkan akses administratif ke pengguna, dan gunakan hanya pengguna root untuk melakukan tugas yang memerlukan akses pengguna root.

AWS mengirimi Anda email konfirmasi setelah proses pendaftaran selesai. Kapan saja, Anda dapat melihat aktivitas akun Anda saat ini dan mengelola akun Anda dengan masuk https://aws.amazon.comke/ dan memilih Akun Saya.

Buat pengguna dengan akses administratif

Setelah Anda mendaftar Akun AWS, amankan Pengguna root akun AWS, aktifkan AWS IAM Identity Center, dan buat pengguna administratif sehingga Anda tidak menggunakan pengguna root untuk tugas sehari-hari.

Amankan Anda Pengguna root akun AWS

1. Masuk ke AWS Management Consolesebagai pemilik akun dengan memilih pengguna Root dan memasukkan alamat Akun AWS email Anda. Di laman berikutnya, masukkan kata sandi.

   Untuk bantuan masuk dengan menggunakan pengguna root, lihat Masuk sebagai pengguna root di AWS Sign-In Panduan Pengguna.
2. Mengaktifkan autentikasi multi-faktor (MFA) untuk pengguna root Anda.

   Untuk petunjuk, lihat Mengaktifkan perangkat MFA virtual untuk pengguna Akun AWS root (konsol) Anda di Panduan Pengguna IAM.

Buat pengguna dengan akses administratif

1. Aktifkan Pusat Identitas IAM.

   Untuk mendapatkan petunjuk, silakan lihat Mengaktifkan AWS IAM Identity Center di Panduan Pengguna AWS IAM Identity Center .
2. Di Pusat Identitas IAM, berikan akses administratif ke pengguna.

   Untuk tutorial tentang menggunakan Direktori Pusat Identitas IAM sebagai sumber identitas Anda, lihat Mengkonfigurasi akses pengguna dengan default Direktori Pusat Identitas IAM di Panduan AWS IAM Identity Center Pengguna.

Masuk sebagai pengguna dengan akses administratif

- Untuk masuk dengan pengguna Pusat Identitas IAM, gunakan URL masuk yang dikirim ke alamat email saat Anda membuat pengguna Pusat Identitas IAM.

  Untuk bantuan masuk menggunakan pengguna Pusat Identitas IAM, lihat Masuk ke portal AWS akses di Panduan AWS Sign-In Pengguna.

Tetapkan akses ke pengguna tambahan

1. Di Pusat Identitas IAM, buat set izin yang mengikuti praktik terbaik menerapkan izin hak istimewa paling sedikit.

   Untuk petunjuknya, lihat Membuat set izin di Panduan AWS IAM Identity Center Pengguna.
2. Tetapkan pengguna ke grup, lalu tetapkan akses masuk tunggal ke grup.

   Untuk petunjuk, lihat Menambahkan grup di Panduan AWS IAM Identity Center Pengguna.

Untuk memberikan akses dan menambahkan izin bagi pengguna, grup, atau peran Anda:

- Pengguna dan grup di AWS IAM Identity Center:

  Buat rangkaian izin. Ikuti instruksi di Buat rangkaian izin di Panduan Pengguna AWS IAM Identity Center .
- Pengguna yang dikelola di IAM melalui penyedia identitas:

  Buat peran untuk federasi identitas. Ikuti instruksi dalam Buat peran untuk penyedia identitas pihak ketiga (federasi) dalam Panduan Pengguna IAM.
- Pengguna IAM:
  - Buat peran yang dapat diambil pengguna Anda. Ikuti instruksi dalam Buat peran untuk pengguna IAM dalam Panduan Pengguna IAM.
  - (Tidak disarankan) Lampirkan kebijakan langsung ke pengguna atau tambahkan pengguna ke grup pengguna. Ikuti petunjuk dalam Menambahkan izin ke pengguna (konsol) dalam Panduan Pengguna IAM.

Memberikan AWS kredensi ke IDT

Untuk mengizinkan IDT mengakses AWS kredensi Anda dan mengirimkan metrik AWS, lakukan hal berikut:

1. Menyimpan AWS kredensi untuk pengguna IAM Anda sebagai variabel lingkungan atau dalam file kredensial:

   a. Untuk menggunakan variabel lingkungan, jalankan perintah berikut:

   ```
   AWS_ACCESS_KEY_ID=access-key
   AWS_SECRET_ACCESS_KEY=secret-access-key
   ```

   b. Untuk menggunakan file kredensial, tambahkan informasi berikut ke `.aws/credentials file:`

   ```
   [profile-name]
   aws_access_key_id=access-key
   aws_secret_access_key=secret-access-key
   ```

2. Konfigurasikan bagian `auth` dari file `config.json`. Lihat informasi yang lebih lengkap di (Opsional) Konfigurasikan config.json.

Pertahankan versi test suite

IDT untuk FreeRTOS mengatur sumber daya pengujian ke dalam rangkaian pengujian dan grup pengujian:

- Rangkaian pengujian adalah kumpulan grup pengujian yang digunakan untuk memverifikasi bahwa perangkat berfungsi dengan versi FreeRTOS tertentu.
- Kelompok uji adalah serangkaian tes individu yang terkait dengan fitur tertentu, seperti pesan BLE dan MQTT.

Mulai dari IDT v3.0.0, rangkaian pengujian diversi menggunakan file. `major minor`. `patch`format mulai dari 1.0.0. Ketika Anda mengunduh IDT, paket termasuk versi test suite terbaru.

Saat Anda memulai IDT di antarmuka baris perintah, IDT memeriksa apakah versi suite pengujian yang lebih baru tersedia. Jika demikian, itu meminta Anda untuk memperbarui ke versi baru. Anda dapat memilih untuk memperbarui atau melanjutkan pengujian Anda saat ini.

> **Note**
>
> IDT mendukung tiga versi test suite terbaru untuk kualifikasi. Untuk informasi selengkapnya, lihat Memahami kebijakan dukungan untuk AWS IoT Device Tester.

Anda dapat mengunduh suite pengujian dengan menggunakan `upgrade-test-suite` perintah. Atau, Anda dapat menggunakan parameter opsional `-upgrade-test-suite` *`flag`* ketika Anda memulai IDT di mana *`flag`* dapat 'y' untuk selalu mengunduh versi terbaru, atau 'n' untuk menggunakan versi yang ada.

Anda juga dapat menjalankan `list-supported-versions` perintah untuk mencantumkan versi FreeRTOS dan test suite yang didukung oleh versi IDT saat ini.

Tes baru mungkin memperkenalkan pengaturan konfigurasi IDT baru. Jika pengaturannya opsional, IDT memberi tahu Anda dan terus menjalankan pengujian. Jika pengaturan diperlukan, IDT memberitahu Anda dan berhenti berjalan. Setelah Anda mengkonfigurasi pengaturan, Anda dapat terus menjalankan tes.

Memecahkan masalah kesalahan

Setiap eksekusi test suite memiliki ID eksekusi unik yang digunakan untuk membuat folder bernama `results/`*`execution-id`* dalam `results` direktori. Log grup pengujian individu berada di bawah `results/`*`execution-id`*`/logs` direktori. Gunakan keluaran konsol IDT untuk FreeRTOS untuk menemukan id eksekusi, id kasus uji, dan id grup uji dari kasus uji yang gagal lalu buka file log untuk kasus uji yang bernama. `results/`*`execution-id`*`/logs/`*`test_group_id`*`__`*`test_case_id`*`.log` Informasi dalam file ini meliputi:

- Output perintah build dan flash penuh.
- Uji output eksekusi.
- IDT lebih verbose untuk keluaran konsol FreeRTOS.

Kami merekomendasikan alur kerja berikut untuk pemecahan masalah:

1. Jika Anda melihat kesalahan "tidak *`user/role`* diizinkan untuk mengakses sumber daya ini", pastikan Anda mengonfigurasi izin seperti yang ditentukan dalamBuat dan konfigurasikan AWS akun.

2. Baca output konsol untuk menemukan informasi, seperti eksekusi UUID dan tugas yang sedang dijalankan.
3. Lihat di `FRQ_Report.xml` file untuk pernyataan kesalahan dari setiap pengujian. Direktori ini berisi log eksekusi dari setiap kelompok uji.
4. Lihat di file log di bawah`/results/`*`execution-id`*`/logs`.
5. Selidiki salah satu bidang masalah berikut:

   - Konfigurasi perangkat, seperti file konfigurasi JSON di `/configs/` folder.
   - Antarmuka perangkat. Periksa log untuk menentukan antarmuka mana yang gagal.
   - Perkakas perangkat. Pastikan bahwa rantai alat untuk membangun dan mem-flash perangkat diinstal dan dikonfigurasi dengan benar.
   - Untuk FRQ 1.xx pastikan bahwa versi kode sumber FreeRTOS yang bersih dan kloning tersedia. Rilis FreeRTOS ditandai sesuai dengan versi FreeRTOS. Untuk mengkloning versi kode tertentu, gunakan perintah berikut:

     ```
     git clone --branch version-number https://github.com/aws/amazon-freertos.git
     cd amazon-freertos
     git submodule update --checkout --init --recursive
     ```

Memecahkan masalah konfigurasi perangkat

Ketika Anda menggunakan IDT untuk FreeRTOS, Anda harus mendapatkan file konfigurasi yang benar di tempat sebelum Anda menjalankan biner. Jika Anda mendapatkan kesalahan penguraian dan konfigurasi, langkah pertama Anda adalah menemukan dan menggunakan templat konfigurasi yang sesuai dengan lingkungan Anda. Template ini terletak di *`IDT_ROOT`*`/configs` direktori.

Jika Anda masih mengalami masalah, lihat proses debugging berikut.

Di mana saya melihat?

Mulailah dengan membaca output konsol untuk menemukan informasi, seperti eksekusi UUID, yang direferensikan seperti `execution-id` dalam dokumentasi ini.

Selanjutnya, lihat `FRQ_Report.xml` file di `/results/`*`execution-id`* direktori.
File ini berisi semua kasus uji yang dijalankan dan cuplikan kesalahan untuk setiap

kegagalan. Untuk mendapatkan semua log eksekusi, cari file `/results/`*`execution-id`*`/logs/`*`test_group_id`*`__`*`test_case_id`*`.log` untuk setiap kasus uji.

Kode kesalahan IDT

Tabel berikut menjelaskan kode kesalahan yang dihasilkan oleh IDT untuk FreeRTOS:

Kode Kesalahan	Nama Kode Kesalahan	Kemungkinan Akar Penyebab	Pemecahan Masalah
201	InvalidInputError	Bidang dalam `device.json` on `config.json` ,, atau `userdata.json` hilang atau dalam format yang salah.	Pastikan bidang yang diperlukan tidak hilang dan dalam format wajib dalam file yang terdaftar. Untuk informasi selengkapnya, lihat Tes pertama papan mikrokontroler Anda.
202	ValidationError	Bidang di`device.json` on ,`config.json` , atau `userdata.json` berisi nilai yang tidak valid.	Periksa pesan kesalahan di sisi kanan kode kesalahan dalam laporan: • AWS Wilayah Tidak Valid - Tentukan AWS wilayah yang valid di file Anda`config.json` on . Untuk informasi selengkapnya tentang AWS wilayah, lihat Wilayah dan Titik Akhir.

Kode Kesalahan	Nama Kode Kesalahan	Kemungkinan Akar Penyebab	Pemecahan Masalah
			• Kredensyal tidak valid - Tetapkan AWS kredensyal yang valid di mesin uji Anda (melalui variabel lingkungan atau file AWS kredensyal). Verifikasi bahwa bidang otentikasi dikonfigurasi dengan benar. Untuk informasi selengkapnya, lihat Buat dan konfigurasikan AWS akun.

Kode Kesalahan	Nama Kode Kesalahan	Kemungkinan Akar Penyebab	Pemecahan Masalah
203	CopySourceCodeError	Tidak dapat menyalin kode sumber FreeRTOS ke direktori tertentu.	Verifikasi item berikut: • Periksa valid `sourcePath` ditentukan dalam `userdata.json` file Anda. • Hapus `build` folder di bawah direktori kode sumber FreeRTOS, jika ada. Untuk informasi selengkapnya, lihat Konfigurasikan pengaturan build, flash, dan uji. • Windows memiliki batas karakter untuk nama jalur file. Nama jalur file yang panjang akan menyebabkan kesalahan.

Kode Kesalahan	Nama Kode Kesalahan	Kemungkinan Akar Penyebab	Pemecahan Masalah
204	BuildSourceError	Tidak dapat mengkompilasi kode sumber FreeRTOS.	Verifikasi item berikut: • Periksa apakah informasi di bawah `buildTool` dalam `userdata.json` file Anda sudah benar. • Jika Anda menggunakan `cmake` sebagai alat build, pastikan yang `{{enableTests}}` ditentukan dalam `buildTool` perintah. Untuk informasi selengkapnya, lihat Konfigurasikan pengaturan build, flash, dan uji. • Jika Anda telah mengekstrak IDT untuk FreeRTOS ke jalur file di sistem Anda yang berisi spasi, misalnya`C:\Users\My Name\Desktop\`, Anda mungkin memerlukan tanda kutip tambahan di dalam perintah

Kode Kesalahan	Nama Kode Kesalahan	Kemungkinan Akar Penyebab	Pemecahan Masalah
			build untuk memastikan jalur diuraikan dengan benar. Hal yang sama mungkin diperlukan untuk perintah flash Anda.
205	FlashOrRunTestError	IDT FreeRTOS tidak dapat mem-flash atau menjalankan FreeRTOS di DUT Anda.	Verifikasi informasi `flashTool` di bawah dalam `userdata.json` file Anda benar. Untuk informasi selengkapnya, lihat Konfigurasikan pengaturan build, flash, dan uji.
206	StartEchoServerError	IDT FreeRTOS tidak dapat memulai server gema untuk tes soket atau aman. WiFi	Verifikasi port yang dikonfigurasi `echoServerConfiguration` di bawah `userdata.json` file Anda tidak digunakan atau diblokir oleh firewall atau pengaturan jaringan.

Kesalahan penguraian file konfigurasi debug

Kadang-kadang, eror ketik dalam konfigurasi JSON dapat menyebabkan penguraian eror. Sebagian besar waktu, masalahnya adalah akibat dari menghilangkan tanda kurung, koma, atau kutipan dari file JSON Anda. IDT untuk FreeRTOS melakukan validasi JSON dan mencetak informasi debugging.

IDT mencetak garis di mana kesalahan terjadi, nomor baris, dan nomor kolom kesalahan sintaks. Informasi ini seharusnya cukup untuk membantu Anda memperbaiki kesalahan, tetapi jika Anda masih mengalami masalah dalam menemukan kesalahan, Anda dapat melakukan validasi secara manual di IDE Anda, editor teks seperti Atom atau Sublime, atau melalui alat online seperti. JSONLint

Kesalahan penguraian hasil tes debug

Saat menjalankan grup pengujian dari FreeRTOS-Libraries-Integration-Tests, seperti FullTransportInterfaceTLS, PKCS11 _Core Penuh, _onboard_ECC Penuh, PKCS11 _onboard_RSA Penuh, Penuh _ _ECC, Penuh _ PKCS11 PKCS11 _RSA, atauPreProvisioned, IDT untuk FreeRTOS mem-parsing hasil pengujian dari PKCS11 perangkat uji dengan PreProvisioned koneksi serial. OTACore Terkadang, output serial tambahan pada perangkat dapat mengganggu penguraian hasil pengujian.

Dalam kasus yang disebutkan di atas, alasan kegagalan kasus uji aneh seperti string yang berasal dari output perangkat yang tidak terkait adalah output. File log kasus uji IDT untuk FreeRTOS (yang mencakup semua IDT keluaran serial untuk FreeRTOS telah diterima selama pengujian) dapat menunjukkan hal berikut:

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

Dalam contoh di atas, output perangkat yang tidak terkait mencegah IDT untuk FreeRTOS mendeteksi hasil pengujian yang LULUS.

Periksa hal berikut untuk memastikan pengujian yang optimal.

- Pastikan makro logging yang digunakan pada perangkat aman untuk utas. Lihat Menerapkan makro logging pustaka untuk informasi selengkapnya.
- Pastikan ada output minimal ke koneksi serial selama pengujian. Output perangkat lain dapat menjadi masalah bahkan jika makro logging Anda benar-benar aman untuk thread, karena hasil pengujian akan ditampilkan dalam panggilan terpisah selama pengujian.

Log kasus uji IDT untuk FreeRTOS idealnya akan menampilkan hasil tes yang tidak terputus seperti di bawah ini:

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

Kegagalan pemeriksaan integritas debug

Jika menggunakan FreeRTOS versi FRQ 1.xx, pemeriksaan integritas berikut berlaku.

Ketika Anda menjalankan grup RTOSIntegrity uji Gratis dan Anda mengalami kegagalan, pertama-tama pastikan bahwa Anda belum memodifikasi file *freertos* direktori apa pun. Jika Anda belum, dan masih melihat masalah, pastikan Anda menggunakan cabang yang benar. Jika Anda menjalankan `list-supported-products` perintah IDT, Anda dapat menemukan cabang *freertos* repo yang diberi tag mana yang harus Anda gunakan.

Jika Anda mengkloning cabang `freertos` repo yang diberi tag yang benar dan masih mengalami masalah, pastikan Anda juga menjalankan perintah. `submodule update` Alur kerja klon untuk `freertos` repo adalah sebagai berikut.

```
git clone --branch version-number https://github.com/aws/amazon-freertos.git
cd amazon-freertos
git submodule update --checkout —init —recursive
```

Daftar file yang dicari pemeriksa integritas ada di `checksums.json` file di *freertos* direktori Anda. Untuk memenuhi syarat port FreeRTOS tanpa modifikasi apa pun pada file dan struktur folder, pastikan tidak ada file yang tercantum di bagian " dan `exhaustive minimal` " file yang telah dimodifikasi. `checksums.json` Untuk menjalankan dengan SDK yang dikonfigurasi, verifikasi bahwa tidak ada file di bawah bagian '`minimal`' yang telah dimodifikasi.

Jika Anda menjalankan IDT dengan SDK dan telah memodifikasi beberapa file di *freertos* direktori Anda, pastikan Anda mengonfigurasi SDK dengan benar di file Anda. `userdata` Jika tidak, pemeriksa Integritas akan memverifikasi semua file di *freertos* direktori.

Kegagalan grup FullWiFi uji debug

Jika Anda menggunakan FRQ 1.xx dan mengalami kegagalan dalam grup FullWiFi pengujian, dan pengujian "`AFQP_WiFiConnectMultipleAP`" gagal, ini bisa jadi karena kedua titik akses tidak berada dalam subnet yang sama dengan komputer host yang menjalankan IDT. Pastikan bahwa kedua titik akses berada di subnet yang sama dengan komputer host yang menjalankan IDT.

Debug kesalahan “parameter yang diperlukan hilang”

Karena fitur baru ditambahkan ke IDT untuk FreeRTOS, perubahan pada file konfigurasi mungkin diperkenalkan. Penggunaan file konfigurasi lama mungkin akan merusak konfigurasi Anda. Jika ini terjadi, *`test_group_id`*`__`*`test_case_id`*`.log` file di bawah `results/`*`execution-id`*`/logs` direktori secara eksplisit mencantumkan semua parameter yang hilang. IDT untuk FreeRTOS memvalidasi skema file konfigurasi JSON Anda untuk memastikan bahwa versi terbaru yang didukung telah digunakan.

Debug kesalahan “tes tidak dapat memulai”

Anda mungkin melihat kesalahan yang menunjukkan kegagalan selama pengujian dimulai. Karena ada beberapa kemungkinan penyebab, periksa area berikut untuk kebenaran:

- Pastikan bahwa nama pool yang Anda sertakan dalam perintah eksekusi Anda benar-benar ada. Ini direferensikan langsung dari `device.json` file Anda.
- Pastikan perangkat atau perangkat di kolam Anda memiliki parameter konfigurasi yang benar.

Debug kesalahan “tidak dapat menemukan awal hasil pengujian”

Anda mungkin melihat kesalahan saat IDT mencoba mengurai hasil output oleh perangkat yang sedang diuji. Ada beberapa kemungkinan penyebab, jadi periksa area berikut untuk kebenaran:

- Pastikan perangkat yang diuji memiliki koneksi yang stabil ke mesin host Anda. Anda dapat memeriksa file log untuk pengujian yang menunjukkan kesalahan ini untuk melihat apa yang diterima IDT.
- Jika menggunakan FRQ 1.xx, dan perangkat yang diuji terhubung melalui jaringan lambat atau antarmuka lain, atau Anda tidak melihat tanda “---------STARTING TESTS---------” di log grup uji FreeRTOS bersama dengan keluaran grup uji FreeRTOS lainnya, Anda dapat mencoba meningkatkan nilai konfigurasi data pengguna Anda. `testStartDelayms` Untuk informasi selengkapnya, lihat Konfigurasikan pengaturan build, flash, dan uji.

Debug kesalahan “Uji kegagalan: diharapkan __ hasil tetapi melihat ___” kesalahan

Anda mungkin melihat kesalahan yang mengarah ke kegagalan pengujian selama pengujian. Tes mengharapkan sejumlah hasil tertentu, dan tidak melihatnya selama pengujian. Beberapa tes FreeRTOS berjalan sebelum IDT melihat output dari perangkat. Jika Anda melihat kesalahan ini, Anda dapat mencoba meningkatkan nilai `testStartDelayms` dalam konfigurasi data pengguna Anda. Untuk informasi selengkapnya, lihat Konfigurasikan pengaturan build, flash, dan uji.

Debug kesalahan “_____ tidak dipilih karena kendala” ConditionalTests

Ini berarti Anda menjalankan pengujian pada kumpulan perangkat yang tidak kompatibel dengan pengujian. Ini mungkin terjadi dengan tes OTA E2E. Misalnya, saat menjalankan grup `OTADataplaneMQTT` pengujian dan di file `device.json` konfigurasi Anda, Anda telah memilih OTA sebagai Tidak atau `OTADataPlaneProtocol` sebagai HTTP. Kelompok uji yang dipilih untuk dijalankan harus sesuai dengan pilihan `device.json` kemampuan Anda.

Debug batas waktu IDT selama pemantauan keluaran perangkat

IDT dapat timeout karena sejumlah alasan. Jika batas waktu terjadi selama fase pemantauan keluaran perangkat pengujian, dan Anda dapat melihat hasil di dalam log kasus uji IDT, itu berarti hasilnya salah diuraikan oleh IDT. Salah satu alasannya adalah pesan log yang disisipkan di tengah hasil tes. Jika ini masalahnya, silakan merujuk ke Panduan Porting FreerTOS untuk detail lebih lanjut tentang bagaimana log UNITY harus disiapkan.

Alasan lain untuk batas waktu selama pemantauan keluaran perangkat bisa jadi perangkat reboot setelah kegagalan kasus uji TLS tunggal. Perangkat kemudian menjalankan gambar yang di-flash dan menyebabkan loop tak terbatas yang terlihat di log. Jika ini terjadi, pastikan perangkat Anda tidak reboot setelah kegagalan pengujian.

Debug kesalahan “tidak diizinkan untuk mengakses sumber daya”

Anda mungkin melihat kesalahan "tidak *`user/role`* diizinkan untuk mengakses sumber daya ini” di output terminal atau di `test_manager.log` file di bawah`/results/`*`execution-id`*`/logs`. Untuk mengatasi masalah ini, lampirkan `AWS IoTDeviceTesterForFreeRTOSFullAccess` kebijakan terkelola untuk pengguna pengetesan Anda. Untuk informasi selengkapnya, lihat Buat dan konfigurasikan AWS akun.

Kesalahan pengujian jaringan debug

Untuk pengujian berbasis jaringan, IDT memulai server gema yang mengikat ke port yang tidak dicadangkan pada mesin host. Jika Anda mengalami kesalahan karena batas waktu atau koneksi yang tidak tersedia dalam pengujian soket WiFi atau aman, pastikan jaringan Anda dikonfigurasi untuk memungkinkan lalu lintas ke port yang dikonfigurasi dalam rentang 1024 - 49151.

Tes soket aman menggunakan port 33333 dan 33334 secara default. WiFi Tes menggunakan port 33335 secara default. Jika ketiga port ini sedang digunakan atau diblokir oleh firewall atau jaringan, Anda dapat memilih untuk menggunakan port yang berbeda di userdata.json untuk pengujian. Untuk informasi selengkapnya, lihat Konfigurasikan pengaturan build, flash, dan uji. Anda dapat menggunakan perintah berikut untuk memeriksa apakah port tertentu sedang digunakan:

- Windows: `netsh advfirewall firewall show rule name=all | grep port`
- Linux: `sudo netstat -pan | grep port`
- macOS: `netstat -nat | grep port`

Kegagalan pembaruan OTA karena payload versi yang sama

Jika kasus pengujian OTA gagal karena versi yang sama ada di perangkat setelah OTA dilakukan, itu mungkin karena sistem build Anda (misalnya cmake) tidak memperhatikan perubahan IDT pada kode sumber FreeRTOS dan tidak membangun biner yang diperbarui. Hal ini menyebabkan OTA dilakukan dengan biner yang sama yang saat ini ada di perangkat, dan pengujian gagal. Untuk mengatasi masalah kegagalan pembaruan OTA, mulailah dengan memastikan bahwa Anda menggunakan versi terbaru yang didukung dari sistem build Anda.

Kegagalan uji OTA pada kasus **`PresignedUrlExpired`** uji

Salah satu prasyarat dari tes ini adalah bahwa waktu pembaruan OTA harus lebih dari 60 detik, jika tidak pengujian akan gagal. Jika ini terjadi, pesan kesalahan berikut ditemukan di log: “Tes membutuhkan waktu kurang dari 60 detik (waktu kedaluwarsa url) untuk selesai. Silakan hubungi kami.”

Debug antarmuka perangkat dan kesalahan port

Bagian ini berisi informasi tentang antarmuka perangkat yang digunakan IDT untuk terhubung ke perangkat Anda.

Platform yang didukung

IDT mendukung Linux, macOS, dan Windows. Ketiga platform memiliki skema penamaan yang berbeda untuk perangkat serial yang melekat padanya:

- Linux: `/dev/tty*`
- macOS: atau `/dev/tty.*` `/dev/cu.*`
- Jendela: COM*

Untuk memeriksa port perangkat Anda:

- Untuk Linux/macOS, buka terminal dan jalankan. `ls /dev/tty*`
- Untuk macOS, buka terminal dan jalankan `ls /dev/tty.*` atau. `ls /dev/cu.*`
- Untuk Windows, buka Device Manager dan perluas grup perangkat serial.

Untuk memverifikasi perangkat mana yang terhubung ke port:

- Untuk Linux, pastikan bahwa `udev` paket diinstal, dan kemudian jalankan`udevadm info --name=`*`PORT`*. Utilitas ini mencetak informasi driver perangkat yang membantu Anda memverifikasi bahwa Anda menggunakan port yang benar.
- Untuk macOS, buka Launchpad dan cari. **`System Information`**
- Untuk Windows, buka Device Manager dan perluas grup perangkat serial.

Antarmuka perangkat

Setiap perangkat yang disematkan berbeda, yang berarti mereka dapat memiliki satu atau lebih port serial. Adalah umum bagi perangkat untuk memiliki dua port saat terhubung ke mesin:

- Port data untuk mem-flash perangkat.
- Port baca untuk membaca output.

  Anda harus mengatur port baca yang benar di `device.json` file Anda. Jika tidak, membaca output dari perangkat mungkin gagal.

  Dalam kasus beberapa port, pastikan untuk menggunakan port baca perangkat di `device.json` file Anda. Misalnya, jika Anda mencolokkan WRover perangkat Espressif dan dua port yang

ditetapkan padanya adalah `/dev/ttyUSB0` dan`/dev/ttyUSB1`, gunakan `/dev/ttyUSB1` dalam file Anda`device.json`.

Untuk Windows, ikuti logika yang sama.

Membaca data perangkat

IDT untuk FreeRTOS menggunakan build perangkat individual dan perkakas flash untuk menentukan konfigurasi port. Jika Anda menguji perangkat Anda dan tidak mendapatkan output, coba pengaturan default berikut:

- Tingkat baud: 115200
- Bit data: 8
- Paritas: Tidak ada
- Hentikan bit: 1
- Kontrol aliran: Tidak ada

Pengaturan ini ditangani oleh IDT untuk FreeRTOS. Anda tidak harus mengaturnya. Namun, Anda dapat menggunakan metode yang sama untuk membaca output perangkat secara manual. Di Linux atau macOS, Anda dapat melakukan ini dengan perintah. `screen` Di Windows, Anda dapat menggunakan program seperti TeraTerm.

```
Screen: screen /dev/cu.usbserial 115200
```

```
TeraTerm: Use the above-provided settings to set the fields explicitly in
the GUI.
```

Masalah rantai alat pengembangan

Bagian ini membahas masalah yang dapat terjadi dengan toolchain Anda.

Code Composer Studio di Ubuntu

Versi Ubuntu yang lebih baru (17.10 dan 18.04) memiliki versi `glibc` paket yang tidak kompatibel dengan Code Composer Studio 7. x versi. Kami menyarankan Anda menginstal Code Composer Studio versi 8.2 atau yang lebih baru.

Gejala ketidakcocokan mungkin termasuk:

- FreeRTOS gagal membangun atau mem-flash ke perangkat Anda.
- Penginstal Code Composer Studio mungkin membeku.
- Tidak ada output log yang ditampilkan di konsol selama proses build atau flash.
- Build command mencoba untuk meluncurkan dalam mode GUI bahkan ketika dipanggil sebagai headless.

Pencatatan log

IDT untuk log FreeRTOS ditempatkan di satu lokasi. Dari direktori root IDT, file-file ini tersedia di bawah`results/`*`execution-id`*`/`:

- `FRQ_Report.xml`
- `awsiotdevicetester_report.xml`
- `logs/`*`test_group_id`*`__`*`test_case_id`*`.log`

`FRQ_Report.xml`dan `logs/`*`test_group_id`*`__`*`test_case_id`*`.log` merupakan log yang paling penting untuk diperiksa. `FRQ_Report.xml`berisi informasi tentang kasus uji mana yang gagal dengan pesan kesalahan tertentu. Anda kemudian dapat menggunakan `logs/`*`test_group_id`*`__`*`test_case_id`*`.log` untuk menggali lebih jauh ke dalam masalah untuk mendapatkan konteks yang lebih baik.

Kesalahan konsol

Saat AWS IoT Device Tester dijalankan, kegagalan dilaporkan ke konsol dengan pesan singkat. Lihat `results/`*`execution-id`*`/logs/`*`test_group_id`*`__`*`test_case_id`*`.log` untuk mempelajari lebih lanjut tentang kesalahan tersebut.

Kesalahan log

Setiap eksekusi test suite memiliki ID eksekusi unik yang digunakan untuk membuat folder bernama`results/`*`execution-id`*. Log kasus uji individu berada di bawah `results/`*`execution-id`*`/logs` direktori. Gunakan output konsol IDT untuk FreeRTOS untuk menemukan id eksekusi, id kasus uji, dan id grup uji kasus uji yang gagal. Kemudian gunakan informasi ini untuk menemukan dan membuka file log untuk kasus uji yang bernama `results/`*`execution-id`*`/logs/`*`test_group_id`*`__`*`test_case_id`*`.log` Informasi dalam file ini mencakup output perintah build dan flash penuh, output eksekusi pengujian, dan lebih banyak keluaran AWS IoT Device Tester konsol verbose.

Masalah bucket S3

Jika Anda menekan CTRL+C saat menjalankan IDT, IDT akan memulai proses pembersihan. Bagian dari pembersihan itu adalah menghapus sumber daya Amazon S3 yang telah dibuat sebagai bagian dari pengujian IDT. Jika pembersihan tidak dapat diselesaikan, Anda mungkin mengalami masalah di mana Anda memiliki terlalu banyak bucket Amazon S3 yang telah dibuat. Ini berarti lain kali Anda menjalankan IDT pengujian akan mulai gagal.

Jika Anda menekan CTRL+C untuk menghentikan IDT, Anda harus membiarkannya menyelesaikan proses pembersihan untuk menghindari masalah ini. Anda juga dapat menghapus bucket Amazon S3 dari akun Anda yang dibuat secara manual.

Memecahkan masalah kesalahan batas waktu

Jika Anda melihat kesalahan batas waktu saat menjalankan rangkaian pengujian, tingkatkan batas waktu dengan menentukan faktor pengganda batas waktu. Faktor ini diterapkan ke nilai batas waktu default. Nilai apa pun yang dikonfigurasi untuk bendera ini harus lebih besar dari atau sama dengan 1.0. Untuk menggunakan pengganda batas waktu, gunakan tanda `--timeout-multiplier` saat menjalankan rangkaian pengujian.

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
```

Fitur seluler dan AWS biaya

Ketika `Cellular` fitur diatur ke `Yes` dalam `device.JSON` file Anda, FullSecureSockets akan menggunakan EC2 instans t.micro untuk menjalankan pengujian dan ini dapat menimbulkan biaya tambahan ke akun Anda. AWS Untuk informasi selengkapnya, lihat EC2 harga Amazon.

Kebijakan pembuatan laporan kualifikasi

Laporan kualifikasi hanya dihasilkan oleh versi AWS IoT Device Tester (IDT) yang mendukung versi FreeRTOS yang dirilis dalam dua tahun terakhir. Jika Anda memiliki pertanyaan tentang kebijakan dukungan, silakan hubungi AWS Dukungan.

Memahami kebijakan AWS Terkelola untuk AWS IoT Device Tester

Kebijakan AWS terkelola adalah kebijakan mandiri yang dibuat dan dikelola oleh AWS. AWS Kebijakan terkelola dirancang untuk memberikan izin bagi banyak kasus penggunaan umum sehingga Anda dapat mulai menetapkan izin kepada pengguna, grup, dan peran.

Perlu diingat bahwa kebijakan AWS terkelola mungkin tidak memberikan izin hak istimewa paling sedikit untuk kasus penggunaan spesifik Anda karena tersedia untuk digunakan semua pelanggan. AWS Kami menyarankan Anda untuk mengurangi izin lebih lanjut dengan menentukan kebijakan yang dikelola pelanggan yang khusus untuk kasus penggunaan Anda.

Anda tidak dapat mengubah izin yang ditentukan dalam kebijakan AWS terkelola. Jika AWS memperbarui izin yang ditentukan dalam kebijakan AWS terkelola, pemutakhiran akan memengaruhi semua identitas utama (pengguna, grup, dan peran) yang dilampirkan kebijakan tersebut. AWS kemungkinan besar akan memperbarui kebijakan AWS terkelola saat baru Layanan AWS diluncurkan atau operasi API baru tersedia untuk layanan yang ada.

Untuk informasi selengkapnya, lihat Kebijakan terkelola AWS dalam Panduan Pengguna IAM.

Topik

- AWS kebijakan terkelola: TDevice TesterForFree RTOSFull Akses AWS Io
- Pembaruan kebijakan AWS terkelola

AWS kebijakan terkelola: TDevice TesterForFree RTOSFull Akses AWS Io

Kebijakan `AWSIoTDeviceTesterForFreeRTOSFullAccess` terkelola berisi AWS IoT Device Tester izin berikut untuk pemeriksaan versi, fitur pembaruan otomatis, dan kumpulan metrik.

Detail izin

Kebijakan ini mencakup izin berikut:

- `iot-device-tester:SupportedVersion`

Memberikan AWS IoT Device Tester izin untuk mengambil daftar produk yang didukung, rangkaian pengujian, dan versi IDT.

- `iot-device-tester:LatestIdt`

  Memberikan AWS IoT Device Tester izin untuk mengambil versi IDT terbaru yang tersedia untuk diunduh.

- `iot-device-tester:CheckVersion`

  Memberikan AWS IoT Device Tester izin untuk memeriksa kompatibilitas versi untuk IDT, suite pengujian, dan produk.

- `iot-device-tester:DownloadTestSuite`

  Memberikan AWS IoT Device Tester izin untuk mengunduh pembaruan rangkaian pengujian.

- `iot-device-tester:SendMetrics`

  Memberikan AWS izin untuk mengumpulkan metrik tentang penggunaan AWS IoT Device Tester internal.

```
{
    "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"
                    ]
                }
            }
        }
    ]
}
```

Pembaruan kebijakan AWS terkelola

Anda dapat melihat detail tentang pembaruan kebijakan AWS terkelola AWS IoT Device Tester sejak layanan ini mulai melacak perubahan ini.

Versi	Perubahan	Deskripsi	Tanggal
7 (Terbaru)	Merestrukturisasi `ec2:CreateTags` kondisi.	Menghapus penggunaa `nForAnyValues` .	6/14/2023
6	Dihapus `freertos:ListHardwarePlatforms` dari kebijakan.	Menghapus izin karena tindakan ini tidak digunakan lagi pada 1 Maret 2023.	6/2/2023

Versi	Perubahan	Deskripsi	Tanggal
5	Menambahkan izin untuk menjalank an tes server echo menggunakan. EC2	Ini untuk memulai dan menghentikan EC2 instance di AWS akun pelanggan.	12/15/2020
4	Ditambahkan `iot:Cance lJobExecution` .	Izin ini membatalkan pekerjaan OTA.	7/17/2020

Versi	Perubahan	Deskripsi	Tanggal
3	Menambahkan izin berikut: • `iot-device-tester: DownloadTestSuite` , • `iot-device-tester: CheckVersion` , • `iot-device-tester: LatestIdt` , • `iot-device-tester: SupportedVersion` .	• `iot-device-tester: DownloadTestSuite` — Memberikan AWS IoT Device Tester izin untuk mengunduh pembaruan rangkaian pengujian, • `iot-device-tester: CheckVersion` — Memberikan AWS IoT Device Tester izin untuk memeriksa kompatibilitas versi untuk IDT, rangkaian pengujian, dan produk, • `iot-device-tester: LatestIdt` — Memberikan AWS IoT Device Tester izin untuk mengambil versi IDT terbaru yang tersedia untuk diunduh,	23/03/2020

Versi	Perubahan	Deskripsi	Tanggal
		• `iot-device-tester: SupportedVersion` — Memberikan AWS IoT Device Tester izin untuk mengambil daftar produk yang didukung, suite pengujian, dan versi IDT.	
2	Menambahkan `iot-device-tester: SendMetrics` izin.	Memberikan AWS izin untuk mengumpulkan metrik tentang penggunaan AWS IoT Device Tester internal.	2/18/2020
1	Versi awal.		2/12/2020

Memahami kebijakan dukungan untuk AWS IoT Device Tester

Important

Per Oktober 2022, AWS IoT Device Tester untuk Kualifikasi AWS IoT FreerTOS (FRQ) 1.0 tidak menghasilkan laporan kualifikasi yang ditandatangani. Anda tidak dapat memenuhi syarat perangkat AWS IoT FreeRTOS baru untuk dicantumkan di AWS Katalog Perangkat Mitra melalui Program Kualifikasi Perangkat menggunakan versi AWS IDT FRQ 1.0. Meskipun Anda tidak dapat memenuhi syarat perangkat FreerTOS menggunakan IDT FRQ 1.0, Anda dapat terus menguji perangkat FreeRTOS Anda dengan FRQ 1.0. Kami menyarankan Anda menggunakan IDT FRQ 2.0 untuk memenuhi syarat dan daftar perangkat FreeRTOS di Katalog Perangkat Mitra.AWS

AWS IoT Device Tester untuk FreeRTOS adalah alat otomatisasi uji untuk memvalidasi port FreeRTOS ke perangkat. Selain itu, Anda dapat memenuhi syarat perangkat FreeRTOS Anda dan mencantumkannya di AWS Katalog Perangkat Mitra. The AWS IoT Device Tester for FreeRTOS mendukung validasi dan kualifikasi pustaka FreeRTOS Long Term Supported (LTS) yang tersedia di Freertos/Freertos-LTS, dan jalur utama FreerTOS tersedia di GitHub Freertos/FreeRTOS. Kami menyarankan Anda menggunakan versi terbaru dari kedua FreeRTOS dan FreeRTOS AWS IoT Device Tester untuk memvalidasi dan memenuhi syarat perangkat Anda.

Untuk FreerTOS-LTS, IDT mendukung validasi dan kualifikasi FreerTOS 202210 versi LTS. Lihat di sini untuk informasi lebih lanjut tentang rilis FreerTOS LTS dan garis waktu pemeliharaannya. Setelah periode dukungan rilis LTS ini berakhir, Anda masih dapat melanjutkan validasi, tetapi IDT tidak akan menghasilkan laporan, yang akan memungkinkan Anda untuk mengirimkan perangkat Anda untuk kualifikasi.

Untuk FreeRTOS arus utama yang tersedia di Freertos/FreeRTOS, kami mendukung validasi dan kualifikasi semua versi yang dirilis dalam enam bulan terakhir, atau dua versi FreeRTOS sebelumnya jika dirilis lebih dari enam bulan terpisah. Lihat di sini untuk versi yang didukung saat ini. Untuk versi FreeRTOS yang tidak didukung, Anda masih dapat melanjutkan validasi, tetapi IDT tidak akan menghasilkan laporan, yang memungkinkan Anda mengirimkan perangkat Anda untuk kualifikasi.

Lihat Versi yang didukung AWS IoT Device Tester untuk versi IDT dan FreeRTOS terbaru yang didukung. Anda dapat menggunakan salah satu versi yang didukung AWS IoT Device Tester dengan versi FreeRTOS yang sesuai untuk menguji atau memenuhi syarat perangkat Anda. Jika Anda terus menggunakanVersi IDT yang tidak didukung untuk FreeRTOS, Anda tidak akan menerima perbaikan bug atau pembaruan terbaru.

Untuk pertanyaan tentang kebijakan dukungan, hubungi AWS Customer Support.

Keamanan di AWS

Keamanan cloud di AWS adalah prioritas tertinggi. Sebagai AWS pelanggan, Anda mendapat manfaat dari pusat data dan arsitektur jaringan yang dibangun untuk memenuhi persyaratan organisasi yang paling sensitif terhadap keamanan.

Keamanan adalah tanggung jawab bersama antara Anda AWS dan Anda. Model tanggung jawab bersama menggambarkan hal ini sebagai keamanan dari cloud dan keamanan di cloud:

- Keamanan cloud — AWS bertanggung jawab untuk melindungi infrastruktur yang menjalankan AWS layanan di AWS Cloud. AWS juga memberi Anda layanan yang dapat Anda gunakan dengan aman. Efektivitas keamanan kami diuji dan diverifikasi secara rutin oleh auditor pihak ketiga sebagai bagian dari program kepatuhan AWS. Untuk mempelajari tentang program kepatuhan yang berlaku untuk AWS layanan, lihat AWS Layanan dalam Lingkup berdasarkan Program Kepatuhan.
- Keamanan di cloud — Tanggung jawab Anda ditentukan oleh AWS layanan yang Anda gunakan. Anda juga bertanggung jawab atas faktor-faktor lain termasuk sensitivitas data, kebutuhan organisasi, serta undang-undang dan peraturan yang berlaku.

Dokumentasi ini akan membantu Anda memahami cara menerapkan model tanggung jawab bersama saat menggunakan AWS. Topik berikut menunjukkan cara mengonfigurasi AWS untuk memenuhi tujuan keamanan dan kepatuhan Anda. Anda juga akan belajar cara menggunakan AWS layanan yang dapat membantu Anda memantau dan mengamankan AWS sumber daya Anda.

Untuk informasi lebih mendalam tentang AWS IoT keamanan, lihat Keamanan dan Identitas untuk AWS IoT.

Topik

- Identity and Access Management untuk FreeRTOS
- Validasi kepatuhan
- Ketahanan di AWS
- Keamanan infrastruktur di FreeRTOS

Identity and Access Management untuk FreeRTOS

AWS Identity and Access Management (IAM) adalah Layanan AWS yang membantu administrator mengontrol akses ke AWS sumber daya dengan aman. Administrator IAM mengontrol siapa yang dapat diautentikasi (masuk) dan diotorisasi (memiliki izin) untuk menggunakan sumber daya FreeRTOS. IAM adalah Layanan AWS yang dapat Anda gunakan tanpa biaya tambahan.

Topik

- Audiens
- Mengautentikasi dengan identitas
- Mengelola akses menggunakan kebijakan
- Bagaimana FreeRTOS bekerja dengan IAM
- Contoh kebijakan berbasis identitas untuk FreeRTOS
- Memecahkan masalah identitas dan akses FreeRTOS

Audiens

Bagaimana Anda menggunakan AWS Identity and Access Management (IAM) berbeda, tergantung pada pekerjaan yang Anda lakukan di FreeRTOS.

Pengguna layanan - Jika Anda menggunakan layanan FreeRTOS untuk melakukan pekerjaan Anda, maka administrator Anda memberi Anda kredensi dan izin yang Anda butuhkan. Saat Anda menggunakan lebih banyak fitur FreeRTOS untuk melakukan pekerjaan Anda, Anda mungkin memerlukan izin tambahan. Memahami cara akses dikelola dapat membantu Anda meminta izin yang tepat dari administrator Anda. Jika Anda tidak dapat mengakses fitur di FreeRTOS, lihat. Memecahkan masalah identitas dan akses FreeRTOS

Administrator layanan - Jika Anda bertanggung jawab atas sumber daya FreeRTOS di perusahaan Anda, Anda mungkin memiliki akses penuh ke FreeRTOS. Tugas Anda adalah menentukan fitur dan sumber daya FreeRTOS mana yang harus diakses pengguna layanan Anda. Kemudian, Anda harus mengirimkan permintaan kepada administrator IAM untuk mengubah izin pengguna layanan Anda. Tinjau informasi di halaman ini untuk memahami konsep dasar IAM. Untuk mempelajari lebih lanjut tentang bagaimana perusahaan Anda dapat menggunakan IAM dengan FreeRTOS, lihat. Bagaimana FreeRTOS bekerja dengan IAM

Administrator IAM — Jika Anda seorang administrator IAM, Anda mungkin ingin mempelajari detail tentang cara menulis kebijakan untuk mengelola akses ke FreeRTOS. Untuk melihat contoh kebijakan berbasis identitas FreeRTOS yang dapat Anda gunakan di IAM, lihat. Contoh kebijakan berbasis identitas untuk FreeRTOS

Mengautentikasi dengan identitas

Otentikasi adalah cara Anda masuk AWS menggunakan kredensi identitas Anda. Anda harus diautentikasi (masuk ke AWS) sebagai Pengguna root akun AWS, sebagai pengguna IAM, atau dengan mengasumsikan peran IAM.

Anda dapat masuk AWS sebagai identitas federasi dengan menggunakan kredensi yang disediakan melalui sumber identitas. AWS IAM Identity Center Pengguna (IAM Identity Center), autentikasi masuk tunggal perusahaan Anda, dan kredensi Google atau Facebook Anda adalah contoh identitas federasi. Saat Anda masuk sebagai identitas terfederasi, administrator Anda sebelumnya menyiapkan federasi identitas menggunakan peran IAM. Ketika Anda mengakses AWS dengan menggunakan federasi, Anda secara tidak langsung mengambil peran.

Bergantung pada jenis pengguna Anda, Anda dapat masuk ke AWS Management Console atau portal AWS akses. Untuk informasi selengkapnya tentang masuk AWS, lihat Cara masuk ke Panduan AWS Sign-In Pengguna Anda Akun AWS.

Jika Anda mengakses AWS secara terprogram, AWS sediakan kit pengembangan perangkat lunak (SDK) dan antarmuka baris perintah (CLI) untuk menandatangani permintaan Anda secara kriptografis dengan menggunakan kredensil Anda. Jika Anda tidak menggunakan AWS alat, Anda harus menandatangani permintaan sendiri. Guna mengetahui informasi selengkapnya tentang penggunaan metode yang disarankan untuk menandatangani permintaan sendiri, lihat AWS Signature Version 4 untuk permintaan API dalam Panduan Pengguna IAM.

Apa pun metode autentikasi yang digunakan, Anda mungkin diminta untuk menyediakan informasi keamanan tambahan. Misalnya, AWS merekomendasikan agar Anda menggunakan otentikasi multi-faktor (MFA) untuk meningkatkan keamanan akun Anda. Untuk mempelajari selengkapnya, lihat Autentikasi multi-faktor dalam Panduan Pengguna AWS IAM Identity Center dan Autentikasi multi-faktor AWS di IAM dalam Panduan Pengguna IAM.

Akun AWS pengguna root

Saat Anda membuat Akun AWS, Anda mulai dengan satu identitas masuk yang memiliki akses lengkap ke semua Layanan AWS dan sumber daya di akun. Identitas ini disebut pengguna Akun AWS root dan diakses dengan masuk dengan alamat email dan kata sandi yang Anda gunakan untuk membuat akun. Kami sangat menyarankan agar Anda tidak menggunakan pengguna root untuk tugas sehari-hari. Lindungi kredensial pengguna root Anda dan gunakan kredensial tersebut untuk melakukan tugas yang hanya dapat dilakukan pengguna root. Untuk daftar lengkap tugas

yang mengharuskan Anda masuk sebagai pengguna root, lihat Tugas yang memerlukan kredensial pengguna root dalam Panduan Pengguna IAM.

Identitas gabungan

Sebagai praktik terbaik, mewajibkan pengguna manusia, termasuk pengguna yang memerlukan akses administrator, untuk menggunakan federasi dengan penyedia identitas untuk mengakses Layanan AWS dengan menggunakan kredensi sementara.

Identitas federasi adalah pengguna dari direktori pengguna perusahaan Anda, penyedia identitas web, direktori Pusat Identitas AWS Directory Service, atau pengguna mana pun yang mengakses Layanan AWS dengan menggunakan kredensil yang disediakan melalui sumber identitas. Ketika identitas federasi mengakses Akun AWS, mereka mengambil peran, dan peran memberikan kredensi sementara.

Untuk manajemen akses terpusat, kami sarankan Anda menggunakan AWS IAM Identity Center. Anda dapat membuat pengguna dan grup di Pusat Identitas IAM, atau Anda dapat menghubungkan dan menyinkronkan ke sekumpulan pengguna dan grup di sumber identitas Anda sendiri untuk digunakan di semua aplikasi Akun AWS dan aplikasi Anda. Untuk informasi tentang Pusat Identitas IAM, lihat Apakah itu Pusat Identitas IAM? dalam Panduan Pengguna AWS IAM Identity Center .

Pengguna dan grup IAM

Pengguna IAM adalah identitas dalam diri Anda Akun AWS yang memiliki izin khusus untuk satu orang atau aplikasi. Jika memungkinkan, kami merekomendasikan untuk mengandalkan kredensial sementara, bukan membuat pengguna IAM yang memiliki kredensial jangka panjang seperti kata sandi dan kunci akses. Namun, jika Anda memiliki kasus penggunaan tertentu yang memerlukan kredensial jangka panjang dengan pengguna IAM, kami merekomendasikan Anda merotasi kunci akses. Untuk informasi selengkapnya, lihat Merotasi kunci akses secara teratur untuk kasus penggunaan yang memerlukan kredensial jangka panjang dalam Panduan Pengguna IAM.

Grup IAM adalah identitas yang menentukan sekumpulan pengguna IAM. Anda tidak dapat masuk sebagai grup. Anda dapat menggunakan grup untuk menentukan izin bagi beberapa pengguna sekaligus. Grup mempermudah manajemen izin untuk sejumlah besar pengguna sekaligus. Misalnya, Anda dapat meminta kelompok untuk menyebutkan IAMAdmins dan memberikan izin kepada grup tersebut untuk mengelola sumber daya IAM.

Pengguna berbeda dari peran. Pengguna secara unik terkait dengan satu orang atau aplikasi, tetapi peran dimaksudkan untuk dapat digunakan oleh siapa pun yang membutuhkannya. Pengguna memiliki kredensial jangka panjang permanen, tetapi peran memberikan kredensial sementara. Untuk

mempelajari selengkapnya, lihat Kasus penggunaan untuk pengguna IAM dalam Panduan Pengguna IAM.

Peran IAM

Peran IAM adalah identitas dalam diri Anda Akun AWS yang memiliki izin khusus. Peran ini mirip dengan pengguna IAM, tetapi tidak terkait dengan orang tertentu. Untuk mengambil peran IAM sementara AWS Management Console, Anda dapat beralih dari pengguna ke peran IAM (konsol). Anda dapat mengambil peran dengan memanggil operasi AWS CLI atau AWS API atau dengan menggunakan URL kustom. Untuk informasi selengkapnya tentang cara menggunakan peran, lihat Metode untuk mengambil peran dalam Panduan Pengguna IAM.

Peran IAM dengan kredensial sementara berguna dalam situasi berikut:

- Akses pengguna terfederasi – Untuk menetapkan izin ke identitas terfederasi, Anda membuat peran dan menentukan izin untuk peran tersebut. Ketika identitas terfederasi mengautentikasi, identitas tersebut terhubung dengan peran dan diberi izin yang ditentukan oleh peran. Untuk informasi tentang peran untuk federasi, lihat Buat peran untuk penyedia identitas pihak ketiga dalam Panduan Pengguna IAM. Jika menggunakan Pusat Identitas IAM, Anda harus mengonfigurasi set izin. Untuk mengontrol apa yang dapat diakses identitas Anda setelah identitas tersebut diautentikasi, Pusat Identitas IAM akan mengorelasikan set izin ke peran dalam IAM. Untuk informasi tentang set izin, lihat Set izin dalam Panduan Pengguna AWS IAM Identity Center .
- Izin pengguna IAM sementara – Pengguna atau peran IAM dapat mengambil peran IAM guna mendapatkan berbagai izin secara sementara untuk tugas tertentu.
- Akses lintas akun – Anda dapat menggunakan peran IAM untuk mengizinkan seseorang (prinsipal tepercaya) di akun lain untuk mengakses sumber daya di akun Anda. Peran adalah cara utama untuk memberikan akses lintas akun. Namun, dengan beberapa Layanan AWS, Anda dapat melampirkan kebijakan secara langsung ke sumber daya (alih-alih menggunakan peran sebagai proxy). Untuk mempelajari perbedaan antara peran dan kebijakan berbasis sumber daya untuk akses lintas akun, lihat Akses sumber daya lintas akun di IAM dalam Panduan Pengguna IAM.
- Akses lintas layanan — Beberapa Layanan AWS menggunakan fitur lain Layanan AWS. Misalnya, saat Anda melakukan panggilan dalam suatu layanan, biasanya layanan tersebut menjalankan aplikasi di Amazon EC2 atau menyimpan objek di Amazon S3. Sebuah layanan mungkin melakukannya menggunakan izin prinsipal yang memanggil, menggunakan peran layanan, atau peran terkait layanan.
  - Sesi akses teruskan (FAS) — Saat Anda menggunakan pengguna atau peran IAM untuk melakukan tindakan AWS, Anda dianggap sebagai prinsipal. Ketika Anda menggunakan

beberapa layanan, Anda mungkin melakukan sebuah tindakan yang kemudian menginisiasi tindakan lain di layanan yang berbeda. FAS menggunakan izin dari pemanggilan utama Layanan AWS, dikombinasikan dengan permintaan Layanan AWS untuk membuat permintaan ke layanan hilir. Permintaan FAS hanya dibuat ketika layanan menerima permintaan yang memerlukan interaksi dengan orang lain Layanan AWS atau sumber daya untuk menyelesaikannya. Dalam hal ini, Anda harus memiliki izin untuk melakukan kedua tindakan tersebut. Untuk detail kebijakan ketika mengajukan permintaan FAS, lihat Sesi akses maju.

- Peran layanan – Peran layanan adalah peran IAM yang dijalankan oleh layanan untuk melakukan tindakan atas nama Anda. Administrator IAM dapat membuat, mengubah, dan menghapus peran layanan dari dalam IAM. Untuk informasi selengkapnya, lihat Buat sebuah peran untuk mendelegasikan izin ke Layanan AWS dalam Panduan pengguna IAM.
- Peran terkait layanan — Peran terkait layanan adalah jenis peran layanan yang ditautkan ke peran layanan. Layanan AWS Layanan tersebut dapat menjalankan peran untuk melakukan tindakan atas nama Anda. Peran terkait layanan muncul di Anda Akun AWS dan dimiliki oleh layanan. Administrator IAM dapat melihat, tetapi tidak dapat mengedit izin untuk peran terkait layanan.

- Aplikasi yang berjalan di Amazon EC2 — Anda dapat menggunakan peran IAM untuk mengelola kredensi sementara untuk aplikasi yang berjalan pada EC2 instance dan membuat AWS CLI atau AWS permintaan API. Ini lebih baik untuk menyimpan kunci akses dalam EC2 instance. Untuk menetapkan AWS peran ke EC2 instance dan membuatnya tersedia untuk semua aplikasinya, Anda membuat profil instance yang dilampirkan ke instance. Profil instance berisi peran dan memungkinkan program yang berjalan pada EC2 instance untuk mendapatkan kredensi sementara. Untuk informasi selengkapnya, lihat Menggunakan peran IAM untuk memberikan izin ke aplikasi yang berjalan di EC2 instans Amazon di Panduan Pengguna IAM.

Mengelola akses menggunakan kebijakan

Anda mengontrol akses AWS dengan membuat kebijakan dan melampirkannya ke AWS identitas atau sumber daya. Kebijakan adalah objek AWS yang, ketika dikaitkan dengan identitas atau sumber daya, menentukan izinnya. AWS mengevaluasi kebijakan ini ketika prinsipal (pengguna, pengguna root, atau sesi peran) membuat permintaan. Izin dalam kebijakan menentukan apakah permintaan diizinkan atau ditolak. Sebagian besar kebijakan disimpan AWS sebagai dokumen JSON. Untuk informasi selengkapnya tentang struktur dan isi dokumen kebijakan JSON, lihat Gambaran umum kebijakan JSON dalam Panduan Pengguna IAM.

Administrator dapat menggunakan kebijakan AWS JSON untuk menentukan siapa yang memiliki akses ke apa. Artinya, prinsipal manakah yang dapat melakukan tindakan pada sumber daya apa, dan dengan kondisi apa.

Secara default, pengguna dan peran tidak memiliki izin. Untuk memberikan izin kepada pengguna untuk melakukan tindakan di sumber daya yang mereka perlukan, administrator IAM dapat membuat kebijakan IAM. Administrator kemudian dapat menambahkan kebijakan IAM ke peran, dan pengguna dapat mengambil peran.

Kebijakan IAM mendefinisikan izin untuk suatu tindakan terlepas dari metode yang Anda gunakan untuk melakukan operasinya. Misalnya, anggaplah Anda memiliki kebijakan yang mengizinkan tindakan `iam:GetRole`. Pengguna dengan kebijakan tersebut bisa mendapatkan informasi peran dari AWS Management Console, API AWS CLI, atau AWS API.

Kebijakan berbasis identitas

Kebijakan berbasis identitas adalah dokumen kebijakan izin JSON yang dapat Anda lampirkan ke sebuah identitas, seperti pengguna IAM, grup pengguna IAM, atau peran IAM. Kebijakan ini mengontrol jenis tindakan yang dapat dilakukan oleh pengguna dan peran, di sumber daya mana, dan berdasarkan kondisi seperti apa. Untuk mempelajari cara membuat kebijakan berbasis identitas, lihat Tentukan izin IAM kustom dengan kebijakan terkelola pelanggan dalam Panduan Pengguna IAM.

Kebijakan berbasis identitas dapat dikategorikan lebih lanjut sebagai kebijakan inline atau kebijakan yang dikelola. Kebijakan inline disematkan langsung ke satu pengguna, grup, atau peran. Kebijakan terkelola adalah kebijakan mandiri yang dapat Anda lampirkan ke beberapa pengguna, grup, dan peran dalam. Akun AWS Kebijakan AWS terkelola mencakup kebijakan terkelola dan kebijakan yang dikelola pelanggan. Untuk mempelajari cara memilih antara kebijakan yang dikelola atau kebijakan inline, lihat Pilih antara kebijakan yang dikelola dan kebijakan inline dalam Panduan Pengguna IAM.

Kebijakan berbasis sumber daya

Kebijakan berbasis sumber daya adalah dokumen kebijakan JSON yang Anda lampirkan ke sumber daya. Contoh kebijakan berbasis sumber daya adalah kebijakan kepercayaan peran IAM dan kebijakan bucket Amazon S3. Dalam layanan yang mendukung kebijakan berbasis sumber daya, administrator layanan dapat menggunakannya untuk mengontrol akses ke sumber daya tertentu. Untuk sumber daya tempat kebijakan dilampirkan, kebijakan menentukan tindakan apa yang dapat dilakukan oleh prinsipal tertentu pada sumber daya tersebut dan dalam kondisi apa. Anda harus

menentukan prinsipal dalam kebijakan berbasis sumber daya. Prinsipal dapat mencakup akun, pengguna, peran, pengguna federasi, atau. Layanan AWS

Kebijakan berbasis sumber daya merupakan kebijakan inline yang terletak di layanan tersebut. Anda tidak dapat menggunakan kebijakan AWS terkelola dari IAM dalam kebijakan berbasis sumber daya.

Daftar kontrol akses (ACLs)

Access control lists (ACLs) mengontrol prinsipal mana (anggota akun, pengguna, atau peran) yang memiliki izin untuk mengakses sumber daya. ACLs mirip dengan kebijakan berbasis sumber daya, meskipun mereka tidak menggunakan format dokumen kebijakan JSON.

Amazon S3, AWS WAF, dan Amazon VPC adalah contoh layanan yang mendukung. ACLs Untuk mempelajari selengkapnya ACLs, lihat Ringkasan daftar kontrol akses (ACL) di Panduan Pengembang Layanan Penyimpanan Sederhana Amazon.

Jenis-jenis kebijakan lain

AWS mendukung jenis kebijakan tambahan yang kurang umum. Jenis-jenis kebijakan ini dapat mengatur izin maksimum yang diberikan kepada Anda oleh jenis kebijakan yang lebih umum.

- Batasan izin – Batasan izin adalah fitur lanjutan tempat Anda mengatur izin maksimum yang dapat diberikan oleh kebijakan berbasis identitas ke entitas IAM (pengguna IAM atau peran IAM). Anda dapat menetapkan batasan izin untuk suatu entitas. Izin yang dihasilkan adalah perpotongan antara kebijakan berbasis identitas milik entitas dan batasan izinnya. Kebijakan berbasis sumber daya yang menentukan pengguna atau peran dalam bidang `Principal` tidak dibatasi oleh batasan izin. Penolakan eksplisit dalam salah satu kebijakan ini akan menggantikan pemberian izin. Untuk informasi selengkapnya tentang batasan izin, lihat Batasan izin untuk entitas IAM dalam Panduan Pengguna IAM.
- Kebijakan kontrol layanan (SCPs) — SCPs adalah kebijakan JSON yang menentukan izin maksimum untuk organisasi atau unit organisasi (OU) di. AWS Organizations AWS Organizations adalah layanan untuk mengelompokkan dan mengelola secara terpusat beberapa Akun AWS yang dimiliki bisnis Anda. Jika Anda mengaktifkan semua fitur dalam suatu organisasi, maka Anda dapat menerapkan kebijakan kontrol layanan (SCPs) ke salah satu atau semua akun Anda. SCP membatasi izin untuk entitas di akun anggota, termasuk masing-masing. Pengguna root akun AWS Untuk informasi selengkapnya tentang Organizations dan SCPs, lihat Kebijakan kontrol layanan di Panduan AWS Organizations Pengguna.
- Kebijakan kontrol sumber daya (RCPs) — RCPs adalah kebijakan JSON yang dapat Anda gunakan untuk menetapkan izin maksimum yang tersedia untuk sumber daya di akun Anda tanpa

memperbarui kebijakan IAM yang dilampirkan ke setiap sumber daya yang Anda miliki. RCP membatasi izin untuk sumber daya di akun anggota dan dapat memengaruhi izin efektif untuk identitas, termasuk Pengguna root akun AWS, terlepas dari apakah itu milik organisasi Anda. Untuk informasi selengkapnya tentang Organizations dan RCPs, termasuk daftar dukungan Layanan AWS tersebut RCPs, lihat Kebijakan kontrol sumber daya (RCPs) di Panduan AWS Organizations Pengguna.

- Kebijakan sesi – Kebijakan sesi adalah kebijakan lanjutan yang Anda berikan sebagai parameter ketika Anda membuat sesi sementara secara programatis untuk peran atau pengguna terfederasi. Izin sesi yang dihasilkan adalah perpotongan antara kebijakan berbasis identitas pengguna atau peran dan kebijakan sesi. Izin juga bisa datang dari kebijakan berbasis sumber daya. Penolakan eksplisit dalam salah satu kebijakan ini akan menggantikan pemberian izin. Untuk informasi selengkapnya, lihat Kebijakan sesi dalam Panduan Pengguna IAM.

Berbagai jenis kebijakan

Ketika beberapa jenis kebijakan berlaku pada suatu permintaan, izin yang dihasilkan lebih rumit untuk dipahami. Untuk mempelajari cara AWS menentukan apakah akan mengizinkan permintaan saat beberapa jenis kebijakan terlibat, lihat Logika evaluasi kebijakan di Panduan Pengguna IAM.

Bagaimana FreeRTOS bekerja dengan IAM

Sebelum Anda menggunakan IAM untuk mengelola akses ke FreeRTOS, pelajari fitur IAM apa yang tersedia untuk digunakan dengan FreeRTOS.

Fitur IAM yang dapat Anda gunakan dengan FreeRTOS

Fitur IAM	Dukungan FreeRTOS
Kebijakan berbasis identitas	Ya
Kebijakan berbasis sumber daya	Tidak
Tindakan kebijakan	Ya
Sumber daya kebijakan	Ya
kunci-kunci persyaratan kebijakan (spesifik layanan)	Ya

Fitur IAM	Dukungan FreeRTOS
ACLs	Tidak
ABAC (tanda dalam kebijakan)	Parsial
Kredensial sementara	Ya
Izin principal	Ya
Peran layanan	Ya
Peran terkait layanan	Tidak

Untuk mendapatkan tampilan tingkat tinggi tentang cara FreeRTOS dan layanan AWS lainnya bekerja dengan sebagian besar fitur IAM, AWS lihat layanan yang bekerja dengan IAM di Panduan Pengguna IAM.

Kebijakan berbasis identitas untuk FreeRTOS

Mendukung kebijakan berbasis identitas: Ya

Kebijakan berbasis identitas adalah dokumen kebijakan izin JSON yang dapat Anda lampirkan ke sebuah identitas, seperti pengguna IAM, grup pengguna IAM, atau peran IAM. Kebijakan ini mengontrol jenis tindakan yang dapat dilakukan oleh pengguna dan peran, di sumber daya mana, dan berdasarkan kondisi seperti apa. Untuk mempelajari cara membuat kebijakan berbasis identitas, lihat Tentukan izin IAM kustom dengan kebijakan terkelola pelanggan dalam Panduan Pengguna IAM.

Dengan kebijakan berbasis identitas IAM, Anda dapat menentukan secara spesifik apakah tindakan dan sumber daya diizinkan atau ditolak, serta kondisi yang menjadi dasar dikabulkan atau ditolaknya tindakan tersebut. Anda tidak dapat menentukan secara spesifik prinsipal dalam sebuah kebijakan berbasis identitas karena prinsipal berlaku bagi pengguna atau peran yang melekat kepadanya. Untuk mempelajari semua elemen yang dapat Anda gunakan dalam kebijakan JSON, lihat Referensi elemen kebijakan JSON IAM dalam Panduan Pengguna IAM.

Contoh kebijakan berbasis identitas untuk FreeRTOS

Untuk melihat contoh kebijakan berbasis identitas FreeRTOS, lihat. Contoh kebijakan berbasis identitas untuk FreeRTOS

Kebijakan berbasis sumber daya dalam FreeRTOS

Mendukung kebijakan berbasis sumber daya: Tidak

Kebijakan berbasis sumber daya adalah dokumen kebijakan JSON yang Anda lampirkan ke sumber daya. Contoh kebijakan berbasis sumber daya adalah kebijakan kepercayaan peran IAM dan kebijakan bucket Amazon S3. Dalam layanan yang mendukung kebijakan berbasis sumber daya, administrator layanan dapat menggunakannya untuk mengontrol akses ke sumber daya tertentu. Untuk sumber daya tempat kebijakan dilampirkan, kebijakan menentukan tindakan apa yang dapat dilakukan oleh prinsipal tertentu pada sumber daya tersebut dan dalam kondisi apa. Anda harus menentukan prinsipal dalam kebijakan berbasis sumber daya. Prinsipal dapat mencakup akun, pengguna, peran, pengguna federasi, atau. Layanan AWS

Untuk mengaktifkan akses lintas akun, Anda dapat menentukan secara spesifik seluruh akun atau entitas IAM di akun lain sebagai prinsipal dalam kebijakan berbasis sumber daya. Menambahkan prinsipal akun silang ke kebijakan berbasis sumber daya hanya setengah dari membangun hubungan kepercayaan. Ketika prinsipal dan sumber daya berbeda Akun AWS, administrator IAM di akun tepercaya juga harus memberikan izin entitas utama (pengguna atau peran) untuk mengakses sumber daya. Mereka memberikan izin dengan melampirkan kebijakan berbasis identitas kepada entitas. Namun, jika kebijakan berbasis sumber daya memberikan akses ke principal dalam akun yang sama, tidak diperlukan kebijakan berbasis identitas tambahan. Untuk informasi selengkapnya, lihat Akses sumber daya lintas akun di IAM dalam Panduan Pengguna IAM.

Tindakan kebijakan untuk FreeRTOS

Mendukung tindakan kebijakan: Ya

Administrator dapat menggunakan kebijakan AWS JSON untuk menentukan siapa yang memiliki akses ke apa. Artinya, prinsipal manakah yang dapat melakukan tindakan pada sumber daya apa, dan dengan kondisi apa.

Elemen `Action` dari kebijakan JSON menjelaskan tindakan yang dapat Anda gunakan untuk mengizinkan atau menolak akses dalam sebuah kebijakan. Tindakan kebijakan biasanya memiliki nama yang sama dengan operasi AWS API terkait. Ada beberapa pengecualian, misalnya tindakan hanya izin yang tidak memiliki operasi API yang cocok. Ada juga beberapa operasi yang memerlukan beberapa tindakan dalam suatu kebijakan. Tindakan tambahan ini disebut tindakan dependen.

Sertakan tindakan dalam kebijakan untuk memberikan izin untuk melakukan operasi terkait.

Untuk melihat daftar tindakan FreeRTOS, lihat Tindakan yang ditentukan oleh FreeRTOS di Referensi Otorisasi Layanan.

Tindakan kebijakan di FreeRTOS menggunakan awalan berikut sebelum tindakan:

```
awes
```

Untuk menetapkan secara spesifik beberapa tindakan dalam satu pernyataan, pisahkan tindakan tersebut dengan koma.

```
"Action": [
      "awes:action1",
      "awes:action2"
         ]
```

Untuk melihat contoh kebijakan berbasis identitas FreeRTOS, lihat. Contoh kebijakan berbasis identitas untuk FreeRTOS

Sumber daya kebijakan untuk FreeRTOS

Mendukung sumber daya kebijakan: Ya

Administrator dapat menggunakan kebijakan AWS JSON untuk menentukan siapa yang memiliki akses ke apa. Artinya, prinsipal manakah yang dapat melakukan tindakan pada sumber daya apa, dan dengan kondisi apa.

Elemen kebijakan JSON `Resource` menentukan objek yang menjadi target penerapan tindakan. Pernyataan harus menyertakan elemen `Resource` atau `NotResource`. Praktik terbaiknya, tentukan sumber daya menggunakan Amazon Resource Name (ARN). Anda dapat melakukan ini untuk tindakan yang mendukung jenis sumber daya tertentu, yang dikenal sebagai izin tingkat sumber daya.

Untuk tindakan yang tidak mendukung izin di tingkat sumber daya, misalnya operasi pencantuman, gunakan wildcard (*) untuk menunjukkan bahwa pernyataan tersebut berlaku untuk semua sumber daya.

```
"Resource": "*"
```

Untuk melihat daftar jenis sumber daya FreeRTOS dan ARNs jenisnya, lihat Sumber daya yang ditentukan oleh FreeRTOS di Referensi Otorisasi Layanan. Untuk mempelajari tindakan mana yang dapat Anda tentukan ARN dari setiap sumber daya, lihat Tindakan yang ditentukan oleh FreeRTOS.

Untuk melihat contoh kebijakan berbasis identitas FreeRTOS, lihat. Contoh kebijakan berbasis identitas untuk FreeRTOS

Kunci kondisi kebijakan untuk FreeRTOS

Mendukung kunci kondisi kebijakan khusus layanan: Yes

Administrator dapat menggunakan kebijakan AWS JSON untuk menentukan siapa yang memiliki akses ke apa. Artinya, prinsipal manakah yang dapat melakukan tindakan pada sumber daya apa, dan dengan kondisi apa.

Elemen `Condition` (atau blok `Condition`) akan memungkinkan Anda menentukan kondisi yang menjadi dasar suatu pernyataan berlaku. Elemen `Condition` bersifat opsional. Anda dapat membuat ekspresi bersyarat yang menggunakan operator kondisi, misalnya sama dengan atau kurang dari, untuk mencocokkan kondisi dalam kebijakan dengan nilai-nilai yang diminta.

Jika Anda menentukan beberapa elemen `Condition` dalam sebuah pernyataan, atau beberapa kunci dalam elemen `Condition` tunggal, maka AWS akan mengevaluasinya menggunakan operasi `AND` logis. Jika Anda menentukan beberapa nilai untuk satu kunci kondisi, AWS mengevaluasi kondisi menggunakan `OR` operasi logis. Semua kondisi harus dipenuhi sebelum izin pernyataan diberikan.

Anda juga dapat menggunakan variabel placeholder saat menentukan kondisi. Sebagai contoh, Anda dapat memberikan izin kepada pengguna IAM untuk mengakses sumber daya hanya jika izin tersebut mempunyai tanda yang sesuai dengan nama pengguna IAM mereka. Untuk informasi selengkapnya, lihat Elemen kebijakan IAM: variabel dan tanda dalam Panduan Pengguna IAM.

AWS mendukung kunci kondisi global dan kunci kondisi khusus layanan. Untuk melihat semua kunci kondisi AWS global, lihat kunci konteks kondisi AWS global di Panduan Pengguna IAM.

Untuk melihat daftar kunci kondisi FreeRTOS, lihat Kunci kondisi untuk FreeRTOS di Referensi Otorisasi Layanan. Untuk mempelajari tindakan dan sumber daya yang dapat Anda gunakan kunci kondisi, lihat Tindakan yang ditentukan oleh FreeRTOS.

Untuk melihat contoh kebijakan berbasis identitas FreeRTOS, lihat. Contoh kebijakan berbasis identitas untuk FreeRTOS

ACLs di FreeRTOS

Mendukung ACLs: Tidak

Access control lists (ACLs) mengontrol prinsipal mana (anggota akun, pengguna, atau peran) yang memiliki izin untuk mengakses sumber daya. ACLs mirip dengan kebijakan berbasis sumber daya, meskipun mereka tidak menggunakan format dokumen kebijakan JSON.

ABAC dengan FreeRTOS

Mendukung ABAC (tag dalam kebijakan): Sebagian

Kontrol akses berbasis atribut (ABAC) adalah strategi otorisasi yang menentukan izin berdasarkan atribut. Dalam AWS, atribut ini disebut tag. Anda dapat melampirkan tag ke entitas IAM (pengguna atau peran) dan ke banyak AWS sumber daya. Penandaan ke entitas dan sumber daya adalah langkah pertama dari ABAC. Kemudian rancanglah kebijakan ABAC untuk mengizinkan operasi ketika tanda milik prinsipal cocok dengan tanda yang ada di sumber daya yang ingin diakses.

ABAC sangat berguna di lingkungan yang berkembang dengan cepat dan berguna di situasi saat manajemen kebijakan menjadi rumit.

Untuk mengendalikan akses berdasarkan tanda, berikan informasi tentang tanda di elemen kondisi dari kebijakan menggunakan kunci kondisi `aws:ResourceTag/`*`key-name`*, `aws:RequestTag/`*`key-name`*, atau `aws:TagKeys`.

Jika sebuah layanan mendukung ketiga kunci kondisi untuk setiap jenis sumber daya, nilainya adalah Ya untuk layanan tersebut. Jika suatu layanan mendukung ketiga kunci kondisi untuk hanya beberapa jenis sumber daya, nilainya adalah Parsial.

Untuk informasi selengkapnya tentang ABAC, lihat Tentukan izin dengan otorisasi ABAC dalam Panduan Pengguna IAM. Untuk melihat tutorial yang menguraikan langkah-langkah pengaturan ABAC, lihat Menggunakan kontrol akses berbasis atribut (ABAC) dalam Panduan Pengguna IAM.

Menggunakan kredensil sementara dengan FreeRTOS

Mendukung kredensial sementara: Ya

Beberapa Layanan AWS tidak berfungsi saat Anda masuk menggunakan kredensil sementara. Untuk informasi tambahan, termasuk yang Layanan AWS bekerja dengan kredensi sementara, lihat Layanan AWS yang bekerja dengan IAM di Panduan Pengguna IAM.

Anda menggunakan kredensi sementara jika Anda masuk AWS Management Console menggunakan metode apa pun kecuali nama pengguna dan kata sandi. Misalnya, ketika Anda mengakses AWS menggunakan tautan masuk tunggal (SSO) perusahaan Anda, proses tersebut secara otomatis membuat kredensil sementara. Anda juga akan secara otomatis membuat kredensial sementara ketika Anda masuk ke konsol sebagai seorang pengguna lalu beralih peran. Untuk informasi selengkapnya tentang peralihan peran, lihat Beralih dari pengguna ke peran IAM (konsol) dalam Panduan Pengguna IAM.

Anda dapat membuat kredensil sementara secara manual menggunakan API AWS CLI atau AWS . Anda kemudian dapat menggunakan kredensil sementara tersebut untuk mengakses. AWS AWS merekomendasikan agar Anda secara dinamis menghasilkan kredensi sementara alih-alih menggunakan kunci akses jangka panjang. Untuk informasi selengkapnya, lihat Kredensial keamanan sementara di IAM.

Izin utama lintas layanan untuk FreeRTOS

Mendukung sesi akses maju (FAS): Ya

Saat Anda menggunakan pengguna atau peran IAM untuk melakukan tindakan AWS, Anda dianggap sebagai prinsipal. Ketika Anda menggunakan beberapa layanan, Anda mungkin melakukan sebuah tindakan yang kemudian menginisiasi tindakan lain di layanan yang berbeda. FAS menggunakan izin dari pemanggilan utama Layanan AWS, dikombinasikan dengan permintaan Layanan AWS untuk membuat permintaan ke layanan hilir. Permintaan FAS hanya dibuat ketika layanan menerima permintaan yang memerlukan interaksi dengan orang lain Layanan AWS atau sumber daya untuk menyelesaikannya. Dalam hal ini, Anda harus memiliki izin untuk melakukan kedua tindakan tersebut. Untuk detail kebijakan ketika mengajukan permintaan FAS, lihat Sesi akses maju.

Peran layanan untuk FreeRTOS

Mendukung peran layanan: Ya

Peran layanan adalah peran IAM yang diambil oleh sebuah layanan untuk melakukan tindakan atas nama Anda. Administrator IAM dapat membuat, mengubah, dan menghapus peran layanan dari dalam IAM. Untuk informasi selengkapnya, lihat Buat sebuah peran untuk mendelegasikan izin ke Layanan AWS dalam Panduan pengguna IAM.

> **Warning**
>
> Mengubah izin untuk peran layanan dapat merusak fungsionalitas FreeRTOS. Edit peran layanan hanya ketika FreeRTOS memberikan panduan untuk melakukannya.

Peran terkait layanan untuk FreeRTOS

Mendukung peran terkait layanan: Tidak

Peran terkait layanan adalah jenis peran layanan yang ditautkan ke. Layanan AWS Layanan tersebut dapat menjalankan peran untuk melakukan tindakan atas nama Anda. Peran terkait layanan muncul di Anda Akun AWS dan dimiliki oleh layanan. Administrator IAM dapat melihat, tetapi tidak dapat mengedit izin untuk peran terkait layanan.

Untuk detail tentang pembuatan atau manajemen peran terkait layanan, lihat Layanan AWS yang berfungsi dengan IAM. Cari layanan dalam tabel yang memiliki `Yes` di kolom Peran terkait layanan. Pilih tautan Ya untuk melihat dokumentasi peran terkait layanan untuk layanan tersebut.

Contoh kebijakan berbasis identitas untuk FreeRTOS

Secara default, pengguna dan peran tidak memiliki izin untuk membuat atau memodifikasi sumber daya FreeRTOS. Mereka juga tidak dapat melakukan tugas dengan menggunakan AWS Management Console, AWS Command Line Interface (AWS CLI), atau AWS API. Untuk memberikan izin kepada pengguna untuk melakukan tindakan di sumber daya yang mereka perlukan, administrator IAM dapat membuat kebijakan IAM. Administrator kemudian dapat menambahkan kebijakan IAM ke peran, dan pengguna dapat mengambil peran.

Untuk mempelajari cara membuat kebijakan berbasis identitas IAM dengan menggunakan contoh dokumen kebijakan JSON ini, lihat Membuat kebijakan IAM (konsol) di Panduan Pengguna IAM.

Untuk detail tentang tindakan dan jenis sumber daya yang ditentukan oleh FreeRTOS, termasuk format untuk setiap jenis sumber daya, lihat Tindakan, sumber daya, dan kunci kondisi untuk FreeRTOS di Referensi Otorisasi Layanan. ARNs

Topik

- Praktik terbaik kebijakan
- Menggunakan konsol FreeRTOS

- Mengizinkan pengguna melihat izin mereka sendiri

Praktik terbaik kebijakan

Kebijakan berbasis identitas menentukan apakah seseorang dapat membuat, mengakses, atau menghapus sumber daya FreeRTOS di akun Anda. Tindakan ini membuat Akun AWS Anda dikenai biaya. Ketika Anda membuat atau mengedit kebijakan berbasis identitas, ikuti panduan dan rekomendasi ini:

- Mulailah dengan kebijakan AWS terkelola dan beralih ke izin hak istimewa paling sedikit — Untuk mulai memberikan izin kepada pengguna dan beban kerja Anda, gunakan kebijakan AWS terkelola yang memberikan izin untuk banyak kasus penggunaan umum. Mereka tersedia di Anda Akun AWS. Kami menyarankan Anda mengurangi izin lebih lanjut dengan menentukan kebijakan yang dikelola AWS pelanggan yang khusus untuk kasus penggunaan Anda. Untuk informasi selengkapnya, lihat Kebijakan yang dikelola AWS atau Kebijakan yang dikelola AWS untuk fungsi tugas dalam Panduan Pengguna IAM.
- Menerapkan izin dengan hak akses paling rendah – Ketika Anda menetapkan izin dengan kebijakan IAM, hanya berikan izin yang diperlukan untuk melakukan tugas. Anda melakukannya dengan mendefinisikan tindakan yang dapat diambil pada sumber daya tertentu dalam kondisi tertentu, yang juga dikenal sebagai izin dengan hak akses paling rendah. Untuk informasi selengkapnya tentang cara menggunakan IAM untuk mengajukan izin, lihat Kebijakan dan izin dalam IAM dalam Panduan Pengguna IAM.
- Gunakan kondisi dalam kebijakan IAM untuk membatasi akses lebih lanjut – Anda dapat menambahkan suatu kondisi ke kebijakan Anda untuk membatasi akses ke tindakan dan sumber daya. Sebagai contoh, Anda dapat menulis kondisi kebijakan untuk menentukan bahwa semua permintaan harus dikirim menggunakan SSL. Anda juga dapat menggunakan ketentuan untuk memberikan akses ke tindakan layanan jika digunakan melalui yang spesifik Layanan AWS, seperti AWS CloudFormation. Untuk informasi selengkapnya, lihat Elemen kebijakan JSON IAM: Kondisi dalam Panduan Pengguna IAM.
- Gunakan IAM Access Analyzer untuk memvalidasi kebijakan IAM Anda untuk memastikan izin yang aman dan fungsional – IAM Access Analyzer memvalidasi kebijakan baru dan yang sudah ada sehingga kebijakan tersebut mematuhi bahasa kebijakan IAM (JSON) dan praktik terbaik IAM. IAM Access Analyzer menyediakan lebih dari 100 pemeriksaan kebijakan dan rekomendasi yang dapat ditindaklanjuti untuk membantu Anda membuat kebijakan yang aman dan fungsional. Untuk informasi selengkapnya, lihat Validasi kebijakan dengan IAM Access Analyzer dalam Panduan Pengguna IAM.

- Memerlukan otentikasi multi-faktor (MFA) - Jika Anda memiliki skenario yang mengharuskan pengguna IAM atau pengguna root di Anda, Akun AWS aktifkan MFA untuk keamanan tambahan. Untuk meminta MFA ketika operasi API dipanggil, tambahkan kondisi MFA pada kebijakan Anda. Untuk informasi selengkapnya, lihat Amankan akses API dengan MFA dalam Panduan Pengguna IAM.

Untuk informasi selengkapnya tentang praktik terbaik dalam IAM, lihat Praktik terbaik keamanan di IAM dalam Panduan Pengguna IAM.

Menggunakan konsol FreeRTOS

Untuk mengakses konsol FreeRTOS, Anda harus memiliki set izin minimum. Izin ini harus memungkinkan Anda untuk daftar dan melihat detail tentang sumber daya FreeRTOS di Anda. Akun AWS Jika Anda membuat kebijakan berbasis identitas yang lebih ketat daripada izin minimum yang diperlukan, konsol tidak akan berfungsi sebagaimana mestinya untuk entitas (pengguna atau peran) dengan kebijakan tersebut.

Anda tidak perlu mengizinkan izin konsol minimum untuk pengguna yang melakukan panggilan hanya ke AWS CLI atau AWS API. Sebagai gantinya, izinkan akses hanya ke tindakan yang sesuai dengan operasi API yang coba mereka lakukan.

Untuk memastikan bahwa pengguna dan peran masih dapat menggunakan konsol FreerTOS, lampirkan juga FreeRTOS atau kebijakan terkelola ke *ConsoleAccess* entitas *ReadOnly* AWS . Untuk informasi selengkapnya, lihat Menambah izin untuk pengguna dalam Panduan Pengguna IAM.

Mengizinkan pengguna melihat izin mereka sendiri

Contoh ini menunjukkan cara membuat kebijakan yang mengizinkan pengguna IAM melihat kebijakan inline dan terkelola yang dilampirkan ke identitas pengguna mereka. Kebijakan ini mencakup izin untuk menyelesaikan tindakan ini di konsol atau menggunakan API atau secara terprogram. AWS CLI AWS

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
```

```
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Memecahkan masalah identitas dan akses FreeRTOS

Gunakan informasi berikut untuk membantu Anda mendiagnosis dan memperbaiki masalah umum yang mungkin Anda temui saat bekerja dengan FreeRTOS dan IAM.

Topik

- Saya tidak berwenang untuk melakukan tindakan di FreeRTOS
- Saya tidak berwenang untuk melakukan iam: PassRole
- Izinkan orang di luar saya Akun AWS untuk mengakses sumber daya FreeRTOS saya

Saya tidak berwenang untuk melakukan tindakan di FreeRTOS

Jika Anda menerima pesan kesalahan bahwa Anda tidak memiliki otorisasi untuk melakukan tindakan, kebijakan Anda harus diperbarui agar Anda dapat melakukan tindakan tersebut.

Contoh kesalahan berikut terjadi ketika pengguna IAM `mateojackson` mencoba menggunakan konsol untuk melihat detail tentang suatu sumber daya *`my-example-widget`* rekaan, tetapi tidak memiliki izin `awes:`*`GetWidget`* rekaan.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 awes:GetWidget on resource: my-example-widget
```

Dalam hal ini, kebijakan untuk pengguna `mateojackson` harus diperbarui untuk mengizinkan akses ke sumber daya *`my-example-widget`* dengan menggunakan tindakan `awes:`*`GetWidget`*.

Jika Anda memerlukan bantuan, hubungi AWS administrator Anda. Administrator Anda adalah orang yang memberi Anda kredensial masuk.

Saya tidak berwenang untuk melakukan iam: PassRole

Jika Anda menerima kesalahan bahwa Anda tidak berwenang untuk melakukan `iam:PassRole` tindakan, kebijakan Anda harus diperbarui agar Anda dapat meneruskan peran ke FreeRTOS.

Beberapa Layanan AWS memungkinkan Anda untuk meneruskan peran yang ada ke layanan tersebut alih-alih membuat peran layanan baru atau peran terkait layanan. Untuk melakukannya, Anda harus memiliki izin untuk meneruskan peran ke layanan.

Contoh kesalahan berikut terjadi ketika pengguna IAM bernama `marymajor` mencoba menggunakan konsol untuk melakukan tindakan di FreeRTOS. Namun, tindakan tersebut memerlukan layanan untuk mendapatkan izin yang diberikan oleh peran layanan. Mary tidak memiliki izin untuk meneruskan peran tersebut pada layanan.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

Dalam kasus ini, kebijakan Mary harus diperbarui agar dia mendapatkan izin untuk melakukan tindakan `iam:PassRole` tersebut.

Jika Anda memerlukan bantuan, hubungi AWS administrator Anda. Administrator Anda adalah orang yang memberi Anda kredensial masuk.

Izinkan orang di luar saya Akun AWS untuk mengakses sumber daya FreeRTOS saya

Anda dapat membuat peran yang dapat digunakan pengguna di akun lain atau orang-orang di luar organisasi Anda untuk mengakses sumber daya Anda. Anda dapat menentukan siapa saja yang

dipercaya untuk mengambil peran tersebut. Untuk layanan yang mendukung kebijakan berbasis sumber daya atau daftar kontrol akses (ACLs), Anda dapat menggunakan kebijakan tersebut untuk memberi orang akses ke sumber daya Anda.

Untuk mempelajari selengkapnya, periksa referensi berikut:

- Untuk mengetahui apakah FreeRTOS mendukung fitur-fitur ini, lihat. Bagaimana FreeRTOS bekerja dengan IAM
- Untuk mempelajari cara menyediakan akses ke sumber daya Anda di seluruh sumber daya Akun AWS yang Anda miliki, lihat Menyediakan akses ke pengguna IAM di pengguna lain Akun AWS yang Anda miliki di Panduan Pengguna IAM.
- Untuk mempelajari cara menyediakan akses ke sumber daya Anda kepada pihak ketiga Akun AWS, lihat Menyediakan akses yang Akun AWS dimiliki oleh pihak ketiga dalam Panduan Pengguna IAM.
- Untuk mempelajari cara memberikan akses melalui federasi identitas, lihat Menyediakan akses ke pengguna terautentikasi eksternal (federasi identitas) dalam Panduan Pengguna IAM.
- Untuk mempelajari perbedaan antara menggunakan peran dan kebijakan berbasis sumber daya untuk akses lintas akun, lihat Akses sumber daya lintas akun di IAM di Panduan Pengguna IAM.

Validasi kepatuhan

Untuk mempelajari apakah an Layanan AWS berada dalam lingkup program kepatuhan tertentu, lihat Layanan AWS di Lingkup oleh Program Kepatuhan Layanan AWS dan pilih program kepatuhan yang Anda minati. Untuk informasi umum, lihat Program AWS Kepatuhan Program AWS .

Anda dapat mengunduh laporan audit pihak ketiga menggunakan AWS Artifact. Untuk informasi selengkapnya, lihat Mengunduh Laporan di AWS Artifact .

Tanggung jawab kepatuhan Anda saat menggunakan Layanan AWS ditentukan oleh sensitivitas data Anda, tujuan kepatuhan perusahaan Anda, dan hukum dan peraturan yang berlaku. AWS menyediakan sumber daya berikut untuk membantu kepatuhan:

- Kepatuhan dan Tata Kelola Keamanan – Panduan implementasi solusi ini membahas pertimbangan arsitektur serta memberikan langkah-langkah untuk menerapkan fitur keamanan dan kepatuhan.
- Referensi Layanan yang Memenuhi Syarat HIPAA — Daftar layanan yang memenuhi syarat HIPAA. Tidak semua memenuhi Layanan AWS syarat HIPAA.

- AWS Sumber Daya AWS — Kumpulan buku kerja dan panduan ini mungkin berlaku untuk industri dan lokasi Anda.
- AWS Panduan Kepatuhan Pelanggan - Memahami model tanggung jawab bersama melalui lensa kepatuhan. Panduan ini merangkum praktik terbaik untuk mengamankan Layanan AWS dan memetakan panduan untuk kontrol keamanan di berbagai kerangka kerja (termasuk Institut Standar dan Teknologi Nasional (NIST), Dewan Standar Keamanan Industri Kartu Pembayaran (PCI), dan Organisasi Internasional untuk Standardisasi (ISO)).
- Mengevaluasi Sumber Daya dengan Aturan dalam Panduan AWS Config Pengembang — AWS Config Layanan menilai seberapa baik konfigurasi sumber daya Anda mematuhi praktik internal, pedoman industri, dan peraturan.
- AWS Security Hub— Ini Layanan AWS memberikan pandangan komprehensif tentang keadaan keamanan Anda di dalamnya AWS. Security Hub menggunakan kontrol keamanan untuk sumber daya AWS Anda serta untuk memeriksa kepatuhan Anda terhadap standar industri keamanan dan praktik terbaik. Untuk daftar layanan dan kontrol yang didukung, lihat Referensi kontrol Security Hub.
- Amazon GuardDuty — Ini Layanan AWS mendeteksi potensi ancaman terhadap beban kerja Akun AWS, kontainer, dan data Anda dengan memantau lingkungan Anda untuk aktivitas yang mencurigakan dan berbahaya. GuardDuty dapat membantu Anda mengatasi berbagai persyaratan kepatuhan, seperti PCI DSS, dengan memenuhi persyaratan deteksi intrusi yang diamanatkan oleh kerangka kerja kepatuhan tertentu.
- AWS Audit ManagerIni Layanan AWS membantu Anda terus mengaudit AWS penggunaan Anda untuk menyederhanakan cara Anda mengelola risiko dan kepatuhan terhadap peraturan dan standar industri.

Ketahanan di AWS

Infrastruktur AWS global dibangun di sekitar AWS Wilayah dan Zona Ketersediaan. AWS Wilayah menyediakan beberapa Zona Ketersediaan yang terpisah dan terisolasi secara fisik, yang terhubung dengan jaringan latensi rendah, throughput tinggi, dan sangat redundan. Dengan Zona Ketersediaan, Anda dapat merancang serta mengoperasikan aplikasi dan basis data yang secara otomatis melakukan failover di antara Zona Ketersediaan tanpa gangguan. Zona Ketersediaan memiliki ketersediaan dan toleransi kesalahan yang lebih baik, dan dapat diskalakan dibandingkan infrastruktur biasa yang terdiri dari satu atau beberapa pusat data.

Untuk informasi selengkapnya tentang AWS Wilayah dan Availability Zone, lihat Infrastruktur AWS Global.

Keamanan infrastruktur di FreeRTOS

AWS layanan terkelola dilindungi oleh prosedur keamanan jaringan AWS global yang dijelaskan dalam whitepaper Amazon Web Services: Tinjauan Proses Keamanan.

Anda menggunakan panggilan API yang AWS dipublikasikan untuk mengakses AWS layanan melalui jaringan. Klien harus mendukung Keamanan Lapisan Pengangkutan (TLS) 1.2 atau versi yang lebih baru. Kami merekomendasikan TLS 1.3 atau versi yang lebih baru. Klien juga harus mendukung cipher suite dengan perfect forward secrecy (PFS) seperti Ephemeral Diffie-Hellman (DHE) atau Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Sebagian besar sistem modern seperti Java 7 dan sistem yang lebih baru mendukung mode ini.

Selain itu, permintaan harus ditandatangani menggunakan ID kunci akses dan kunci akses rahasia yang terkait dengan prinsipal IAM. Atau Anda bisa menggunakan AWS Security Token Service (AWS STS) untuk membuat kredensial keamanan sementara guna menandatangani permintaan.

Panduan Migrasi Repositori Github Amazon-freertos

Jika Anda memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori amazon-freertos yang sekarang tidak digunakan lagi, ikuti langkah-langkah ini:

1. Tetap up to date dengan perbaikan keamanan terbaru yang tersedia untuk umum. Periksa halaman pustaka FreerTOS LTS untuk pembaruan, atau berlangganan repositori GitHub Freertos-LTS untuk menerima patch LTS terbaru dengan perbaikan bug kritis dan keamanan. Anda dapat mengunduh atau mengkloning patch FreerTOS LTS terbaru yang diperlukan langsung dari masing-masing repositori. GitHub
2. Pertimbangkan refactoring implementasi antarmuka transportasi jaringan untuk mengoptimalkan platform perangkat keras Anda. Abstrak APIs seperti soket aman dan Wifi tidak APIs diperlukan oleh perpustakaan CoreMQTT terbaru. Lihat Transport Interface untuk detail lebih lanjut.

Lampiran

Tabel berikut memberikan rekomendasi untuk semua proyek demo, pustaka lama, dan abstrak APIs dalam repositori Amazon-Freertos.

Perpustakaan dan demo yang dimigrasi

Nama	Tipe	Rekomendasi	
CoreHTTP	demo dan perpustakaan	Kloning atau unduh pustaka CoreHTTP langsung dari repositori CoreHTTP (sub-modul jika menggunakan git) di organisasi FreerTOS Github. Demo CoreHTTP berada dalam distribusi FreerTOS utama. Untuk lebih jelasnya, lihat halaman CoreHTTP.	
CoreMQTT	demo dan perpustakaan	Kloning atau unduh pustaka CoreMQTT langsung dari repositori CoreMQTT (sub-modul jika menggunakan git) di organisasi FreeRTOS Github. Demo CoreMQTT	

Nama	Tipe	Rekomendasi	
		berada dalam distribusi FreeRTOS utama. Untuk detail lebih lanjut, lihat halaman CoreMQTT.	
Agen CoreMQTT	demo dan perpustakaan	Kloning atau unduh pustaka CoreMQTT-Agent langsung dari repositori CoreMQTT-Agent (sub-modul jika menggunakan git) di organisasi FreeRTOS Github. Demo CoreMQTT-Agent ada di repositori CoreMQTT-Agent-Demos. Untuk detail lebih lanjut, lihat halaman Agen CoreMQTT.	
device_defender_for_aws	demo dan perpustakaan	Pustaka AWS IoT Device Defender ada di repositorinya di organisasi.AWS GitHub Kloning atau unduh (sub-modul jika menggunakan git) langsung dari repositori AWS IoT Device Defender. Demo AWS IoT Device Defender berada dalam distribusi FreeRTOS utama. Untuk detail selengkapnya, lihat halaman AWS IoT Device Defender.	
device_shadow_for_aws	demo dan perpustakaan	Pustaka AWS IoT Device Shadow ada di repositorinya di organisasi.AWS GitHub Kloning atau unduh (sub-modul jika menggunakan git) langsung dari repositori AWS IoT Device Shadow). Demo AWS IoT Device Shadow berada di distribusi FreerTOS utama. Untuk detail selengkapnya, lihat halaman AWS IoT Device Shadow.	

Nama	Tipe	Rekomendasi
jobs_for_aws	demo dan perpustakaan	Perpustakaan AWS IoT Jobs ada di repositori di organisasi.AWS GitHub Kloning atau unduh (sub-modul jika menggunakan git) langsung dari repositori AWS IoT Jobs. Demo AWS IoT Jobs berada di distribus i FreeRTOS utama. Untuk detail selengkapnya, lihat halaman AWS IoT Pekerjaan.
OTA	demo dan perpustakaan	Pustaka Pembaruan AWS IoT Over-The-Air (OTA) ada di repositori di organisasi.AWS GitHub Kloning atau unduh (sub-modul jika menggunakan git) langsung dari repositori AWS IoT OTA. Demo AWS IoT OTA berada dalam distribusi FreeRTOS utama. Untuk lebih jelasnya, lihat halaman AWS IoT OTA.
CLI dan Freertos_ Plus_CLI	demo dan perpustakaan	Ada contoh CLI yang berjalan di. WinSim Lihat halaman Antarmuka Baris Perintah FreeRTOS Plus untuk lebih jelasnya. Integrasi referensi IoT FreeRTOS Unggulan pada platform RT1 NXP i.MX STM32060 dan U5, juga memberikan contoh CLI pada perangkat keras yang sebenarnya.

Nama	Tipe	Rekomendasi	
pencatatan log	makro	Ada implementasi makro logging untuk platform perangkat keras tertentu yang digunakan oleh beberapa pustaka FreeRTOS. Lihat halaman logging untuk cara menerapkan makro logging. Lihat salah satu referensi IoT unggulan FreeRTOS untuk contoh yang berjalan pada perangkat keras yang sebenarnya.	
greengrass_connectivity	demo	[Migrasi sedang berlangsung] Proyek demo ini mengasumsikan bahwa konektivitas cloud tersedia sebelum menghubungkan ke perangkat AWS IoT Greengrass. Sebuah proyek baru yang menunjukkan otentikasi lokal dan kemampuan penemuan sedang dalam pengembangan. Harapkan proyek demo baru akan segera diterbitkan di organisasi FreerTOS Github.	

Pustaka dan demo usang

Nama	Tipe	Rekomendasi	
BLE	demo dan perpustakaan	Pustaka FreeRTOS BLE mengimplementasikan protokol MQTT berpemilik dan mendukung penerbitan dan berlangganan topik MQTT melalui Bluetooth Low Energy (BLE) melalui perangkat proxy seperti ponsel. Ini tidak lagi diamanatkan. Gunakan tumpukan BLE Anda sendiri atau	

Nama	Tipe	Rekomendasi	
		opsi pihak ketiga seperti Nimble untuk mengoptimalkan proyek Anda dengan sebaik-baiknya.	
dev_mode_ key_provisioning	demo	Integrasi referensi IoT FreeRTOS Unggulan pada platform RT1 NXP i.MX STM32060, U5, ESP32atau - C3 memberikan contoh penyediaan penting menggunakan CLI.	
posix	abstraksi dan demo	Tidak direkomendasikan untuk digunakan.	
wifi_provisioning	contoh	Contoh ini menunjukkan cara menyediakan WiFi kredensyal pada perangkat menggunakan library Amazon-Freertos BLE. Lihat referensi IoT Unggulan FreeRTOS ESP32pada platform C3 untuk contoh penyediaan melalui BLE. WiFi	
Abstrak warisan APIs	code	Ini adalah APIs yang dibuat untuk menyediakan antarmuka abstrak untuk berbagai tumpukan perangkat lunak pihak ketiga, modul konektivi tas, dan platform MCU dari berbagai vendor. Misalnya, ada antarmuka untuk WiFi abstraksi, soket aman, dan sebagainya. Mereka didukung di repositori Amazon-Freertos dan ada di folder. `/libraries/abstrac tions/` Ini tidak APIs diperluka n saat menggunakan pustaka FreerTOS LTS.	

Pustaka dan demo pada tabel di atas tidak akan mendapatkan patch keamanan atau perbaikan bug.

Perpustakaan pihak ketiga

Ketika demo di Amazon-Freertos menggunakan pustaka pihak ketiga, kami sarankan Anda mensub-modulnya langsung dari repositori pihak ketiga mereka.

- CMock: kloning (submodule jika Anda menggunakan git) langsung dari repositori Cmock.
- jsmn: tidak direkomendasikan dan tidak lagi didukung.
- lwip: kloningnya (submodule jika Anda menggunakan git) langsung dari repositori lwip-tcpip.
- lwip_osal: lihat Integrasi Referensi Unggulan FreeRTOS di i.MX RT1 060 atau U5 untuk cara mengimplementasikan lwip_osal pada platform/papan perangkat keras Anda. STM32
- mbedtls: kloningnya (submodule jika Anda menggunakan git) langsung dari repositori MBed-TLS. Konfigurasi dan utilitas mbedtls dapat digunakan kembali; buat salinan lokal dalam kasus ini.
- pkcs11: kloningnya (submodule jika Anda menggunakan git) langsung dari PKCS11 pustaka inti atau repositori OASIS PKCS 11.
- tinycbor: kloningnya (submodule jika Anda menggunakan git) langsung dari repositori tinycbor.
- tinycrypt: kami menyarankan Anda menggunakan akselerator kripto dari platform MCU Anda, jika tersedia. Jika Anda ingin terus menggunakan tinycrypt, kloningnya (submodule jika Anda menggunakan git) langsung dari repositori tinycrypt.
- tracealyzer_recorder: kloningnya (submodul jika Anda menggunakan git) langsung dari repositori perekam jejak Percepio.
- unity: kloningnya (submodule jika Anda menggunakan git) langsung dari repositori ThrowTheSwitch/Unity.
- win_pcap: win_pcap tidak lagi dipertahankan. Kami menyarankan Anda menggunakan libslirp, libpcap (posix), atau npcap sebagai gantinya.

Tes porting dan tes integrasi

Semua pengujian di bawah `/tests` folder yang diperlukan untuk memvalidasi integrasi pustaka FreeRTOS dimigrasikan ke repositori. FreeRTOS-Libraries-Integration-Tests Ini dapat digunakan untuk menguji implementasi PAL dan integrasi perpustakaan. Tes yang sama digunakan oleh AWS IoT Device Tester (IDT) untuk Program Kualifikasi AWS Perangkat untuk FreeRTOS.

Dokumentasi yang diarsipkan FreeRTOS

Konten ini diarsipkan dan tidak lagi diperbarui. Ini dapat mereferensikan informasi yang sudah ketinggalan zaman atau fitur yang tidak digunakan lagi. Untuk panduan terbaru tentang FreeRTOS, silakan merujuk ke informasi terbaru dalam Panduan Pengguna FreeRTOS ini atau dokumentasi relevan lainnya. AWS Untuk informasi terkini, lihatApa itu FreeRTOS?.

Topik

- Arsip Panduan Pengguna FreeRTOS
- Konten Panduan Pengguna FreeRTOS sebelumnya

Arsip Panduan Pengguna FreeRTOS

Versi sebelumnya dari Panduan Pengguna FreeRTOS tersedia untuk digunakan dengan rilis FreeRTOS LTS (dukungan jangka panjang).

- Panduan Pengguna FreeRTOS untuk FreeRTOS versi 202210.00
- Panduan Pengguna FreeRTOS untuk FreeRTOS versi 202012.00

Konten Panduan Pengguna FreeRTOS sebelumnya

Konten ini sudah usang tetapi disediakan di sini untuk referensi.

Lihat Memulai dengan FreeRTOS tautan ke konten terbaru.

Memulai dengan FreeRTOS

> **Important**
>
> Halaman ini mengacu pada repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-Freertos yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Tutorial Memulai dengan FreeRTOS ini menunjukkan kepada Anda cara mengunduh dan mengkonfigurasi FreeRTOS pada mesin host, dan kemudian mengkompilasi dan menjalankan aplikasi demo sederhana pada papan mikrokontroler yang memenuhi syarat.

Sepanjang tutorial ini, kami berasumsi bahwa Anda akrab dengan AWS IoT dan AWS IoT konsol. Jika tidak, kami sarankan Anda menyelesaikan tutorial AWS IoT Memulai sebelum melanjutkan.

Topik:

- Aplikasi demo FreeRTOS
- Langkah pertama
- Pemecahan masalah memulai
- Menggunakan CMake dengan FreeRTOS
- Penyediaan kunci mode pengembang
- Panduan memulai khusus dewan
- Langkah selanjutnya dengan FreeRTOS

Aplikasi demo FreeRTOS

> **Important**
>
> Halaman ini mengacu pada repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-Freertos yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Aplikasi demo dalam tutorial ini adalah demo Agen CoreMQTT yang didefinisikan dalam file. *freertos*/demos/coreMQTT_Agent/mqtt_agent_task.c Ini menggunakan Perpustakaan CoreMQTT untuk terhubung ke AWS Cloud dan kemudian secara berkala mempublikasikan pesan ke topik MQTT yang diselenggarakan oleh broker MQTT.AWS IoT

Hanya satu aplikasi demo FreeRTOS yang dapat berjalan pada satu waktu. Saat Anda membangun proyek demo FreeRTOS, demo pertama yang diaktifkan di *freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h file header adalah aplikasi yang berjalan. Untuk tutorial ini, Anda tidak perlu mengaktifkan atau menonaktifkan demo apa pun. Demo Agen CoreMQTT diaktifkan secara default.

Untuk informasi lebih lanjut tentang aplikasi demo yang disertakan dengan FreeRTOS, lihat. Demo FreeRTOS

Langkah pertama

> **Important**
>
> Halaman ini mengacu pada repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-FreerTOS yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Untuk mulai menggunakan FreeRTOS AWS IoT dengan, Anda harus memiliki AWS akun, pengguna dengan izin untuk mengakses AWS IoT dan layanan cloud FreeRTOS. Anda juga harus mengunduh FreeRTOS dan mengonfigurasi proyek demo FreeRTOS papan Anda untuk bekerja dengannya. AWS IoT Bagian berikut memandu Anda melalui persyaratan ini.

> **Note**
>
> - Jika Anda menggunakan Espressif ESP32 - DevKit C,, atau ESP32 -WROOM-32SE ESP-WROVER-KIT, lewati langkah-langkah ini dan pergi ke. Memulai dengan Espressif ESP32 - DevKit C dan ESP-WROVER-KIT
> - Jika Anda menggunakan Nordic n RF5284 0-DK, lewati langkah-langkah ini dan pergi ke. Memulai dengan Nordic n 0-DK RF5284

1. Menyiapkan AWS akun dan izin
2. Mendaftarkan papan MCU Anda dengan AWS IoT
3. Mengunduh FreeRTOS
4. Mengkonfigurasi demo FreeRTOS

Menyiapkan AWS akun dan izin

Mendaftar untuk Akun AWS

Jika Anda tidak memiliki Akun AWS, selesaikan langkah-langkah berikut untuk membuatnya.

Untuk mendaftar untuk Akun AWS

1. Buka https://portal.aws.amazon.com/billing/pendaftaran.
2. Ikuti petunjuk online.

   Bagian dari prosedur pendaftaran melibatkan menerima panggilan telepon atau pesan teks dan memasukkan kode verifikasi pada keypad telepon.

   Saat Anda mendaftar untuk sebuah Akun AWS, sebuah Pengguna root akun AWSdibuat. Pengguna root memiliki akses ke semua Layanan AWS dan sumber daya di akun. Sebagai praktik keamanan terbaik, tetapkan akses administratif ke pengguna, dan gunakan hanya pengguna root untuk melakukan tugas yang memerlukan akses pengguna root.

AWS mengirimi Anda email konfirmasi setelah proses pendaftaran selesai. Kapan saja, Anda dapat melihat aktivitas akun Anda saat ini dan mengelola akun Anda dengan masuk https://aws.amazon.comke/ dan memilih Akun Saya.

Buat pengguna dengan akses administratif

Setelah Anda mendaftar Akun AWS, amankan Pengguna root akun AWS, aktifkan AWS IAM Identity Center, dan buat pengguna administratif sehingga Anda tidak menggunakan pengguna root untuk tugas sehari-hari.

Amankan Pengguna root akun AWS

1. Masuk ke AWS Management Consolesebagai pemilik akun dengan memilih pengguna Root dan memasukkan alamat Akun AWS email Anda. Di laman berikutnya, masukkan kata sandi.

   Untuk bantuan masuk dengan menggunakan pengguna root, lihat Masuk sebagai pengguna root di AWS Sign-In Panduan Pengguna.
2. Mengaktifkan autentikasi multi-faktor (MFA) untuk pengguna root Anda.

   Untuk petunjuk, lihat Mengaktifkan perangkat MFA virtual untuk pengguna Akun AWS root (konsol) Anda di Panduan Pengguna IAM.

Buat pengguna dengan akses administratif

1. Aktifkan Pusat Identitas IAM.

Untuk mendapatkan petunjuk, silakan lihat Mengaktifkan AWS IAM Identity Center di Panduan Pengguna AWS IAM Identity Center .

2. Di Pusat Identitas IAM, berikan akses administratif ke pengguna.

   Untuk tutorial tentang menggunakan Direktori Pusat Identitas IAM sebagai sumber identitas Anda, lihat Mengkonfigurasi akses pengguna dengan default Direktori Pusat Identitas IAM di Panduan AWS IAM Identity Center Pengguna.

Masuk sebagai pengguna dengan akses administratif

- Untuk masuk dengan pengguna Pusat Identitas IAM, gunakan URL masuk yang dikirim ke alamat email saat Anda membuat pengguna Pusat Identitas IAM.

  Untuk bantuan masuk menggunakan pengguna Pusat Identitas IAM, lihat Masuk ke portal AWS akses di Panduan AWS Sign-In Pengguna.

Tetapkan akses ke pengguna tambahan

1. Di Pusat Identitas IAM, buat set izin yang mengikuti praktik terbaik menerapkan izin hak istimewa paling sedikit.

   Untuk petunjuknya, lihat Membuat set izin di Panduan AWS IAM Identity Center Pengguna.

2. Tetapkan pengguna ke grup, lalu tetapkan akses masuk tunggal ke grup.

   Untuk petunjuk, lihat Menambahkan grup di Panduan AWS IAM Identity Center Pengguna.

Untuk memberikan akses dan menambahkan izin bagi pengguna, grup, atau peran Anda:

- Pengguna dan grup di AWS IAM Identity Center:

  Buat rangkaian izin. Ikuti instruksi di Buat rangkaian izin di Panduan Pengguna AWS IAM Identity Center .

- Pengguna yang dikelola di IAM melalui penyedia identitas:

  Buat peran untuk federasi identitas. Ikuti instruksi dalam Buat peran untuk penyedia identitas pihak ketiga (federasi) dalam Panduan Pengguna IAM.

- Pengguna IAM:

- Buat peran yang dapat diambil pengguna Anda. Ikuti instruksi dalam Buat peran untuk pengguna IAM dalam Panduan Pengguna IAM.
- (Tidak disarankan) Lampirkan kebijakan langsung ke pengguna atau tambahkan pengguna ke grup pengguna. Ikuti petunjuk dalam Menambahkan izin ke pengguna (konsol) dalam Panduan Pengguna IAM.

Mendaftarkan papan MCU Anda dengan AWS IoT

Papan Anda harus terdaftar AWS IoT untuk berkomunikasi dengan AWS Cloud. Untuk mendaftarkan papan Anda AWS IoT, Anda harus memiliki:

Sebuah AWS IoT kebijakan

AWS IoT Kebijakan ini memberikan izin perangkat Anda untuk mengakses AWS IoT sumber daya. Itu disimpan di AWS Cloud.

AWS IoT Sesuatu

Suatu AWS IoT hal memungkinkan Anda untuk mengelola perangkat Anda AWS IoT. Itu disimpan di AWS Cloud.

Kunci pribadi dan sertifikat X.509

Kunci pribadi dan sertifikat memungkinkan perangkat Anda untuk mengautentikasi AWS IoT.

Untuk mendaftarkan papan Anda, ikuti prosedur di bawah ini.

Untuk membuat AWS IoT kebijakan

1. Untuk membuat kebijakan IAM, Anda harus mengetahui AWS Wilayah dan nomor AWS akun Anda.

   Untuk menemukan nomor AWS akun Anda, buka Konsol AWS Manajemen, cari dan perluas menu di bawah nama akun Anda di sudut kanan atas, dan pilih Akun Saya. ID akun Anda ditampilkan di bawah Pengaturan Akun.

   Untuk menemukan AWS wilayah untuk AWS akun Anda, gunakan AWS Command Line Interface. Untuk menginstal AWS CLI, ikuti petunjuk di Panduan AWS Command Line Interface Pengguna. Setelah Anda menginstal AWS CLI, buka jendela prompt perintah dan masukkan perintah berikut:

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

Outputnya akan terlihat seperti ini:

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

Dalam contoh ini, wilayahnya adalah`us-west-2`.

> **Note**
>
> Sebaiknya gunakan titik akhir ATS seperti yang terlihat pada contoh.

2. Jelajahi ke AWS IoT konsol.
3. Di panel navigasi, pilih Aman, pilih Kebijakan, lalu pilih Buat.
4. Masukkan nama untuk mengidentifikasi kebijakan Anda.
5. Di bagian Tambahkan pernyataan, pilih Mode lanjutan. Salin dan tempel JSON berikut ke jendela editor kebijakan. Ganti *`aws-region`* dan *`aws-account`* dengan AWS Wilayah dan ID akun Anda.

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

Kebijakan ini memberikan izin berikut:

`iot:Connect`

Memberi perangkat Anda izin untuk terhubung ke broker AWS IoT pesan dengan ID klien apa pun.

`iot:Publish`

Memberi perangkat Anda izin untuk mempublikasikan pesan MQTT pada topik MQTT apa pun.

`iot:Subscribe`

Memberi perangkat Anda izin untuk berlangganan filter topik MQTT apa pun.

`iot:Receive`

Memberi perangkat Anda izin untuk menerima pesan dari broker AWS IoT pesan tentang topik MQTT apa pun.

6. Pilih Buat.

Untuk membuat IoT, kunci pribadi, dan sertifikat untuk perangkat Anda

1. Jelajahi ke AWS IoT konsol.
2. Di panel navigasi, pilih Kelola, lalu pilih Things.
3. Jika Anda tidak memiliki hal-hal IoT yang terdaftar di akun Anda, halaman Anda belum memiliki hal apa pun akan ditampilkan. Jika Anda melihat halaman ini, pilih Daftarkan sesuatu. Jika tidak, pilih Buat.
4. Pada halaman Creating AWS IoT things, pilih Create a single.
5. Pada halaman Tambahkan perangkat Anda ke hal registri, masukkan nama untuk barang Anda, lalu pilih Berikutnya.

6. Pada halaman Tambahkan sertifikat untuk hal Anda, di bawah Pembuatan sertifikat sekali klik, pilih Buat sertifikat.
7. Unduh kunci pribadi dan sertifikat Anda dengan memilih tautan Unduh untuk masing-masing.
8. Pilih Aktifkan untuk mengaktifkan sertifikat Anda. Sertifikat harus diaktifkan sebelum digunakan.
9. Pilih Lampirkan kebijakan untuk melampirkan kebijakan ke sertifikat yang memberi perangkat Anda akses ke AWS IoT operasi.
10. Pilih kebijakan yang baru saja Anda buat dan pilih Daftar hal.

Setelah papan Anda terdaftar AWS IoT, Anda dapat melanjutkanMengunduh FreeRTOS.

Mengunduh FreeRTOS

Anda dapat mengunduh FreeRTOS dari repositori FreeRTOS. GitHub

Setelah Anda mengunduh FreeRTOS, Anda dapat melanjutkan. Mengkonfigurasi demo FreeRTOS

Mengkonfigurasi demo FreeRTOS

Anda harus mengedit beberapa file konfigurasi di direktori FreeRTOS Anda sebelum Anda dapat mengkompilasi dan menjalankan demo apa pun di papan Anda.

Untuk mengonfigurasi titik AWS IoT akhir

Anda harus menyediakan FreeRTOS dengan titik akhir AWS IoT Anda sehingga aplikasi yang berjalan di papan Anda dapat mengirim permintaan ke titik akhir yang benar.

1. Jelajahi ke AWS IoT konsol.
2. Pada panel navigasi kiri, pilih Pengaturan.

   AWS IoT Titik akhir Anda ditampilkan di titik akhir data Perangkat. Itu seharusnya terlihat seperti *1234567890123*`-ats.iot.`*us-east-1*`.amazonaws.com`. Catat titik akhir ini.
3. Di panel navigasi, pilih Kelola, lalu pilih Things.

   Perangkat Anda harus memiliki nama AWS IoT benda. Catat nama ini.
4. Buka `demos/include/aws_clientcredential.h`.
5. Tentukan nilai untuk konstanta berikut:

   - `#define clientcredentialMQTT_BROKER_ENDPOINT "`*Your AWS IoT endpoint*`";`

- `#define clientcredentialIOT_THING_NAME "The AWS IoT thing name of your board"`

Untuk mengkonfigurasi Wi-Fi Anda

Jika papan Anda terhubung ke internet melalui koneksi Wi-Fi, Anda harus menyediakan FreeRTOS dengan kredensyal Wi-Fi untuk terhubung ke jaringan. Jika papan Anda tidak mendukung Wi-Fi, Anda dapat melewati langkah-langkah ini.

1. `demos/include/aws_clientcredential.h`.
2. Tentukan nilai untuk `#define` konstanta berikut:

   - `#define clientcredentialWIFI_SSID "The SSID for your Wi-Fi network"`
   - `#define clientcredentialWIFI_PASSWORD "The password for your Wi-Fi network"`
   - `#define clientcredentialWIFI_SECURITY The security type of your Wi-Fi network`

     Jenis keamanan yang valid adalah:

     - `eWiFiSecurityOpen`(Terbuka, tidak ada keamanan)
     - `eWiFiSecurityWEP`(Keamanan WEP)
     - `eWiFiSecurityWPA`(Keamanan WPA)
     - `eWiFiSecurityWPA2`(WPA2 keamanan)

Untuk memformat AWS IoT kredensi Anda

FreeRTOS harus memiliki AWS IoT sertifikat dan kunci pribadi yang terkait dengan barang terdaftar Anda dan kebijakan izinnya untuk berhasil berkomunikasi AWS IoT dengan atas nama perangkat Anda.

> **Note**
>
> Untuk mengonfigurasi AWS IoT kredensyal Anda, Anda harus memiliki kunci pribadi dan sertifikat yang Anda unduh dari AWS IoT konsol saat Anda mendaftarkan perangkat Anda. Setelah mendaftarkan perangkat sebagai AWS IoT sesuatu, Anda dapat mengambil sertifikat perangkat dari AWS IoT konsol, tetapi Anda tidak dapat mengambil kunci pribadi.

FreeRTOS adalah proyek bahasa C, dan sertifikat dan kunci pribadi harus diformat khusus untuk ditambahkan ke proyek.

1. Di jendela browser, buka`tools/certificate_configuration/CertificateConfigurator.html`.
2. Di bawah file PEM Sertifikat, pilih *`ID`*`-certificate.pem.crt` yang Anda unduh dari AWS IoT konsol.
3. Di bawah file PEM Kunci Pribadi, pilih *`ID`*`-private.pem.key` yang Anda unduh dari AWS IoT konsol.
4. Pilih Generate dan save aws_clientcredential_keys.h, dan kemudian simpan file di. `demos/include` Ini menimpa file yang ada di direktori.

> **Note**
>
> Sertifikat dan kunci pribadi dikodekan keras untuk tujuan demonstrasi saja. Aplikasi tingkat produksi harus menyimpan file-file ini di lokasi yang aman.

Setelah Anda mengonfigurasi FreeRTOS, Anda dapat melanjutkan ke panduan Memulai untuk papan Anda untuk mengatur perangkat keras platform Anda dan lingkungan pengembangan perangkat lunaknya, lalu mengkompilasi dan menjalankan demo di papan Anda. Untuk instruksi khusus papan, lihat. Panduan memulai khusus dewan Aplikasi demo yang digunakan dalam tutorial Memulai adalah demo CoreMQTT Mutual Authentication, yang terletak di. `demos/coreMQTT/mqtt_demo_mutual_auth.c`

Pemecahan masalah memulai

> **Important**
>
> Halaman ini mengacu pada repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-Freertos yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Topik berikut dapat membantu Anda memecahkan masalah yang Anda temui saat memulai dengan FreeRTOS:

Topik

- Kiat pemecahan masalah umum untuk memulai
- Menginstal emulator terminal

Untuk pemecahan masalah khusus papan, lihat Memulai dengan FreeRTOS panduan untuk papan Anda.

Kiat pemecahan masalah umum untuk memulai

Tidak ada pesan yang muncul di AWS IoT konsol setelah saya menjalankan proyek demo Hello World. Apa yang harus saya lakukan?

Coba yang berikut ini:

1. Buka jendela terminal untuk melihat output logging sampel. Ini dapat membantu Anda menentukan apa yang salah.
2. Periksa apakah kredensi jaringan Anda valid.

Log yang ditampilkan di terminal saya saat menjalankan demo terpotong. Bagaimana saya bisa menambah panjangnya?

Tingkatkan nilai `configLOGGING_MAX_MESSAGE_LENGTH` menjadi 255 dalam `FreeRTOSconfig.h` file untuk demo yang Anda jalankan:

```
#define configLOGGING_MAX_MESSAGE_LENGTH    255
```

Menginstal emulator terminal

Emulator terminal dapat membantu Anda mendiagnosis masalah atau memverifikasi bahwa kode perangkat Anda berjalan dengan benar. Ada berbagai emulator terminal yang tersedia untuk Windows, macOS, dan Linux.

Anda harus menghubungkan papan Anda ke komputer Anda sebelum Anda mencoba untuk membuat koneksi serial ke papan Anda dengan emulator terminal.

Gunakan pengaturan berikut untuk mengonfigurasi emulator terminal Anda:

Pengaturan Terminal	Nilai
Tingkat BAUD	115200
Data	8 bit
Paritas	none
Menghentikan	1 bit
Kontrol aliran	none

Menemukan port serial papan Anda

Jika Anda tidak mengetahui port serial board Anda, Anda dapat mengeluarkan salah satu perintah berikut dari baris perintah atau terminal untuk mengembalikan port serial untuk semua perangkat yang terhubung ke komputer host Anda:

Windows

```
chgport
```

Linux

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

macOS

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

Menggunakan CMake dengan FreeRTOS

Important

Halaman ini mengacu pada repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki

proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-Freertos yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Anda dapat menggunakan CMake untuk menghasilkan file build proyek dari kode sumber aplikasi FreeRTOS, dan untuk membangun dan menjalankan kode sumber.

Anda juga dapat menggunakan IDE untuk mengedit, men-debug, mengkompilasi, mem-flash, dan menjalankan kode pada perangkat yang memenuhi syarat Freertos. Setiap panduan Memulai khusus papan mencakup instruksi untuk menyiapkan IDE untuk platform tertentu. Jika Anda lebih suka bekerja tanpa IDE, Anda dapat menggunakan alat pengeditan dan debugging kode pihak ketiga lainnya untuk mengembangkan dan men-debug kode Anda, dan kemudian gunakan CMake untuk membangun dan menjalankan aplikasi.

Dukungan papan berikut CMake:

- Espressif ESP32 - C DevKit
- Espressif ESP-WROVER-KIT
- Kit Konektivitas IoT Infineon XMC48 00
- Kit Pemula Marvell MW32 0 AWS IoT
- Kit Pemula Marvell MW322 AWS IoT
- Bundel MZEF Keingintahuan PIC32 Microchip
- Kit Pengembangan Nordik n RF5284 0 DK
- STMicroelectronicsSTM32Kit Penemuan L4 IoT Node
- Texas Instruments CC322 0SF-LAUNCHXL
- Simulator Microsoft Windows

Lihat topik di bawah ini untuk informasi lebih lanjut tentang penggunaan CMake dengan FreeRTOS.

Topik

- Prasyarat
- Mengembangkan aplikasi FreeRTOS dengan editor kode pihak ketiga dan alat debugging
- Membangun FreeRTOS dengan CMake

Prasyarat

Pastikan mesin host Anda memenuhi prasyarat berikut sebelum melanjutkan:

- Rantai alat kompilasi perangkat Anda harus mendukung sistem operasi mesin. CMake mendukung semua versi Windows, macOS, dan Linux

  Subsistem Windows untuk Linux (WSL) tidak didukung. Gunakan asli CMake pada mesin Windows.
- Anda harus menginstal CMake versi 3.13 atau lebih tinggi.

  Anda dapat mengunduh distribusi biner CMake dari CMake.org.

  > **Note**
  >
  > Jika Anda mengunduh distribusi biner CMake, pastikan Anda menambahkan CMake executable ke variabel lingkungan PATH sebelum Anda menggunakan CMake from command line.

  Anda juga dapat mengunduh dan menginstal CMake menggunakan pengelola paket, seperti homebrew di macOS, dan scoop atau chocolatey di Windows.

  > **Note**
  >
  > Versi CMake paket yang disediakan di manajer paket untuk banyak distribusi Linux adalah out-of-date. Jika manajer paket distribusi Anda tidak menyediakan versi terbaru CMake, Anda dapat mencoba manajer paket alternatif, seperti `linuxbrew` ataunix.
- Anda harus memiliki sistem build asli yang kompatibel.

  CMake dapat menargetkan banyak sistem build asli, termasuk GNU Make atau Ninja. Baik Make dan Ninja dapat diinstal dengan manajer paket di Linux, macOS dan Windows. Jika Anda menggunakan Make di Windows, Anda dapat menginstal versi mandiri dari Equation, atau Anda dapat menginstal MinGW, yang dibuat bundel.

  > **Note**
  >
  > Make executable di MinGW disebut`mingw32-make.exe`, bukan. `make.exe`

Kami menyarankan Anda menggunakan Ninja, karena lebih cepat daripada Make dan juga menyediakan dukungan asli untuk semua sistem operasi desktop.

Mengembangkan aplikasi FreeRTOS dengan editor kode pihak ketiga dan alat debugging

Anda dapat menggunakan editor kode dan ekstensi debugging atau alat debugging pihak ketiga untuk mengembangkan aplikasi untuk FreeRTOS.

Jika, misalnya, Anda menggunakan Visual Studio Code sebagai editor kode Anda, Anda dapat menginstal ekstensi Cortex-Debug VS Code sebagai debugger. Ketika Anda selesai mengembangkan aplikasi Anda, Anda dapat memanggil alat CMake baris perintah untuk membangun proyek Anda dari dalam VS Code. Untuk informasi selengkapnya tentang penggunaan CMake untuk membangun aplikasi FreerTOS, lihat. Membangun FreeRTOS dengan CMake

Untuk debugging, Anda dapat memberikan VS Code dengan konfigurasi debug yang mirip dengan berikut ini:

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

Membangun FreeRTOS dengan CMake

CMake menargetkan sistem operasi host Anda sebagai sistem target secara default. Untuk menggunakannya untuk kompilasi silang, CMake memerlukan file toolchain, yang menentukan kompiler yang ingin Anda gunakan. Di FreeRTOS, kami menyediakan file toolchain default di. *freertos*`/tools/cmake/toolchains` Cara menyediakan file ini CMake tergantung pada apakah Anda menggunakan antarmuka baris CMake perintah atau GUI. Untuk lebih jelasnya, ikuti Menghasilkan file build (CMake alat baris perintah) petunjuk di bawah ini. Untuk informasi selengkapnya tentang kompilasi silang CMake, lihat CrossCompilingdi wiki resmi CMake .

Membangun proyek CMake berbasis

1. Jalankan CMake untuk menghasilkan file build untuk sistem build asli, seperti Make atau Ninja.

   Anda dapat menggunakan alat CMake baris perintah atau CMake GUI untuk membuat file build untuk sistem build asli Anda.

   Untuk informasi tentang membuat file build FreeRTOS, lihat dan. Menghasilkan file build (CMake alat baris perintah) Menghasilkan file build (CMake GUI)
2. Panggil sistem build asli untuk membuat proyek menjadi executable.

   Untuk informasi tentang membuat file build FreeRTOS, lihat. Membangun FreeRTOS dari file build yang dihasilkan

Menghasilkan file build (CMake alat baris perintah)

Anda dapat menggunakan alat CMake baris perintah (cmake) untuk menghasilkan file build untuk FreeRTOS. Untuk menghasilkan file build, Anda perlu menentukan papan target, kompiler, dan lokasi kode sumber dan direktori build.

Anda dapat menggunakan opsi berikut untuk cmake:

- `-DVENDOR`- Menentukan papan target.
- `-DCOMPILER`- Menentukan compiler.
- `-S`— Menentukan lokasi kode sumber.
- `-B`— Menentukan lokasi file build yang dihasilkan.

> **Note**
>
> Kompiler harus dalam `PATH` variabel sistem, atau Anda harus menentukan lokasi kompiler.

Misalnya, jika vendornya adalah Texas Instruments, dan papannya adalah CC322 0 Launchpad, dan kompilernya adalah GCC untuk ARM, Anda dapat mengeluarkan perintah berikut untuk membangun file sumber dari direktori saat ini ke direktori bernama: *`build-directory`*

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

> **Note**
>
> Jika Anda menggunakan Windows, Anda harus menentukan sistem build asli karena CMake menggunakan Visual Studio secara default. Sebagai contoh:
>
> ```
> cmake -DVENDOR=ti -DBOARD=cc3220_launchpad -DCOMPILER=arm-ti -S . -B build-directory -G Ninja
> ```
>
> Atau:
>
> ```
> cmake -DVENDOR=ti -DBOARD=cc3220_launchpad -DCOMPILER=arm-ti -S . -B build-directory -G "MinGW Makefiles"
> ```

Ekspresi reguler `${VENDOR}.*` dan `${BOARD}.*` digunakan untuk mencari papan yang cocok, jadi Anda tidak perlu menggunakan nama lengkap vendor dan papan untuk `BOARD` opsi `VENDOR` dan opsi. Nama sebagian berfungsi, asalkan ada satu kecocokan. Misalnya, perintah berikut menghasilkan file build yang sama dari sumber yang sama:

```
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
```

Anda dapat menggunakan `CMAKE_TOOLCHAIN_FILE` opsi jika Anda ingin menggunakan file toolchain yang tidak terletak di direktori `cmake/toolchains` default. Sebagai contoh:

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

Jika file toolchain tidak menggunakan jalur absolut untuk kompiler Anda, dan Anda tidak menambahkan kompiler Anda ke variabel `PATH` lingkungan, CMake mungkin tidak dapat menemukannya. Untuk memastikan bahwa CMake menemukan file toolchain Anda, Anda dapat menggunakan `AFR_TOOLCHAIN_PATH` opsi. Opsi ini mencari jalur direktori toolchain yang ditentukan dan subfolder toolchain di bawahnya. `bin` Sebagai contoh:

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

Untuk mengaktifkan debugging, atur CMAKE_BUILD_TYPE kedebug. Dengan opsi ini diaktifkan, CMake menambahkan flag debug ke opsi kompilasi, dan membangun FreeRTOS dengan simbol debug.

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

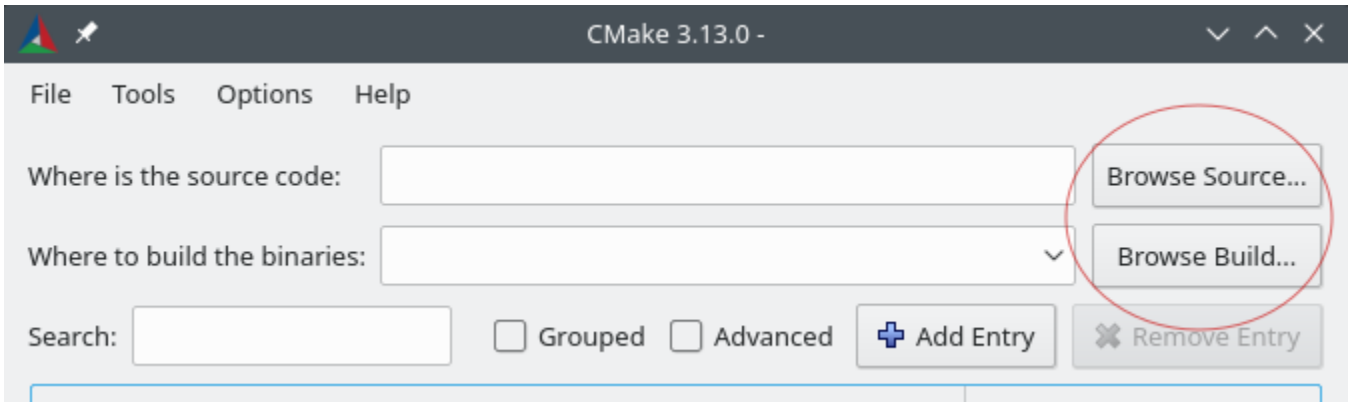
Anda juga dapat mengatur CMAKE_BUILD_TYPE to `release` untuk menambahkan flag optimasi ke opsi kompilasi.

Menghasilkan file build (CMake GUI)

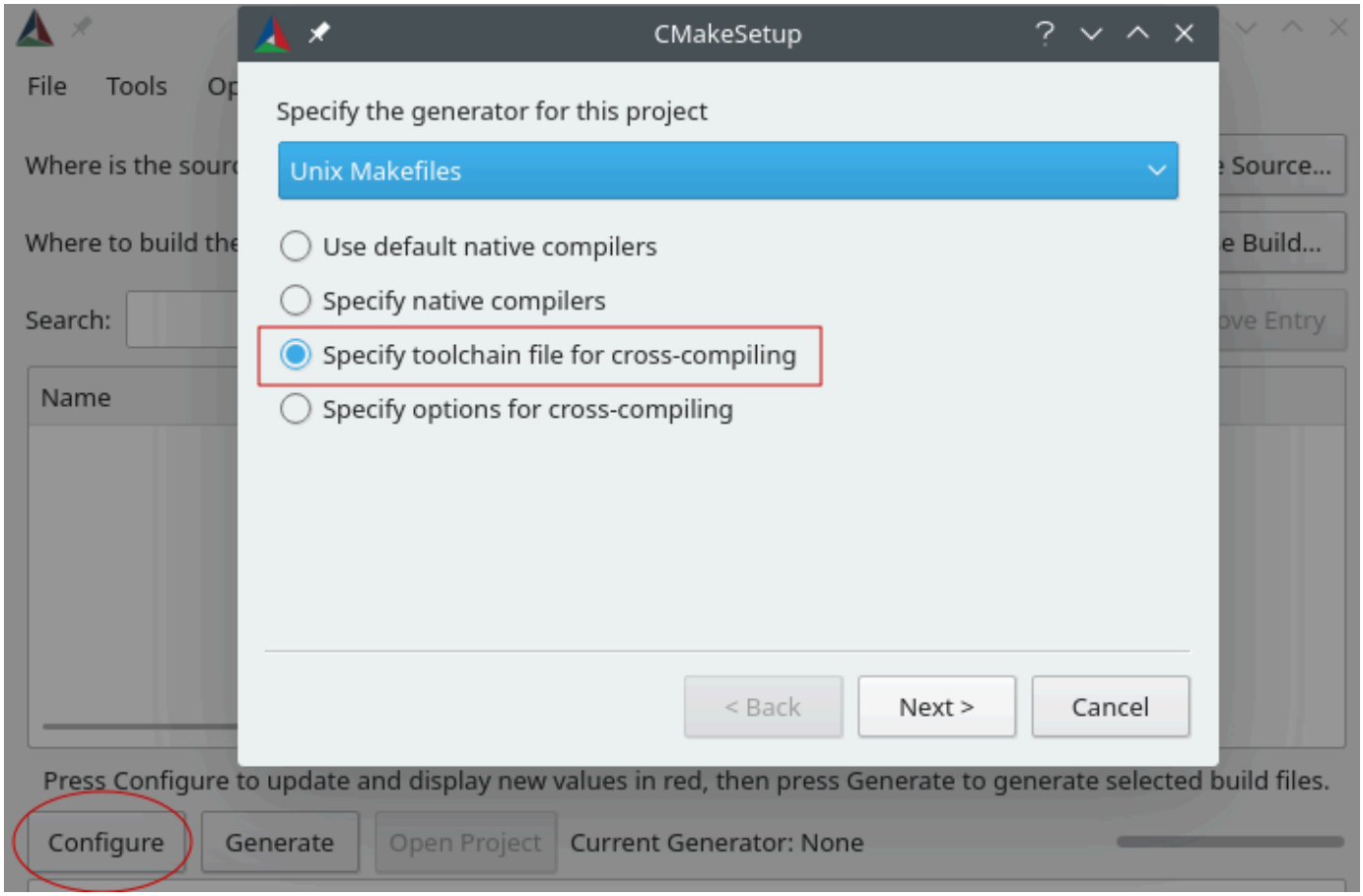
Anda dapat menggunakan CMake GUI untuk menghasilkan file build FreeRTOS.

Untuk menghasilkan file build dengan CMake GUI

1. Dari baris perintah, masalah `cmake-gui` untuk memulai GUI.
2. Pilih Browse Source dan tentukan input sumber, lalu pilih Browse Build dan tentukan output build.



3. Pilih Konfigurasi, dan di bawah Tentukan generator build untuk proyek ini, temukan dan pilih sistem build yang ingin Anda gunakan untuk membangun file build yang dihasilkan. jika Anda tidak melihat jendela pop up, Anda mungkin menggunakan kembali direktori build yang ada. Dalam hal ini, hapus CMake cache dengan memilih Hapus Cache dari menu File.



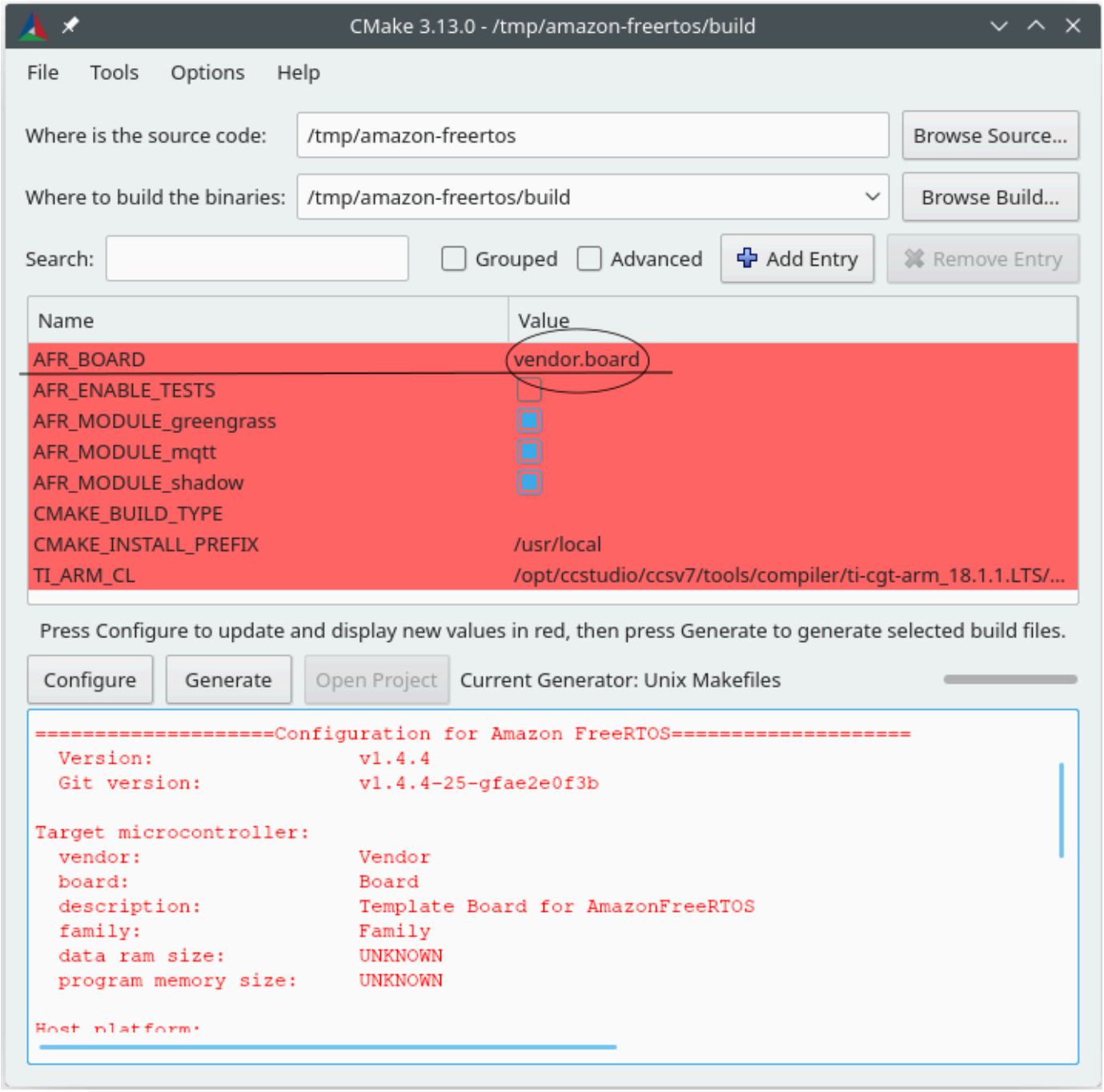
4. Pilih Tentukan file toolchain untuk kompilasi silang, lalu pilih Berikutnya.
5. Pilih file toolchain (misalnya,*freertos*`/tools/cmake/toolchains/arm-ti.cmake`), lalu pilih Selesai.

   Konfigurasi default untuk FreeRTOS adalah papan templat, yang tidak menyediakan target lapisan portabel apa pun. Akibatnya, sebuah jendela muncul dengan pesanError in configuration process.

> **Note**
>
> Jika Anda melihat kesalahan berikut:
>
> ```
> CMake Error at tools/cmake/toolchains/find_compiler.cmake:23 (message):
> Compiler not found, you can specify search path with AFR_TOOLCHAIN_PATH.
> ```

Ini berarti kompiler tidak ada dalam variabel PATH lingkungan Anda. Anda dapat mengatur AFR_TOOLCHAIN_PATH variabel dalam GUI untuk memberi tahu CMake di mana Anda menginstal kompiler Anda. Jika Anda tidak melihat AFR_TOOLCHAIN_PATH variabelnya, pilih Tambah Entri. Di jendela pop up, di bawah Nama, ketik**AFR_TOOLCHAIN_PATH**. Di bawah Compiler Path ketik jalur ke kompiler Anda. misalnya,. `C:/toolchains/arm-none-eabi-gcc`

6. GUI sekarang akan terlihat seperti ini:



Pilih AFR_BOARD, pilih papan Anda, lalu pilih Konfigurasi lagi.

7. Pilih Hasilkan. CMake menghasilkan file sistem build (misalnya, file makefiles atau ninja), dan file-file ini muncul di direktori build yang Anda tentukan pada langkah pertama. Ikuti petunjuk di bagian selanjutnya untuk menghasilkan gambar biner.

Membangun FreeRTOS dari file build yang dihasilkan

Membangun dengan sistem build asli

Anda dapat membangun FreeRTOS dengan sistem build asli dengan memanggil perintah sistem build dari direktori binari keluaran.

Misalnya, jika direktori keluaran file build Anda`<build_dir>`, dan Anda menggunakan Make sebagai sistem build asli, jalankan perintah berikut:

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

Membangun dengan CMake

Anda juga dapat menggunakan alat CMake baris perintah untuk membangun FreeRTOS. CMake menyediakan lapisan abstraksi untuk memanggil sistem build asli. Sebagai contoh:

```
cmake --build build_dir
```

Berikut adalah beberapa kegunaan umum lainnya dari CMake mode build tool baris perintah:

```
# 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
```

Untuk informasi selengkapnya tentang mode CMake build, lihat CMake dokumentasi.

Penyediaan kunci mode pengembang

> **Important**
>
> Halaman ini mengacu pada repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-Freertos yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Pengantar

Bagian ini membahas dua opsi untuk mendapatkan sertifikat klien X.509 tepercaya ke perangkat IoT untuk pengujian lab. Bergantung pada kemampuan perangkat, berbagai operasi terkait penyediaan mungkin atau mungkin tidak didukung, termasuk pembuatan kunci ECDSA onboard, impor kunci pribadi, dan pendaftaran sertifikat X.509. Selain itu, kasus penggunaan yang berbeda membutuhkan tingkat perlindungan kunci yang berbeda, mulai dari penyimpanan flash onboard hingga penggunaan perangkat keras kripto khusus. Bagian ini memberikan logika untuk bekerja dalam kemampuan kriptografi perangkat Anda.

Opsi #1: impor kunci pribadi dari AWS IoT

Untuk tujuan pengujian lab, jika perangkat Anda mengizinkan impor kunci pribadi, ikuti petunjuknyaMengkonfigurasi demo FreeRTOS.

Opsi #2: pembuatan kunci pribadi onboard

Jika perangkat Anda memiliki elemen aman, atau jika Anda lebih suka membuat key pair dan sertifikat perangkat Anda sendiri, ikuti petunjuknya di sini.

Konfigurasi Awal

Pertama, lakukan langkah-langkahnyaMengkonfigurasi demo FreeRTOS, tetapi lewati langkah terakhir (yaitu, jangan lakukan Untuk memformat AWS IoT kredensil Anda). Hasil bersihnya adalah `demos/include/aws_clientcredential.h` file tersebut telah diperbarui dengan pengaturan Anda, tetapi `demos/include/aws_clientcredential_keys.h` file tersebut belum.

Konfigurasi Proyek Demo

Buka demo CoreMQTT Mutual Authentication seperti yang dijelaskan dalam panduan untuk papan Anda di. Panduan memulai khusus dewan Dalam proyek, buka file `aws_dev_mode_key_provisioning.c` dan ubah definisi`keyprovisioningFORCE_GENERATE_NEW_KEY_PAIR`, yang diatur ke nol secara default, menjadi satu:

```
#define keyprovisioningFORCE_GENERATE_NEW_KEY_PAIR 1
```

Kemudian bangun dan jalankan proyek demo dan lanjutkan ke langkah berikutnya.

Ekstraksi Kunci Publik

Karena perangkat belum disediakan dengan kunci pribadi dan sertifikat klien, demo akan gagal untuk mengautentikasi. AWS IoT Namun, demo CoreMQTT Mutual Authentication dimulai dengan menjalankan penyediaan kunci mode pengembang, menghasilkan pembuatan kunci pribadi jika belum ada. Anda akan melihat sesuatu seperti berikut di dekat awal output konsol serial.

```
7 910 [IP-task] Device public key, 91 bytes:
3059 3013 0607 2a86 48ce 3d02 0106 082a
8648 ce3d 0301 0703 4200 04cd 6569 ceb8
1bb9 1e72 339f e8cf 60ef 0f9f b473 33ac
6f19 1813 6999 3fa0 c293 5fae 08f1 1ad0
41b7 345c e746 1046 228e 5a5f d787 d571
dcb2 4e8d 75b3 2586 e2cc 0c
```

Salin enam baris byte kunci ke dalam file bernama`DevicePublicKeyAsciiHex.txt`. Kemudian gunakan alat baris perintah “xxd” untuk mengurai byte hex menjadi biner:

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

Gunakan “openssl” untuk memformat kunci publik perangkat biner yang dikodekan (DER) sebagai PEM:

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

Jangan lupa untuk menonaktifkan pengaturan pembuatan kunci sementara yang Anda aktifkan di atas. Jika tidak, perangkat akan membuat key pair lain, dan Anda harus mengulangi langkah-langkah sebelumnya:

```
#define keyprovisioningFORCE_GENERATE_NEW_KEY_PAIR 0
```

Pengaturan Infrastruktur Kunci Publik

Ikuti petunjuk dalam Mendaftarkan Sertifikat CA Anda untuk membuat hierarki sertifikat untuk sertifikat lab perangkat Anda. Berhenti sebelum menjalankan urutan yang dijelaskan di bagian Membuat Sertifikat Perangkat Menggunakan Sertifikat CA Anda.

Dalam hal ini, perangkat tidak akan menandatangani permintaan sertifikat (yaitu, Permintaan Layanan Sertifikat atau CSR) karena logika pengkodean X.509 yang diperlukan untuk membuat dan menandatangani CSR telah dikeluarkan dari proyek demo FreeRTOS untuk mengurangi ukuran ROM. Sebagai gantinya, untuk tujuan pengujian lab, buat kunci pribadi di workstation Anda dan gunakan untuk menandatangani CSR.

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

Setelah Otoritas Sertifikat dibuat dan didaftarkan AWS IoT, gunakan perintah berikut untuk menerbitkan sertifikat klien berdasarkan CSR perangkat yang ditandatangani pada langkah sebelumnya:

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

Meskipun CSR ditandatangani dengan kunci pribadi sementara, sertifikat yang dikeluarkan hanya dapat digunakan dengan kunci pribadi perangkat yang sebenarnya. Mekanisme yang sama dapat digunakan dalam produksi jika Anda menyimpan kunci penandatangan CSR di perangkat keras terpisah, dan mengonfigurasi otoritas sertifikat Anda sehingga hanya mengeluarkan sertifikat untuk permintaan yang telah ditandatangani oleh kunci tertentu. Kunci itu juga harus tetap berada di bawah kendali administrator yang ditunjuk.

Impor Sertifikat

Dengan sertifikat yang dikeluarkan, langkah selanjutnya adalah mengimpornya ke perangkat Anda. Anda juga perlu mengimpor sertifikat Certificate Authority (CA) Anda,

karena diperlukan agar otentikasi pertama kali AWS IoT berhasil saat menggunakan JITP. Dalam `aws_clientcredential_keys.h` file dalam proyek Anda, atur `keyCLIENT_CERTIFICATE_PEM` makro menjadi isi DeviceCert.pem dan atur makro menjadi isi dari`keyJITR_DEVICE_CERTIFICATE_AUTHORITY_PEM`. `rootCA.pem`

Otorisasi Perangkat

Impor `deviceCert.pem` ke AWS IoT registri seperti yang dijelaskan dalam Gunakan Sertifikat Anda Sendiri. Anda harus membuat AWS IoT hal baru, melampirkan sertifikat PENDING dan kebijakan untuk barang Anda, lalu tandai sertifikat sebagai AKTIF. Semua langkah ini dapat dilakukan secara manual di AWS IoT konsol.

Setelah sertifikat klien baru AKTIF dan terkait dengan sesuatu dan kebijakan, jalankan demo CoreMQTT Mutual Authentication lagi. Kali ini, koneksi ke broker AWS IoT MQTT akan berhasil.

Panduan memulai khusus dewan

> **Important**
>
> Halaman ini mengacu pada repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-Freertos yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Setelah Anda menyelesaikanLangkah pertama, lihat panduan papan Anda untuk instruksi khusus papan tentang memulai dengan FreeRTOS:

- Memulai dengan Kit Pengembangan Cypress CYW9439 07 AEVAL1 F
- Memulai dengan Kit Pengembangan Cypress CYW9549 07 AEVAL1 F
- Memulai dengan kit Cypress CY8 CKIT-064S0S2-4343W
- Memulai dengan Infineon XMC48 00 IoT Connectivity Kit
- Memulai dengan MW32x AWS IoT Starter Kit
- Memulai dengan kit pengembangan MediaTek MT7697 Hx
- Memulai dengan Microchip Curiosity PIC32 MZ EF
- Memulai dengan NuMaker-IoT-M Nuvoton 487

- Memulai dengan Modul LPC54 IoT NXP 018
- Memulai dengan Renesas Starter Kit+untuk N-2MB RX65
- Memulai dengan STMicroelectronics STM32 L4 Discovery Kit IoT Node
- Memulai dengan Texas Instruments CC322 0SF-LAUNCHXL
- Memulai dengan Simulator Perangkat Windows
- Memulai dengan Xilinx Avnet MicroZed Industrial IoT Kit

> **Note**
>
> Anda tidak perlu menyelesaikan panduan Memulai dengan FreeRTOS mandiri berikut ini: Langkah pertama
>
> - Memulai dengan Microchip ATECC6 08A Secure Element dengan simulator Windows
> - Memulai dengan Espressif ESP32 - DevKit C dan ESP-WROVER-KIT
> - Memulai dengan ESP32 Espressif -WROOM-32SE
> - Memulai dengan ESP32 Espressif -S2
> - Memulai dengan Infineon OPTIGA Trust X dan 00 XMC48 IoT Connectivity Kit
> - Memulai dengan Nordic n 0-DK RF5284

Memulai dengan Kit Pengembangan Cypress CYW9439 07 AEVAL1 F

> **Important**
>
> Integrasi referensi ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-Freertos yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Tutorial ini memberikan instruksi untuk memulai dengan Cypress CYW9439 07 AEVAL1 F Development Kit. Jika Anda tidak memiliki Kit Pengembangan Cypress CYW9439 07 AEVAL1 F, kunjungi Katalog Perangkat AWS Mitra untuk membelinya dari mitra kami.

> **Note**
>
> Tutorial ini memandu Anda melalui pengaturan dan menjalankan demo CoreMQTT Mutual Authentication. Port FreeRTOS untuk papan ini saat ini tidak mendukung demo server dan klien TCP.

Sebelum memulai, Anda harus mengonfigurasi AWS IoT dan mengunduh FreeRTOS Anda untuk menghubungkan perangkat Anda ke Cloud. AWS Lihat Langkah pertama untuk instruksi.

> **Important**
>
> - Dalam topik ini, jalur ke direktori unduhan FreerTOS disebut sebagai. *`freertos`*
> - Karakter ruang di *`freertos`* jalur dapat menyebabkan kegagalan build. Saat Anda mengkloning atau menyalin repositori, pastikan jalur yang Anda buat tidak berisi karakter spasi.
> - Panjang maksimum jalur file di Microsoft Windows adalah 260 karakter. Jalur direktori unduhan FreerTOS yang panjang dapat menyebabkan kegagalan build.
> - Karena kode sumber mungkin berisi tautan simbolis, jika Anda menggunakan Windows untuk mengekstrak arsip, Anda mungkin harus:
>   - Aktifkan Mode Pengembang atau,
>   - Gunakan konsol yang ditinggikan sebagai administrator.
>
>   Dengan cara ini, Windows dapat membuat tautan simbolis dengan benar saat mengekstrak arsip. Jika tidak, tautan simbolis akan ditulis sebagai file normal yang berisi jalur tautan simbolis sebagai teks atau kosong. Untuk informasi lebih lanjut, lihat entri blog Symlinks di Windows 10! .
>
>   Jika Anda menggunakan Git di bawah Windows, Anda harus mengaktifkan Mode Pengembang atau Anda harus:
>   - Setel `core.symlinks` ke true dengan perintah berikut:
>
>     ```
>     git config --global core.symlinks true
>     ```
>   - Gunakan konsol yang ditinggikan sebagai administrator setiap kali Anda menggunakan perintah git yang menulis ke sistem (misalnya,git pull,git clone, dangit submodule update --init --recursive).

- Seperti disebutkan dalamMengunduh FreeRTOS, port FreeRTOS untuk Cypress saat ini hanya tersedia di. GitHub

Gambaran Umum

Tutorial ini berisi petunjuk untuk langkah-langkah memulai berikut:

1. Menginstal perangkat lunak pada mesin host untuk mengembangkan dan men-debug aplikasi tertanam untuk papan mikrokontroler Anda.
2. Menyusun silang aplikasi demo FreeRTOS ke gambar biner.
3. Memuat gambar biner aplikasi ke papan Anda, dan kemudian menjalankan aplikasi.
4. Berinteraksi dengan aplikasi yang berjalan di papan Anda di seluruh koneksi serial, untuk tujuan pemantauan dan debugging.

Menyiapkan lingkungan pengembangan Anda

Unduh dan instal WICED Studio SDK

Dalam panduan Memulai ini, Anda menggunakan Cypress WICED Studio SDK untuk memprogram papan Anda dengan demo FreeRTOS. Kunjungi situs web WICED Software untuk mengunduh WICED Studio SDK dari Cypress. Anda harus mendaftar untuk akun Cypress gratis untuk mengunduh perangkat lunak. WICED Studio SDK kompatibel dengan sistem operasi Windows, macOS, dan Linux.

> **Note**
>
> Beberapa sistem operasi memerlukan langkah-langkah instalasi tambahan. Pastikan Anda membaca dan mengikuti semua petunjuk instalasi untuk sistem operasi dan versi WICED Studio yang sedang Anda instal.

Tetapkan variabel lingkungan

Sebelum Anda menggunakan WICED Studio untuk memprogram papan Anda, Anda harus membuat variabel lingkungan untuk direktori instalasi WICED Studio SDK. Jika WICED Studio berjalan saat Anda membuat variabel Anda, Anda perlu me-restart aplikasi setelah Anda mengatur variabel Anda.

> **Note**
>
> Penginstal WICED Studio membuat dua folder terpisah yang diberi nama `WICED-Studio-`*`m`*`.`*`n`* pada mesin Anda di mana `m` dan `n` merupakan nomor versi mayor dan minor masing-masing. Dokumen ini mengasumsikan nama folder `WICED-Studio-6.2` tetapi pastikan untuk menggunakan nama yang benar untuk versi yang Anda instal. Saat Anda menentukan variabel `WICED_STUDIO_SDK_PATH` lingkungan, pastikan untuk menentukan jalur instalasi lengkap WICED Studio SDK, dan bukan jalur instalasi WICED Studio IDE. Di Windows dan macOS, `WICED-Studio-`*`m`*`.`*`n`* folder untuk SDK dibuat di `Documents` folder secara default.

Untuk membuat variabel lingkungan pada Windows

1. Buka Control Panel, pilih System, lalu pilih Advanced System Settings.
2. Pada tab Advanced, pilih Variabel Lingkungan.
3. Di bawah Variabel pengguna, pilih Baru.
4. Untuk nama Variabel, masukkan**`WICED_STUDIO_SDK_PATH`**. Untuk nilai Variabel, masukkan direktori instalasi WICED Studio SDK.

Untuk membuat variabel lingkungan di Linux atau macOS

1. Buka `/etc/profile` file di mesin Anda, dan tambahkan yang berikut ini ke baris terakhir file:

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

2. Mulai ulang mesin Anda.
3. Buka terminal dan jalankan perintah berikut:

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

   ```
   perl platform_adjust_make.pl
   ```

   ```
   chmod +x make
   ```

Membangun koneksi serial

Untuk membuat koneksi serial antara mesin host Anda dan papan Anda

1. Hubungkan papan ke komputer host Anda dengan kabel USB Standard-A ke Micro-B.
2. Identifikasi nomor port seri USB untuk koneksi ke papan di komputer host Anda.
3. Mulai terminal serial dan buka koneksi dengan pengaturan berikut:
   - Tingkat baud: 115200
   - Data: 8 bit
   - Paritas: Tidak ada
   - Hentikan bit: 1
   - Kontrol aliran: Tidak ada

Untuk informasi selengkapnya tentang memasang terminal dan menyiapkan koneksi serial, lihatMenginstal emulator terminal.

Memantau pesan MQTT di cloud

Sebelum menjalankan proyek demo FreeRTOS, Anda dapat mengatur klien MQTT di konsol untuk memantau pesan AWS IoT yang dikirim perangkat Anda ke Cloud. AWS

Untuk berlangganan topik MQTT dengan klien MQTT AWS IoT

1. Masuk ke konsol AWS IoT tersebut.
2. Di panel navigasi, pilih Uji, lalu pilih klien pengujian MQTT untuk membuka klien MQTT.
3. Dalam Subscription topic, masukkan***your-thing-name*****/example/topic**, lalu pilih Subscribe to topic.

Bangun dan jalankan proyek demo FreeRTOS

Setelah Anda mengatur koneksi serial ke papan Anda, Anda dapat membangun proyek demo FreeRTOS, mem-flash demo ke papan Anda, dan kemudian menjalankan demo.

Untuk membangun dan menjalankan proyek demo FreeRTOS di WICED Studio

1. Luncurkan WICED Studio.

2. Dari menu File, pilih Impor. Perluas `General` folder, pilih Existing Projects ke Workspace, lalu pilih Next.
3. Di Pilih direktori root, pilih Browse... , navigasikan ke jalur*freertos*`/projects/cypress/CYW943907AEVAL1F/wicedstudio`, lalu pilih OK.
4. Di bawah Proyek, centang kotak hanya untuk proyek aws_demo. Pilih Selesai untuk mengimpor proyek. Proyek target aws_demo akan muncul di jendela Make Target.
5. Buka menu Platform WICED dan pilih Filter WICED nonaktif.
6. Di jendela Make Target, perluas aws_demo, klik kanan `demo.aws_demo` file, lalu pilih Build Target untuk membangun dan mengunduh demo ke papan Anda. Demo akan berjalan secara otomatis setelah dibangun dan diunduh ke papan Anda.

Pemecahan Masalah

- Jika Anda menggunakan Windows, Anda mungkin menerima kesalahan berikut ketika Anda membangun dan menjalankan proyek demo:

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

  Untuk memecahkan masalah kesalahan ini, lakukan hal berikut:

  1. Jelajahi *WICED-Studio-SDK-PATH*`\WICED-Studio-6.2\43xxx_Wi-Fi\tools\OpenOCD\Win32` dan klik dua kali. `openocd-all-brcm-libftdi.exe`
  2. Jelajahi *WICED-Studio-SDK-PATH*`\WICED-Studio-6.2\43xxx_Wi-Fi\tools\drivers\CYW9WCD1EVAL1` dan klik dua kali. `InstallDriver.exe`
- Jika Anda menggunakan Linux atau macOS, Anda mungkin menerima kesalahan berikut saat membuat dan menjalankan proyek demo:

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

  Untuk memecahkan masalah kesalahan ini, gunakan perintah berikut untuk memperbarui paket libusb-dev:

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

Untuk informasi pemecahan masalah umum tentang Memulai FreeRTOS, lihat. Pemecahan masalah memulai

Memulai dengan Kit Pengembangan Cypress CYW9549 07 AEVAL1 F

> **Important**
>
> Integrasi referensi ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-Freertos yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Tutorial ini memberikan instruksi untuk memulai dengan Cypress CYW9549 07 AEVAL1 F Development Kit. Jika Anda tidak memiliki Kit Pengembangan Cypress CYW9549 07 AEVAL1 F, kunjungi Katalog Perangkat AWS Mitra untuk membelinya dari mitra kami.

> **Note**
>
> Tutorial ini memandu Anda melalui pengaturan dan menjalankan demo CoreMQTT Mutual Authentication. Port FreeRTOS untuk papan ini saat ini tidak mendukung demo server dan klien TCP.

Sebelum memulai, Anda harus mengonfigurasi AWS IoT dan mengunduh FreeRTOS Anda untuk menghubungkan perangkat Anda ke Cloud. AWS Lihat Langkah pertama untuk instruksi. Dalam tutorial ini, jalur ke direktori unduhan FreeRTOS disebut sebagai. *`freertos`*

> **Important**
>
> - Dalam topik ini, jalur ke direktori unduhan FreeRTOS disebut sebagai. *`freertos`*
> - Karakter ruang di *`freertos`* jalur dapat menyebabkan kegagalan build. Saat Anda mengkloning atau menyalin repositori, pastikan jalur yang Anda buat tidak berisi karakter spasi.
> - Panjang maksimum jalur file di Microsoft Windows adalah 260 karakter. Jalur direktori unduhan FreeRTOS yang panjang dapat menyebabkan kegagalan build.
> - Karena kode sumber mungkin berisi tautan simbolis, jika Anda menggunakan Windows untuk mengekstrak arsip, Anda mungkin harus:

- Aktifkan Mode Pengembang atau,
- Gunakan konsol yang ditinggikan sebagai administrator.

Dengan cara ini, Windows dapat membuat tautan simbolis dengan benar saat mengekstrak arsip. Jika tidak, tautan simbolis akan ditulis sebagai file normal yang berisi jalur tautan simbolis sebagai teks atau kosong. Untuk informasi lebih lanjut, lihat entri blog Symlinks di Windows 10! .

Jika Anda menggunakan Git di bawah Windows, Anda harus mengaktifkan Mode Pengembang atau Anda harus:

- Setel `core.symlinks` ke true dengan perintah berikut:

```
git config --global core.symlinks true
```

- Gunakan konsol yang ditinggikan sebagai administrator setiap kali Anda menggunakan perintah git yang menulis ke sistem (misalnya,git pull,git clone, dangit submodule update --init --recursive).

- Seperti disebutkan dalamMengunduh FreeRTOS, port FreeRTOS untuk Cypress saat ini hanya tersedia di. GitHub

Gambaran Umum

Tutorial ini berisi petunjuk untuk langkah-langkah memulai berikut:

1. Menginstal perangkat lunak pada mesin host untuk mengembangkan dan men-debug aplikasi tertanam untuk papan mikrokontroler Anda.
2. Menyusun silang aplikasi demo FreeRTOS ke gambar biner.
3. Memuat gambar biner aplikasi ke papan Anda, dan kemudian menjalankan aplikasi.
4. Berinteraksi dengan aplikasi yang berjalan di papan Anda di seluruh koneksi serial, untuk tujuan pemantauan dan debugging.

Menyiapkan lingkungan pengembangan Anda

Unduh dan instal WICED Studio SDK

Dalam panduan Memulai ini, Anda menggunakan Cypress WICED Studio SDK untuk memprogram papan Anda dengan demo FreeRTOS. Kunjungi situs web WICED Software untuk mengunduh

WICED Studio SDK dari Cypress. Anda harus mendaftar untuk akun Cypress gratis untuk mengunduh perangkat lunak. WICED Studio SDK kompatibel dengan sistem operasi Windows, macOS, dan Linux.

> **Note**
>
> Beberapa sistem operasi memerlukan langkah-langkah instalasi tambahan. Pastikan Anda membaca dan mengikuti semua petunjuk instalasi untuk sistem operasi dan versi WICED Studio yang sedang Anda instal.

Tetapkan variabel lingkungan

Sebelum Anda menggunakan WICED Studio untuk memprogram papan Anda, Anda harus membuat variabel lingkungan untuk direktori instalasi WICED Studio SDK. Jika WICED Studio berjalan saat Anda membuat variabel Anda, Anda perlu me-restart aplikasi setelah Anda mengatur variabel Anda.

> **Note**
>
> Penginstal WICED Studio membuat dua folder terpisah yang diberi nama `WICED-Studio-`*`m`*`.`*`n`* pada mesin Anda di mana `m` dan `n` merupakan nomor versi mayor dan minor masing-masing. Dokumen ini mengasumsikan nama folder `WICED-Studio-6.2` tetapi pastikan untuk menggunakan nama yang benar untuk versi yang Anda instal. Saat Anda menentukan variabel `WICED_STUDIO_SDK_PATH` lingkungan, pastikan untuk menentukan jalur instalasi lengkap WICED Studio SDK, dan bukan jalur instalasi WICED Studio IDE. Di Windows dan macOS, `WICED-Studio-`*`m`*`.`*`n`* folder untuk SDK dibuat di `Documents` folder secara default.

Untuk membuat variabel lingkungan pada Windows

1. Buka Control Panel, pilih System, lalu pilih Advanced System Settings.
2. Pada tab Advanced, pilih Variabel Lingkungan.
3. Di bawah Variabel pengguna, pilih Baru.
4. Untuk nama Variabel, masukkan**`WICED_STUDIO_SDK_PATH`**. Untuk nilai Variabel, masukkan direktori instalasi WICED Studio SDK.

Untuk membuat variabel lingkungan di Linux atau macOS

1. Buka `/etc/profile` file di mesin Anda, dan tambahkan yang berikut ini ke baris terakhir file:

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

2. Mulai ulang mesin Anda.
3. Buka terminal dan jalankan perintah berikut:

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

```
perl platform_adjust_make.pl
```

```
chmod +x make
```

Membangun koneksi serial

Untuk membuat koneksi serial antara mesin host Anda dan papan Anda

1. Hubungkan papan ke komputer host Anda dengan kabel USB Standard-A ke Micro-B.
2. Identifikasi nomor port seri USB untuk koneksi ke papan di komputer host Anda.
3. Mulai terminal serial dan buka koneksi dengan pengaturan berikut:
   - Tingkat baud: 115200
   - Data: 8 bit
   - Paritas: Tidak ada
   - Hentikan bit: 1
   - Kontrol aliran: Tidak ada

Untuk informasi selengkapnya tentang memasang terminal dan menyiapkan koneksi serial, lihatMenginstal emulator terminal.

Memantau pesan MQTT di cloud

Sebelum menjalankan proyek demo FreeRTOS, Anda dapat mengatur klien MQTT di konsol untuk memantau pesan AWS IoT yang dikirim perangkat Anda ke Cloud. AWS

Untuk berlangganan topik MQTT dengan klien MQTT AWS IoT

1. Masuk ke konsol AWS IoT tersebut.
2. Di panel navigasi, pilih Uji, lalu pilih klien pengujian MQTT untuk membuka klien MQTT.
3. Dalam Subscription topic, masukkan***your-thing-name*****/example/topic**, lalu pilih Subscribe to topic.

Bangun dan jalankan proyek demo FreeRTOS

Setelah Anda mengatur koneksi serial ke papan Anda, Anda dapat membangun proyek demo FreeRTOS, mem-flash demo ke papan Anda, dan kemudian menjalankan demo.

Untuk membangun dan menjalankan proyek demo FreeRTOS di WICED Studio

1. Luncurkan WICED Studio.
2. Dari menu File, pilih Impor. Perluas `General` folder, pilih Existing Projects ke Workspace, lalu pilih Next.
3. Di Pilih direktori root, pilih Browse... , navigasikan ke jalur*freertos*`/projects/cypress/CYW954907AEVAL1F/wicedstudio`, lalu pilih OK.
4. Di bawah Proyek, centang kotak hanya untuk proyek aws_demo. Pilih Selesai untuk mengimpor proyek. Proyek target aws_demo akan muncul di jendela Make Target.
5. Perluas menu Platform WICED dan pilih Filter WICED nonaktif.
6. Di jendela Make Target, perluas aws_demo, klik kanan `demo.aws_demo` file, lalu pilih Build Target untuk membangun dan mengunduh demo ke papan Anda. Demo akan berjalan secara otomatis setelah dibangun dan diunduh ke papan Anda.

Pemecahan Masalah

- Jika Anda menggunakan Windows, Anda mungkin menerima kesalahan berikut ketika Anda membangun dan menjalankan proyek demo:

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

  Untuk memecahkan masalah kesalahan ini, lakukan hal berikut:

1. Jelajahi *WICED-Studio-SDK-PATH*\WICED-Studio-6.2\43xxx_Wi-Fi\tools\OpenOCD\Win32 dan klik dua kali. openocd-all-brcm-libftdi.exe
2. Jelajahi *WICED-Studio-SDK-PATH*\WICED-Studio-6.2\43xxx_Wi-Fi\tools\drivers\CYW9WCD1EVAL1 dan klik dua kali. InstallDriver.exe

- Jika Anda menggunakan Linux atau macOS, Anda mungkin menerima kesalahan berikut saat membuat dan menjalankan proyek demo:

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

Untuk memecahkan masalah kesalahan ini, gunakan perintah berikut untuk memperbarui paket libusb-dev:

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

Untuk informasi pemecahan masalah umum tentang Memulai FreeRTOS, lihat. Pemecahan masalah memulai

Memulai dengan kit Cypress CY8 CKIT-064S0S2-4343W

> **Important**
>
> Integrasi referensi ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-Freertos yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Tutorial ini memberikan petunjuk untuk memulai dengan CY8kit CKIT-064S0S2-4343W. Jika Anda belum memilikinya, Anda dapat menggunakan tautan itu untuk membeli kit. Anda juga dapat menggunakan tautan itu untuk mengakses panduan pengguna kit.

Memulai

Sebelum Anda mulai, Anda harus mengkonfigurasi AWS IoT dan FreeRTOS untuk menghubungkan perangkat Anda ke Cloud. AWS Untuk petunjuk, silakan lihat Langkah pertama. Setelah Anda menyelesaikan prasyarat, Anda akan memiliki paket FreeRTOS dengan kredensyal. AWS IoT Core

> **Note**
>
> Dalam tutorial ini, jalur ke direktori unduhan FreeRTOS yang dibuat di bagian “Langkah pertama” disebut sebagai. *freertos*

Menyiapkan lingkungan pengembangan

FreeRTOS bekerja dengan alur build atau CMake Make. Anda dapat menggunakan ModusToolbox alur Make build Anda. Anda dapat menggunakan Eclipse IDE yang dikirimkan dengan ModusToolbox atau IDE mitra seperti IAR EW-ARM, Arm MDK, atau Microsoft Visual Studio Code. Eclipse IDE kompatibel dengan sistem operasi Windows, macOS, dan Linux.

Sebelum Anda mulai, unduh dan instal perangkat ModusToolbox lunak terbaru. Untuk informasi selengkapnya, lihat Panduan ModusToolbox Instalasi.

Memperbarui alat untuk ModusToolbox 2.1 atau yang lebih lama

Jika Anda menggunakan ModusToolbox 2.1 Eclipse IDE untuk memprogram kit ini, Anda harus memperbarui alat OpenOCD dan Firmware-loader.

Dalam langkah-langkah berikut, secara default *ModusToolbox* jalur untuk:

- Windows adalah`C:\Users\`*user_name*`\ModusToolbox`.
- Linux adalah *user_home*`/ModusToolbox` atau di mana Anda memilih untuk mengekstrak file arsip.
- macOS berada di bawah folder Aplikasi dalam volume yang Anda pilih di wizard.

Memperbarui OpenOCD

Kit ini membutuhkan Cypress OpenOCD 4.0.0 atau yang lebih baru untuk berhasil menghapus dan memprogram chip.

Untuk memperbarui Cypress OpenOCD

1. Buka halaman rilis Cypress OpenOCD.
2. Unduh file arsip untuk OS Anda (Windows/Mac/Linux).
3. Hapus file yang ada di*ModusToolbox*`/tools_2.x/openocd`.

4. Ganti file *ModusToolbox*/tools_2.x/openocd dengan konten yang diekstrak dari arsip yang Anda unduh pada langkah sebelumnya.

Memperbarui Firmware-loader

Kit ini membutuhkan Cypress Firmware-loader 3.0.0 atau yang lebih baru.

Untuk memperbarui Cypress Firmware-loader

1. Buka halaman rilis Cypress Firmware-loader.
2. Unduh file arsip untuk OS Anda (Windows/Mac/Linux).
3. Hapus file yang ada di*ModusToolbox*/tools_2.x/fw-loader.
4. Ganti file *ModusToolbox*/tools_2.x/fw-loader dengan konten yang diekstrak dari arsip yang Anda unduh pada langkah sebelumnya.

Atau, Anda dapat menggunakan CMake untuk menghasilkan file build proyek dari kode sumber aplikasi FreeRTOS, membangun proyek menggunakan alat build pilihan Anda, dan kemudian memprogram kit menggunakan OpenOCD. Jika Anda lebih suka menggunakan alat GUI untuk pemrograman dengan CMake alur, unduh dan instal Cypress Programmer dari halaman web Cypress Programming Solutions. Untuk informasi selengkapnya, lihat Menggunakan CMake dengan FreeRTOS.

Menyiapkan perangkat keras Anda

Ikuti langkah-langkah ini untuk mengatur perangkat keras kit.

1. Menyediakan kit Anda

   Ikuti Panduan Penyediaan untuk instruksi Kit CY8 CKIT-064S0S2-4343W untuk menyediakan kit Anda dengan aman. AWS IoT

   Kit ini membutuhkan CySecureTools 3.1.0 atau yang lebih baru.
2. Siapkan koneksi serial

   a. Connect kit ke komputer host Anda.

   b. Port Serial USB untuk kit secara otomatis disebutkan di komputer host. Identifikasi nomor port. Di Windows, Anda dapat mengidentifikasinya menggunakan Device Manager di bawah Ports (COM & LPT).

c. Mulai terminal serial dan buka koneksi dengan pengaturan berikut:

- Tingkat baud: 115200
- Data: 8 bit
- Paritas: Tidak ada
- Hentikan bit: 1
- Kontrol aliran: Tidak ada

Bangun dan jalankan proyek Demo FreeRTOS

Di bagian ini Anda membangun dan menjalankan demo.

1. Pastikan untuk mengikuti langkah-langkah dalam Panduan Penyediaan Kit CY8 CKIT-064S0S2-4343W.
2. Bangun Demo FreeRTOS.

   a. Buka Eclipse IDE untuk ModusToolbox dan pilih, atau buat, ruang kerja.

   b. Dari menu File, pilih Impor.

      Expand General, pilih Existing Project Into Workspace, lalu pilih Next.

   c. Di Root Directory, masukkan `freertos/projects/cypress/CY8CKIT-064S0S2-4343W/mtb/aws_demos` dan kemudian pilih nama proyek`aws_demos`. Itu harus dipilih secara default.

   d. Pilih Selesai untuk mengimpor proyek ke ruang kerja Anda.

   e. Bangun aplikasi dengan melakukan salah satu hal berikut:

      - Dari Quick Panel, pilih Build aws_demos Application.
      - Pilih Project dan pilih Build All.

      Pastikan proyek dikompilasi tanpa kesalahan.

3. Memantau Pesan MQTT di Cloud

   Sebelum menjalankan demo, Anda dapat mengatur klien MQTT di AWS IoT konsol untuk memantau pesan yang dikirim perangkat Anda ke Cloud. AWS Untuk berlangganan topik MQTT dengan klien AWS IoT MQTT, ikuti langkah-langkah ini.

   a. Masuk ke konsol AWS IoT tersebut.

b. Di panel navigasi, pilih Uji, lalu pilih klien pengujian MQTT untuk membuka klien MQTT.

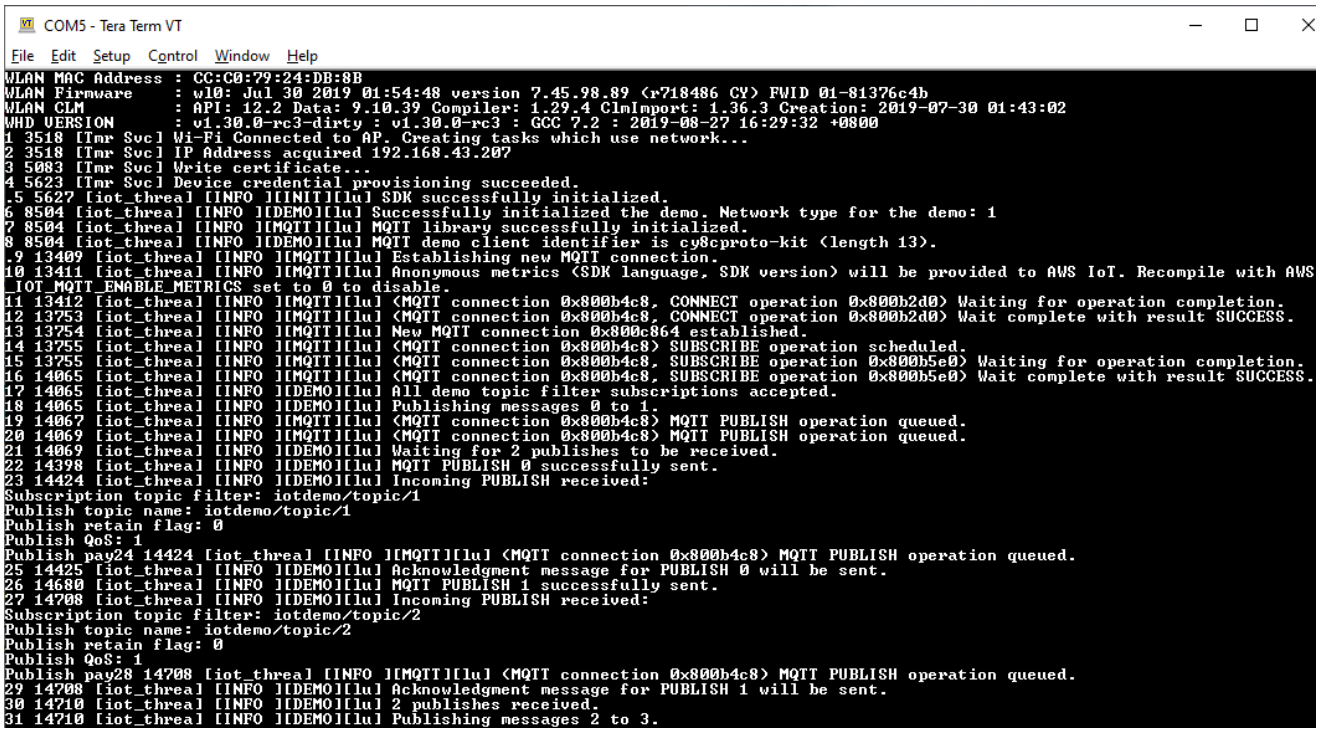
c. Untuk topik Langganan, masukkan***your-thing-name*****/example/topic**, lalu pilih Berlangganan topik.

4. Jalankan proyek demo FreeRTOS

a. Pilih proyek `aws_demos` di ruang kerja.

b. Dari Quick Panel, pilih aws_demos Program (3). KitProg Ini memprogram papan dan aplikasi demo mulai berjalan setelah pemrograman selesai.

c. Anda dapat melihat status aplikasi yang sedang berjalan di terminal serial. Gambar berikut menunjukkan bagian dari output terminal.



Demo MQTT menerbitkan pesan tentang empat topik berbeda (`iotdemo/topic/`*n*, di mana n = 1 hingga 4) dan berlangganan semua topik tersebut untuk menerima pesan yang sama kembali. Ketika pesan diterima, demo menerbitkan pesan pengakuan tentang topik tersebut. `iotdemo/acknowledgements` Daftar berikut menjelaskan pesan debug yang muncul di output terminal, dengan referensi ke nomor seri pesan. Pada output, detail driver WICED Host Driver (WHD) dicetak terlebih dahulu tanpa penomoran seri.

1. 1 hingga 4 — Perangkat terhubung ke Access Point (AP) yang dikonfigurasi dan disediakan dengan menghubungkan ke AWS server menggunakan titik akhir dan sertifikat yang dikonfigurasi.

2. 5 hingga 13 - pustaka CoreMQTT diinisialisasi dan perangkat membuat koneksi MQTT.
3. 14 hingga 17 — Perangkat berlangganan semua topik untuk menerima pesan yang dipublikasikan kembali.
4. 18 hingga 30 - Perangkat menerbitkan dua pesan dan menunggu untuk menerimanya kembali. Ketika setiap pesan diterima, perangkat mengirimkan pesan pengakuan.

Siklus publikasi, penerimaan, dan pengakuan yang sama berlanjut sampai semua pesan dipublikasikan. Dua pesan diterbitkan per siklus hingga jumlah siklus yang dikonfigurasi selesai.

5. Menggunakan CMake dengan FreeRTOS

Anda juga dapat menggunakan CMake untuk membangun dan menjalankan aplikasi demo. Untuk menyiapkan CMake dan sistem build asli, lihatPrasyarat.

a. Gunakan perintah berikut untuk menghasilkan file build. Tentukan papan target dengan `-DBOARD` opsi.

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

Jika Anda menggunakan Windows, Anda harus menentukan sistem build asli menggunakan `-G` opsi karena CMake menggunakan Visual Studio secara default.

Example

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

Jika tidak `arm-none-eabi-gcc` ada di jalur shell Anda, Anda juga perlu mengatur `AFR_TOOLCHAIN_PATH` CMake variabel.

Example

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

b. Gunakan perintah berikut untuk membangun proyek menggunakan CMake.

```
cmake --build build_dir
```

c. Akhirnya, program `cm0.hex` dan `cm4.hex` file yang dihasilkan di bawah *`build_dir`* dengan menggunakan Cypress Programmer.

Menjalankan demo lainnya

Aplikasi demo berikut telah diuji dan diverifikasi untuk bekerja dengan rilis saat ini. Anda dapat menemukan demo ini di bawah *`freertos`*`/demos` direktori. Untuk informasi tentang cara menjalankan demo ini, lihatDemo FreeRTOS.

- Demo Bluetooth Energi Rendah
- Over-the-Air Pembaruan demo
- Demo Klien Echo Soket Aman
- AWS IoT Demo Device Shadow

Debugging

KitProg3 pada kit mendukung debugging melalui protokol SWD.

- Untuk men-debug aplikasi FreeRTOS, pilih proyek aws_demos di ruang kerja dan kemudian pilih aws_demos Debug (3) dari Quick Panel. KitProg

Pembaruan OTA

PSoC 64 MCUs telah lulus semua tes kualifikasi FreeRTOS yang diperlukan. Namun, fitur opsional over-the-air (OTA) yang diterapkan di pustaka AWS firmware PSo C 64 Standard Secure masih menunggu evaluasi. Fitur OTA sebagaimana diimplementasikan saat ini lulus semua tes kualifikasi OTA kecuali aws_ota_test_case_rollback_if_unable_to_connect_after_update.py.

Ketika gambar OTA yang berhasil divalidasi diterapkan ke perangkat menggunakan PSo C64 Standard Secure — AWS MCU dan perangkat tidak dapat berkomunikasi dengannya AWS IoT Core, perangkat tidak dapat secara otomatis mengembalikan ke gambar bagus asli yang diketahui. Ini mungkin mengakibatkan perangkat tidak dapat dijangkau AWS IoT Core untuk pembaruan lebih lanjut. Fungsionalitas ini masih dalam pengembangan oleh tim Cypress.

Untuk informasi lebih lanjut, lihat Pembaruan OTA dengan AWS dan CY8 Kit CKIT-064S0S2-4343W. Jika Anda memiliki pertanyaan lebih lanjut atau memerlukan dukungan teknis, hubungi Komunitas Pengembang Cypress.

Memulai dengan Microchip ATECC6 08A Secure Element dengan simulator Windows

> **Important**
>
> Integrasi referensi ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-Freertos yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Tutorial ini memberikan instruksi untuk memulai dengan Microchip ATECC6 08A Secure Element dengan Windows Simulator.

Anda memerlukan perangkat keras berikut:

- Microchip ATECC6 08A elemen aman clickboard
- SAMD21 XPlained Pro
- MikroBus Xplained Pro Adaptor

Sebelum memulai, Anda harus mengonfigurasi AWS IoT dan mengunduh FreeRTOS Anda untuk menghubungkan perangkat Anda ke Cloud. AWS Lihat Langkah pertama untuk instruksi. Dalam tutorial ini, jalur ke direktori unduhan FreeRTOS disebut sebagai. *freertos*

Gambaran Umum

Tutorial ini berisi langkah-langkah berikut:

1. Hubungkan papan Anda ke mesin host.
2. Instal perangkat lunak pada mesin host untuk mengembangkan dan men-debug aplikasi tertanam untuk papan mikrokontroler Anda.
3. Kompilasi silang aplikasi demo FreerTOS ke gambar biner.
4. Muat gambar biner aplikasi ke papan Anda, lalu jalankan aplikasi.

Siapkan perangkat keras Microchip ATECC6 08A

Sebelum Anda dapat berinteraksi dengan perangkat Microchip ATECC6 08A Anda, Anda harus terlebih dahulu memprogram. SAMD21

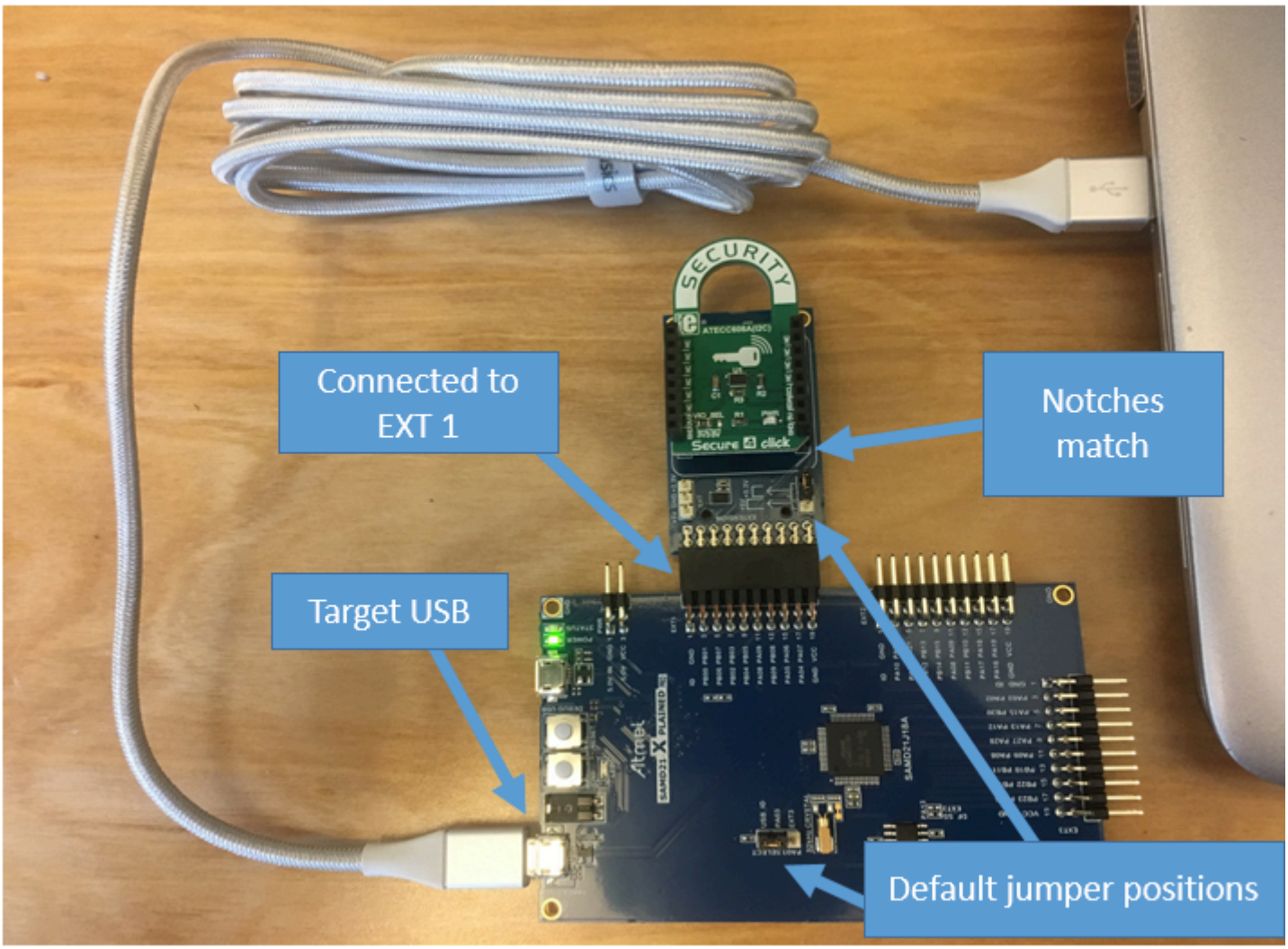
Untuk mengatur papan SAMD21 XPlained Pro

1. Ikuti CryptoAuthSSH-XSTK (DM320109) - Tautan Firmware Terbaru untuk mengunduh file.zip yang berisi instruksi (PDF) dan biner yang dapat diprogram ke D21.
2. Unduh dan instal Atmel Studio 7 IDP. Pastikan Anda memilih arsitektur driver SMART ARM MCU selama instalasi.
3. Gunakan kabel USB 2.0 Micro B untuk memasang konektor “Debug USB” ke komputer Anda, dan ikuti instruksi dalam PDF. (Konektor “Debug USB” adalah port USB yang paling dekat dengan LED POWER dan pin.)

Untuk menghubungkan perangkat keras

1. Cabut kabel micro USB dari Debug USB.
2. Colokkan adaptor MikroBus XPlained Pro ke SAMD21 papan di lokasi. EXT1
3. Colokkan papan ATECC6 08A Secure 4 Click ke adaptor XPlained MikroBusX Pro. Pastikan sudut papan klik yang berlekuk cocok dengan ikon berlekuk di papan adaptor.
4. Colokkan kabel micro USB ke Target USB.

Pengaturan Anda akan terlihat seperti berikut ini.



Siapkan lingkungan pengembangan Anda

Mendaftar untuk Akun AWS

Jika Anda tidak memiliki Akun AWS, selesaikan langkah-langkah berikut untuk membuatnya.

Untuk mendaftar untuk Akun AWS

1. Buka https://portal.aws.amazon.com/billing/pendaftaran.
2. Ikuti petunjuk online.

   Bagian dari prosedur pendaftaran melibatkan menerima panggilan telepon atau pesan teks dan memasukkan kode verifikasi pada keypad telepon.

   Saat Anda mendaftar untuk sebuah Akun AWS, sebuah Pengguna root akun AWSdibuat. Pengguna root memiliki akses ke semua Layanan AWS dan sumber daya di akun. Sebagai

praktik keamanan terbaik, tetapkan akses administratif ke pengguna, dan gunakan hanya pengguna root untuk melakukan tugas yang memerlukan akses pengguna root.

AWS mengirimi Anda email konfirmasi setelah proses pendaftaran selesai. Kapan saja, Anda dapat melihat aktivitas akun Anda saat ini dan mengelola akun Anda dengan masuk https://aws.amazon.comke/ dan memilih Akun Saya.

Buat pengguna dengan akses administratif

Setelah Anda mendaftar Akun AWS, amankan Pengguna root akun AWS, aktifkan AWS IAM Identity Center, dan buat pengguna administratif sehingga Anda tidak menggunakan pengguna root untuk tugas sehari-hari.

Amankan Pengguna root akun AWS

1. Masuk ke AWS Management Consolesebagai pemilik akun dengan memilih pengguna Root dan memasukkan alamat Akun AWS email Anda. Di laman berikutnya, masukkan kata sandi.

   Untuk bantuan masuk dengan menggunakan pengguna root, lihat Masuk sebagai pengguna root di AWS Sign-In Panduan Pengguna.
2. Mengaktifkan autentikasi multi-faktor (MFA) untuk pengguna root Anda.

   Untuk petunjuk, lihat Mengaktifkan perangkat MFA virtual untuk pengguna Akun AWS root (konsol) Anda di Panduan Pengguna IAM.

Buat pengguna dengan akses administratif

1. Aktifkan Pusat Identitas IAM.

   Untuk mendapatkan petunjuk, silakan lihat Mengaktifkan AWS IAM Identity Center di Panduan Pengguna AWS IAM Identity Center .
2. Di Pusat Identitas IAM, berikan akses administratif ke pengguna.

   Untuk tutorial tentang menggunakan Direktori Pusat Identitas IAM sebagai sumber identitas Anda, lihat Mengkonfigurasi akses pengguna dengan default Direktori Pusat Identitas IAM di Panduan AWS IAM Identity Center Pengguna.

Masuk sebagai pengguna dengan akses administratif

- Untuk masuk dengan pengguna Pusat Identitas IAM, gunakan URL masuk yang dikirim ke alamat email saat Anda membuat pengguna Pusat Identitas IAM.

  Untuk bantuan masuk menggunakan pengguna Pusat Identitas IAM, lihat Masuk ke portal AWS akses di Panduan AWS Sign-In Pengguna.

Tetapkan akses ke pengguna tambahan

1. Di Pusat Identitas IAM, buat set izin yang mengikuti praktik terbaik menerapkan izin hak istimewa paling sedikit.

   Untuk petunjuknya, lihat Membuat set izin di Panduan AWS IAM Identity Center Pengguna.
2. Tetapkan pengguna ke grup, lalu tetapkan akses masuk tunggal ke grup.

   Untuk petunjuk, lihat Menambahkan grup di Panduan AWS IAM Identity Center Pengguna.

Untuk memberikan akses dan menambahkan izin bagi pengguna, grup, atau peran Anda:

- Pengguna dan grup di AWS IAM Identity Center:

  Buat rangkaian izin. Ikuti instruksi di Buat rangkaian izin di Panduan Pengguna AWS IAM Identity Center .
- Pengguna yang dikelola di IAM melalui penyedia identitas:

  Buat peran untuk federasi identitas. Ikuti instruksi dalam Buat peran untuk penyedia identitas pihak ketiga (federasi) dalam Panduan Pengguna IAM.
- Pengguna IAM:
  - Buat peran yang dapat diambil pengguna Anda. Ikuti instruksi dalam Buat peran untuk pengguna IAM dalam Panduan Pengguna IAM.
  - (Tidak disarankan) Lampirkan kebijakan langsung ke pengguna atau tambahkan pengguna ke grup pengguna. Ikuti petunjuk dalam Menambahkan izin ke pengguna (konsol) dalam Panduan Pengguna IAM.

Pengaturan

1. Unduh repo FreeRTOS dari repositori FreerTOS. GitHub

   Untuk mengunduh FreeRTOS dari: GitHub

   1. Jelajahi repositori FreeRTOS GitHub .
   2. Pilih Kloning atau unduh.
   3. Dari baris perintah di komputer Anda, kloning repositori ke direktori di mesin host Anda.

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

      Important

      - Dalam topik ini, jalur ke direktori unduhan FreeRTOS disebut sebagai. *freertos*
      - Karakter ruang di *freertos* jalur dapat menyebabkan kegagalan build. Saat Anda mengkloning atau menyalin repositori, pastikan jalur yang Anda buat tidak berisi karakter spasi.
      - Panjang maksimum jalur file di Microsoft Windows adalah 260 karakter. Jalur direktori unduhan FreeRTOS yang panjang dapat menyebabkan kegagalan build.
      - Karena kode sumber mungkin berisi tautan simbolis, jika Anda menggunakan Windows untuk mengekstrak arsip, Anda mungkin harus:
        - Aktifkan Mode Pengembang atau,
        - Gunakan konsol yang ditinggikan sebagai administrator.

        Dengan cara ini, Windows dapat membuat tautan simbolis dengan benar saat mengekstrak arsip. Jika tidak, tautan simbolis akan ditulis sebagai file normal yang berisi jalur tautan simbolis sebagai teks atau kosong. Untuk informasi lebih lanjut, lihat entri blog Symlinks di Windows 10! .

        Jika Anda menggunakan Git di bawah Windows, Anda harus mengaktifkan Mode Pengembang atau Anda harus:

        - Setel `core.symlinks` ke true dengan perintah berikut:

          ```
          git config -\-global core.symlinks true
          ```

- Gunakan konsol yang ditinggikan sebagai administrator setiap kali Anda menggunakan perintah git yang menulis ke sistem (misalnya,git pull,git clone, dangit submodule update -\-init -\-recursive).

4. Dari *freertos* direktori, periksa cabang yang akan digunakan.

2. Siapkan lingkungan pengembangan Anda.

   a. Instal versi terbaru Win PCap.

   b. Instal Microsoft Visual Studio.

      Visual Studio versi 2017 dan 2019 diketahui berfungsi. Semua edisi versi Visual Studio ini didukung (Komunitas, Profesional, atau Perusahaan).

      Selain IDE, instal pengembangan Desktop dengan komponen C++. Kemudian, di bawah Opsional, instal SDK Windows 10 terbaru.

   c. Pastikan Anda memiliki koneksi Ethernet terprogram yang aktif.

Bangun dan jalankan proyek demo FreeRTOS

> **Important**
>
> Perangkat Microchip ATECC6 08A memiliki inisialisasi satu kali yang dikunci ke perangkat saat pertama kali proyek dijalankan (selama panggilan ke). `C_InitToken` Namun, proyek demo FreeRTOS dan proyek pengujian memiliki konfigurasi yang berbeda. Jika perangkat terkunci selama konfigurasi proyek demo, semua pengujian dalam proyek pengujian tidak akan berhasil.

Untuk membangun dan menjalankan proyek demo FreeRTOS dengan Visual Studio IDE

1. Muat proyek ke Visual Studio.

   Dari menu File, pilih Buka. Pilih File/Solusi, navigasikan ke ***freertos*\projects\microchip\ecc608a_plus_winsim\visual_studio\aws_demos\aws_demos.sln** file, lalu pilih Buka.

2. Targetkan ulang proyek demo.

Proyek demo tergantung pada Windows SDK, tetapi tidak memiliki versi Windows SDK yang ditentukan. Secara default, IDE mungkin mencoba membangun demo dengan versi SDK yang tidak ada di mesin Anda. Untuk mengatur versi Windows SDK, klik kanan aws_demos, lalu pilih Retarget Projects. Ini membuka jendela Review Solution Actions. Pilih versi Windows SDK yang ada di mesin Anda (gunakan nilai awal dalam daftar drop-down), lalu pilih OK.

3. Membangun dan menjalankan proyek.

   Dari menu Build, pilih Build Solution, dan pastikan solusi dibuat tanpa kesalahan. Pilih Debug, Mulai Debugging untuk menjalankan proyek. Pada proses pertama, Anda perlu mengkonfigurasi antarmuka perangkat Anda dan mengkompilasi ulang. Untuk informasi selengkapnya, lihat Konfigurasikan antarmuka jaringan Anda.

4. Penyediaan Microchip ATECC6 08A.

   Microchip telah menyediakan beberapa alat skrip untuk membantu pengaturan bagian ATECC6 08A. Arahkan ke*freertos*`\vendors\microchip\secure_elements\app\example_trust_chain_tool`, dan buka file README.md.

   Ikuti petunjuk dalam `README.md` file untuk menyediakan perangkat Anda. Langkah-langkahnya meliputi:

   1. Buat dan daftarkan otoritas sertifikat dengan AWS.
   2. Hasilkan kunci Anda di Microchip ATECC6 08A dan ekspor kunci publik dan nomor seri perangkat.
   3. Hasilkan sertifikat untuk perangkat dan daftarkan sertifikat itu dengan AWS.
   4. Muat sertifikat CA dan sertifikat perangkat ke perangkat.

5. Bangun dan jalankan sampel FreeRTOS.

   Jalankan kembali proyek demo lagi. Kali ini Anda harus terhubung!

Pemecahan Masalah

Untuk informasi pemecahan masalah umum, lihat. Pemecahan masalah memulai

Memulai dengan Espressif ESP32 - DevKit C dan ESP-WROVER-KIT

> **Important**
>
> Integrasi referensi ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-Freertos yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

> **Note**
>
> Untuk mengeksplorasi cara mengintegrasikan pustaka dan demo modular FreeRTOS dalam proyek Espressif IDF Anda sendiri, lihat integrasi referensi unggulan kami untuk platform -C3. ESP32

Ikuti tutorial ini untuk memulai dengan Espressif ESP32 - DevKit C dilengkapi dengan modul ESP32 -WROOM-32, ESP32 -SOLO-1, atau ESP-WROVER dan modul. ESP-WROVER-KIT-VB Untuk membeli satu dari mitra kami di katalog Perangkat AWS Mitra, gunakan tautan berikut:

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

Versi papan pengembangan ini didukung di FreeRTOS.

Untuk informasi lebih lanjut tentang versi terbaru papan ini, lihat ESP32- DevKit C V4 atau ESP-WROVER-KITv4.1 di situs web Espressif.

> **Note**
>
> Saat ini, port FreeRTOS ESP32 untuk -WROVER-KIT dan DevKit ESP C tidak mendukung fitur Symmetric multiprocessing (SMP).

Gambaran Umum

Tutorial ini memandu Anda melalui langkah-langkah berikut:

1. Menghubungkan papan Anda ke mesin host.
2. Menginstal perangkat lunak pada mesin host untuk mengembangkan dan men-debug aplikasi tertanam untuk papan mikrokontroler Anda.
3. Menyusun silang aplikasi demo FreeRTOS ke gambar biner.
4. Memuat gambar biner aplikasi ke papan Anda, dan kemudian menjalankan aplikasi.
5. Berinteraksi dengan aplikasi yang berjalan di papan Anda di seluruh koneksi serial, untuk tujuan pemantauan dan debugging.

Prasyarat

Sebelum Anda memulai dengan FreeRTOS di papan Espressif Anda, Anda harus mengatur akun dan izin Anda. AWS

Mendaftar untuk Akun AWS

Jika Anda tidak memiliki Akun AWS, selesaikan langkah-langkah berikut untuk membuatnya.

Untuk mendaftar untuk Akun AWS

1. Buka https://portal.aws.amazon.com/billing/pendaftaran.
2. Ikuti petunjuk online.

   Bagian dari prosedur pendaftaran melibatkan menerima panggilan telepon atau pesan teks dan memasukkan kode verifikasi pada keypad telepon.

   Saat Anda mendaftar untuk sebuah Akun AWS, sebuah Pengguna root akun AWSdibuat. Pengguna root memiliki akses ke semua Layanan AWS dan sumber daya di akun. Sebagai praktik keamanan terbaik, tetapkan akses administratif ke pengguna, dan gunakan hanya pengguna root untuk melakukan tugas yang memerlukan akses pengguna root.

AWS mengirimi Anda email konfirmasi setelah proses pendaftaran selesai. Kapan saja, Anda dapat melihat aktivitas akun Anda saat ini dan mengelola akun Anda dengan masuk https://aws.amazon.comke/ dan memilih Akun Saya.

Buat pengguna dengan akses administratif

Setelah Anda mendaftar Akun AWS, amankan Pengguna root akun AWS, aktifkan AWS IAM Identity Center, dan buat pengguna administratif sehingga Anda tidak menggunakan pengguna root untuk tugas sehari-hari.

Amankan Anda Pengguna root akun AWS

1. Masuk ke AWS Management Consolesebagai pemilik akun dengan memilih pengguna Root dan memasukkan alamat Akun AWS email Anda. Di laman berikutnya, masukkan kata sandi.

   Untuk bantuan masuk dengan menggunakan pengguna root, lihat Masuk sebagai pengguna root di AWS Sign-In Panduan Pengguna.
2. Mengaktifkan autentikasi multi-faktor (MFA) untuk pengguna root Anda.

   Untuk petunjuk, lihat Mengaktifkan perangkat MFA virtual untuk pengguna Akun AWS root (konsol) Anda di Panduan Pengguna IAM.

Buat pengguna dengan akses administratif

1. Aktifkan Pusat Identitas IAM.

   Untuk mendapatkan petunjuk, silakan lihat Mengaktifkan AWS IAM Identity Center di Panduan Pengguna AWS IAM Identity Center .
2. Di Pusat Identitas IAM, berikan akses administratif ke pengguna.

   Untuk tutorial tentang menggunakan Direktori Pusat Identitas IAM sebagai sumber identitas Anda, lihat Mengkonfigurasi akses pengguna dengan default Direktori Pusat Identitas IAM di Panduan AWS IAM Identity Center Pengguna.

Masuk sebagai pengguna dengan akses administratif

- Untuk masuk dengan pengguna Pusat Identitas IAM, gunakan URL masuk yang dikirim ke alamat email saat Anda membuat pengguna Pusat Identitas IAM.

  Untuk bantuan masuk menggunakan pengguna Pusat Identitas IAM, lihat Masuk ke portal AWS akses di Panduan AWS Sign-In Pengguna.

Tetapkan akses ke pengguna tambahan

1. Di Pusat Identitas IAM, buat set izin yang mengikuti praktik terbaik menerapkan izin hak istimewa paling sedikit.

   Untuk petunjuknya, lihat Membuat set izin di Panduan AWS IAM Identity Center Pengguna.
2. Tetapkan pengguna ke grup, lalu tetapkan akses masuk tunggal ke grup.

   Untuk petunjuk, lihat Menambahkan grup di Panduan AWS IAM Identity Center Pengguna.

Untuk memberikan akses dan menambahkan izin bagi pengguna, grup, atau peran Anda:

- Pengguna dan grup di AWS IAM Identity Center:

  Buat rangkaian izin. Ikuti instruksi di Buat rangkaian izin di Panduan Pengguna AWS IAM Identity Center .
- Pengguna yang dikelola di IAM melalui penyedia identitas:

  Buat peran untuk federasi identitas. Ikuti instruksi dalam Buat peran untuk penyedia identitas pihak ketiga (federasi) dalam Panduan Pengguna IAM.
- Pengguna IAM:
  - Buat peran yang dapat diambil pengguna Anda. Ikuti instruksi dalam Buat peran untuk pengguna IAM dalam Panduan Pengguna IAM.
  - (Tidak disarankan) Lampirkan kebijakan langsung ke pengguna atau tambahkan pengguna ke grup pengguna. Ikuti petunjuk dalam Menambahkan izin ke pengguna (konsol) dalam Panduan Pengguna IAM.

Memulai

> **Note**
>
> Perintah Linux dalam tutorial ini mengharuskan Anda menggunakan shell Bash.

1. Siapkan perangkat keras Espressif.
   - Untuk informasi tentang menyiapkan perangkat keras papan pengembangan ESP32 - DevKit C, lihat Panduan Memulai ESP32 - DevKit C V4.

- Untuk informasi tentang menyiapkan perangkat keras papan ESP-WROVER-KIT pengembangan, lihat Panduan Memulai ESP-WROVER-KIT V4.1.

> **Important**
>
> Ketika Anda mencapai bagian Memulai dari panduan Espressif, berhenti, dan kemudian kembali ke instruksi di halaman ini.

2. Unduh Amazon GitHubFreeRTOS dari. (Untuk instruksi, lihat file README.md.)
3. Siapkan lingkungan pengembangan Anda.

   Untuk berkomunikasi dengan papan Anda, Anda harus menginstal rantai alat. Espressif menyediakan ESP-IDF untuk mengembangkan perangkat lunak untuk papan mereka. Karena ESP-IDF memiliki versi sendiri dari FreeRTOS Kernel terintegrasi sebagai komponen, Amazon FreeRTOS menyertakan versi kustom ESP-IDF v4.2 yang memiliki FreeRTOS Kernel dihapus. Ini memperbaiki masalah dengan file duplikat saat Anda mengkompilasi. Untuk menggunakan versi kustom ESP-IDF v4.2 yang disertakan dengan Amazon FreeRTOS, ikuti petunjuk di bawah ini untuk sistem operasi mesin host Anda.

   Windows

   1. Unduh Pemasang Online Universal ESP-IDF untuk Windows.
   2. Jalankan Pemasang Online Universal.
   3. Ketika Anda sampai ke langkah Unduh atau gunakan ESP-IDF, pilih Gunakan direktori ESP-IDF yang ada dan atur Pilih direktori ESP-IDF yang ada ke. *`freertos`*`/vendors/espressif/esp-idf`
   4. Selesaikan instalasi.

   macOS

   1. Ikuti petunjuk dalam Pengaturan Standar prasyarat Toolchain (ESP-IDF v4.2) untuk macOS.

      > **Important**
      >
      > Ketika Anda mencapai petunjuk “Dapatkan ESP-IDF” di bawah Langkah Berikutnya, berhenti, dan kemudian kembali ke instruksi di halaman ini.

2. Buka jendela baris perintah.
3. Arahkan ke direktori unduhan FreeRTOS, lalu jalankan skrip berikut untuk mengunduh dan menginstal rantai alat espressif untuk platform Anda.

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

4. Tambahkan alat toolchain ESP-IDF ke jalur terminal Anda dengan perintah berikut.

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

Linux

1. Ikuti petunjuk dalam Pengaturan Standar prasyarat Toolchain (ESP-IDF v4.2) untuk Linux.

> **Important**
>
> Ketika Anda mencapai petunjuk “Dapatkan ESP-IDF” di bawah Langkah Berikutnya, berhenti, dan kemudian kembali ke instruksi di halaman ini.

2. Buka jendela baris perintah.
3. Arahkan ke direktori unduhan FreeRTOS, lalu jalankan skrip berikut untuk mengunduh dan menginstal rantai alat Espressif untuk platform Anda.

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

4. Tambahkan alat toolchain ESP-IDF ke jalur terminal Anda dengan perintah berikut.

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

4. Buat koneksi serial.

   a. Untuk membuat koneksi serial antara mesin host Anda dan ESP32 - DevKit C, Anda harus menginstal driver CP21 0x USB ke UART Bridge VCP. Anda dapat mengunduh driver ini dari Silicon Labs.

      Untuk membuat koneksi serial antara mesin host Anda dan ESP32 -WROVER-KIT, Anda harus menginstal driver port COM virtual FTDI. Anda dapat mengunduh driver ini dari FTDI.

   b. Ikuti langkah-langkah untuk Membangun Koneksi Serial dengan ESP32.

c. Setelah Anda membuat koneksi serial, catat port serial untuk koneksi papan Anda. Anda membutuhkannya untuk mem-flash demo.

Konfigurasikan aplikasi demo FreeRTOS

Untuk tutorial ini, file konfigurasi FreeRTOS terletak di. *freertos*`/vendors/espressif/boards/`*board-name*`/aws_demos/config_files/FreeRTOSConfig.h` (Misalnya, jika `AFR_BOARD espressif.esp32_devkitc` dipilih, file konfigurasi terletak di*freertos*`/vendors/espressif/boards/esp32/aws_demos/config_files/FreeRTOSConfig.h`.)

1. Jika Anda menjalankan macOS atau Linux, buka prompt terminal. Jika Anda menjalankan Windows, buka aplikasi “ESP-IDF 4.x CMD” (jika Anda menyertakan opsi ini saat menginstal rantai alat ESP-IDF), atau aplikasi “Command Prompt” sebaliknya.
2. Untuk memverifikasi bahwa Anda telah menginstal Python3, jalankan

   ```
   python --version
   ```

   Versi yang diinstal ditampilkan. Jika Anda tidak menginstal Python 3.0.1 atau yang lebih baru, Anda dapat menginstalnya dari situs web Python.
3. Anda memerlukan AWS Command Line Interface (CLI) untuk menjalankan AWS IoT perintah. Jika Anda menjalankan Windows, gunakan `easy_install awscli` perintah untuk menginstal AWS CLI di aplikasi “Command” atau “ESP-IDF 4.x CMD”.

   Jika Anda menjalankan macOS atau Linux, lihat Menginstal CLI AWS.
4. Jalankan .

   ```
   aws configure
   ```

   dan konfigurasikan AWS CLI dengan ID kunci AWS akses Anda, kunci akses rahasia, dan Wilayah default AWS . Untuk informasi selengkapnya, lihat Mengonfigurasi AWS CLI.
5. Gunakan perintah berikut untuk menginstal AWS SDK untuk Python (boto3):

   - Di Windows, di aplikasi “Command” atau “ESP-IDF 4.x CMD”, jalankan

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

> **Note**
>
> Lihat dokumentasi Boto3 untuk detailnya.

- Di macOS atau Linux, jalankan

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

dan kemudian lari

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

FreeRTOS menyertakan `SetupAWS.py` skrip untuk membuatnya lebih mudah untuk mengatur papan Espressif Anda untuk terhubung. AWS IoT Untuk mengkonfigurasi skrip, buka *`freertos`*`/tools/aws_config_quick_start/configure.json` dan atur atribut berikut:

`afr_source_dir`

Jalur lengkap ke *`freertos`* direktori di komputer Anda. Pastikan Anda menggunakan garis miring maju untuk menentukan jalur ini.

`thing_name`

Nama yang ingin Anda tetapkan untuk AWS IoT hal yang mewakili papan Anda.

`wifi_ssid`

SSID jaringan Wi-Fi Anda.

`wifi_password`

Kata sandi untuk jaringan Wi-Fi Anda.

`wifi_security`

Jenis keamanan untuk jaringan Wi-Fi Anda.

Berikut ini adalah jenis keamanan yang valid:

- `eWiFiSecurityOpen`(Terbuka, tidak ada keamanan)
- `eWiFiSecurityWEP`(Keamanan WEP)

- `eWiFiSecurityWPA`(Keamanan WPA)
- `eWiFiSecurityWPA2`(WPA2 keamanan)

6. Jalankan skrip konfigurasi.

   a. Jika Anda menjalankan macOS atau Linux, buka prompt terminal. Jika Anda menjalankan Windows, buka aplikasi “ESP-IDF 4.x CMD” atau “Command”.

   b. Arahkan ke *freertos*`/tools/aws_config_quick_start` direktori dan jalankan

   ```
   python SetupAWS.py setup
   ```

   Script melakukan hal berikut:

   - Membuat hal IoT, sertifikat, dan kebijakan.
   - Melampirkan kebijakan IoT ke sertifikat dan sertifikat untuk AWS IoT benda itu.
   - Mengisi `aws_clientcredential.h` file dengan AWS IoT titik akhir, SSID Wi-Fi, dan kredensil Anda.
   - Memformat sertifikat dan kunci pribadi Anda dan menuliskannya ke file `aws_clientcredential_keys.h` header.

   > **Note**
   >
   > Sertifikat di-hardcode hanya untuk tujuan demonstrasi. Aplikasi tingkat produksi harus menyimpan file-file ini di lokasi yang aman.

   Untuk informasi selengkapnya`SetupAWS.py`, lihat `README.md` di *freertos*`/tools/aws_config_quick_start` direktori.

Memantau pesan MQTT di cloud

Sebelum menjalankan proyek demo FreeRTOS, Anda dapat mengatur klien MQTT di konsol untuk memantau pesan AWS IoT yang dikirim perangkat Anda ke Cloud. AWS

Untuk berlangganan topik MQTT dengan klien MQTT AWS IoT

1. Navigasikan ke konsol AWS IoT tersebut.

2. Di panel navigasi, pilih Uji, lalu pilih MQTT Test Client.
3. Dalam Subscription topic, masukkan*your-thing-name*/example/topic, lalu pilih Subscribe to topic.

Ketika proyek demo berhasil berjalan di perangkat Anda, Anda melihat “Hello World!” dikirim beberapa kali ke topik yang Anda berlangganan.

Bangun, flash, dan jalankan proyek demo FreeRTOS menggunakan skrip idf.py

Anda dapat menggunakan utilitas IDF Espressif (`idf.py`) untuk membangun proyek dan mem-flash binari ke perangkat Anda.

> **Note**
>
> Beberapa pengaturan mungkin mengharuskan Anda menggunakan opsi `"-p port-name"` port `idf.py` untuk menentukan port yang benar, seperti pada contoh berikut.
>
> ```
> idf.py -p /dev/cu.usbserial-00101301B flash
> ```

Membangun dan mem-flash FreeRTOS di Windows, Linux, dan macOS (ESP-IDF v4.2)

1. Arahkan ke root direktori unduhan FreeRTOS Anda.
2. Di jendela baris perintah, masukkan perintah berikut untuk menambahkan alat ESP-IDF ke PATH terminal Anda.

   Windows (aplikasi “Perintah”)

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

   Windows (aplikasi “ESP-IDF 4.x CMD”)

   (Ini sudah dilakukan saat Anda membuka aplikasi.)

   Linux/macOS

   ```
   source vendors/espressif/esp-idf/export.sh
   ```
3. Konfigurasikan cmake di `build` direktori dan buat gambar firmware dengan perintah berikut.

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

Anda akan melihat output seperti berikut.

```
   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'
```

Jika tidak ada kesalahan, build akan menghasilkan file biner firmware .bin.

4. Hapus memori flash papan pengembangan Anda dengan perintah berikut.

```
idf.py erase_flash
```

5. Gunakan `idf.py` skrip untuk mem-flash biner aplikasi ke papan Anda.

```
idf.py flash
```

6. Pantau output dari port serial papan Anda dengan perintah berikut.

```
idf.py monitor
```

> **Note**
>
> Anda dapat menggabungkan perintah-perintah ini seperti pada contoh berikut.
>
> ```
> idf.py erase_flash flash monitor
> ```

Untuk pengaturan mesin host tertentu, Anda harus menentukan port saat Anda mem-flash papan seperti pada contoh berikut.

```
idf.py erase_flash flash monitor -p /dev/ttyUSB1
```

Membangun dan Flash FreeRTOS dengan CMake

Selain `idf.py` skrip yang disediakan oleh SDK IDF untuk membangun dan menjalankan kode Anda, Anda juga dapat membangun proyek dengan. CMake Saat ini, ia mendukung Unix Makefiles atau sistem build Ninja.

Untuk membangun dan mem-flash proyek

1. Di jendela baris perintah, arahkan ke root direktori unduhan FreeRTOS Anda.
2. Jalankan skrip berikut untuk menambahkan alat ESP-IDF ke PATH shell Anda.

   Windows

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

   Linux/macOS

   ```
   source vendors/espressif/esp-idf/export.sh
   ```
3. Masukkan perintah berikut untuk menghasilkan file build.

   Dengan 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
   ```

Dengan 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. Bangun proyek.

   Dengan Unix Makefiles

   ```
   make -C ./YOUR_BUILD_DIRECTORY -j8
   ```

   Dengan Ninja

   ```
   ninja -C ./YOUR_BUILD_DIRECTORY -j8
   ```

5. Hapus flash dan kemudian flash papan.

   Dengan Unix Makefiles

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

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

   Dengan Ninja

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

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

Jalankan demo Bluetooth Low Energy

FreeRTOS mendukung konektivitasPerpustakaan Bluetooth Rendah Energi.

Untuk menjalankan proyek demo FreeRTOS di Bluetooth Low Energy, Anda harus menjalankan Aplikasi Demo SDK Seluler FreeRTOS Bluetooth Low Energy di perangkat seluler iOS atau Android.

Untuk mengatur aplikasi demo SDK seluler FreeRTOS Bluetooth Low Energy

1. Ikuti instruksi Ponsel SDKs untuk perangkat Bluetooth FreeRTOS untuk mengunduh dan menginstal SDK untuk platform seluler Anda di komputer host Anda.
2. Ikuti petunjuk Aplikasi demo SDK Seluler Energi Rendah FreeRTOS Bluetooth untuk menyiapkan aplikasi seluler demo di perangkat seluler Anda.

Untuk petunjuk tentang cara menjalankan demo MQTT melalui Bluetooth Low Energy di papan tulis Anda, lihat. MQTT melalui Bluetooth Energi Rendah

Untuk petunjuk tentang cara menjalankan demo penyediaan Wi-Fi di papan tulis Anda, lihat. Penyediaan Wi-Fi

Menggunakan FreeRTOS dalam proyek Anda sendiri untuk CMake ESP32

Jika Anda ingin menggunakan FreeRTOS dalam proyek Anda CMake sendiri, Anda dapat mengaturnya sebagai subdirektori dan membangunnya bersama dengan aplikasi Anda. Pertama, dapatkan salinan FreeRTOS dari. GitHub Anda juga dapat mengaturnya sebagai submodul Git dengan perintah berikut sehingga lebih mudah untuk memperbarui di masa mendatang.

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

Jika versi yang lebih baru dirilis, Anda dapat memperbarui salinan lokal Anda dengan perintah ini.

```
# 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"
```

Jika proyek Anda memiliki struktur direktori berikut:

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

Maka berikut ini adalah contoh `CMakeLists.txt` file tingkat atas yang dapat digunakan untuk membangun aplikasi Anda bersama dengan 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)
```

Untuk membangun proyek, jalankan CMake perintah berikut. Pastikan ESP32 compiler berada dalam variabel lingkungan PATH.

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

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

Untuk mem-flash aplikasi ke papan Anda, jalankan perintah berikut.

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

Menggunakan komponen dari FreeRTOS

Setelah berjalan CMake, Anda dapat menemukan semua komponen yang tersedia dalam output ringkasan. Seharusnya terlihat seperti contoh berikut.

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

Anda dapat mereferensikan komponen apa pun dari `Modules to build` daftar. Untuk menautkannya ke aplikasi Anda, letakkan `AFR::` namespace di depan nama, misalnya,, `AFR::core_mqtt``AFR::ota`, dan sebagainya.

Tambahkan komponen khusus menggunakan ESP-IDF

Anda dapat menambahkan lebih banyak komponen saat menggunakan ESP-IDF. Misalnya, dengan asumsi Anda ingin menambahkan komponen yang dipanggil`example_component`, dan proyek Anda terlihat seperti ini:

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

Berikut ini adalah contoh `CMakeLists.txt` file untuk komponen Anda.

```
add_library(example_component src/example_component.c)
```

```
target_include_directories(example_component PUBLIC include)
```

Kemudian, di `CMakeLists.txt` file tingkat atas, tambahkan komponen dengan memasukkan baris berikut setelahnya`add_subdirectory(freertos)`.

```
add_subdirectory(component/example_component)
```

Kemudian, modifikasi `target_link_libraries` untuk menyertakan komponen Anda.

```
target_link_libraries(my_app PRIVATE AFR::core_mqtt PRIVATE example_component)
```

Komponen ini sekarang secara otomatis ditautkan ke kode aplikasi Anda secara default. Anda sekarang dapat menyertakan file header dan memanggil fungsi yang didefinisikan.

Ganti konfigurasi untuk FreeRTOS

Saat ini tidak ada pendekatan yang terdefinisi dengan baik untuk mendefinisikan ulang konfigurasi di luar pohon sumber FreerTos. Secara default, CMake akan mencari *freertos*`/demos/include/` direktori *freertos*`/vendors/espressif/boards/esp32/aws_demos/config_files/` dan. Namun, Anda dapat menggunakan solusi untuk memberi tahu kompiler agar mencari direktori lain terlebih dahulu. Misalnya, Anda dapat menambahkan folder lain untuk konfigurasi FreeRTOS.

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

File di bawah `freertos-configs` disalin dari *freertos*`/demos/include/` direktori *freertos*`/vendors/espressif/boards/esp32/aws_demos/config_files/` dan. Kemudian, di `CMakeLists.txt` file tingkat atas Anda, tambahkan baris ini sebelumnya `add_subdirectory(freertos)` sehingga kompiler akan mencari direktori ini terlebih dahulu.

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

Menyediakan sdkconfig Anda sendiri untuk ESP-IDF

Jika Anda ingin menyediakan sendirisdkconfig.default, Anda dapat mengatur CMake variabelIDF_SDKCONFIG_DEFAULTS, dari baris perintah:

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

Jika Anda tidak menentukan lokasi untuk file Anda sendiri, FreeRTOS menggunakan sdkconfig.default file default yang terletak di. *freertos*/vendors/espressif/boards/esp32/aws_demos/sdkconfig.defaults

Untuk informasi selengkapnya, lihat Konfigurasi Proyek di Referensi API Espressif dan, jika Anda mengalami masalah setelah berhasil dikompilasi, lihat bagian tentang Opsi yang tidak digunakan lagi dan penggantiannya di halaman tersebut.

Ringkasan

Jika Anda memiliki proyek dengan komponen yang disebutexample_component, dan Anda ingin mengganti beberapa konfigurasi, berikut adalah contoh lengkap dari file tingkat CMakeLists.txt atas.

```
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)
```

Pemecahan Masalah

- Jika Anda menjalankan macOS dan sistem operasi tidak mengenali Anda ESP-WROVER-KIT, pastikan Anda tidak menginstal driver D2XX. Untuk menghapus instalannya, ikuti petunjuk di Panduan Instalasi Driver FTDI untuk macOS X.
- Utilitas monitor yang disediakan oleh ESP-IDF (dan dipanggil menggunakan make monitor) membantu Anda memecahkan kode alamat. Untuk alasan ini, ini dapat membantu Anda mendapatkan beberapa backtrace yang berarti jika aplikasi berhenti bekerja. Untuk informasi selengkapnya, lihat Penguraian Kode Alamat Otomatis di situs web Espressif.
- Dimungkinkan juga GDBstub untuk mengaktifkan komunikasi dengan gdb tanpa memerlukan perangkat keras JTAG khusus. Untuk informasi selengkapnya, lihat Meluncurkan GDB dengan GDBStub di situs web Espressif.
- Untuk informasi tentang pengaturan lingkungan berbasis OpenOCD jika debugging berbasis perangkat keras JTAG diperlukan, lihat JTAG Debugging di situs web Espressif.
- Jika tidak `pyserial` dapat `pip` diinstal menggunakan macOS, unduh dari situs web pyserial.
- Jika papan direset terus menerus, coba hapus lampu kilat dengan memasukkan perintah berikut di terminal.

  ```
  make erase_flash
  ```

- Jika Anda melihat kesalahan saat menjalankan`idf_monitor.py`, gunakan Python 2.7.
- Pustaka yang diperlukan dari ESP-IDF disertakan dalam FreeRTOS, jadi tidak perlu mengunduhnya secara eksternal. Jika variabel `IDF_PATH` lingkungan disetel, kami sarankan Anda menghapusnya sebelum Anda membangun FreeRTOS.
- Di Windows, dibutuhkan waktu 3-4 menit untuk membangun proyek. Untuk mengurangi waktu pembuatan, Anda dapat menggunakan `-j4` sakelar pada perintah make.

  ```
  make flash monitor -j4
  ```

- Jika perangkat Anda mengalami masalah saat terhubung AWS IoT, buka `aws_clientcredential.h` file, dan verifikasi bahwa variabel konfigurasi didefinisikan dengan benar dalam file. `clientcredentialMQTT_BROKER_ENDPOINT[]`Seharusnya terlihat seperti`1234567890123-ats.iot.us-east-1.amazonaws.com`.
- Jika Anda mengikuti langkah-langkah Menggunakan FreeRTOS dalam proyek Anda sendiri untuk CMake ESP32 dan Anda melihat kesalahan referensi yang tidak ditentukan dari tautan, biasanya karena pustaka atau demo dependen yang hilang. Untuk menambahkannya, perbarui `CMakeLists.txt` file (di bawah direktori root) menggunakan CMake fungsi standar`target_link_libraries`.
- ESP-IDF v4.2 mendukung penggunaan xtensa\ -esp32\ -elf\ -gcc 8\ .2\ .0\. rantai alat. Jika Anda menggunakan versi sebelumnya dari rantai alat Xtensa, unduh versi yang diperlukan.
- Jika Anda melihat log kesalahan seperti berikut tentang dependensi python yang tidak terpenuhi untuk 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
```

  Instal dependensi python di platform Anda menggunakan perintah Python berikut:

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

Untuk informasi pemecahan masalah selengkapnya, lihat Pemecahan masalah memulai.

Debugging

Kode debugging pada Espressif ESP32 - DevKit C dan ESP-WROVER-KIT (ESP-IDF v4.2)

Bagian ini menunjukkan cara men-debug perangkat keras Espressif menggunakan ESP-IDF v4.2. Anda memerlukan kabel JTAG ke USB. Kami menggunakan kabel USB ke MPSSE (misalnya, FTDI C232HM-DDHSL-0).

Pengaturan ESP- DevKit C JTAG

Untuk kabel FTDI C232HM-DDHSL-0, ini adalah koneksi ke 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             |
```

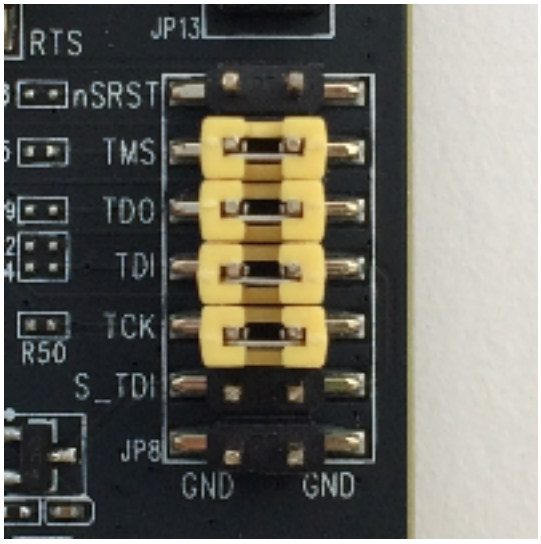
ESP-WROVER-KIT Pengaturan JTAG

Untuk kabel FTDI C232HM-DDHSL-0, ini adalah koneksi ke -WROVER-KIT. ESP32

```
| 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             |
```

Tabel ini dikembangkan dari lembar data FTDI C232HM-DDHSL-0. Untuk informasi lebih lanjut, lihat bagian “Koneksi Kabel MPSSE C232HM dan Detail Mekanik di lembar data.

Untuk mengaktifkan JTAG pada ESP-WROVER-KIT, tempatkan jumper pada pin TMS, TDO, TDI, TCK, dan S_TDI seperti yang ditunjukkan di sini.



Debugging pada Windows (ESP-IDF v4.2)

Untuk mengatur debugging pada Windows

1. Hubungkan sisi USB FTDI C232HM-DDHSL-0 ke komputer Anda dan sisi lain seperti yang dijelaskan dalam. Kode debugging pada Espressif ESP32 - DevKit C dan ESP-WROVER-KIT (ESP-IDF v4.2) Perangkat FTDI C232HM-DDHSL-0 akan muncul di Device Manager di bawah Universal Serial Bus Controllers.
2. Di bawah daftar perangkat bus serial universal, klik kanan perangkat C232HM-DDHSL-0, lalu pilih Properties.

   > **Note**
   >
   > Perangkat mungkin terdaftar sebagai Port Serial USB.

   Untuk melihat properti perangkat, di jendela properti, pilih tab Detail. Jika perangkat tidak terdaftar, instal driver Windows untuk FTDI C232HM-DDHSL-0.
3. Pada tab Detail, pilih Properti, lalu pilih Hardware IDs. Anda akan melihat sesuatu seperti ini di bidang Nilai.

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

   Dalam contoh ini, ID vendor adalah 0403 dan ID produk adalah 6014.

Verifikasi ini IDs cocok dengan IDs in`projects/espressif/esp32/make/aws_demos/esp32_devkitj_v1.cfg`. IDs Ini ditentukan dalam baris yang dimulai dengan `ftdi_vid_pid` diikuti oleh ID vendor dan ID produk.

```
ftdi_vid_pid 0x0403 0x6014
```

4. Unduh OpenOCD untuk Windows.
5. Buka zip file ke `C:\` dan tambahkan `C:\openocd-esp32\bin` ke jalur sistem Anda.
6. OpenOCD membutuhkan libusb, yang tidak diinstal secara default pada Windows. Untuk menginstal libusb:
   a. Unduh zadig.exe.
   b. Jalankan `zadig.exe`. Dari menu Opsi, pilih Daftar Semua Perangkat.
   c. Dari menu tarik-turun, pilih C232HM-DDHSL-0.
   d. Di bidang driver target, di sebelah kanan panah hijau, pilih WinUSB.
   e. Untuk daftar di bawah bidang driver target, pilih panah, lalu pilih Install Driver. Pilih Ganti Driver.
7. Buka prompt perintah, arahkan ke root direktori unduhan FreeRTOS Anda, dan jalankan perintah berikut.

```
idf.py openocd
```

Biarkan command prompt ini terbuka.

8. Buka prompt perintah baru, arahkan ke root direktori unduhan FreeRTOS Anda, dan jalankan

```
idf.py flash monitor
```

9. Buka prompt perintah lain, arahkan ke root direktori unduhan FreeRTOS Anda, dan tunggu hingga demo mulai berjalan di papan Anda. Ketika itu terjadi, jalankan

```
idf.py gdb
```

Program harus berhenti dalam `main` fungsi.

> **Note**
>
> ESP32 Mendukung maksimal dua break point.

Debugging di macOS (ESP-IDF v4.2)

1. Unduh driver FTDI untuk macOS.
2. Unduh OpenOCD.
3. Ekstrak file.tar yang diunduh dan atur jalurnya `.bash_profile` ke`OCD_INSTALL_DIR/openocd-esp32/bin`.
4. Gunakan perintah berikut untuk menginstal `libusb` di macOS.

```
brew install libusb
```

5. Gunakan perintah berikut untuk membongkar driver port serial.

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

6. Gunakan perintah berikut untuk membongkar driver port serial.

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

7. Jika Anda menjalankan versi macOS lebih lambat dari 10.9, gunakan perintah berikut untuk membongkar driver Apple FTDI.

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

8. Gunakan perintah berikut untuk mendapatkan ID produk dan ID vendor kabel FTDI. Ini mencantumkan perangkat USB yang terpasang.

```
system_profiler SPUSBDataType
```

   Output dari `system_profiler` akan terlihat seperti berikut ini.

```
   DEVICE:

   Product ID: product-ID
```

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

9. Buka file `projects/espressif/esp32/make/aws_demos/esp32_devkitj_v1.cfg`. ID vendor dan ID produk untuk perangkat Anda ditentukan dalam baris yang dimulai dengan`ftdi_vid_pid`. Ubah IDs untuk mencocokkan IDs dari `system_profiler` output pada langkah sebelumnya.
10. Buka jendela terminal, arahkan ke root direktori unduhan FreeRTOS Anda, dan gunakan perintah berikut untuk menjalankan OpenOCD.

    ```
    idf.py openocd
    ```

    Biarkan jendela terminal ini terbuka.
11. Buka terminal baru, dan gunakan perintah berikut untuk memuat driver port serial FTDI.

    ```
    sudo kextload -b com.FTDI.driver.FTDIUSBSerialDriver
    ```
12. Arahkan ke root direktori unduhan FreeRTOS Anda, dan jalankan

    ```
    idf.py flash monitor
    ```
13. Buka terminal baru lainnya, arahkan ke root direktori unduhan FreeRTOS Anda, dan jalankan

    ```
    idf.py gdb
    ```

    Program harus berhenti di`main`.

Debugging di Linux (ESP-IDF v4.2)

1. Unduh OpenOCD. Ekstrak tarball dan ikuti petunjuk penginstalan di file readme.
2. Gunakan perintah berikut untuk menginstal libusb di Linux.

    ```
    sudo apt-get install libusb-1.0
    ```
3. Buka terminal dan masukkan ls -l /dev/ttyUSB* untuk mencantumkan semua perangkat USB yang terhubung ke komputer Anda. Ini membantu Anda memeriksa apakah port USB papan dikenali oleh sistem operasi. Anda akan melihat output seperti berikut.

    ```
    $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. Masuk dan kemudian masuk dan putar daya ke papan untuk membuat perubahan berlaku. Dalam prompt terminal, daftarkan perangkat USB. Pastikan pemilik grup telah berubah dari `dialout` ke`plugdev`.

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

   `/dev/ttyUSBn`Antarmuka dengan angka yang lebih rendah digunakan untuk komunikasi JTAG. Antarmuka lainnya dirutekan ke ESP32 port serial (UART) dan digunakan untuk mengunggah kode ke memori flash ESP32.

5. Di jendela terminal, arahkan ke root direktori unduhan FreeRTOS Anda, dan gunakan perintah berikut untuk menjalankan OpenOCD.

```
idf.py openocd
```

6. Buka terminal lain, arahkan ke root direktori unduhan FreeRTOS Anda, dan jalankan perintah berikut.

```
idf.py flash monitor
```

7. Buka terminal lain, navigasikan root direktori unduhan FreeRTOS Anda, dan jalankan perintah berikut:

```
idf.py gdb
```

   Program harus berhenti`main()`.

Memulai dengan ESP32 Espressif -WROOM-32SE

> **Important**
>
> Integrasi referensi ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-Freertos yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

> **Note**
>
> - Untuk mengeksplorasi cara mengintegrasikan pustaka dan demo modular FreeRTOS dalam proyek Espressif IDF Anda sendiri, lihat integrasi referensi unggulan kami untuk platform -C3. ESP32
> - Saat ini, port FreeRTOS ESP32 untuk -WROOM-32SE tidak mendukung fitur multiprocessing simetris (SMP).

Tutorial ini menunjukkan kepada Anda bagaimana memulai dengan Espressif ESP32 - WROOM-32SE. Untuk membeli satu dari mitra kami di katalog Perangkat AWS Mitra, lihat ESP32-WROOM-32SE.

Gambaran Umum

Tutorial ini memandu Anda melalui langkah-langkah berikut:

1. Hubungkan papan Anda ke mesin host.
2. Instal perangkat lunak pada mesin host Anda untuk mengembangkan dan men-debug aplikasi tertanam untuk papan mikrokontroler Anda.
3. Kompilasi silang aplikasi demo FreeRTOS ke gambar biner.
4. Muat gambar biner aplikasi ke papan Anda, lalu jalankan aplikasi.
5. Pantau dan debug aplikasi yang sedang berjalan dengan menggunakan koneksi serial.

Prasyarat

Sebelum Anda memulai dengan FreeRTOS di papan Espressif Anda, Anda harus mengatur akun dan izin Anda. AWS

Mendaftar untuk Akun AWS

Jika Anda tidak memiliki Akun AWS, selesaikan langkah-langkah berikut untuk membuatnya.

Untuk mendaftar untuk Akun AWS

1. Buka https://portal.aws.amazon.com/billing/pendaftaran.
2. Ikuti petunjuk online.

   Bagian dari prosedur pendaftaran melibatkan menerima panggilan telepon atau pesan teks dan memasukkan kode verifikasi pada keypad telepon.

   Saat Anda mendaftar untuk sebuah Akun AWS, sebuah Pengguna root akun AWSdibuat. Pengguna root memiliki akses ke semua Layanan AWS dan sumber daya di akun. Sebagai praktik keamanan terbaik, tetapkan akses administratif ke pengguna, dan gunakan hanya pengguna root untuk melakukan tugas yang memerlukan akses pengguna root.

AWS mengirimi Anda email konfirmasi setelah proses pendaftaran selesai. Kapan saja, Anda dapat melihat aktivitas akun Anda saat ini dan mengelola akun Anda dengan masuk https://aws.amazon.comke/ dan memilih Akun Saya.

Buat pengguna dengan akses administratif

Setelah Anda mendaftar Akun AWS, amankan Pengguna root akun AWS, aktifkan AWS IAM Identity Center, dan buat pengguna administratif sehingga Anda tidak menggunakan pengguna root untuk tugas sehari-hari.

Amankan Anda Pengguna root akun AWS

1. Masuk ke AWS Management Consolesebagai pemilik akun dengan memilih pengguna Root dan memasukkan alamat Akun AWS email Anda. Di laman berikutnya, masukkan kata sandi.

   Untuk bantuan masuk dengan menggunakan pengguna root, lihat Masuk sebagai pengguna root di AWS Sign-In Panduan Pengguna.
2. Mengaktifkan autentikasi multi-faktor (MFA) untuk pengguna root Anda.

Untuk petunjuk, lihat Mengaktifkan perangkat MFA virtual untuk pengguna Akun AWS root (konsol) Anda di Panduan Pengguna IAM.

Buat pengguna dengan akses administratif

1. Aktifkan Pusat Identitas IAM.

   Untuk mendapatkan petunjuk, silakan lihat Mengaktifkan AWS IAM Identity Center di Panduan Pengguna AWS IAM Identity Center .

2. Di Pusat Identitas IAM, berikan akses administratif ke pengguna.

   Untuk tutorial tentang menggunakan Direktori Pusat Identitas IAM sebagai sumber identitas Anda, lihat Mengkonfigurasi akses pengguna dengan default Direktori Pusat Identitas IAM di Panduan AWS IAM Identity Center Pengguna.

Masuk sebagai pengguna dengan akses administratif

- Untuk masuk dengan pengguna Pusat Identitas IAM, gunakan URL masuk yang dikirim ke alamat email saat Anda membuat pengguna Pusat Identitas IAM.

  Untuk bantuan masuk menggunakan pengguna Pusat Identitas IAM, lihat Masuk ke portal AWS akses di Panduan AWS Sign-In Pengguna.

Tetapkan akses ke pengguna tambahan

1. Di Pusat Identitas IAM, buat set izin yang mengikuti praktik terbaik menerapkan izin hak istimewa paling sedikit.

   Untuk petunjuknya, lihat Membuat set izin di Panduan AWS IAM Identity Center Pengguna.

2. Tetapkan pengguna ke grup, lalu tetapkan akses masuk tunggal ke grup.

   Untuk petunjuk, lihat Menambahkan grup di Panduan AWS IAM Identity Center Pengguna.

Untuk memberikan akses dan menambahkan izin bagi pengguna, grup, atau peran Anda:

- Pengguna dan grup di AWS IAM Identity Center:

Buat rangkaian izin. Ikuti instruksi di Buat rangkaian izin di Panduan Pengguna AWS IAM Identity Center .

- Pengguna yang dikelola di IAM melalui penyedia identitas:

  Buat peran untuk federasi identitas. Ikuti instruksi dalam Buat peran untuk penyedia identitas pihak ketiga (federasi) dalam Panduan Pengguna IAM.
- Pengguna IAM:
  - Buat peran yang dapat diambil pengguna Anda. Ikuti instruksi dalam Buat peran untuk pengguna IAM dalam Panduan Pengguna IAM.
  - (Tidak disarankan) Lampirkan kebijakan langsung ke pengguna atau tambahkan pengguna ke grup pengguna. Ikuti petunjuk dalam Menambahkan izin ke pengguna (konsol) dalam Panduan Pengguna IAM.

Memulai

> **Note**
>
> Perintah Linux dalam tutorial ini mengharuskan Anda menggunakan shell Bash.

1. Siapkan perangkat keras Espressif.

   Untuk informasi tentang menyiapkan perangkat keras papan pengembangan ESP32 - WROOM-32SE, lihat Panduan Memulai ESP32- DevKit C V4.

   > **Important**
   >
   > Ketika Anda mencapai bagian Instalasi Langkah demi Langkah dari panduan ini, ikuti sampai Anda menyelesaikan Langkah 4 (Siapkan variabel lingkungan). Berhenti setelah Anda menyelesaikan Langkah 4 dan ikuti langkah-langkah yang tersisa di sini.

2. Unduh Amazon GitHubFreeRTOS dari. (Untuk instruksi, lihat file README.md.)
3. Siapkan lingkungan pengembangan Anda.

   Untuk berkomunikasi dengan papan Anda, Anda harus menginstal rantai alat. Espressif menyediakan ESP-IDF untuk mengembangkan perangkat lunak untuk papan mereka. Karena ESP-IDF memiliki versi sendiri dari FreeRTOS Kernel terintegrasi sebagai komponen, Amazon

FreeRTOS menyertakan versi kustom ESP-IDF v4.2 yang memiliki FreeRTOS Kernel dihapus. Ini memperbaiki masalah dengan file duplikat saat Anda mengkompilasi. Untuk menggunakan versi kustom ESP-IDF v4.2 yang disertakan dengan Amazon FreeRTOS, ikuti petunjuk di bawah ini untuk sistem operasi mesin host Anda.

Windows

1. Unduh Pemasang Online Universal ESP-IDF untuk Windows.
2. Jalankan Pemasang Online Universal.
3. Ketika Anda sampai ke langkah Unduh atau gunakan ESP-IDF, pilih Gunakan direktori ESP-IDF yang ada dan atur Pilih direktori ESP-IDF yang ada ke. *freertos*`/vendors/espressif/esp-idf`
4. Selesaikan instalasi.

macOS

1. Ikuti petunjuk dalam Pengaturan Standar prasyarat Toolchain (ESP-IDF v4.2) untuk macOS.

   > **Important**
   >
   > Ketika Anda mencapai petunjuk “Dapatkan ESP-IDF” di bawah Langkah Berikutnya, berhenti, dan kemudian kembali ke petunjuk di halaman ini.

2. Buka jendela baris perintah.
3. Arahkan ke direktori unduhan FreeRTOS, lalu jalankan skrip berikut untuk mengunduh dan menginstal rantai alat espressif untuk platform Anda.

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

4. Tambahkan alat toolchain ESP-IDF ke jalur terminal Anda dengan perintah berikut.

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

Linux

1. Ikuti petunjuk dalam Pengaturan Standar prasyarat Toolchain (ESP-IDF v4.2) untuk Linux.

> **Important**
>
> Ketika Anda mencapai petunjuk “Dapatkan ESP-IDF” di bawah Langkah Berikutnya, berhenti, dan kemudian kembali ke petunjuk di halaman ini.

2. Buka jendela baris perintah.
3. Arahkan ke direktori unduhan FreeRTOS, lalu jalankan skrip berikut untuk mengunduh dan menginstal rantai alat Espressif untuk platform Anda.

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

4. Tambahkan alat toolchain ESP-IDF ke jalur terminal Anda dengan perintah berikut.

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

4. Buat koneksi serial.

   a. Untuk membuat koneksi serial antara mesin host Anda dan ESP32 -WROOM-32SE, instal driver CP21 0x USB ke UART Bridge VCP. Anda dapat mengunduh driver ini dari Silicon Labs.

   b. Ikuti langkah-langkah untuk Membuat Koneksi Serial dengan ESP32.

   c. Setelah Anda membuat koneksi serial, catat port serial untuk koneksi papan Anda. Anda membutuhkannya untuk mem-flash demo.

Konfigurasikan aplikasi demo FreeRTOS

Untuk tutorial ini, file konfigurasi FreeRTOS terletak di. *freertos*`/vendors/espressif/boards/`*board-name*`/aws_demos/config_files/FreeRTOSConfig.h` (Misalnya, jika `AFR_BOARD espressif.esp32_devkitc` dipilih, file konfigurasi terletak di*freertos*`/vendors/espressif/boards/esp32/aws_demos/config_files/FreeRTOSConfig.h`.)

> **Important**
>
> Perangkat ATECC6 08A memiliki inisialisasi satu kali yang dikunci ke perangkat saat pertama kali proyek dijalankan (selama panggilan ke). `C_InitToken` Namun, proyek demo FreeRTOS dan proyek pengujian memiliki konfigurasi yang berbeda. Jika perangkat terkunci

selama konfigurasi proyek demo, tidak semua pengujian dalam proyek pengujian akan berhasil.

1. Konfigurasikan Proyek Demo FreeRTOS dengan mengikuti langkah-langkah di. Mengkonfigurasi demo FreeRTOS Saat Anda sampai ke langkah terakhir Untuk memformat AWS IoT kredensil Anda, hentikan, dan lakukan langkah-langkah berikut.
2. Microchip telah menyediakan beberapa alat skrip untuk membantu pengaturan bagian ATECC6 08A. Arahkan ke *freertos*`/vendors/microchip/example_trust_chain_tool` direktori, dan buka `README.md` file.
3. Untuk menyediakan perangkat Anda, ikuti instruksi dalam `README.md` file. Langkah-langkahnya meliputi:
    1. Buat dan daftarkan otoritas sertifikat dengan AWS.
    2. Hasilkan kunci Anda pada ATECC6 08A dan ekspor kunci publik dan nomor seri perangkat.
    3. Buat sertifikat untuk perangkat dan daftarkan sertifikat itu AWS.
4. Muat sertifikat CA dan sertifikat perangkat ke perangkat dengan mengikuti petunjuk untukPenyediaan kunci mode pengembang.

Memantau pesan MQTT di Cloud AWS

Sebelum menjalankan proyek demo FreeRTOS, Anda dapat mengatur klien MQTT di konsol untuk memantau pesan AWS IoT yang dikirim perangkat Anda ke Cloud. AWS

Untuk berlangganan topik MQTT dengan klien MQTT AWS IoT

1. Masuk ke konsol AWS IoT tersebut.
2. Di panel navigasi, pilih Uji, lalu pilih MQTT Test Client.
3. Dalam Subscription topic, masukkan *your-thing-name*`/example/topic` lalu pilih Subscribe to topic.

Bangun, flash, dan jalankan proyek demo FreeRTOS menggunakan skrip idf.py

Anda dapat menggunakan utilitas IDF Espressif (`idf.py`) untuk menghasilkan file build, membangun biner aplikasi, dan mem-flash binari ke perangkat Anda.

> **Note**
>
> Beberapa pengaturan mungkin mengharuskan Anda menggunakan opsi port "`-p port-name`" dengan `idf.py` untuk menentukan port yang benar, seperti pada contoh berikut.
>
> ```
> idf.py -p /dev/cu.usbserial-00101301B flash
> ```

Membangun dan mem-flash FreeRTOS di Windows, Linux, dan macOS (ESP-IDF v4.2)

1. Arahkan ke root direktori unduhan FreeRTOS Anda.
2. Di jendela baris perintah, masukkan perintah berikut untuk menambahkan alat ESP-IDF ke PATH terminal Anda:

   Windows (aplikasi “Perintah”)

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

   Windows (aplikasi “ESP-IDF 4.x CMD”)

   (Ini sudah dilakukan saat Anda membuka aplikasi.)

   Linux/macOS

   ```
   source vendors/espressif/esp-idf/export.sh
   ```
3. Konfigurasikan cmake di `build` direktori dan buat gambar firmware dengan perintah berikut.

   ```
   idf.py -DVENDOR=espressif -DBOARD=esp32_ecc608a_devkitc -DCOMPILER=xtensa-esp32
    build
   ```

   Anda akan melihat output seperti contoh berikut ini.

   ```
   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'
```

Jika tidak ada kesalahan, build akan menghasilkan file biner firmware .bin.

4. Hapus memori flash papan pengembangan Anda dengan perintah berikut.

```
idf.py erase_flash
```

5. Gunakan `idf.py` skrip untuk mem-flash biner aplikasi ke papan Anda.

```
idf.py flash
```

6. Pantau output dari port serial papan Anda dengan perintah berikut.

```
idf.py monitor
```

> **Note**
>
> - Anda dapat menggabungkan perintah ini seperti pada contoh berikut.
>
>   ```
>   idf.py erase_flash flash monitor
>   ```
>
> - Untuk pengaturan mesin host tertentu, Anda harus menentukan port saat Anda mem-flash papan seperti pada contoh berikut.
>
>   ```
>   idf.py erase_flash flash monitor -p /dev/ttyUSB1
>   ```

Membangun dan Flash FreeRTOS dengan CMake

Selain menggunakan `idf.py` skrip yang disediakan oleh SDK IDF untuk membangun dan menjalankan kode Anda, Anda juga dapat membangun proyek dengan. CMake Saat ini mendukung Unix Makefile dan sistem build Ninja.

Untuk membangun dan mem-flash proyek

1. Di jendela baris perintah, arahkan ke root direktori unduhan FreeRTOS Anda.
2. Jalankan skrip berikut untuk menambahkan alat ESP-IDF ke PATH shell Anda.

   Windows

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

   Linux/macOS

   ```
   source vendors/espressif/esp-idf/export.sh
   ```
3. Masukkan perintah berikut untuk menghasilkan file build.

   Dengan 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
   ```

   Dengan 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. Hapus flash dan kemudian flash papan.

   Dengan Unix Makefiles

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

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

Dengan Ninja

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

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

Informasi tambahan

Untuk informasi selengkapnya tentang penggunaan dan pemecahan masalah ESP32 papan Espressif, lihat topik berikut:

- Menggunakan FreeRTOS dalam proyek Anda sendiri untuk CMake ESP32
- Pemecahan Masalah
- Debugging

Memulai dengan ESP32 Espressif -S2

> **Important**
>
> Integrasi referensi ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-Freertos yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

> **Note**
>
> Untuk mengeksplorasi cara mengintegrasikan pustaka dan demo modular FreeRTOS dalam proyek Espressif IDF Anda sendiri, lihat integrasi referensi unggulan kami untuk platform -C3. ESP32

Tutorial ini menunjukkan kepada Anda bagaimana memulai dengan papan pengembangan Espressif ESP32 -S2 SoC dan -S2-Saola-1. ESP32

Gambaran Umum

Tutorial ini memandu Anda melalui langkah-langkah berikut:

1. Hubungkan papan Anda ke mesin host.
2. Instal perangkat lunak pada mesin host Anda untuk mengembangkan dan men-debug aplikasi tertanam untuk papan mikrokontroler Anda.
3. Kompilasi silang aplikasi demo FreerTOS ke gambar biner.
4. Muat gambar biner aplikasi ke papan Anda, lalu jalankan aplikasi.
5. Pantau dan debug aplikasi yang sedang berjalan menggunakan koneksi serial.

Prasyarat

Sebelum Anda memulai dengan FreeRTOS di papan Espressif Anda, Anda harus mengatur akun dan izin Anda. AWS

Mendaftar untuk Akun AWS

Jika Anda tidak memiliki Akun AWS, selesaikan langkah-langkah berikut untuk membuatnya.

Untuk mendaftar untuk Akun AWS

1. Buka https://portal.aws.amazon.com/billing/pendaftaran.
2. Ikuti petunjuk online.

   Bagian dari prosedur pendaftaran melibatkan menerima panggilan telepon atau pesan teks dan memasukkan kode verifikasi pada keypad telepon.

   Saat Anda mendaftar untuk sebuah Akun AWS, sebuah Pengguna root akun AWSdibuat. Pengguna root memiliki akses ke semua Layanan AWS dan sumber daya di akun. Sebagai praktik keamanan terbaik, tetapkan akses administratif ke pengguna, dan gunakan hanya pengguna root untuk melakukan tugas yang memerlukan akses pengguna root.

AWS mengirimi Anda email konfirmasi setelah proses pendaftaran selesai. Kapan saja, Anda dapat melihat aktivitas akun Anda saat ini dan mengelola akun Anda dengan masuk https://aws.amazon.comke/ dan memilih Akun Saya.

Buat pengguna dengan akses administratif

Setelah Anda mendaftar Akun AWS, amankan Pengguna root akun AWS, aktifkan AWS IAM Identity Center, dan buat pengguna administratif sehingga Anda tidak menggunakan pengguna root untuk tugas sehari-hari.

Amankan Pengguna root akun AWS

1. Masuk ke AWS Management Consolesebagai pemilik akun dengan memilih pengguna Root dan memasukkan alamat Akun AWS email Anda. Di laman berikutnya, masukkan kata sandi.

   Untuk bantuan masuk dengan menggunakan pengguna root, lihat Masuk sebagai pengguna root di AWS Sign-In Panduan Pengguna.
2. Mengaktifkan autentikasi multi-faktor (MFA) untuk pengguna root Anda.

   Untuk petunjuk, lihat Mengaktifkan perangkat MFA virtual untuk pengguna Akun AWS root (konsol) Anda di Panduan Pengguna IAM.

Buat pengguna dengan akses administratif

1. Aktifkan Pusat Identitas IAM.

   Untuk mendapatkan petunjuk, silakan lihat Mengaktifkan AWS IAM Identity Center di Panduan Pengguna AWS IAM Identity Center .
2. Di Pusat Identitas IAM, berikan akses administratif ke pengguna.

   Untuk tutorial tentang menggunakan Direktori Pusat Identitas IAM sebagai sumber identitas Anda, lihat Mengkonfigurasi akses pengguna dengan default Direktori Pusat Identitas IAM di Panduan AWS IAM Identity Center Pengguna.

Masuk sebagai pengguna dengan akses administratif

- Untuk masuk dengan pengguna Pusat Identitas IAM, gunakan URL masuk yang dikirim ke alamat email saat Anda membuat pengguna Pusat Identitas IAM.

  Untuk bantuan masuk menggunakan pengguna Pusat Identitas IAM, lihat Masuk ke portal AWS akses di Panduan AWS Sign-In Pengguna.

Tetapkan akses ke pengguna tambahan

1. Di Pusat Identitas IAM, buat set izin yang mengikuti praktik terbaik menerapkan izin hak istimewa paling sedikit.

   Untuk petunjuknya, lihat Membuat set izin di Panduan AWS IAM Identity Center Pengguna.
2. Tetapkan pengguna ke grup, lalu tetapkan akses masuk tunggal ke grup.

   Untuk petunjuk, lihat Menambahkan grup di Panduan AWS IAM Identity Center Pengguna.

Untuk memberikan akses dan menambahkan izin bagi pengguna, grup, atau peran Anda:

- Pengguna dan grup di AWS IAM Identity Center:

  Buat rangkaian izin. Ikuti instruksi di Buat rangkaian izin di Panduan Pengguna AWS IAM Identity Center .
- Pengguna yang dikelola di IAM melalui penyedia identitas:

  Buat peran untuk federasi identitas. Ikuti instruksi dalam Buat peran untuk penyedia identitas pihak ketiga (federasi) dalam Panduan Pengguna IAM.
- Pengguna IAM:
  - Buat peran yang dapat diambil pengguna Anda. Ikuti instruksi dalam Buat peran untuk pengguna IAM dalam Panduan Pengguna IAM.
  - (Tidak disarankan) Lampirkan kebijakan langsung ke pengguna atau tambahkan pengguna ke grup pengguna. Ikuti petunjuk dalam Menambahkan izin ke pengguna (konsol) dalam Panduan Pengguna IAM.

Memulai

> **Note**
>
> Perintah Linux dalam tutorial ini mengharuskan Anda menggunakan shell Bash.

1. Siapkan perangkat keras Espressif.

   Untuk informasi tentang menyiapkan perangkat keras papan pengembangan ESP32 -S2, lihat Panduan Memulai ESP32-S2-Saola-1.

> **Important**
>
> Ketika Anda mencapai bagian Memulai dari panduan Espressif, berhenti, dan kemudian kembali ke instruksi di halaman ini.

2. Unduh Amazon GitHubFreeRTOS dari. (Untuk instruksi, lihat file README.md.)
3. Siapkan lingkungan pengembangan Anda.

   Untuk berkomunikasi dengan papan Anda, Anda harus menginstal rantai alat. Espressif menyediakan ESP-IDF untuk mengembangkan perangkat lunak untuk papan mereka. Karena ESP-IDF memiliki versi sendiri dari FreeRTOS Kernel terintegrasi sebagai komponen, Amazon FreeRTOS menyertakan versi kustom ESP-IDF v4.2 yang memiliki FreeRTOS Kernel dihapus. Ini memperbaiki masalah dengan file duplikat saat Anda mengkompilasi. Untuk menggunakan versi kustom ESP-IDF v4.2 yang disertakan dengan Amazon FreeRTOS, ikuti petunjuk di bawah ini untuk sistem operasi mesin host Anda.

   Windows

   1. Unduh Pemasang Online Universal ESP-IDF untuk Windows.
   2. Jalankan Pemasang Online Universal.
   3. Ketika Anda sampai ke langkah Unduh atau gunakan ESP-IDF, pilih Gunakan direktori ESP-IDF yang ada dan atur Pilih direktori ESP-IDF yang ada ke. *`freertos`*`/vendors/espressif/esp-idf`
   4. Selesaikan instalasi.

   macOS

   1. Ikuti petunjuk dalam Pengaturan Standar prasyarat Toolchain (ESP-IDF v4.2) untuk macOS.

      > **Important**
      >
      > Ketika Anda mencapai petunjuk “Dapatkan ESP-IDF” di bawah Langkah Berikutnya, berhenti, dan kemudian kembali ke petunjuk di halaman ini.

   2. Buka jendela baris perintah.

3. Arahkan ke direktori unduhan FreeRTOS, lalu jalankan skrip berikut untuk mengunduh dan menginstal rantai alat espressif untuk platform Anda.

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

4. Tambahkan alat toolchain ESP-IDF ke jalur terminal Anda dengan perintah berikut.

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

Linux

1. Ikuti petunjuk dalam Pengaturan Standar prasyarat Toolchain (ESP-IDF v4.2) untuk Linux.

> **Important**
>
> Ketika Anda mencapai petunjuk “Dapatkan ESP-IDF” di bawah Langkah Berikutnya, berhenti, dan kemudian kembali ke petunjuk di halaman ini.

2. Buka jendela baris perintah.
3. Arahkan ke direktori unduhan FreeRTOS, lalu jalankan skrip berikut untuk mengunduh dan menginstal rantai alat Espressif untuk platform Anda.

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

4. Tambahkan alat toolchain ESP-IDF ke jalur terminal Anda dengan perintah berikut.

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

4. Buat koneksi serial.

   a. Untuk membuat koneksi serial antara mesin host Anda dan ESP32 - DevKit C, instal driver CP21 0x USB ke UART Bridge VCP. Anda dapat mengunduh driver ini dari Silicon Labs.

   b. Ikuti langkah-langkah untuk Membuat Koneksi Serial dengan ESP32.

   c. Setelah Anda membuat koneksi serial, catat port serial untuk koneksi papan Anda. Anda membutuhkannya untuk mem-flash demo.

Konfigurasikan aplikasi demo FreeRTOS

Untuk tutorial ini, file konfigurasi FreeRTOS terletak di. *freertos*/vendors/espressif/boards/*board-name*/aws_demos/config_files/FreeRTOSConfig.h (Misalnya, jika AFR_BOARD espressif.esp32_devkitc dipilih, file konfigurasi terletak di*freertos*/vendors/espressif/boards/esp32/aws_demos/config_files/FreeRTOSConfig.h.)

1. Jika Anda menjalankan macOS atau Linux, buka prompt terminal. Jika Anda menjalankan Windows, buka aplikasi “ESP-IDF 4.x CMD” (jika Anda menyertakan opsi ini saat menginstal rantai alat ESP-IDF), atau aplikasi “Command Prompt” sebaliknya.
2. Untuk memverifikasi bahwa Anda telah menginstal Python3, jalankan yang berikut ini:

   ```
   python --version
   ```

   Versi yang diinstal ditampilkan. Jika Anda tidak menginstal Python 3.0.1 atau yang lebih baru, Anda dapat menginstalnya dari situs web Python.
3. Anda memerlukan AWS Command Line Interface (CLI) untuk menjalankan AWS IoT perintah. Jika Anda menjalankan Windows, gunakan `easy_install awscli` perintah untuk menginstal AWS CLI di aplikasi “Command” atau “ESP-IDF 4.x CMD”.

   Jika Anda menjalankan macOS atau Linux, lihat Menginstal CLI AWS.
4. Jalankan .

   ```
   aws configure
   ```

   dan konfigurasikan AWS CLI dengan ID kunci AWS akses Anda, kunci akses rahasia, dan Wilayah default AWS . Untuk informasi selengkapnya, lihat Mengonfigurasi AWS CLI.
5. Gunakan perintah berikut untuk menginstal AWS SDK untuk Python (boto3):

   - Di Windows, di aplikasi “Command” atau “ESP-IDF 4.x CMD”, jalankan

     ```
     easy_install boto3
     ```
   - Di macOS atau Linux, jalankan

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

     dan kemudian lari

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

FreeRTOS menyertakan `SetupAWS.py` skrip untuk membuatnya lebih mudah untuk mengatur papan Espressif Anda untuk terhubung. AWS IoT

Untuk menjalankan skrip konfigurasi

1. Untuk mengkonfigurasi skrip, buka *`freertos`*`/tools/aws_config_quick_start/configure.json` dan atur atribut berikut:

   `afr_source_dir`

   Jalur lengkap ke *`freertos`* direktori di komputer Anda. Pastikan Anda menggunakan garis miring maju untuk menentukan jalur ini.

   `thing_name`

   Nama yang ingin Anda tetapkan untuk AWS IoT hal yang mewakili papan Anda.

   `wifi_ssid`

   SSID jaringan Wi-Fi Anda.

   `wifi_password`

   Kata sandi untuk jaringan Wi-Fi Anda.

   `wifi_security`

   Jenis keamanan untuk jaringan Wi-Fi Anda. Berikut ini adalah jenis keamanan yang valid:

   - `eWiFiSecurityOpen`(Terbuka, tidak ada keamanan)
   - `eWiFiSecurityWEP`(Keamanan WEP)
   - `eWiFiSecurityWPA`(Keamanan WPA)
   - `eWiFiSecurityWPA2`(WPA2 keamanan)
2. Jika Anda menjalankan macOS atau Linux, buka prompt terminal. Jika Anda menjalankan Windows, buka aplikasi “ESP-IDF 4.x CMD” atau “Command”.
3. Arahkan ke *`freertos`*`/tools/aws_config_quick_start` direktori dan jalankan

```
python SetupAWS.py setup
```

Script melakukan hal berikut:

- Menciptakan AWS IoT sesuatu, sertifikat, dan kebijakan.
- Melampirkan AWS IoT kebijakan ke sertifikat dan sertifikat untuk AWS IoT benda itu.
- Mengisi `aws_clientcredential.h` file dengan AWS IoT titik akhir, SSID Wi-Fi, dan kredensialnya.
- Memformat sertifikat dan kunci pribadi Anda dan menuliskannya ke file `aws_clientcredential_keys.h` header.

> **Note**
>
> Sertifikat di-hardcode hanya untuk tujuan demonstrasi. Aplikasi tingkat produksi harus menyimpan file-file ini di lokasi yang aman.

Untuk informasi selengkapnya`SetupAWS.py`, lihat `README.md` di *`freertos`*`/tools/aws_config_quick_start` direktori.

Memantau pesan MQTT di Cloud AWS

Sebelum menjalankan proyek demo FreeRTOS, Anda dapat mengatur klien MQTT di konsol untuk memantau pesan AWS IoT yang dikirim perangkat Anda ke Cloud. AWS

Untuk berlangganan topik MQTT dengan klien MQTT AWS IoT

1. Masuk ke konsol AWS IoT tersebut.
2. Di panel navigasi, pilih Uji, lalu pilih MQTT Test Client.
3. Dalam Subscription topic, masukkan*`your-thing-name`*`/example/topic`, lalu pilih Subscribe to topic.

Ketika proyek demo berhasil berjalan di perangkat Anda, Anda melihat “Hello World!” dikirim beberapa kali ke topik yang Anda berlangganan.

Bangun, flash, dan jalankan proyek demo FreeRTOS menggunakan skrip idf.py

Anda dapat menggunakan utilitas IDF Espressif untuk menghasilkan file build, membangun biner aplikasi, dan mem-flash papan Anda.

Membangun dan mem-flash FreeRTOS di Windows, Linux, dan macOS (ESP-IDF v4.2)

Gunakan `idf.py` skrip untuk membangun proyek dan mem-flash binari ke perangkat Anda.

> **Note**
>
> Beberapa pengaturan mungkin mengharuskan Anda menggunakan opsi `-p port-name` port `idf.py` untuk menentukan port yang benar, seperti pada contoh berikut.
>
> ```
> idf.py -p /dev/cu.usbserial-00101301B flash
> ```

Untuk membangun dan mem-flash proyek

1. Arahkan ke root direktori unduhan FreeRTOS Anda.
2. Di jendela baris perintah, masukkan perintah berikut untuk menambahkan alat ESP-IDF ke PATH terminal Anda:

   Windows (aplikasi “Perintah”)

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

   Windows (aplikasi “ESP-IDF 4.x CMD”)

   (Ini sudah dilakukan saat Anda membuka aplikasi.)

   Linux/macOS

   ```
   source vendors/espressif/esp-idf/export.sh
   ```
3. Konfigurasikan cmake di `build` direktori dan buat gambar firmware dengan perintah berikut.

   ```
   idf.py -DVENDOR=espressif -DBOARD=esp32s2_saola_1 -DCOMPILER=xtensa-esp32s2 build
   ```

   Anda akan melihat output seperti contoh berikut ini.

```
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'
```

Jika tidak ada kesalahan, build menghasilkan file biner firmware .bin.

4. Hapus memori flash papan pengembangan Anda dengan perintah berikut.

```
idf.py erase_flash
```

5. Gunakan `idf.py` skrip untuk mem-flash biner aplikasi ke papan Anda.

```
idf.py flash
```

6. Pantau output dari port serial papan Anda dengan perintah berikut.

```
idf.py monitor
```

> **Note**
>
> - Anda dapat menggabungkan perintah ini seperti pada contoh berikut.

```
idf.py erase_flash flash monitor
```

- Untuk pengaturan mesin host tertentu, Anda harus menentukan port saat Anda mem-flash papan seperti pada contoh berikut.

```
idf.py erase_flash flash monitor -p /dev/ttyUSB1
```

Membangun dan Flash FreeRTOS dengan CMake

Selain menggunakan `idf.py` skrip yang disediakan oleh SDK IDF untuk membangun dan menjalankan kode Anda, Anda juga dapat membangun proyek dengan. CMake Saat ini mendukung Unix Makefile dan sistem build Ninja.

Untuk membangun dan mem-flash proyek

1. Di jendela baris perintah, arahkan ke root direktori unduhan FreeRTOS Anda.
2. Jalankan skrip berikut untuk menambahkan alat ESP-IDF ke PATH shell Anda.
   - Windows

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

   - Linux/macOS

     ```
     source vendors/espressif/esp-idf/export.sh
     ```
3. Masukkan perintah berikut untuk menghasilkan file build.
   - Dengan 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
     ```

   - Dengan 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. Bangun proyek.

- Dengan Unix Makefiles

```
make -C ./YOUR_BUILD_DIRECTORY -j8
```

- Dengan Ninja

```
ninja -C ./YOUR_BUILD_DIRECTORY -j8
```

5. Hapus flash dan kemudian flash papan.

- Dengan Unix Makefiles

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

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

- Dengan Ninja

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

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

Informasi tambahan

Untuk informasi selengkapnya tentang penggunaan dan pemecahan masalah ESP32 papan Espressif, lihat topik berikut:

- Menggunakan FreeRTOS dalam proyek Anda sendiri untuk CMake ESP32
- Pemecahan Masalah
- Debugging

Memulai dengan Infineon XMC48 00 IoT Connectivity Kit

> **Important**
>
> Integrasi referensi ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki

proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-Freertos yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Tutorial ini memberikan instruksi untuk memulai dengan Infineon XMC48 00 IoT Connectivity Kit. Jika Anda tidak memiliki Kit Konektivitas IoT Infineon XMC48 00, kunjungi Katalog AWS Perangkat Mitra untuk membelinya dari mitra kami.

Jika Anda ingin membuka koneksi serial dengan papan untuk melihat informasi logging dan debugging, Anda memerlukan USB/Serial converter, in addition to the XMC4800 IoT Connectivity Kit. The CP2104 is a common USB/Serial konverter 3.3V yang tersedia secara luas di papan seperti Adafruit's 04 Friend. CP21

Sebelum memulai, Anda harus mengonfigurasi AWS IoT dan mengunduh FreeRTOS Anda untuk menghubungkan perangkat Anda ke Cloud. AWS Lihat Langkah pertama untuk instruksi. Dalam tutorial ini, jalur ke direktori unduhan FreeRTOS disebut sebagai. `freertos`

Gambaran Umum

Tutorial ini berisi petunjuk untuk langkah-langkah memulai berikut:

1. Menginstal perangkat lunak pada mesin host untuk mengembangkan dan men-debug aplikasi tertanam untuk papan mikrokontroler Anda.
2. Menyusun silang aplikasi demo FreeRTOS ke gambar biner.
3. Memuat gambar biner aplikasi ke papan Anda, dan kemudian menjalankan aplikasi.
4. Berinteraksi dengan aplikasi yang berjalan di papan Anda di seluruh koneksi serial, untuk tujuan pemantauan dan debugging.

Siapkan lingkungan pengembangan Anda

FreeRTOS menggunakan lingkungan pengembangan DAVE Infineon untuk memprogram 00. XMC48 Sebelum Anda mulai, Anda perlu mengunduh dan menginstal DAVE dan beberapa driver J-Link untuk berkomunikasi dengan debugger on-board.

Instal DAVE

1. Buka halaman unduhan perangkat lunak DAVE Infineon.

2. Pilih paket DAVE untuk sistem operasi Anda dan kirimkan informasi pendaftaran Anda. Setelah mendaftar dengan Infineon, Anda akan menerima email konfirmasi dengan tautan untuk mengunduh file.zip.
3. Unduh paket DAVE. File zip (`DAVE_version_os_date.zip`), dan unzip ke lokasi di mana Anda ingin menginstal DAVE (misalnya,). `C:\DAVE4`

> **Note**
>
> Beberapa pengguna Windows telah melaporkan masalah menggunakan Windows Explorer untuk membuka zip file. Kami menyarankan Anda menggunakan program pihak ketiga seperti 7-Zip.

4. Untuk meluncurkan DAVE, jalankan file yang dapat dieksekusi yang ditemukan di folder yang tidak di-zip. `DAVE_version_os_date.zip`

Untuk informasi selengkapnya, lihat Panduan Mulai Cepat DAVE.

Instal driver Segger J-Link

Untuk berkomunikasi dengan probe debugging on-board XMC48 00 Relax EtherCat board, Anda memerlukan driver yang disertakan dalam paket Perangkat Lunak dan Dokumentasi J-Link. Anda dapat mengunduh paket Perangkat Lunak dan Dokumentasi J-Link dari halaman unduhan perangkat lunak J-Link Segger.

Buat koneksi serial

Menyiapkan koneksi serial adalah opsional, tetapi disarankan. Koneksi serial memungkinkan papan Anda mengirim informasi logging dan debugging dalam bentuk yang dapat Anda lihat di mesin pengembangan Anda.

Aplikasi demo XMC48 00 menggunakan koneksi serial UART pada pin P0.0 dan P0.1, yang diberi label pada silkscreen papan 00 XMC48 Relax EtherCat. Untuk mengatur koneksi serial:

1. Hubungkan pin berlabel “RX<P0.0" ke pin “TX” USB/konverter serial Anda.
2. Hubungkan pin berlabel “TX> P0.1" ke pin “RX” USB/konverter serial Anda.
3. Hubungkan pin Ground konverter serial Anda ke salah satu pin berlabel “GND” di papan Anda. Perangkat harus berbagi kesamaan.

Daya disuplai dari port debugging USB, jadi jangan sambungkan pin tegangan positif adaptor serial Anda ke papan.

> **Note**
>
> Beberapa kabel serial menggunakan level pensinyalan 5V. Papan XMC48 00 dan modul Klik Wi-Fi memerlukan 3.3V. Jangan gunakan jumper IOREF papan untuk mengubah sinyal papan menjadi 5V.

Dengan kabel yang terhubung, Anda dapat membuka koneksi serial pada emulator terminal seperti Layar GNU. Baud rate diatur ke 115200 secara default dengan 8 bit data, tidak ada paritas, dan 1 stop bit.

Memantau pesan MQTT di cloud

Sebelum menjalankan demo FreeRTOS, Anda dapat mengatur klien MQTT di konsol untuk memantau pesan AWS IoT yang dikirim perangkat Anda ke Cloud. AWS

Untuk berlangganan topik MQTT dengan klien MQTT AWS IoT

1. Masuk ke konsol AWS IoT tersebut.
2. Di panel navigasi, pilih Uji, lalu pilih klien pengujian MQTT untuk membuka klien MQTT.
3. Dalam Subscription topic, masukkan***your-thing-name*/example/topic**, lalu pilih Subscribe to topic.

Ketika proyek demo berhasil berjalan di perangkat Anda, Anda melihat “Hello World!” dikirim beberapa kali ke topik yang Anda berlangganan.

Bangun dan jalankan proyek demo FreeRTOS

Impor demo FreeRTOS ke DAVE

1. Mulai DAVE.
2. Di DAVE, pilih File, Impor. Di jendela Impor, perluas folder Infineon, pilih DAVE Project, lalu pilih Berikutnya.
3. Di jendela Impor Proyek DAVE, pilih Pilih Direktori Root, pilih Browse, dan kemudian pilih proyek demo XMC48 00.

Di direktori tempat Anda membuka ritsleting unduhan FreerTOS Anda, proyek demo berada di. `projects/infineon/xmc4800_iotkit/dave4/aws_demos`

Pastikan bahwa Copy Projects Into Workspace tidak dicentang.

4. Pilih Selesai.

`aws_demos`Proyek harus diimpor ke ruang kerja Anda dan diaktifkan.

5. Dari menu Project, pilih Build Active Project.

Pastikan proyek dibangun tanpa kesalahan.

Jalankan proyek demo FreeRTOS

1. Gunakan kabel USB untuk menghubungkan Kit Konektivitas XMC48 00 IoT Anda ke komputer Anda. Papan memiliki dua konektor microUSB. Gunakan yang berlabel “X101", di mana Debug muncul di sebelahnya di layar sutra papan.
2. Dari menu Project, pilih Rebuild Active Project untuk membangun kembali `aws_demos` dan memastikan bahwa perubahan konfigurasi Anda diambil.
3. Dari Project Explorer, klik kanan`aws_demos`, pilih Debug As, dan kemudian pilih DAVE C/C++ Application.
4. Klik dua kali GDB SEGGER J-Link Debugging untuk membuat konfirmasi debug. Pilih Debug.
5. Ketika debugger berhenti di breakpoint di**`main()`**, dari menu Run, pilih Lanjutkan.

Di AWS IoT konsol, klien MQTT dari langkah 4-5 harus menampilkan pesan MQTT yang dikirim oleh perangkat Anda. Jika Anda menggunakan koneksi serial, Anda melihat sesuatu seperti ini pada 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----
```

Bangun demo FreeRTOS dengan CMake

Jika Anda memilih untuk tidak menggunakan IDE untuk pengembangan FreeRTOS, Anda dapat CMake menggunakan alternatif untuk membangun dan menjalankan aplikasi demo atau aplikasi yang telah Anda kembangkan menggunakan editor kode pihak ketiga dan alat debugging.

> **Note**
>
> Bagian ini mencakup penggunaan CMake pada Windows dengan MinGW sebagai sistem build asli. Untuk informasi selengkapnya tentang penggunaan CMake dengan sistem operasi dan opsi lain, lihatMenggunakan CMake dengan FreeRTOS. (MinGW adalah lingkungan pengembangan minimalis untuk aplikasi Microsoft Windows asli.)

Untuk membangun demo FreeRTOS dengan CMake

1. Siapkan GNU Arm Embedded Toolchain.

   a. Unduh toolchain versi Windows dari halaman unduhan Arm Embedded Toolchain.

   > **Note**
   >
   > Kami menyarankan Anda mengunduh versi selain “8-2018-q4-major”, karena bug yang dilaporkan dengan utilitas “objcopy” dalam versi itu.

   b. Buka installer toolchain yang diunduh, dan ikuti petunjuk wizard penginstalan untuk menginstal toolchain.

   > **Important**
   >
   > Pada halaman akhir wizard instalasi, pilih Tambahkan jalur ke variabel lingkungan untuk menambahkan jalur toolchain ke variabel lingkungan jalur sistem.

2. Instal CMake dan MingW.

   Untuk instruksi, lihat CMake Prasyarat.

3. Buat folder untuk berisi file build yang dihasilkan (*`build-folder`*).

4. Ubah direktori ke direktori unduhan FreeRTOS Anda *`freertos`* (), dan gunakan perintah berikut untuk menghasilkan file build:

   ```
   cmake -DVENDOR=infineon -DBOARD=xmc4800_iotkit -DCOMPILER=arm-gcc -S . -B build-folder -G "MinGW Makefiles" -DAFR_ENABLE_TESTS=0
   ```

5. Ubah direktori ke direktori build (*`build-folder`*), dan gunakan perintah berikut untuk membangun biner:

   ```
   cmake --build . --parallel 8
   ```

   Perintah ini membangun biner keluaran `aws_demos.hex` ke direktori build.

6. Flash dan jalankan gambar dengan JLINK.

   a. Dari direktori build (*`build-folder`*), gunakan perintah berikut untuk membuat skrip flash:

```
echo loadfile aws_demos.hex > flash.jlink
```

```
echo r >> flash.jlink
```

```
echo g >> flash.jlink
```

```
echo q >> flash.jlink
```

b. Flash gambar menggunakan executable JLNIK.

```
JLINK_PATH\JLink.exe  -device XMC4800-2048 -if SWD -speed auto -CommanderScript
 flash.jlink
```

Log aplikasi harus terlihat melalui koneksi serial yang Anda buat dengan papan.

Pemecahan Masalah

Jika Anda belum melakukannya, pastikan untuk mengonfigurasi AWS IoT dan mengunduh FreeRTOS Anda untuk menghubungkan perangkat Anda ke Cloud. AWS Lihat Langkah pertama untuk instruksi.

Untuk informasi pemecahan masalah umum tentang Memulai FreeRTOS, lihat. Pemecahan masalah memulai

Memulai dengan Infineon OPTIGA Trust X dan 00 XMC48 IoT Connectivity Kit

> **Important**
>
> Integrasi referensi ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-Freertos yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Tutorial ini memberikan instruksi untuk memulai dengan Infineon OPTIGA Trust X Secure Element dan 00 XMC48 IoT Connectivity Kit. Dibandingkan dengan Memulai dengan Infineon XMC48 00 IoT

Connectivity Kit tutorial, panduan ini menunjukkan kepada Anda cara memberikan kredenal aman menggunakan Elemen Aman Infineon OPTIGA Trust X.

Anda memerlukan perangkat keras berikut:

1. Host MCU - Infineon XMC48 00 IoT Connectivity Kit, kunjungi Katalog Perangkat AWS Mitra untuk membeli satu dari mitra kami.
2. Paket Ekstensi Keamanan:
   - Elemen Aman - Infineon OPTIGA Trust X.

     Kunjungi Katalog Perangkat AWS Mitra untuk membelinya dari mitra kami.
   - Papan Personalisasi - Papan Personalisasi Infineon OPTIGA.
   - Papan Adaptor - Adaptor Infineon MyIo T.

Untuk mengikuti langkah-langkah di sini, Anda harus membuka koneksi serial dengan papan untuk melihat informasi logging dan debugging. (Salah satu langkah mengharuskan Anda untuk menyalin kunci publik dari output debugging serial dari papan dan menempelkannya ke file.) Untuk melakukan ini, Anda memerlukan konverter USB/serial 3.3V selain Kit Konektivitas XMC48 IoT 00. Konverter USB/serial JBtek EL-PN-47310126 diketahui berfungsi untuk demo ini. Anda juga memerlukan tiga kabel male-to-male jumper (untuk menerima (RX), transmisi (TX), dan ground (GND)) untuk menghubungkan kabel serial ke papan MyIo Infineon T Adapter.

Sebelum memulai, Anda harus mengonfigurasi AWS IoT dan mengunduh FreeRTOS Anda untuk menghubungkan perangkat Anda ke Cloud. AWS Untuk petunjuk, silakan lihat Opsi #2: pembuatan kunci pribadi onboard. Dalam tutorial ini, jalur ke direktori unduhan FreeRTOS disebut sebagai. *`freertos`*

Gambaran Umum

Tutorial ini berisi langkah-langkah berikut:

1. Instal perangkat lunak pada mesin host untuk mengembangkan dan men-debug aplikasi tertanam untuk papan mikrokontroler Anda.
2. Kompilasi silang aplikasi demo FreerTOS ke gambar biner.
3. Muat gambar biner aplikasi ke papan Anda, lalu jalankan aplikasi.
4. Untuk tujuan pemantauan dan debugging, berinteraksi dengan aplikasi yang berjalan di papan Anda di seluruh koneksi serial.

Siapkan lingkungan pengembangan Anda

FreeRTOS menggunakan lingkungan pengembangan DAVE Infineon untuk memprogram 00. XMC48 Sebelum Anda mulai, unduh dan instal DAVE dan beberapa driver J-Link untuk berkomunikasi dengan debugger on-board.

Instal DAVE

1. Buka halaman unduhan perangkat lunak DAVE Infineon.
2. Pilih paket DAVE untuk sistem operasi Anda dan kirimkan informasi pendaftaran Anda. Setelah Anda mendaftar, Anda akan menerima email konfirmasi dengan tautan untuk mengunduh file.zip.
3. Unduh paket DAVE. File zip (`DAVE_`*`version_os_date`*`.zip`), dan unzip ke lokasi di mana Anda ingin menginstal DAVE (misalnya,). `C:\DAVE4`

   > **Note**
   >
   > Beberapa pengguna Windows telah melaporkan masalah menggunakan Windows Explorer untuk membuka zip file. Kami menyarankan Anda menggunakan program pihak ketiga seperti 7-Zip.

4. Untuk meluncurkan DAVE, jalankan file yang dapat dieksekusi yang ditemukan di folder yang tidak di-zip. `DAVE_`*`version_os_date`*`.zip`

Untuk informasi selengkapnya, lihat Panduan Mulai Cepat DAVE.

Instal driver Segger J-Link

Untuk berkomunikasi dengan probe debugging on-board kit Konektivitas IoT XMC48 00, Anda memerlukan driver yang disertakan dalam paket Perangkat Lunak dan Dokumentasi J-Link. Anda dapat mengunduh paket Perangkat Lunak dan Dokumentasi J-Link dari halaman unduhan perangkat lunak J-Link Segger.

Buat koneksi serial

Hubungkan kabel konverter USB/Serial ke Adaptor Infineon Shield2Go. Ini memungkinkan papan Anda untuk mengirim informasi logging dan debugging dalam bentuk yang dapat Anda lihat di mesin pengembangan Anda. Untuk mengatur koneksi serial:

1. Hubungkan pin RX ke pin TX USB/konverter serial Anda.

2. Hubungkan pin TX ke pin RX USB/konverter serial Anda.
3. Hubungkan pin ground konverter serial Anda ke salah satu pin GND di papan Anda. Perangkat harus berbagi kesamaan.

Daya disuplai dari port debugging USB, jadi jangan sambungkan pin tegangan positif adaptor serial Anda ke papan.

> **Note**
>
> Beberapa kabel serial menggunakan level pensinyalan 5V. Papan XMC48 00 dan modul Klik Wi-Fi memerlukan 3.3V. Jangan gunakan jumper IOREF papan untuk mengubah sinyal papan menjadi 5V.

Dengan kabel yang terhubung, Anda dapat membuka koneksi serial pada emulator terminal seperti Layar GNU. Baud rate diatur ke 115200 secara default dengan 8 bit data, tidak ada paritas, dan 1 stop bit.

Memantau pesan MQTT di cloud

Sebelum menjalankan proyek demo FreeRTOS, Anda dapat mengatur klien MQTT di konsol untuk memantau pesan AWS IoT yang dikirim perangkat Anda ke Cloud. AWS

Untuk berlangganan topik MQTT dengan klien MQTT AWS IoT

1. Masuk ke konsol AWS IoT tersebut.
2. Di panel navigasi, pilih Uji, lalu pilih klien pengujian MQTT untuk membuka klien MQTT.
3. Dalam Subscription topic, masukkan***your-thing-name*****/example/topic**, lalu pilih Subscribe to topic.

Ketika proyek demo berhasil berjalan di perangkat Anda, Anda melihat “Hello World!” dikirim beberapa kali ke topik yang Anda berlangganan.

Bangun dan jalankan proyek demo FreeRTOS

Impor demo FreeRTOS ke DAVE

1. Mulai DAVE.

2. Di DAVE, pilih File, lalu pilih Impor. Perluas folder Infineon, pilih DAVE Project, lalu pilih Berikutnya.
3. Di jendela Impor Proyek DAVE, pilih Pilih Direktori Root, pilih Browse, dan kemudian pilih proyek demo XMC48 00.

   Di direktori tempat Anda membuka ritsleting unduhan FreerTOS Anda, proyek demo berada di. `projects/infineon/xmc4800_plus_optiga_trust_x/dave4/aws_demos/dave4`

   Pastikan bahwa Copy Projects Into Workspace dihapus.
4. Pilih Selesai.

   `aws_demos`Proyek harus diimpor ke ruang kerja Anda dan diaktifkan.
5. Dari menu Project, pilih Build Active Project.

   Pastikan proyek dibangun tanpa kesalahan.

Jalankan proyek demo FreeRTOS

1. Dari menu Project, pilih Rebuild Active Project untuk membangun kembali `aws_demos` dan mengonfirmasi bahwa perubahan konfigurasi Anda diambil.
2. Dari Project Explorer, klik kanan`aws_demos`, pilih Debug As, dan kemudian pilih DAVE C/C++ Application.
3. Klik dua kali GDB SEGGER J-Link Debugging untuk membuat konfirmasi debug. Pilih Debug.
4. Ketika debugger berhenti di breakpoint di**`main()`**, dari menu Run, pilih Lanjutkan.

Pada titik ini, lanjutkan dengan langkah ekstraksi kunci publikOpsi #2: pembuatan kunci pribadi onboard. Setelah semua langkah selesai, buka AWS IoT konsol. Klien MQTT yang Anda atur sebelumnya harus menampilkan pesan MQTT yang dikirim oleh perangkat Anda. Melalui koneksi serial perangkat, Anda akan melihat sesuatu seperti ini pada 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----
```

Membangun demo FreeRTOS dengan CMake

Bagian ini mencakup penggunaan CMake pada Windows dengan MinGW sebagai sistem build asli. Untuk informasi selengkapnya tentang penggunaan CMake dengan sistem operasi dan opsi lain, lihatMenggunakan CMake dengan FreeRTOS. (MinGW adalah lingkungan pengembangan minimalis untuk aplikasi Microsoft Windows asli.)

Jika Anda memilih untuk tidak menggunakan IDE untuk pengembangan FreeRTOS, Anda dapat CMake menggunakan untuk membangun dan menjalankan aplikasi demo atau aplikasi yang telah Anda kembangkan menggunakan editor kode pihak ketiga dan alat debugging.

Untuk membangun demo FreeRTOS dengan CMake

1. Siapkan GNU Arm Embedded Toolchain.

   a. Unduh toolchain versi Windows dari halaman unduhan Arm Embedded Toolchain.

   > **Note**
   >
   > Karena bug yang dilaporkan dalam utilitas objcopy, kami sarankan Anda mengunduh versi selain “8-2018-q4-major.”

   b. Buka installer toolchain yang diunduh, dan ikuti instruksi di wizard.

   c. Pada halaman akhir wizard instalasi, pilih Tambahkan jalur ke variabel lingkungan untuk menambahkan jalur toolchain ke variabel lingkungan jalur sistem.

2. Instal CMake dan MingW.

   Untuk instruksi, lihat CMake Prasyarat.

3. Buat folder untuk berisi file build yang dihasilkan (*build-folder*).

4. Ubah direktori ke direktori unduhan FreeRTOS Anda *freertos* (), dan gunakan perintah berikut untuk menghasilkan file 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. Ubah direktori ke direktori build (*build-folder*), dan gunakan perintah berikut untuk membangun biner:

```
cmake --build . --parallel 8
```

   Perintah ini membangun biner keluaran `aws_demos.hex` ke direktori build.

6. Flash dan jalankan gambar dengan JLINK.

   a. Dari direktori build (*build-folder*), gunakan perintah berikut untuk membuat skrip flash:

```
echo loadfile aws_demos.hex > flash.jlink
echo r >> flash.jlink
echo g >> flash.jlink
echo q >> flash.jlink
```

b. Flash gambar menggunakan executable JLNIK.

```
JLINK_PATH\JLink.exe  -device XMC4800-2048 -if SWD -speed auto -CommanderScript
 flash.jlink
```

Log aplikasi harus terlihat melalui koneksi serial yang Anda buat dengan papan. Lanjutkan ke langkah ekstraksi kunci publikOpsi #2: pembuatan kunci pribadi onboard. Setelah semua langkah selesai, buka AWS IoT konsol. Klien MQTT yang Anda atur sebelumnya harus menampilkan pesan MQTT yang dikirim oleh perangkat Anda.

Pemecahan Masalah

Untuk informasi pemecahan masalah umum, lihat. Pemecahan masalah memulai

Memulai dengan MW32x AWS IoT Starter Kit

> **Important**
>
> Integrasi referensi ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-FreerTOS yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

AWS IoT Starter Kit adalah kit pengembangan berdasarkan 88 MW32 0/88MW322, Mikrokontroler Cortex M4 terintegrasi terbaru dari NXP, yang mengintegrasikan Wi-Fi 802.11b/g/n pada satu chip mikrokontroler. Kit pengembangan bersertifikat FCC. Untuk informasi selengkapnya, lihat Katalog Perangkat AWS Mitra untuk membeli satu dari mitra kami. MW322 Modul 88 MW32 0/88 juga bersertifikat FCC dan tersedia untuk kustomisasi dan penjualan volume.

Panduan memulai ini menunjukkan kepada Anda cara mengkompilasi silang aplikasi Anda bersama dengan SDK di komputer host, lalu memuat file biner yang dihasilkan ke papan menggunakan alat yang disediakan dengan SDK. Ketika aplikasi mulai berjalan di papan tulis, Anda dapat men-debug atau berinteraksi dengannya dari konsol Serial di komputer host Anda.

Ubuntu 16.04 adalah platform host yang didukung untuk pengembangan dan debugging. Anda mungkin dapat menggunakan platform lain, tetapi ini tidak didukung secara resmi. Anda harus memiliki izin untuk menginstal perangkat lunak pada platform host. Alat eksternal berikut yang diperlukan untuk membangun SDK:

- Platform host Ubuntu 16.04
- Rantai alat ARM versi 4_9_2015q3
- Eclipse 4.9.0 IDE

Rantai alat ARM diperlukan untuk mengkompilasi silang aplikasi Anda dan SDK. SDK memanfaatkan versi terbaru dari rantai alat untuk mengoptimalkan jejak gambar dan menyesuaikan lebih banyak fungsionalitas ke dalam ruang yang lebih sedikit. Panduan ini mengasumsikan bahwa Anda menggunakan toolchain versi 4_9_2015q3. Menggunakan versi toolchain yang lebih lama tidak disarankan. Kit pengembangan sudah di-flash dengan firmware proyek Demo Mikrokontroler Nirkabel.

Topik

- Menyiapkan perangkat keras Anda
- Menyiapkan lingkungan pengembangan
- Bangun dan jalankan proyek demo FreeRTOS
- Debugging
- Pemecahan Masalah

Menyiapkan perangkat keras Anda

Hubungkan MW32x papan ke laptop Anda dengan menggunakan kabel mini-USB ke USB. Hubungkan kabel mini-USB ke satu-satunya konektor mini-USB yang ada di papan tulis. Anda tidak perlu jumper change.

Jika papan terhubung ke laptop atau komputer desktop, Anda tidak memerlukan catu daya eksternal.

Koneksi USB ini menyediakan yang berikut:

- Akses konsol ke papan tulis. Port tty/com virtual terdaftar dengan host pengembangan yang dapat digunakan untuk mengakses konsol.
- Akses JTAG ke papan. Ini dapat digunakan untuk memuat atau membongkar gambar firmware ke dalam RAM atau flash papan, atau untuk tujuan debugging.

Menyiapkan lingkungan pengembangan

Untuk tujuan pengembangan, persyaratan minimum adalah rantai alat ARM dan alat yang dibundel dengan SDK. Bagian berikut memberikan detail tentang pengaturan toolchain ARM.

Rantai Alat GNU

SDK secara resmi mendukung toolchain GCC Compiler. Toolchain cross-compiler untuk GNU ARM tersedia di GNU Arm Embedded Toolchain 4.9-2015-q3-update.

Sistem build dikonfigurasi untuk menggunakan toolchain GNU secara default. Makefiles berasumsi bahwa binari toolchain kompiler GNU tersedia di PATH pengguna dan dapat dipanggil dari Makefiles. Makefiles juga berasumsi bahwa nama file dari binari toolchain GNU diawali dengan. `arm-none-eabi-`

Toolchain GCC dapat digunakan dengan GDB untuk men-debug dengan OpenOCD (dibundel dengan SDK). Ini menyediakan perangkat lunak yang berinteraksi dengan JTAG.

Kami merekomendasikan versi 4_9_2015q3 dari toolchain. gcc-arm-embedded

Prosedur Pengaturan Rantai Alat Linux

Ikuti langkah-langkah ini untuk mengatur toolchain GCC di Linux.

1. Unduh tarball toolchain yang tersedia di GNU Arm Embedded Toolchain 4.9-2015-q3-update. File tersebut adalah`gcc-arm-none-eabi-4_9-2015q3-20150921-linux.tar.bz2`.
2. Salin file ke direktori pilihan Anda. Pastikan tidak ada spasi dalam nama direktori.
3. Gunakan perintah berikut untuk menghapus file.

   ```
   tar -vxf filename
   ```

4. Tambahkan jalur rantai alat yang diinstal ke PATH sistem. Misalnya, tambahkan baris berikut di akhir `.profile` file yang terletak di `/home/user-name` direktori.

   ```
   PATH=$PATH:path to gcc-arm-none-eabit-4_9_2015_q3/bin
   ```

> **Note**
>
> Distribusi Ubuntu yang lebih baru mungkin datang dengan versi Debian dari GCC Cross Compiler. Jika demikian, Anda harus menghapus Cross Compiler asli dan ikuti prosedur pengaturan di atas.

Bekerja dengan host pengembangan Linux

Distribusi desktop Linux modern seperti Ubuntu atau Fedora dapat digunakan. Namun, kami menyarankan Anda meningkatkan ke rilis terbaru. Langkah-langkah berikut telah diverifikasi untuk bekerja di Ubuntu 16.04 dan berasumsi bahwa Anda menggunakan versi itu.

Instalasi Paket

SDK menyertakan skrip untuk mengaktifkan pengaturan cepat lingkungan pengembangan Anda pada mesin Linux yang baru disiapkan. Script mencoba mendeteksi secara otomatis jenis mesin dan menginstal perangkat lunak yang sesuai, termasuk pustaka C, pustaka USB, pustaka FTDI, ncurses, python, dan lateks. Di bagian ini, nama direktori generik *amzsdk_bundle-x.y.z* menunjukkan direktori root AWS SDK. Nama direktori sebenarnya mungkin berbeda. Anda harus memiliki hak akses root.

- Arahkan ke *amzsdk_bundle-x.y.z/* direktori dan jalankan perintah ini.

```
./lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/bin/installpkgs.sh
```

Menghindari sudo

Dalam panduan ini, `flashprog` operasi menggunakan `flashprog.py` skrip untuk mem-flash NAND papan, seperti yang dijelaskan di bawah ini. Demikian pula, `ramload` operasi menggunakan `ramload.py` skrip untuk menyalin gambar firmware dari host langsung ke RAM mikrokontroler, tanpa mem-flash NAND.

Anda dapat mengkonfigurasi host pengembangan Linux Anda untuk melakukan `flashprog` dan `ramload` operasi tanpa memerlukan `sudo` perintah setiap kali. Untuk melakukan ini, jalankan perintah berikut.

```
./lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/bin/perm_fix.sh
```

> **Note**
>
> Anda harus mengonfigurasi izin host pengembangan Linux Anda dengan cara ini untuk memastikan pengalaman Eclipse IDE yang lancar.

Menyiapkan Konsol Serial

Masukkan kabel USB ke slot USB host Linux. Ini memicu deteksi perangkat. Anda akan melihat pesan seperti berikut dalam `/var/log/messages` file, atau setelah menjalankan `dmesg` perintah.

```
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
```

Verifikasi bahwa dua perangkat ttyUSB telah dibuat. TtyUSB kedua adalah konsol serial. Pada contoh di atas, ini diberi nama “ttyUSB1”.

Dalam panduan ini, kami menggunakan minicom untuk melihat output konsol serial. Anda juga dapat menggunakan program serial lain seperti`putty`. Jalankan perintah berikut untuk menjalankan minicom dalam mode pengaturan.

```
minicom -s
```

Di minicom, navigasikan ke Serial Port Setup dan tangkap pengaturan berikut.

```
| 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
```

Anda dapat menyimpan pengaturan ini di minicom untuk penggunaan masa depan. Jendela minicom sekarang menampilkan pesan dari konsol serial.

Pilih jendela konsol serial dan tekan tombol Enter. Ini menampilkan hash (#) di layar.

> **Note**
>
> Papan pengembangan termasuk perangkat silikon FTDI. Perangkat FTDI memperlihatkan dua antarmuka USB untuk host. Antarmuka pertama dikaitkan dengan fungsionalitas JTAG dari MCU dan antarmuka kedua dikaitkan dengan UARTx port fisik MCU.

Menginstal OpenOCD

OpenOCD adalah perangkat lunak yang menyediakan debugging, pemrograman dalam sistem, dan pengujian pemindaian batas untuk perangkat target tertanam.

OpenOCD versi 0.9 diperlukan. Ini juga diperlukan untuk fungsionalitas Eclipse. Jika versi sebelumnya, seperti versi 0.7, diinstal pada host Linux Anda, hapus repositori itu dengan perintah yang sesuai untuk distribusi Linux yang saat ini Anda gunakan.

Jalankan perintah Linux standar untuk menginstal OpenOCD,

```
apt-get install openocd
```

Jika perintah di atas tidak menginstal versi 0.9 atau yang lebih baru, gunakan prosedur berikut untuk mengunduh dan mengkompilasi kode sumber openocd.

Untuk menginstal OpenOCD

1. Jalankan perintah berikut untuk menginstal libusb-1.0.

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

2. Unduh kode sumber openocd 0.9.0 dari http://openocd.org/.
3. Ekstrak openocd dan arahkan ke direktori tempat Anda mengekstraknya.

4. Konfigurasikan openocd dengan perintah berikut.

```
./configure --enable-ftdi --enable-jlink
```

5. Jalankan utilitas make untuk mengkompilasi opencd.

```
make install
```

Menyiapkan Eclipse

> **Note**
>
> Bagian ini mengasumsikan bahwa Anda telah menyelesaikan langkah-langkah di Menghindari sudo

Eclipse adalah IDE pilihan untuk pengembangan aplikasi dan debugging. Ini menyediakan IDE yang kaya dan ramah pengguna dengan dukungan debugging terintegrasi, termasuk debugging sadar utas. Bagian ini menjelaskan pengaturan Eclipse umum untuk semua host pengembangan yang didukung.

Untuk mengatur Eclipse

1. Unduh dan instal Java Run Time Environment (JRE).

   Eclipse mengharuskan Anda menginstal JRE. Kami menyarankan Anda menginstal ini terlebih dahulu, meskipun dapat diinstal setelah Anda menginstal Eclipse. Versi JRE (32/64 bit) harus cocok dengan versi Eclipse (32/64 bit). Anda dapat mengunduh JRE dari Java SE Runtime Environment 8 Downloads di situs web Oracle.
2. Unduh dan instal “Eclipse IDE for C/C++ Developers” dari http://www.eclipse.org. Eclipse versi 4.9.0 atau yang lebih baru didukung. Instalasi hanya mengharuskan Anda untuk mengekstrak arsip yang diunduh. Anda menjalankan platform khusus Eclipse executable untuk memulai aplikasi.

Bangun dan jalankan proyek demo FreeRTOS

Ada dua cara untuk menjalankan proyek demo FreeRTOS:

- Gunakan baris perintah.
- Gunakan Eclipse IDE.

Topik ini mencakup kedua opsi.

Penyediaan

- Bergantung pada apakah Anda menggunakan aplikasi pengujian atau demo, atur data penyediaan di salah satu file berikut:
  - `./tests/common/include/aws_clientcredential.h`
  - `./demos/common/include/aws_clientcredential.h`

  Sebagai contoh:

```
#define clientcredentialWIFI_SSID "Wi-Fi SSID"
#define clientcredentialWIFI_PASSWORD "Wi-Fi password"
#define clientcredentialWIFI_SECURITY "Wi-Fi security"
```

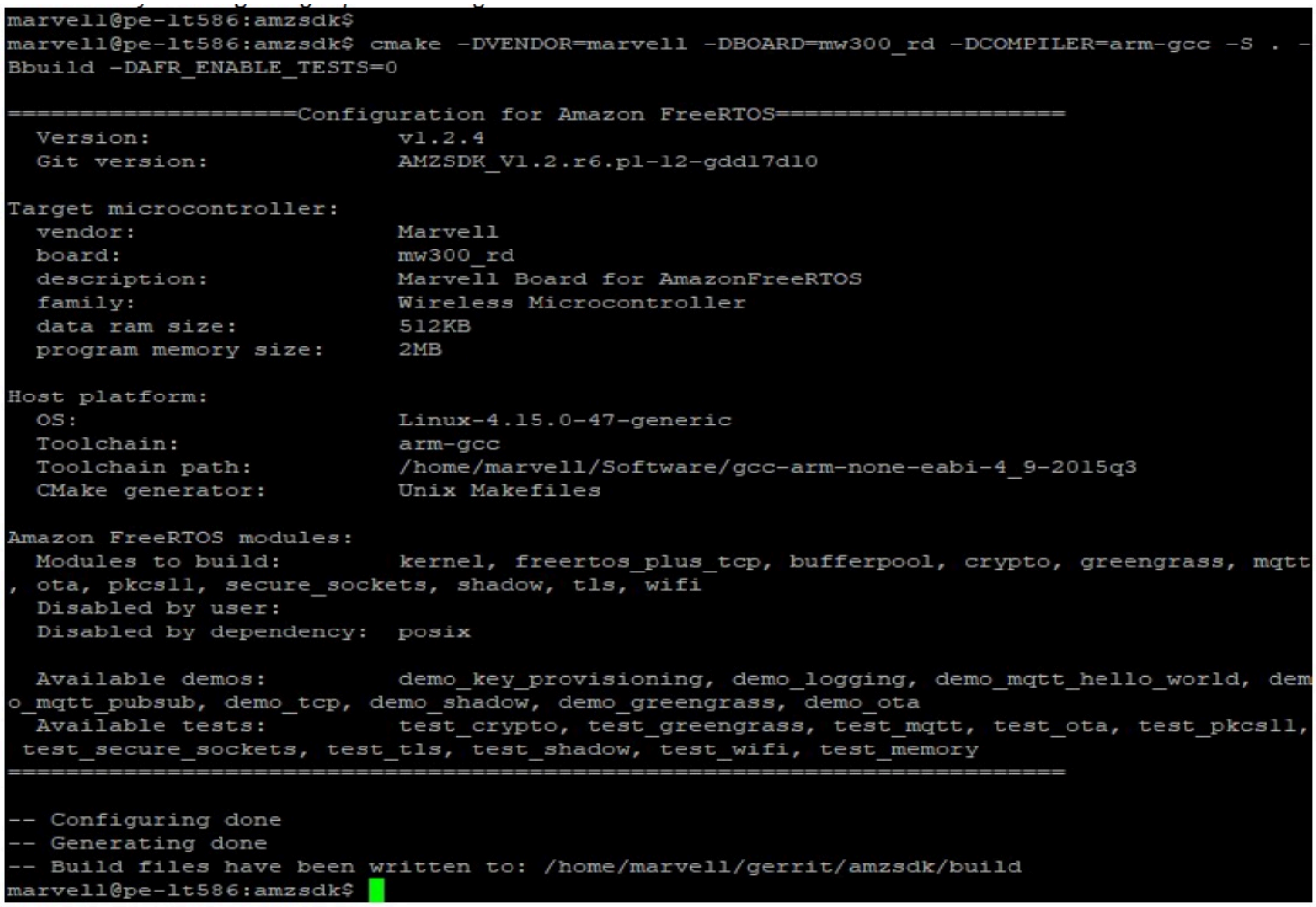
  > **Note**
  >
  > Anda dapat memasukkan nilai keamanan Wi-Fi berikut:
  >
  > - `eWiFiSecurityOpen`
  > - `eWiFiSecurityWEP`
  > - `eWiFiSecurityWPA`
  > - `eWiFiSecurityWPA2`
  >
  > SSID dan kata sandi harus dilampirkan dalam tanda kutip ganda.

Bangun dan jalankan demo FreeRTOS menggunakan baris perintah

1. Gunakan perintah berikut untuk mulai membangun aplikasi demo.

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

Pastikan Anda mendapatkan output yang sama seperti yang ditunjukkan pada contoh berikut.



2. Arahkan ke direktori build.

```
cd build
```

3. Jalankan utilitas make untuk membangun aplikasi.

```
make all -j4
```

Pastikan Anda mendapatkan output yang sama seperti yang ditunjukkan pada gambar berikut:

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

4. Gunakan perintah berikut untuk membangun aplikasi pengujian.

```
cmake -DVENDOR=marvell -DBOARD=mw320 -DCOMPILER=arm-gcc -S .  -Bbuild -
DAFR_ENABLE_TESTS=1
cd build
make all -j4
```

   Jalankan `cmake` perintah setiap kali Anda beralih antara `aws_demos project` dan`aws_tests project`.

5. Tulis gambar firmware ke flash papan pengembangan. Firmware akan dijalankan setelah papan pengembangan diatur ulang. Anda harus membangun SDK sebelum Anda mem-flash gambar ke mikrokontroler.

   a. Sebelum Anda mem-flash gambar firmware, siapkan flash papan pengembangan dengan komponen umum Layout dan Boot2. Gunakan perintah berikut.

```
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
```

      `flashprog`Perintah memulai yang berikut:

- Layout - Utilitas flashprog pertama kali diinstruksikan untuk menulis tata letak ke flash. Tata letaknya mirip dengan informasi partisi untuk flash. Tata letak default terletak di`/lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/OpenOCD/mw300/layout.txt`.
- Boot2 — Ini adalah boot-loader yang digunakan oleh WMSDK. `flashprog`Perintah juga menulis bootloader ke flash. Ini adalah tugas bootloader untuk memuat gambar firmware mikrokontroler setelah di-flash. Pastikan Anda mendapatkan output yang sama seperti yang ditunjukkan pada gambar di bawah ini.

```
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. Firmware menggunakan chipset Wi-Fi untuk fungsinya, dan chipset Wi-Fi memiliki firmware sendiri yang juga harus ada dalam flash. Anda menggunakan `flashprog.py` utilitas untuk mem-flash firmware Wi-Fi dengan cara yang sama seperti yang Anda lakukan untuk mem-flash boot-loader Boot2 dan firmware MCU. Gunakan perintah berikut untuk mem-flash 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
```

Pastikan output dari perintah mirip dengan gambar di bawah ini.

```
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. Gunakan perintah berikut untuk mem-flash 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. Setel ulang papan. Anda akan melihat log untuk aplikasi demo.

e. Untuk menjalankan aplikasi pengujian, flash `aws_tests.bin` biner yang terletak di direktori yang sama.

```
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
```

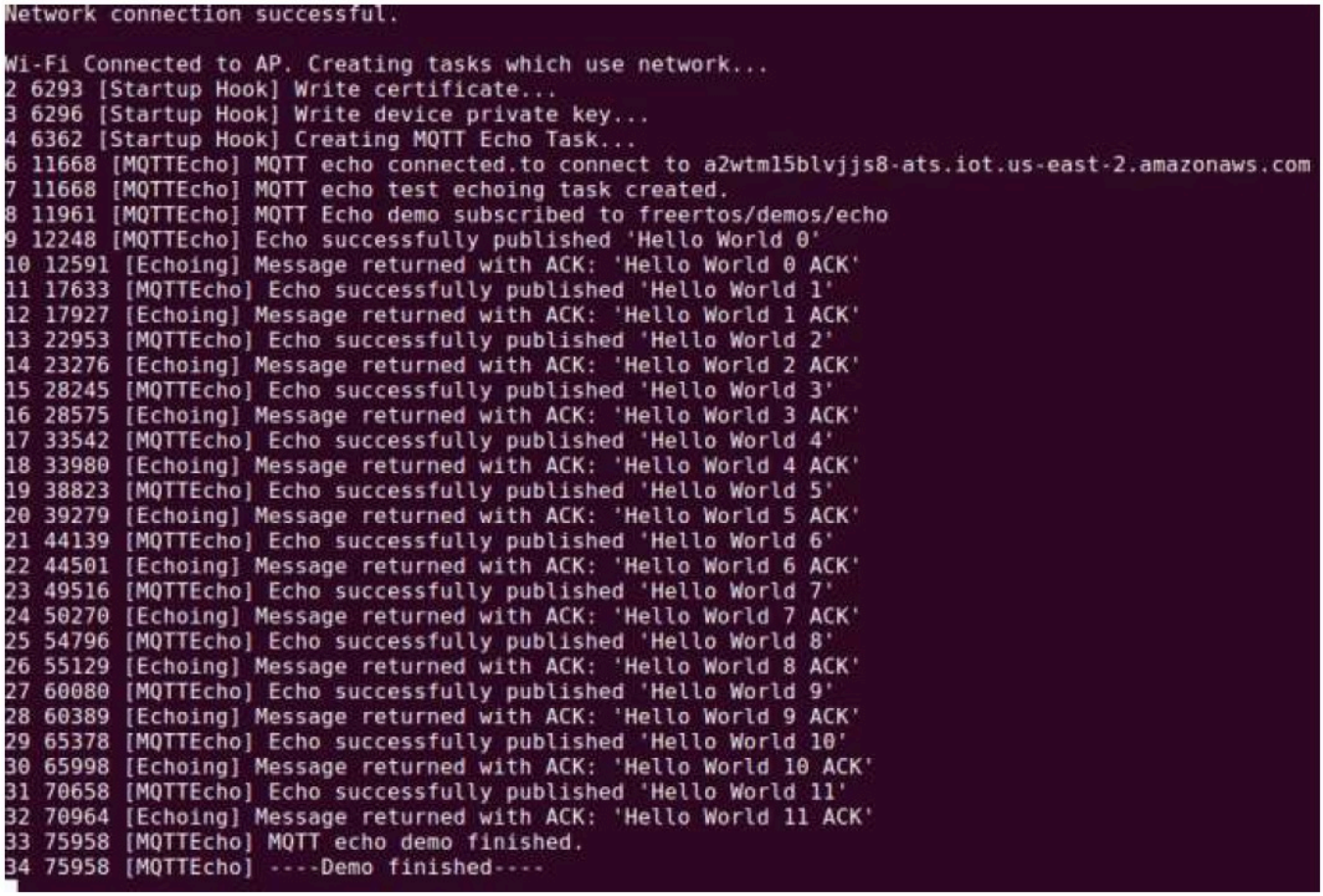
Output perintah Anda harus mirip dengan yang ditunjukkan pada gambar di bawah ini.

```
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. Setelah Anda mem-flash firmware dan mengatur ulang papan, aplikasi demo akan dimulai seperti yang ditunjukkan pada gambar di bawah ini.



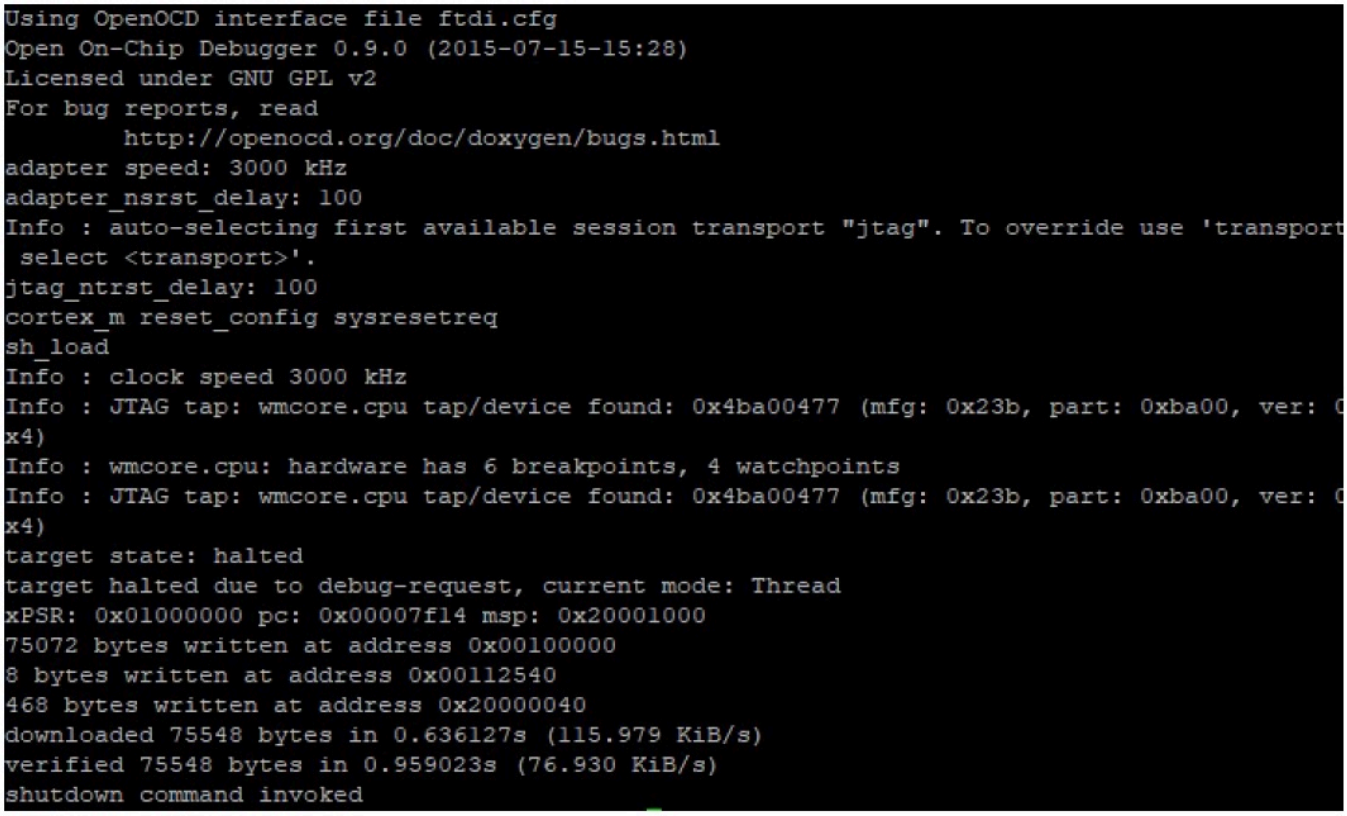
7. (Opsional) Sebagai metode alternatif untuk menguji gambar Anda, gunakan utilitas flashprog untuk menyalin gambar mikrokontroler dari host langsung ke RAM mikrokontroler. Gambar tidak disalin dalam flash, sehingga akan hilang setelah Anda me-reboot mikrokontroler.

   Memuat gambar firmware ke dalam SRAM adalah operasi yang lebih cepat karena segera meluncurkan file eksekusi. Metode ini digunakan terutama untuk pengembangan berulang.

   Gunakan perintah berikut untuk memuat firmware ke dalam 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
   ```

   Output perintah ditunjukkan pada gambar di bawah ini.



Ketika eksekusi perintah selesai, Anda akan melihat log dari aplikasi demo.

Bangun dan jalankan demo FreeRTOS menggunakan Eclipse IDE

1. Sebelum Anda mengatur ruang kerja Eclipse, Anda harus menjalankan perintah. `cmake`

   Jalankan perintah berikut untuk bekerja dengan proyek `aws_demos` Eclipse.

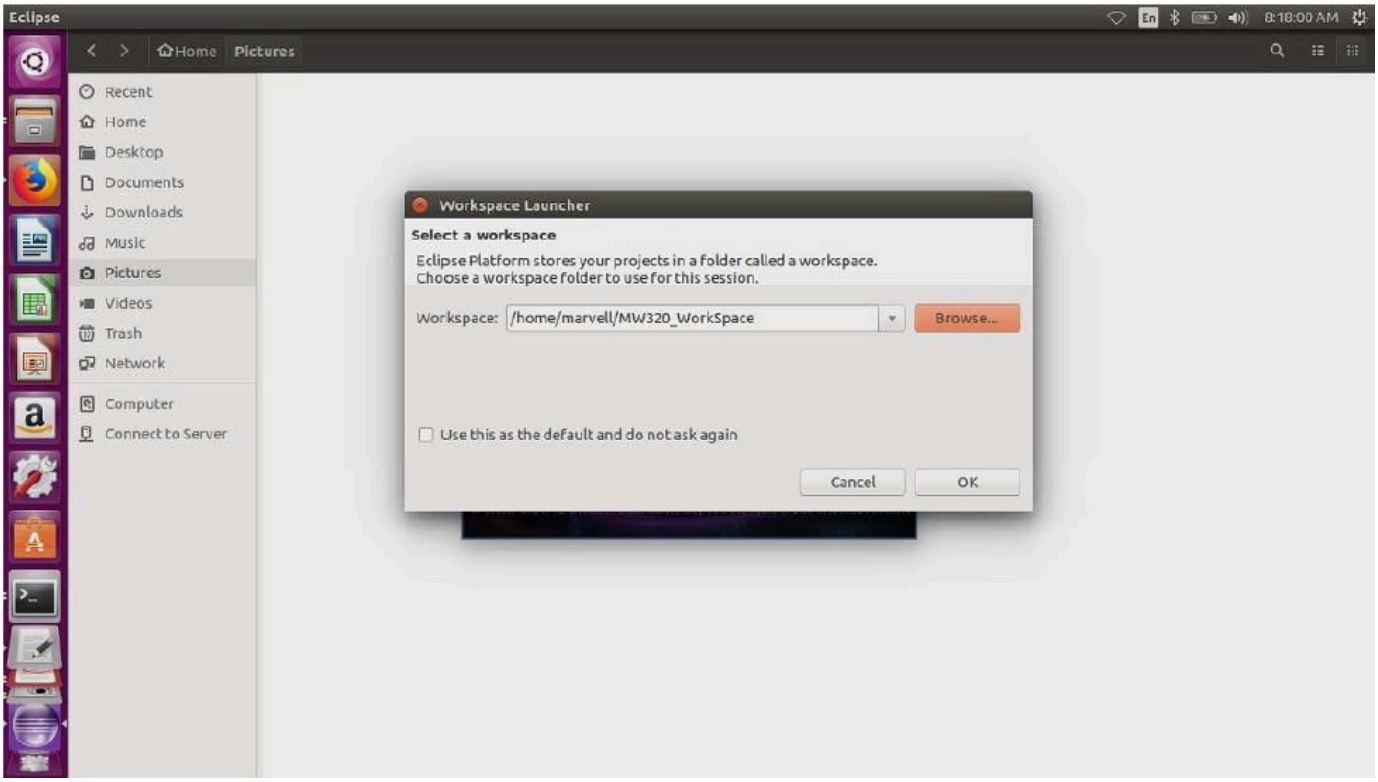
   ```
   cmake -DVENDOR=marvell -DBOARD=mw320 -DCOMPILER=arm-gcc -S .  -Bbuild -
   DAFR_ENABLE_TESTS=0
   ```

   Jalankan perintah berikut untuk bekerja dengan proyek `aws_tests` Eclipse.

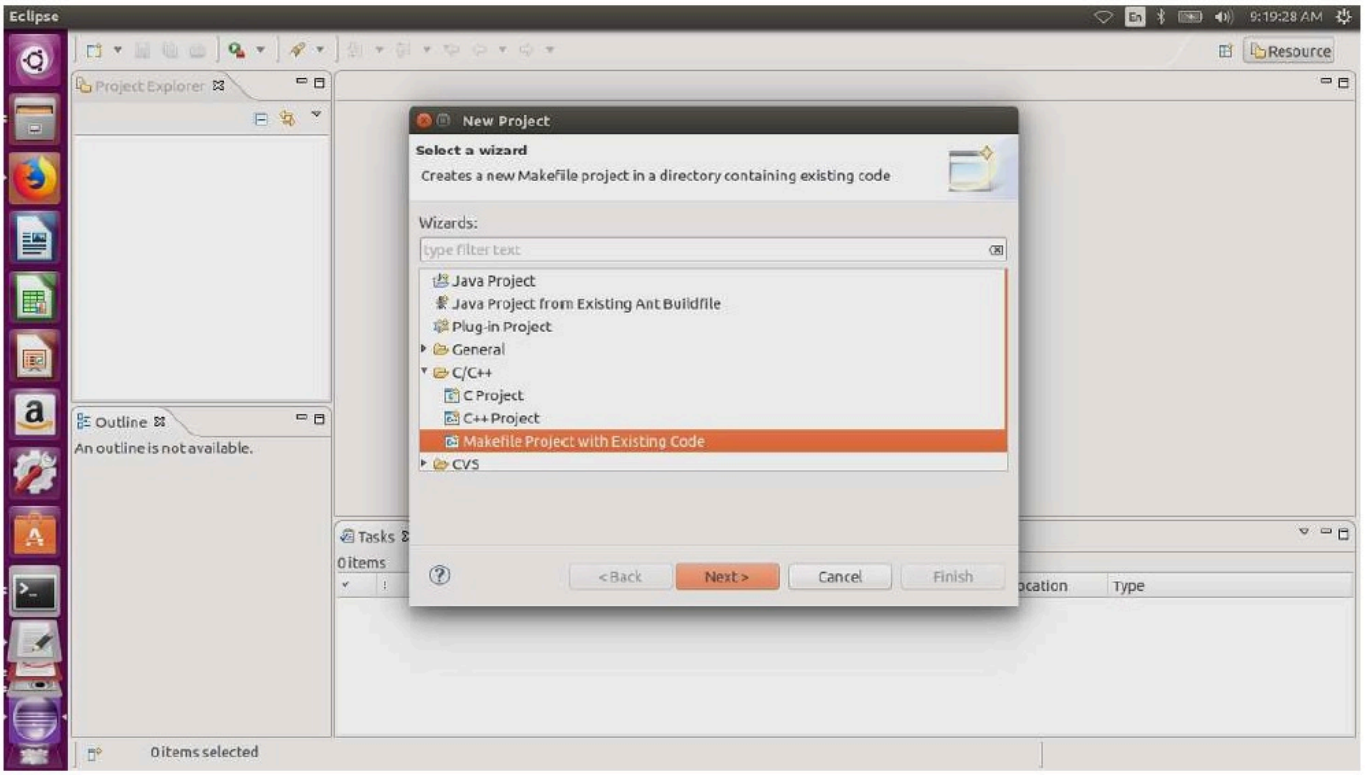
   ```
   cmake -DVENDOR=marvell -DBOARD=mw320 -DCOMPILER=arm-gcc -S .  -Bbuild -
   DAFR_ENABLE_TESTS=1
   ```

> **Tip**
>
> Jalankan `cmake` perintah setiap kali Anda beralih antara `aws_demos` proyek dan `aws_tests` proyek.

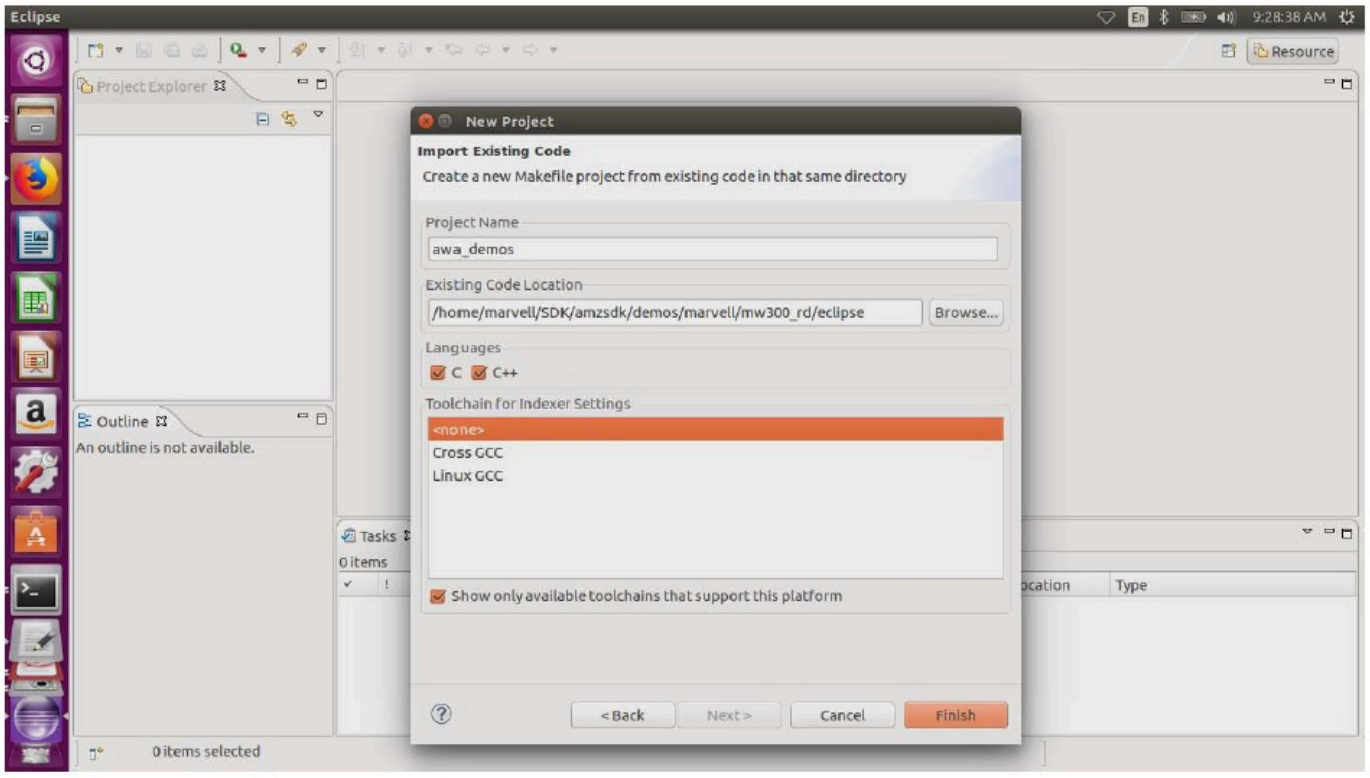
2. Buka Eclipse dan, ketika diminta, pilih ruang kerja Eclipse Anda seperti yang ditunjukkan pada gambar di bawah ini.



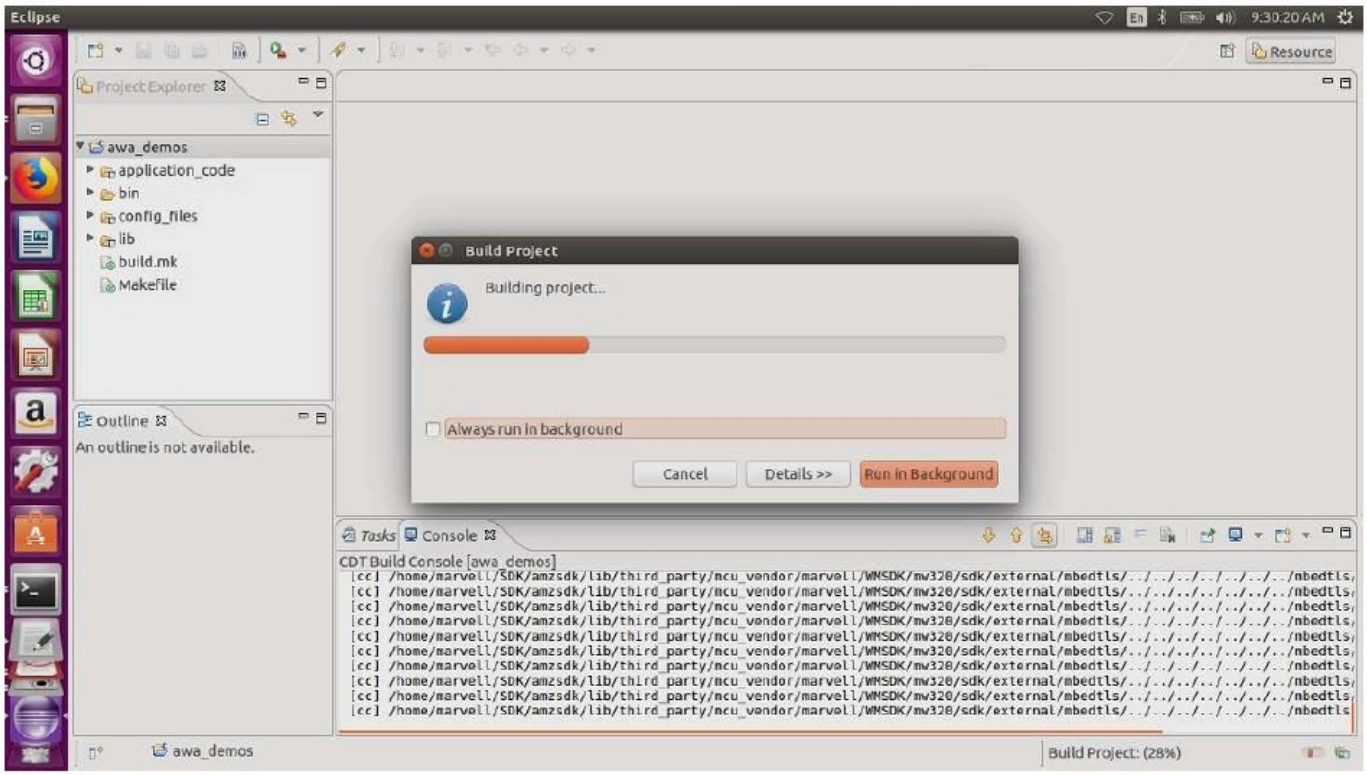
3. Pilih opsi untuk membuat Proyek Makefile: dengan Kode yang Ada seperti yang ditunjukkan pada gambar di bawah ini.



4. Pilih Browse, tentukan direktori kode yang ada, lalu pilih Selesai.



5. Di panel navigasi, pilih aws_demos di project explorer. Klik kanan aws_demo untuk membuka menu, lalu pilih Build.



Jika build berhasil, itu menghasilkan `build/cmake/vendors/marvell/mw300_rd/aws_demos.bin` file.

6. Gunakan alat baris perintah untuk mem-flash file Layout (`layout.txt`), biner Boot2 (`boot2.bin`), biner firmware MCU (`aws_demos.bin`), dan firmware Wi-Fi.

   a. Sebelum Anda mem-flash gambar firmware, siapkan flash papan pengembangan dengan komponen umum, Layout dan Boot2. Gunakan perintah berikut.

   ```
   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
   ```

   `flashprog`Perintah memulai yang berikut:

   - Layout - Utilitas flashprog pertama kali diinstruksikan untuk menulis tata letak ke flash. Tata letaknya mirip dengan informasi partisi untuk flash. Tata letak default terletak di/`lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/OpenOCD/mw300/layout.txt`.

- Boot2 — Ini adalah boot-loader yang digunakan oleh WMSDK. Perintah flashprog juga menulis bootloader ke flash. Ini adalah tugas bootloader untuk memuat gambar firmware mikrokontroler setelah di-flash. Pastikan Anda mendapatkan output yang sama seperti yang ditunjukkan pada gambar di bawah ini.

```
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

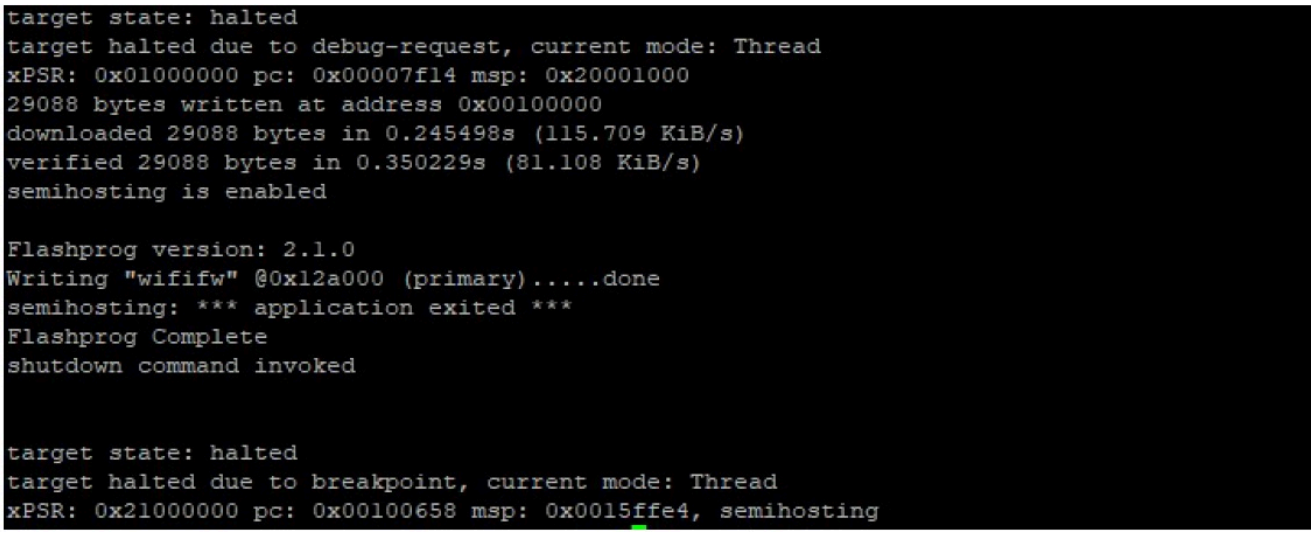
Flashprog version: 2.1.0
Erasing primary flash...done
Writing new flash layout...done
Writing "boot2" @0x0 (primary)...done
semihosting: *** application exited ***
Flashprog Complete
shutdown command invoked


target state: halted
target halted due to breakpoint, current mode: Thread
xPSR: 0x21000000 pc: 0x00100658 msp: 0x0015ffe4, semihosting
```

b. Firmware menggunakan chipset Wi-Fi untuk fungsinya, dan chipset Wi-Fi memiliki firmware sendiri yang juga harus ada dalam flash. Anda menggunakan `flashprog.py` utilitas untuk mem-flash firmware Wi-Fi dengan cara yang sama seperti yang Anda lakukan untuk mem-flash boot-loader boot2 dan firmware MCU. Gunakan perintah berikut untuk mem-flash 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
```

Pastikan output dari perintah mirip dengan gambar di bawah ini.



c. Gunakan perintah berikut untuk mem-flash firmware MCU.

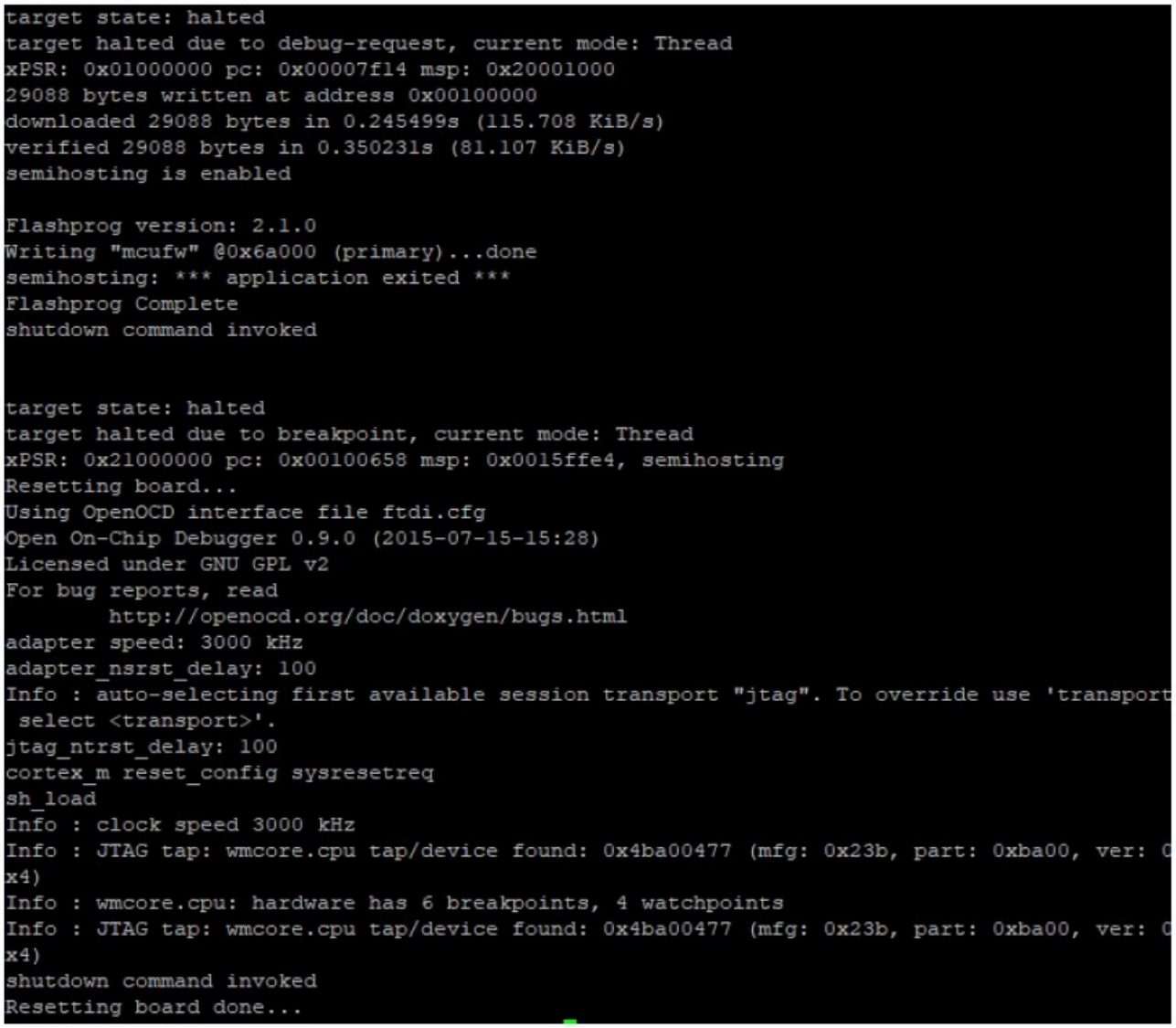
```
cd amzsdk_bundle-x.y.z
./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py --mcufw build/
cmake/vendors/marvell/mw300_rd/aws_demos.bin -r
```

d. Setel ulang papan. Anda akan melihat log untuk aplikasi demo.

e. Untuk menjalankan aplikasi pengujian, flash `aws_tests.bin` biner yang terletak di direktori yang sama.

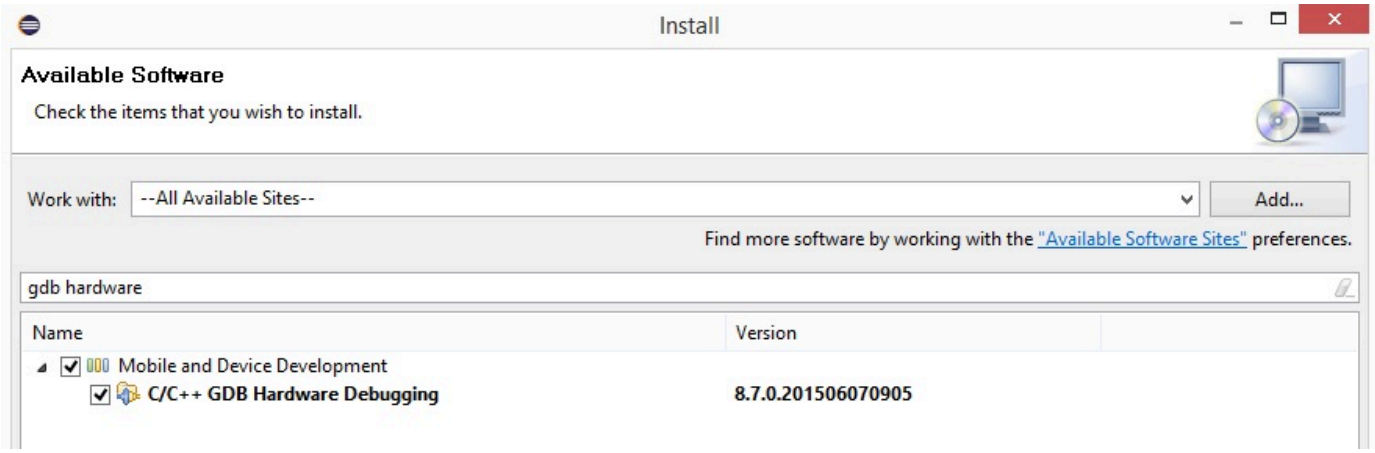
```
cd amzsdk_bundle-x.y.z
./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py --mcufw build/
cmake/vendors/marvell/mw300_rd/aws_tests.bin -r
```

Output perintah Anda harus mirip dengan yang ditunjukkan pada gambar di bawah ini.



Debugging

- Mulai Eclipse dan pilih Bantuan dan kemudian pilih Instal perangkat lunak baru. Di menu Bekerja dengan, pilih Semua Situs yang Tersedia. Masukkan teks filterGDB `Hardware`. Pilih opsi C/C++ GDB Hardware Debugging dan instal plugin.



Pemecahan Masalah

Masalah jaringan

Periksa kredensyal jaringan Anda. Lihat “Penyediaan” di. Bangun dan jalankan proyek demo FreeRTOS

Mengaktifkan log tambahan

- Aktifkan log khusus papan.

  Aktifkan panggilan ke `wmstdio_init(UART0_ID, 0)` dalam fungsi `prvMiscInitialization` dalam `main.c` file untuk tes atau demo.

- Mengaktifkan log Wi-Fi

  Aktifkan makro `CONFIG_WLCMGR_DEBUG` dalam *`freertos`*`/vendors/marvell/WMSDK/mw320/sdk/src/incl/autoconf.h` file.

Menggunakan GDB

Kami menyarankan Anda menggunakan file `arm-none-eabi-gdb` dan gdb perintah yang dikemas bersama dengan SDK. Buka direktori tersebut.

```
cd freertos/lib/third_party/mcu_vendor/marvell/WMSDK/mw320
```

Jalankan perintah berikut (pada satu baris) untuk terhubung ke GDB.

```
arm-none-eabi-gdb -x ./sdk/tools/OpenOCD/gdbinit ../../../../../../build/cmake/
vendors/marvell/mw300 _rd/aws_demos.axf
```

Memulai dengan kit pengembangan MediaTek MT7697 Hx

> **Important**
>
> Integrasi referensi ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-Freertos yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Tutorial ini memberikan instruksi untuk memulai dengan MediaTek MT7697 Hx Development Kit. Jika Anda tidak memiliki MediaTek MT7697 Hx Development Kit, kunjungi Katalog Perangkat AWS Mitra untuk membelinya dari mitra kami.

Sebelum memulai, Anda harus mengonfigurasi AWS IoT dan mengunduh FreeRTOS Anda untuk menghubungkan perangkat Anda ke Cloud. AWS Lihat Langkah pertama untuk instruksi. Dalam tutorial ini, jalur ke direktori unduhan FreeRTOS disebut sebagai. *`freertos`*

Gambaran Umum

Tutorial ini berisi petunjuk untuk langkah-langkah memulai berikut:

1. Menginstal perangkat lunak pada mesin host untuk mengembangkan dan men-debug aplikasi tertanam untuk papan mikrokontroler Anda.
2. Menyusun silang aplikasi demo FreeRTOS ke gambar biner.
3. Memuat gambar biner aplikasi ke papan Anda, dan kemudian menjalankan aplikasi.
4. Berinteraksi dengan aplikasi yang berjalan di papan Anda di seluruh koneksi serial, untuk tujuan pemantauan dan debugging.

Siapkan lingkungan pengembangan Anda

Sebelum Anda mengatur lingkungan Anda, sambungkan komputer Anda ke port USB pada MediaTek MT7697 Hx Development Kit.

Unduh dan instal Keil MDK

Anda dapat menggunakan Keil Microcontroller Development Kit (MDK) berbasis GUI untuk mengkonfigurasi, membangun, dan menjalankan proyek FreeRTOS di papan Anda. Keil MDK mencakup µVision IDE dan µVision Debugger.

> **Note**
>
> Keil MDK hanya didukung pada mesin Windows 7, Windows 8, dan Windows 10 64-bit.

Untuk mengunduh dan menginstal Keil MDK

1. Buka halaman Keil MDK Getting Started, dan pilih Download MDK-Core.
2. Masukkan dan kirimkan informasi Anda untuk mendaftar dengan Keil.
3. Klik kanan MDK executable dan simpan installer Keil MDK ke komputer Anda.
4. Buka installer Keil MDK dan ikuti langkah-langkah untuk menyelesaikannya. Pastikan Anda menginstal paket MediaTek perangkat (MT76x7 Seri).

Buat koneksi serial

Hubungkan papan ke komputer host Anda dengan kabel USB. Port COM muncul di Windows Device Manager. Untuk debugging, Anda dapat membuka sesi ke port dengan alat utilitas terminal seperti HyperTerminal atau TeraTerm.

Memantau pesan MQTT di cloud

Sebelum menjalankan proyek demo FreeRTOS, Anda dapat mengatur klien MQTT di konsol untuk memantau pesan AWS IoT yang dikirim perangkat Anda ke Cloud. AWS

Untuk berlangganan topik MQTT dengan klien MQTT AWS IoT

1. Masuk ke konsol AWS IoT tersebut.
2. Di panel navigasi, pilih Uji, lalu pilih klien pengujian MQTT untuk membuka klien MQTT.
3. Dalam Subscription topic, masukkan***your-thing-name***`example/topic`, lalu pilih Subscribe to topic.

Ketika proyek demo berhasil berjalan di perangkat Anda, Anda melihat “Hello World!” dikirim beberapa kali ke topik yang Anda berlangganan.

Bangun dan jalankan proyek demo FreeRTOS dengan Keil MDK

Untuk membangun proyek demo FreeRTOS di Keil µVision

1. Dari menu Start, buka Keil µVision 5.
2. Buka file `projects/mediatek/mt7697hx-dev-kit/uvision/aws_demos/aws_demos.uvprojx` proyek.
3. Dari menu, pilih Project, lalu pilih Build target.

   Setelah kode dibangun, Anda melihat file demo yang dapat dieksekusi di. `projects/mediatek/mt7697hx-dev-kit/uvision/aws_demos/out/Objects/aws_demo.axf`

Untuk menjalankan proyek demo FreeRTOS

1. Atur MediaTek MT7697 Hx Development Kit ke mode PROGRAM.

   Untuk mengatur kit ke mode PROGRAM, tekan dan tahan tombol PROG. Dengan tombol PROG masih ditekan, tekan dan lepaskan tombol RESET, lalu lepaskan tombol PROG.
2. Dari menu, pilih Flash, lalu pilih Configure Flash Tools.
3. Di Pilihan untuk Target **`aws_demo`** ", pilih tab Debug. Pilih Gunakan, atur debugger ke CMSIS-DAP Debugger, lalu pilih OK.
4. Dari menu, pilih Flash, lalu pilih Unduh.

   µVision memberi tahu Anda saat unduhan selesai.
5. Gunakan utilitas terminal untuk membuka jendela konsol serial. Atur port serial ke 115200 bps, none-parity, 8-bit, dan 1 stop-bit.
6. Pilih tombol RESET pada MediaTek MT7697 Hx Development Kit Anda.

Pemecahan Masalah

Mendebug proyek FreeRTOS di Keil µVision

Saat ini, Anda harus mengedit MediaTek paket yang disertakan dengan Keil µVision sebelum Anda dapat men-debug proyek demo FreeRTOS dengan Keil µVision. MediaTek

Untuk mengedit MediaTek paket untuk debugging proyek FreerTOS

1. Temukan dan buka `Keil_v5\ARM\PACK\.Web\MediaTek.MTx.pdsc` file di folder instalasi Keil MDK Anda.
2. Ganti semua contoh `flag = Read32(0x20000000);` dengan`flag = Read32(0x0010FBFC);`.
3. Ganti semua contoh `Write32(0x20000000, 0x76877697);` dengan`Write32(0x0010FBFC, 0x76877697);`.

Untuk memulai debugging proyek

1. Dari menu, pilih Flash, lalu pilih Configure Flash Tools.
2. Pilih tab Target, lalu pilih Read/Write Memory Areas. Konfirmasikan IRAM2 itu IRAM1 dan keduanya dipilih.
3. Pilih tab Debug, lalu pilih CMSIS-DAP Debugger.
4. Buka`vendors/mediatek/boards/mt7697hx-dev-kit/aws_demos/application_code/main.c`, dan atur makro `MTK_DEBUGGER` ke`1`.
5. Membangun kembali proyek demo di µVision.
6. Atur MediaTek MT7697 Hx Development Kit ke mode PROGRAM.

   Untuk mengatur kit ke mode PROGRAM, tekan dan tahan tombol PROG. Dengan tombol PROG masih ditekan, tekan dan lepaskan tombol RESET, lalu lepaskan tombol PROG.
7. Dari menu, pilih Flash, lalu pilih Unduh.

   µVision memberi tahu Anda saat unduhan selesai.
8. Tekan tombol RESET pada MediaTek MT7697 Hx Development Kit Anda.
9. Dari menu µVision, pilih Debug, lalu pilih Start/Stop Debug Session. Jendela Call Stack + Locals terbuka saat Anda memulai sesi debug.
10. Dari menu, pilih Debug, lalu pilih Stop untuk menjeda eksekusi kode. Penghitung program berhenti di baris berikut:

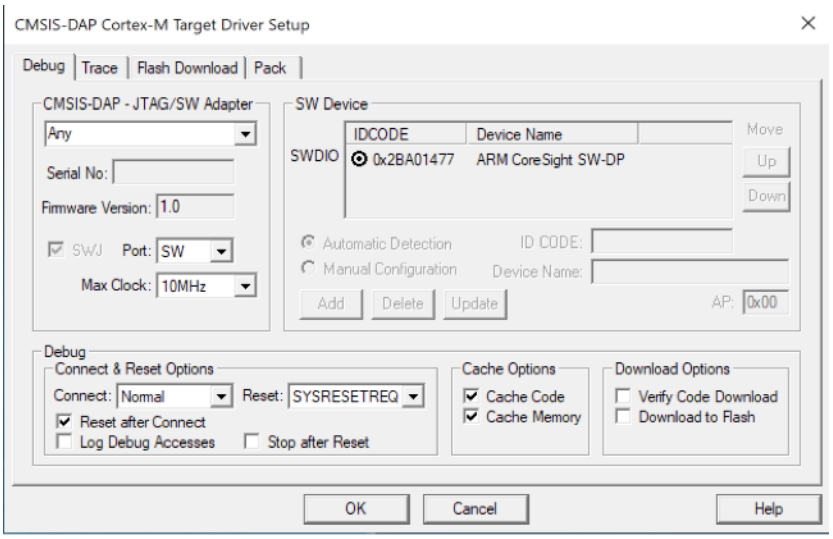
    ```
    { volatile int wait_ice = 1 ; while ( wait_ice ) ; }
    ```
11. Di jendela Call Stack + Locals, ubah nilainya `wait_ice` menjadi`0`.
12. Tetapkan breakpoint dalam kode sumber proyek Anda, dan jalankan kode.

Memecahkan masalah pengaturan debugger IDE

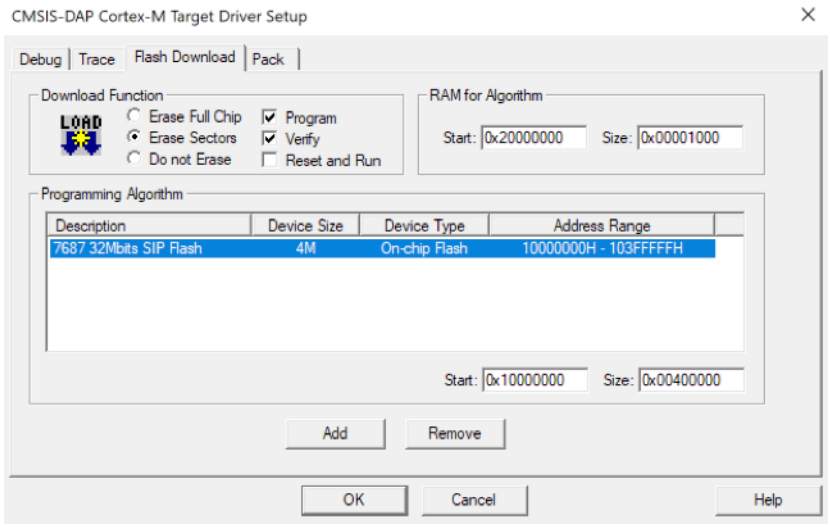
Jika Anda mengalami kesulitan men-debug aplikasi, pengaturan debugger Anda mungkin salah.

Untuk memverifikasi bahwa pengaturan debugger Anda sudah benar

1. Buka Keil µVision.
2. Klik kanan `aws_demos` proyek, pilih Options, dan di bawah tab Utilities, pilih Settings, di samping "-- Use Debug Driver --".
3. Verifikasi bahwa pengaturan di bawah tab Debug muncul sebagai berikut:



4. Verifikasi bahwa pengaturan di bawah tab Unduh Flash muncul sebagai berikut:



Untuk informasi pemecahan masalah umum tentang Memulai FreeRTOS, lihat. Pemecahan masalah memulai

Memulai dengan Microchip Curiosity PIC32 MZ EF

> **Important**
>
> Integrasi referensi ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-Freertos yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

> **Note**
>
> Sesuai dengan Microchip, kami menghapus Curiosity PIC32 MZEF (DM320104) dari cabang utama repositori Freertos Reference Integration dan tidak akan lagi membawanya dalam rilis baru. Microchip telah mengeluarkan pemberitahuan resmi bahwa PIC32 MZEF (DM320104) tidak lagi direkomendasikan untuk desain baru. Proyek PIC32 MZEF dan kode sumber masih dapat diakses melalui tag rilis sebelumnya. Microchip merekomendasikan agar pelanggan menggunakan papan Pengembangan Curiosity PIC32 MZ-EF-2.0 (DM320209) untuk desain baru. PIC32MZv1 Platform ini masih dapat ditemukan di v202012.00 dari repositori Integrasi Referensi FreeRTOS. Namun, platform ini tidak lagi didukung oleh Referensi FreeRTOS v202107.00.

Tutorial ini memberikan instruksi untuk memulai dengan Microchip Curiosity PIC32 MZ EF. Jika Anda tidak memiliki paket Microchip Curiosity PIC32 MZ EF, kunjungi Katalog Perangkat AWS Mitra untuk membelinya dari mitra kami.

Bundel ini mencakup item-item berikut:

- Dewan Pengembangan Curiosity PIC32 MZ EF
- MikroElectronika Papan Klik UART USB
- MikroElectronika WiFi 7 klik Papan
- PIC32 LAN8720 papan putri PHY

Anda juga memerlukan item berikut untuk debugging:

- MPLAB Snap In-Circuit Debugger

- (Opsional) PICkit 3 Kit Kabel Pemrograman

Sebelum memulai, Anda harus mengonfigurasi AWS IoT dan mengunduh FreeRTOS Anda untuk menghubungkan perangkat Anda ke Cloud. AWS Lihat Langkah pertama untuk instruksi.

Important

- Dalam topik ini, jalur ke direktori unduhan FreerTOS disebut sebagai. *freertos*
- Karakter ruang di *freertos* jalur dapat menyebabkan kegagalan build. Saat Anda mengkloning atau menyalin repositori, pastikan jalur yang Anda buat tidak berisi karakter spasi.
- Panjang maksimum jalur file di Microsoft Windows adalah 260 karakter. Jalur direktori unduhan FreeRTOS yang panjang dapat menyebabkan kegagalan build.
- Karena kode sumber mungkin berisi tautan simbolis, jika Anda menggunakan Windows untuk mengekstrak arsip, Anda mungkin harus:
  - Aktifkan Mode Pengembang atau,
  - Gunakan konsol yang ditinggikan sebagai administrator.

  Dengan cara ini, Windows dapat membuat tautan simbolis dengan benar saat mengekstrak arsip. Jika tidak, tautan simbolis akan ditulis sebagai file normal yang berisi jalur tautan simbolis sebagai teks atau kosong. Untuk informasi lebih lanjut, lihat entri blog Symlinks di Windows 10! .

  Jika Anda menggunakan Git di bawah Windows, Anda harus mengaktifkan Mode Pengembang atau Anda harus:
  - Setel `core.symlinks` ke true dengan perintah berikut:

    ```
    git config --global core.symlinks true
    ```
  - Gunakan konsol yang ditinggikan sebagai administrator setiap kali Anda menggunakan perintah git yang menulis ke sistem (misalnya,git pull,git clone, dangit submodule update --init --recursive).

Gambaran Umum

Tutorial ini berisi petunjuk untuk langkah-langkah memulai berikut:

1. Menghubungkan papan Anda ke mesin host.
2. Menginstal perangkat lunak pada mesin host untuk mengembangkan dan men-debug aplikasi tertanam untuk papan mikrokontroler Anda.
3. Menyusun silang aplikasi demo FreeRTOS ke gambar biner.
4. Memuat gambar biner aplikasi ke papan Anda, dan kemudian menjalankan aplikasi.
5. Berinteraksi dengan aplikasi yang berjalan di papan Anda di seluruh koneksi serial, untuk tujuan pemantauan dan debugging.

Siapkan perangkat keras Microchip Curiosity PIC32 MZ EF

1. Hubungkan MikroElectronika USB UART klik Board ke konektor MicroBus 1 pada Microchip PIC32 Curiosity MZ EF.
2. Hubungkan papan putri PIC32 LAN872 0 PHY ke header J18 pada Microchip Curiosity PIC32 MZ EF.
3. Hubungkan Papan klik UART MikroElectronika USB ke komputer Anda menggunakan kabel USB A ke USB Mini-B.
4. Untuk menghubungkan papan Anda ke internet, gunakan salah satu opsi berikut:

   - Untuk menggunakan Wi-Fi, sambungkan Papan klik MikroElectronika Wi-Fi 7 ke konektor MicroBus 2 pada Microchip Curiosity PIC32 MZ EF. Lihat Mengkonfigurasi demo FreeRTOS.
   - Untuk menggunakan Ethernet untuk menghubungkan Microchip Curiosity PIC32 MZ EF Board ke internet, sambungkan papan putri PIC32 LAN872 0 PHY ke header J18 pada Microchip Curiosity MZ EF. PIC32 Hubungkan salah satu ujung kabel Ethernet ke papan putri LAN872 0 PHY. Hubungkan ujung lainnya ke router Anda atau port internet lainnya. Anda juga harus mendefinisikan makro `PIC32_USE_ETHERNET` preprocessor.
5. Jika belum dilakukan, solder konektor sudut ke header ICSP pada Microchip Curiosity PIC32 MZ EF.
6. Connect salah satu ujung kabel ICSP dari PICkit 3 Programming Cable Kit ke Microchip Curiosity PIC32 MZ EF.

   Jika Anda tidak memiliki Kit Kabel Pemrograman PICkit 3, Anda dapat menggunakan jumper kawat M-F Dupont untuk menghubungkan koneksi sebagai gantinya. Perhatikan bahwa lingkaran putih menandakan posisi Pin 1.
7. Hubungkan ujung lain dari kabel ICSP (atau jumper) ke MPLAB Snap Debugger. Pin 1 dari Konektor Pemrograman SIL 8-pin ditandai dengan segitiga hitam di kanan bawah papan.

Pastikan bahwa setiap kabel ke Pin 1 pada Microchip Curiosity PIC32 MZ EF, ditandai dengan lingkaran putih, sejajar dengan Pin 1 pada MPLAB Snap Debugger.

Untuk informasi selengkapnya tentang MPLAB Snap Debugger, lihat Lembar Informasi Debugger Dalam Sirkuit MPLAB Snap.

Siapkan perangkat keras Microchip Curiosity PIC32 MZ EF menggunakan PICkit On Board (PKOB)

Kami menyarankan Anda mengikuti prosedur pengaturan di bagian sebelumnya. Namun, Anda dapat mengevaluasi dan menjalankan demo FreeRTOS dengan debugging dasar menggunakan programmer/debugger On Board (PKOB) PICkit terintegrasi dengan mengikuti langkah-langkah ini.

1. Hubungkan MikroElectronika USB UART klik Board ke konektor MicroBus 1 pada Microchip PIC32 Curiosity MZ EF.
2. Untuk menghubungkan papan Anda ke internet, lakukan salah satu hal berikut:
   - Untuk menggunakan Wi-Fi, sambungkan Papan klik MikroElectronika Wi-Fi 7 ke konektor MicroBus 2 pada Microchip Curiosity PIC32 MZ EF. (Ikuti langkah-langkah “Untuk mengkonfigurasi Wi-Fi Anda” diMengkonfigurasi demo FreeRTOS.
   - Untuk menggunakan Ethernet untuk menghubungkan Microchip Curiosity PIC32 MZ EF Board ke internet, sambungkan papan putri PIC32 LAN872 0 PHY ke header J18 pada Microchip Curiosity MZ EF. PIC32 Hubungkan salah satu ujung kabel Ethernet ke papan putri LAN872 0 PHY. Hubungkan ujung lainnya ke router Anda atau port internet lainnya. Anda juga harus mendefinisikan makro `PIC32_USE_ETHERNET` preprocessor.
3. Hubungkan port USB micro-B bernama “USB DEBUG” pada Microchip Curiosity PIC32 MZ EF Board ke komputer Anda menggunakan kabel USB tipe A ke USB micro-B.
4. Hubungkan Papan klik UART MikroElectronika USB ke komputer Anda menggunakan kabel USB A ke USB Mini-B.

Siapkan lingkungan pengembangan Anda

> **Note**
>
> Proyek FreeRTOS untuk perangkat ini didasarkan pada MPLAB Harmony v2. Untuk membangun proyek, Anda perlu menggunakan versi alat MPLAB yang kompatibel dengan

Harmony v2, seperti v2.10 MPLAB XC32 Compiler dan versi 2.X.X MPLAB Harmony Configurator (MHC).

1. Instal Python versi 3.x atau yang lebih baru.
2. Instal MPLAB X IDE:

> **Note**
>
> FreeRTOS AWS Reference Integrations v202007.00 saat ini didukung hanya pada .35. MPLabv5 Versi Integrasi AWS Referensi FreeRTOS sebelumnya didukung pada .40. MPLabv5

MPLabv5.35 unduhan

- MPLAB X Lingkungan Pengembangan Terpadu untuk Windows
- MPLAB X Lingkungan Pengembangan Terpadu untuk macOS
- MPLAB X Lingkungan Pengembangan Terpadu untuk Linux

MPLab Unduhan terbaru (MPLabv5.40)

- MPLAB X Lingkungan Pengembangan Terpadu untuk Windows
- MPLAB X Lingkungan Pengembangan Terpadu untuk macOS
- MPLAB X Lingkungan Pengembangan Terpadu untuk Linux

3. Instal MPLAB XC32 Compiler:
   - MPLAB XC32 /32++ Kompiler untuk Windows
   - MPLAB XC32 /32++ Kompiler untuk macOS
   - MPLAB XC32 /32++ Kompiler untuk Linux
4. Mulai emulator terminal UART dan buka koneksi dengan pengaturan berikut:
   - Tingkat baud: 115200
   - Data: 8 bit
   - Paritas: Tidak ada

- Hentikan bit: 1
- Kontrol aliran: Tidak ada

Memantau pesan MQTT di cloud

Sebelum menjalankan proyek demo FreeRTOS, Anda dapat mengatur klien MQTT di konsol untuk memantau pesan AWS IoT yang dikirim perangkat Anda ke Cloud. AWS

Untuk berlangganan topik MQTT dengan klien MQTT AWS IoT

1. Masuk ke konsol AWS IoT tersebut.
2. Di panel navigasi, pilih Uji, lalu pilih klien pengujian MQTT untuk membuka klien MQTT.
3. Dalam Subscription topic, masukkan***your-thing-name*****/example/topic**, lalu pilih Subscribe to topic.

Ketika proyek demo berhasil berjalan di perangkat Anda, Anda melihat “Hello World!” dikirim beberapa kali ke topik yang Anda berlangganan.

Bangun dan jalankan proyek demo FreeRTOS

Buka demo FreeRTOS di MPLAB IDE

1. Buka MPLAB IDE. Jika Anda memiliki lebih dari satu versi kompiler yang diinstal, Anda harus memilih kompiler yang ingin Anda gunakan dari dalam IDE.
2. Dari menu File, pilih Open Project.
3. Jelajahi dan bukaprojects/microchip/curiosity_pic32mzef/mplab/aws_demos.
4. Pilih Buka proyek.

> **Note**
>
> Saat Anda membuka proyek untuk pertama kalinya, Anda mungkin mendapatkan pesan kesalahan tentang kompiler. Di IDE, navigasikan ke Tools, Options, Embedded, lalu pilih compiler yang Anda gunakan untuk proyek Anda.

Untuk menggunakan Ethernet untuk terhubung, Anda harus menentukan makro preprocessor. `PIC32_USE_ETHERNET`

Untuk menggunakan Ethernet untuk terhubung menggunakan MPLAB IDE

1. Di MPLAB IDE, klik kanan proyek dan pilih Properties.
2. Dalam kotak dialog Project Properties, pilih ***compiler-name***(Global Options) untuk memperluasnya, lalu pilih ***compiler-name***-gcc.
3. Untuk kategori Opsi, pilih Preprocessing dan pesan, lalu tambahkan `PIC32_USE_ETHERNET` string ke makro Preprocessor.

Jalankan proyek demo FreeRTOS

1. Membangun kembali proyek Anda.
2. Pada tab Projects, klik kanan folder **`aws_demos`** tingkat atas, lalu pilih Debug.
3. Ketika debugger berhenti di breakpoint**`main()`**, dari menu Run, pilih Lanjutkan.

Membangun demo FreeRTOS dengan CMake

Jika Anda memilih untuk tidak menggunakan IDE untuk pengembangan FreeRTOS, Anda dapat CMake menggunakan alternatif untuk membangun dan menjalankan aplikasi demo atau aplikasi yang telah Anda kembangkan menggunakan editor kode pihak ketiga dan alat debugging.

Untuk membangun demo FreeRTOS dengan CMake

1. Buat direktori untuk berisi file build yang dihasilkan, seperti*build-directory*.
2. Gunakan perintah berikut untuk menghasilkan file build dari kode sumber.

```
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**
>
> Anda harus menentukan jalur yang benar ke binari Hexmate dan toolchain, seperti dan path. `C:\Program Files (x86)\Microchip\MPLABX\v5.35\mplab_platform \bin` `C:\Program Files\Microchip\xc32\v2.40\bin`

3. Ubah direktori ke direktori build (*build-directory*), lalu jalankan `make` dari direktori itu.

Untuk informasi selengkapnya, lihat Menggunakan CMake dengan FreeRTOS.

Untuk menggunakan Ethernet untuk terhubung, Anda harus menentukan makro preprocessor. `PIC32_USE_ETHERNET`

Pemecahan Masalah

Untuk informasi pemecahan masalah, lihat Pemecahan masalah memulai.

Memulai dengan Nordic n 0-DK RF5284

> **Important**
>
> Integrasi referensi ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-Freertos yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Tutorial ini memberikan instruksi untuk memulai dengan Nordic n RF5284 0-DK. Jika Anda tidak memiliki Nordic n RF5284 0-DK, kunjungi Katalog Perangkat AWS Mitra untuk membeli satu dari mitra kami.

Sebelum Anda mulai, Anda perluSiapkan AWS IoT dan Amazon Cognito untuk FreerTOS Bluetooth Low Energy.

Untuk menjalankan demo FreeRTOS Bluetooth Low Energy, Anda juga memerlukan perangkat seluler iOS atau Android dengan kemampuan Bluetooth dan Wi-Fi.

> **Note**
>
> Jika Anda menggunakan perangkat iOS, Anda memerlukan Xcode untuk membangun aplikasi seluler demo. Jika Anda menggunakan perangkat Android, Anda dapat menggunakan Android Studio untuk membuat aplikasi seluler demo.

Gambaran Umum

Tutorial ini berisi petunjuk untuk langkah-langkah memulai berikut:

1. Menghubungkan papan Anda ke mesin host.

2. Menginstal perangkat lunak pada mesin host untuk mengembangkan dan men-debug aplikasi tertanam untuk papan mikrokontroler Anda.
3. Menyusun silang aplikasi demo FreeRTOS ke gambar biner.
4. Memuat gambar biner aplikasi ke papan Anda, dan kemudian menjalankan aplikasi.
5. Berinteraksi dengan aplikasi yang berjalan di papan Anda di seluruh koneksi serial, untuk tujuan pemantauan dan debugging.

Siapkan perangkat keras Nordik

Hubungkan komputer host Anda ke port USB berlabel J2, yang terletak tepat di atas dudukan baterai sel koin di papan Nordic n RF5284 0 Anda.

Untuk informasi selengkapnya tentang pengaturan Nordic n RF5284 0-DK, lihat Panduan Pengguna Kit Pengembangan n RF5284 0.

Siapkan lingkungan pengembangan Anda

Unduh dan instal Segger Embedded Studio

FreeRTOS mendukung Segger Embedded Studio sebagai lingkungan pengembangan untuk Nordic n 0-DK. RF5284

Untuk mengatur lingkungan Anda, Anda perlu mengunduh dan menginstal Segger Embedded Studio di komputer host Anda.

Untuk mengunduh dan menginstal Segger Embedded Studio

1. Buka halaman Unduhan Studio Tertanam Segger, dan pilih opsi Embedded Studio for ARM untuk sistem operasi Anda.
2. Jalankan penginstal dan ikuti petunjuk hingga selesai.

Siapkan aplikasi demo FreeRTOS Bluetooth Low Energy Mobile SDK

Untuk menjalankan proyek demo FreeRTOS di Bluetooth Low Energy, Anda perlu menjalankan aplikasi demo FreeRTOS Bluetooth Low Energy Mobile SDK di perangkat seluler Anda.

Untuk mengatur aplikasi Demo SDK Seluler FreeRTOS Bluetooth Low Energy

1. Ikuti instruksi Ponsel SDKs untuk perangkat Bluetooth FreeRTOS untuk mengunduh dan menginstal SDK untuk platform seluler Anda di komputer host Anda.

2. Ikuti petunjuk Aplikasi demo SDK Seluler Energi Rendah FreeRTOS Bluetooth untuk menyiapkan aplikasi seluler demo di perangkat seluler Anda.

Buat koneksi serial

Segger Embedded Studio menyertakan emulator terminal yang dapat Anda gunakan untuk menerima pesan log di seluruh koneksi serial ke papan Anda.

Untuk membuat koneksi serial dengan Segger Embedded Studio

1. Buka Studio Tertanam Segger.
2. Dari menu atas, pilih Target, Connect J-Link.
3. Dari menu atas, pilih Tools, Terminal Emulator, Properties, dan atur properti seperti yang diinstruksikan. Menginstal emulator terminal
4. Dari menu atas, pilih Tools, Terminal Emulator, Connect ***`port`*** (115200, N, 8,1).

> **Note**
>
> Emulator terminal studio tertanam Segger tidak mendukung kemampuan input. Untuk ini, gunakan emulator terminal seperti PuTTy, Tera Term, atau Layar GNU. Konfigurasikan terminal untuk terhubung ke papan Anda dengan koneksi serial seperti yang diinstruksikan. Menginstal emulator terminal

Unduh dan konfigurasikan FreeRTOS

Setelah Anda mengatur perangkat keras dan lingkungan Anda, Anda dapat mengunduh FreeRTOS.

Unduh FreeRTOS

Untuk mengunduh FreeRTOS untuk Nordic RF5284 n 0-DK, buka halaman GitHub FreeRTOS dan kloning repositori. Lihat file README.md untuk instruksi.

> **Important**
>
> - Dalam topik ini, jalur ke direktori unduhan FreeRTOS disebut sebagai. ***`freertos`***
> - Karakter ruang di ***`freertos`*** jalur dapat menyebabkan kegagalan build. Saat mengkloning atau menyalin repositori, pastikan jalur yang Anda buat tidak berisi karakter spasi.

- Panjang maksimum jalur file di Microsoft Windows adalah 260 karakter. Jalur direktori unduhan FreeRTOS yang panjang dapat menyebabkan kegagalan build.
- Karena kode sumber mungkin berisi tautan simbolis, jika Anda menggunakan Windows untuk mengekstrak arsip, Anda mungkin harus:
  - Aktifkan Mode Pengembang atau,
  - Gunakan konsol yang ditinggikan sebagai administrator.

  Dengan cara ini, Windows dapat membuat tautan simbolis dengan benar saat mengekstrak arsip. Jika tidak, tautan simbolis akan ditulis sebagai file normal yang berisi jalur tautan simbolis sebagai teks atau kosong. Untuk informasi lebih lanjut, lihat entri blog Symlinks di Windows 10! .

  Jika Anda menggunakan Git di bawah Windows, Anda harus mengaktifkan Mode Pengembang atau Anda harus:
  - Setel `core.symlinks` ke true dengan perintah berikut:

    ```
    git config --global core.symlinks true
    ```
  - Gunakan konsol yang ditinggikan sebagai administrator setiap kali Anda menggunakan perintah git yang menulis ke sistem (misalnya,git pull,git clone, dangit submodule update --init --recursive).

Konfigurasikan proyek Anda

Untuk mengaktifkan demo, Anda perlu mengonfigurasi proyek Anda agar berfungsi AWS IoT. Untuk mengonfigurasi proyek Anda agar berfungsi AWS IoT, perangkat Anda harus terdaftar sebagai AWS IoT sesuatu. Anda seharusnya mendaftarkan perangkat Anda saat AndaSiapkan AWS IoT dan Amazon Cognito untuk FreerTOS Bluetooth Low Energy.

Untuk mengonfigurasi titik AWS IoT akhir

1. Masuk ke konsol AWS IoT tersebut.
2. Pada panel navigasi, silakan pilih Pengaturan.

   AWS IoT Titik akhir Anda muncul di kotak teks titik akhir data Perangkat. Itu seharusnya terlihat seperti *1234567890123*`-ats.iot.`*us-east-1*`.amazonaws.com`. Catat titik akhir ini.

3. Di panel navigasi, pilih Kelola, lalu pilih Things. Catat nama AWS IoT benda untuk perangkat Anda.
4. Dengan AWS IoT titik akhir dan nama AWS IoT benda Anda, buka *freertos*/demos/include/aws_clientcredential.h di IDE Anda, dan tentukan nilai untuk #define konstanta berikut:
   - clientcredentialMQTT_BROKER_ENDPOINT *Your AWS IoT endpoint*
   - clientcredentialIOT_THING_NAME *Your board's AWS IoT thing name*

Untuk mengaktifkan demo

1. Periksa apakah Demo Bluetooth Low Energy GATT diaktifkan. Pergi kevendors/nordic/boards/nrf52840-dk/aws_demos/config_files/iot_ble_config.h, dan tambahkan #define IOT_BLE_ADD_CUSTOM_SERVICES (1) ke daftar pernyataan define.
2. Bukavendors/nordic/boards/nrf52840-dk/aws_demos/config_files/aws_demo_config.h, dan tentukan salah satu CONFIG_OTA_MQTT_BLE_TRANSPORT_DEMO_ENABLED atau CONFIG_OTA_HTTP_BLE_TRANSPORT_DEMO_ENABLED seperti dalam contoh ini.

```
/* 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. Karena chip Nordic dilengkapi dengan RAM yang sangat sedikit (250KB), konfigurasi BLE mungkin perlu diubah untuk memungkinkan entri tabel GATT yang lebih besar dibandingkan dengan ukuran masing-masing atribut. Dengan cara ini Anda dapat menyesuaikan jumlah memori yang didapat aplikasi. Untuk melakukan ini, ganti definisi atribut berikut dalam file*freertos*/vendors/nordic/boards/nrf52840-dk/aws_demos/config_files/sdk_config.h:

- NRF_SDH_BLE_VS_UUID_COUNT

  Jumlah vendor spesifik UUIDs. Tingkatkan jumlah ini sebanyak 1 saat Anda menambahkan UUID khusus vendor baru.

- NRF_SDH_BLE_GATTS_ATTR_TAB_SIZE

  Ukuran Tabel Atribut dalam byte. Ukurannya harus kelipatan 4. Nilai ini menunjukkan jumlah set memori yang didedikasikan untuk tabel atribut (termasuk ukuran karakteristik), jadi ini akan bervariasi dari proyek ke proyek. Jika Anda melebihi ukuran tabel atribut Anda akan mendapatkan kesalahan NRF_ERROR_NO_MEM. Jika Anda memodifikasi NRF_SDH_BLE_GATTS_ATTR_TAB_SIZE biasanya Anda juga harus mengkonfigurasi ulang pengaturan RAM.

(Untuk pengujian, lokasi file adalah*freertos*`/vendors/nordic/boards/nrf52840-dk/aws_tests/config_files/sdk_config.h`.)

Bangun dan jalankan proyek demo FreeRTOS

Setelah Anda mengunduh FreeRTOS dan mengkonfigurasi proyek demo Anda, Anda siap untuk membangun dan menjalankan proyek demo di papan Anda.

> **Important**
>
> Jika ini adalah pertama kalinya Anda menjalankan demo di papan ini, Anda perlu mem-flash bootloader ke papan sebelum demo dapat berjalan.
> Untuk membangun dan mem-flash bootloader, ikuti langkah-langkah di bawah ini, tetapi alih-alih menggunakan file `projects/nordic/nrf52840-dk/ses/aws_demos/aws_demos.emProject` proyek, gunakan`projects/nordic/nrf52840-dk/ses/aws_demos/bootloader/bootloader.emProject`.

Untuk membangun dan menjalankan demo FreeRTOS Bluetooth Low Energy dari Segger Embedded Studio

1. Buka Studio Tertanam Segger. Dari menu atas, pilih File, pilih Open Solution, dan kemudian navigasikan ke file proyek `projects/nordic/nrf52840-dk/ses/aws_demos/aws_demos.emProject`

2. Jika Anda menggunakan emulator terminal Segger Embedded Studio, pilih Tools dari menu atas, lalu pilih Terminal Emulator, Terminal Emulator untuk menampilkan informasi dari koneksi serial Anda.

   Jika Anda menggunakan alat terminal lain, Anda dapat memantau alat itu untuk output dari koneksi serial Anda.

3. Klik kanan proyek `aws_demos` demo di Project Explorer, dan pilih Build.

   > **Note**
   >
   > Jika ini adalah pertama kalinya Anda menggunakan Segger Embedded Studio, Anda mungkin melihat peringatan “Tidak ada lisensi untuk penggunaan komersial”. Segger Embedded Studio dapat digunakan secara gratis untuk perangkat Nordic Semiconductor. Minta lisensi gratis kemudian, selama pengaturan pilih Aktifkan Lisensi Gratis Anda, dan ikuti petunjuknya.

4. Pilih Debug, lalu pilih Go.

   Setelah demo dimulai, ia menunggu untuk dipasangkan dengan perangkat seluler di Bluetooth Low Energy.

5. Ikuti instruksi untuk MQTT melalui Aplikasi Demo Energi Rendah Bluetooth untuk menyelesaikan demo dengan aplikasi demo FreeRTOS Bluetooth Low Energy Mobile SDK sebagai proxy MQTT seluler.

Pemecahan Masalah

Untuk informasi pemecahan masalah umum tentang Memulai FreeRTOS, lihat. Pemecahan masalah memulai

Memulai dengan NuMaker-IoT-M Nuvoton 487

> **Important**
>
> Integrasi referensi ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-Freertos yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Tutorial ini memberikan instruksi untuk memulai dengan papan pengembangan Nuvoton NuMaker-IoT-M 487. Mikrokontroler seri, dan termasuk modul RJ45 Ethernet dan Wi-Fi bawaan. Jika Anda tidak memiliki Nuvoton NuMaker-IoT-M 487, kunjungi Katalog Perangkat AWS Mitra untuk membelinya dari mitra kami.

Sebelum Anda mulai, Anda harus mengkonfigurasi AWS IoT dan perangkat lunak FreeRTOS Anda untuk menghubungkan papan pengembangan Anda ke Cloud. AWS Untuk petunjuk, silakan lihat Langkah pertama. Dalam tutorial ini, jalur ke direktori unduhan FreeRTOS disebut sebagai. *freertos*

Gambaran Umum

Tutorial ini memandu Anda melalui langkah-langkah berikut:

1. Instal perangkat lunak pada mesin host Anda untuk mengembangkan dan men-debug aplikasi tertanam untuk papan mikrokontroler Anda.
2. Kompilasi silang aplikasi demo FreerTOS ke gambar biner.
3. Muat gambar biner aplikasi ke papan Anda, lalu jalankan aplikasi.

Siapkan lingkungan pengembangan Anda

Edisi Keil MDK Nuvoton dirancang untuk mengembangkan dan men-debug aplikasi untuk papan Nuvoton M487. Versi Keil MDK v5 Essential, Plus, atau Pro juga harus berfungsi untuk MCU Nuvoton M487 (Cortex-M4 core). Anda dapat mengunduh edisi Keil MDK Nuvoton dengan diskon harga untuk seri Nuvoton Cortex-M4. MCUs Keil MDK hanya didukung pada Windows.

Untuk menginstal alat pengembangan untuk NuMaker-IoT-M 487

1. Unduh Keil MDK Nuvoton Edition dari situs web Keil MDK.
2. Instal MDK Keil di mesin host Anda menggunakan lisensi Anda. Keil MDK mencakup Keil µVision IDE, rantai alat kompilasi C/C++, dan debugger µVision.

   Jika Anda mengalami masalah selama instalasi, hubungi Nuvoton untuk bantuan.
3. Instal Nu-Link_Keil_Driver_v3.06.7215R (atau versi terbaru), yang ada di halaman Alat Pengembangan Nuvoton.

Bangun dan jalankan proyek demo FreeRTOS

Untuk membangun proyek demo FreeRTOS

1. Buka Keil µVision IDE.
2. Pada menu File, pilih Buka. Dalam kotak dialog Open file, pastikan pemilih jenis file diatur ke Project Files.
3. Pilih salah satu proyek demo Wi-Fi atau Ethernet untuk dibangun.
   - Untuk membuka proyek demo Wi-Fi, pilih proyek target `aws_demos.uvproj` di *`freertos`*`\projects\nuvoton\numaker_iot_m487_wifi\uvision\aws_demos` direktori.
   - Untuk membuka proyek demo Ethernet, pilih proyek target `aws_demos_eth.uvproj` di *`freertos`*`\projects\nuvoton\numaker_iot_m487_wifi\uvision\aws_demos_eth` direktori.
4. Untuk memastikan pengaturan Anda benar untuk mem-flash papan, klik kanan `aws_demo` proyek di IDE, lalu pilih Opsi. (Lihat Pemecahan Masalah untuk lebih jelasnya.)
5. Pada tab Utilities, verifikasi bahwa Use Target Driver for Flash Programming dipilih, dan Nuvoton Nu-Link Debugger ditetapkan sebagai driver target.
6. Pada tab Debug, di sebelah Nuvoton Nu-Link Debugger, pilih Pengaturan.
7. Verifikasi bahwa Jenis Chip diatur ke M480.
8. Di panel navigasi Keil µVision IDE Project, pilih project. `aws_demos` Pada menu Project, pilih Build Target.

Anda dapat menggunakan klien MQTT di AWS IoT konsol untuk memantau pesan yang dikirim perangkat Anda ke Cloud. AWS

Untuk berlangganan topik MQTT dengan klien MQTT AWS IoT

1. Masuk ke konsol AWS IoT tersebut.
2. Di panel navigasi, pilih Uji, lalu pilih klien pengujian MQTT untuk membuka klien MQTT.
3. Dalam Subscription topic, masukkan***`your-thing-name`*****`/example/topic`**, lalu pilih Subscribe to topic.

Untuk menjalankan proyek demo FreeRTOS

1. Hubungkan papan Numaker-IoT-M 487 Anda ke mesin host Anda (komputer).
2. Membangun kembali proyek.
3. Di Keil µVision IDE, pada menu Flash, pilih Unduh.
4. Pada menu Debug, pilih Start/Stop Debug Session.
5. Ketika debugger berhenti di breakpoint`main()`, buka menu Run, lalu pilih Run (F5).

   Anda akan melihat pesan MQTT yang dikirim oleh perangkat Anda di klien MQTT di konsol. AWS IoT

Menggunakan CMake dengan FreeRTOS

Anda juga dapat menggunakan CMake untuk membangun dan menjalankan aplikasi demo FreeRTOS atau aplikasi yang telah Anda kembangkan menggunakan editor kode pihak ketiga dan alat debugging.

Pastikan Anda telah menginstal sistem CMake build. Ikuti instruksi diMenggunakan CMake dengan FreeRTOS, lalu ikuti langkah-langkah di bagian ini.

> **Note**
>
> Pastikan jalur ke lokasi kompiler (Keil) ada di variabel sistem Path Anda, misalnya,. `C:\Keil_v5\ARM\ARMCC\bin`

Anda juga dapat menggunakan klien MQTT di AWS IoT konsol untuk memantau pesan yang dikirim perangkat Anda ke Cloud. AWS

Untuk berlangganan topik MQTT dengan klien MQTT AWS IoT

1. Masuk ke konsol AWS IoT tersebut.
2. Di panel navigasi, pilih Uji, lalu pilih klien pengujian MQTT untuk membuka klien MQTT.
3. Dalam Subscription topic, masukkan***your-thing-name*****/example/topic**, lalu pilih Subscribe to topic.

Untuk menghasilkan file build dari file sumber dan menjalankan proyek demo

1. Di mesin host Anda, buka prompt perintah dan arahkan ke *freertos* folder.
2. Buat folder untuk berisi file build yang dihasilkan. Kami akan merujuk ke folder ini sebagai*BUILD_FOLDER*.
3. Buat file build untuk demo Wi-Fi atau Ethernet.

   - Untuk Wi-Fi:

     Arahkan ke direktori yang berisi file sumber untuk proyek demo FreeRTOS. Kemudian, buat file build dengan menjalankan perintah berikut.

     ```
     cmake -DVENDOR=nuvoton -DBOARD=numaker_iot_m487_wifi -DCOMPILER=arm-keil -S . -
     B BUILD_FOLDER -G Ninja
     ```

   - Untuk Ethernet:

     Arahkan ke direktori yang berisi file sumber untuk proyek demo FreeRTOS. Kemudian, buat file build dengan menjalankan perintah berikut.

     ```
     cmake -DVENDOR=nuvoton -DBOARD=numaker_iot_m487_wifi -DCOMPILER=arm-keil -
     DAFR_ENABLE_ETH=1 -S . -B BUILD_FOLDER -G Ninja
     ```

4. Hasilkan biner untuk flash ke M487 dengan menjalankan perintah berikut.

   ```
   cmake --build BUILD_FOLDER
   ```

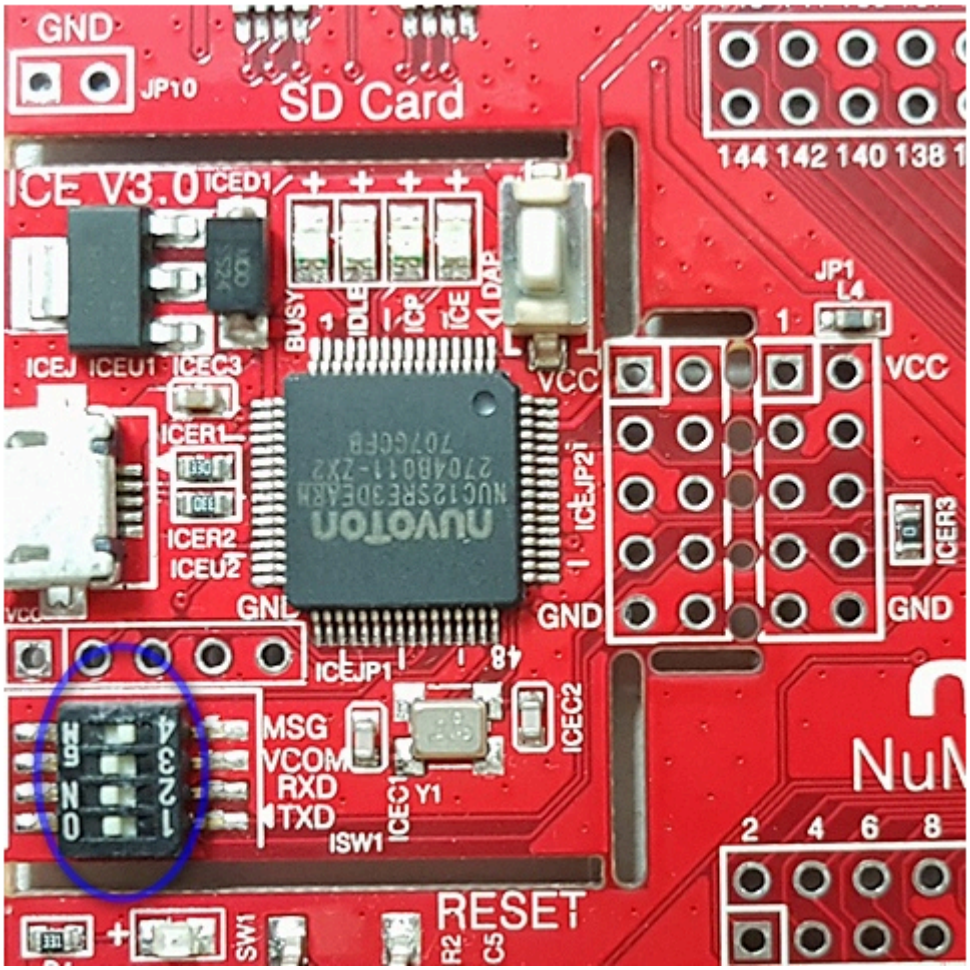
   Pada titik ini, file biner `aws_demos.bin` harus ada di *BUILD_FOLDER*`/vendors/Nuvoton/boards/numaker_iot_m487_wifi` folder.

5. Untuk mengkonfigurasi papan untuk mode flashing, pastikan sakelar MSG (No.4 dari ISW1 pada ICE) diaktifkan. Saat Anda mencolokkan papan, jendela (dan drive) akan ditetapkan. (Lihat Pemecahan Masalah).
6. Buka emulator terminal untuk melihat pesan melalui UART. Ikuti petunjuk di Menginstal emulator terminal.
7. Jalankan proyek demo dengan menyalin biner yang dihasilkan ke perangkat.

   Jika Anda berlangganan topik MQTT dengan klien MQTT, Anda akan melihat pesan AWS IoT MQTT yang dikirim oleh perangkat Anda di konsol. AWS IoT

Pemecahan Masalah

- Jika windows Anda tidak dapat mengenali perangkat`VCOM`, instal driver port serial NuMaker windows dari tautan Nu-Link USB Driver v1.6.
- Jika Anda menghubungkan perangkat Anda ke Keil MDK (IDE) melalui Nu-Link, pastikan sakelar MSG (No.4 dari ISW1 pada ICE) MATI, seperti yang ditunjukkan.



Jika Anda mengalami masalah saat menyiapkan lingkungan pengembangan atau menghubungkan ke papan Anda, hubungi Nuvoton.

Mendebug proyek FreeRTOS di Keil µVision

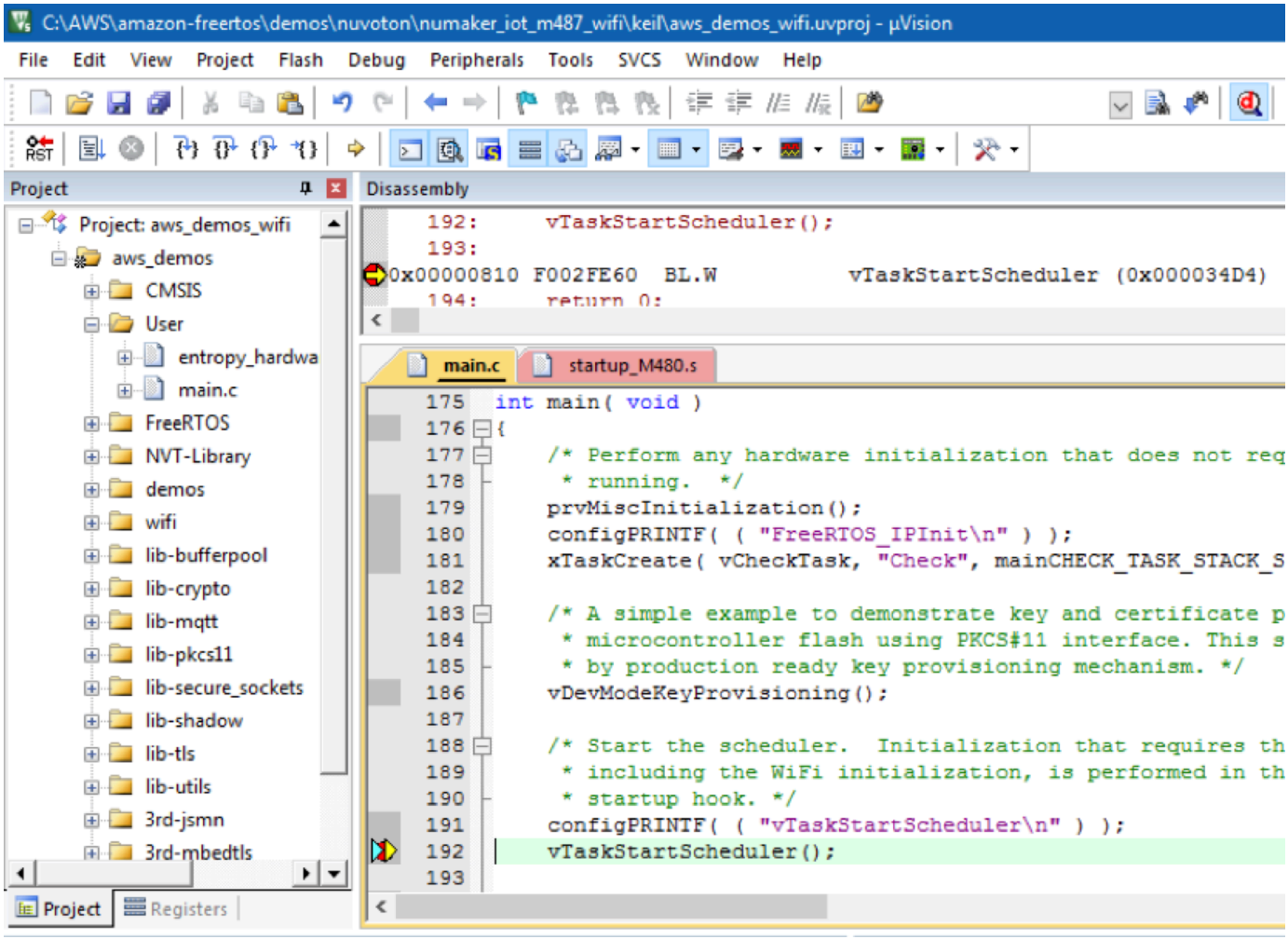
Untuk memulai sesi debug di Keil µVision

1. Buka Keil µVision.
2. Ikuti langkah-langkah untuk membangun proyek demo FreeRTOS di. Bangun dan jalankan proyek demo FreeRTOS

3. Pada menu Debug, pilih Start/Stop Debug Session.

   Jendela Call Stack + Locals muncul saat Anda memulai sesi debug. µVision mem-flash demo ke papan, menjalankan demo, dan berhenti di awal fungsi. `main()`

4. Tetapkan breakpoint dalam kode sumber proyek Anda, lalu jalankan kodenya. Proyek ini akan terlihat seperti berikut ini.



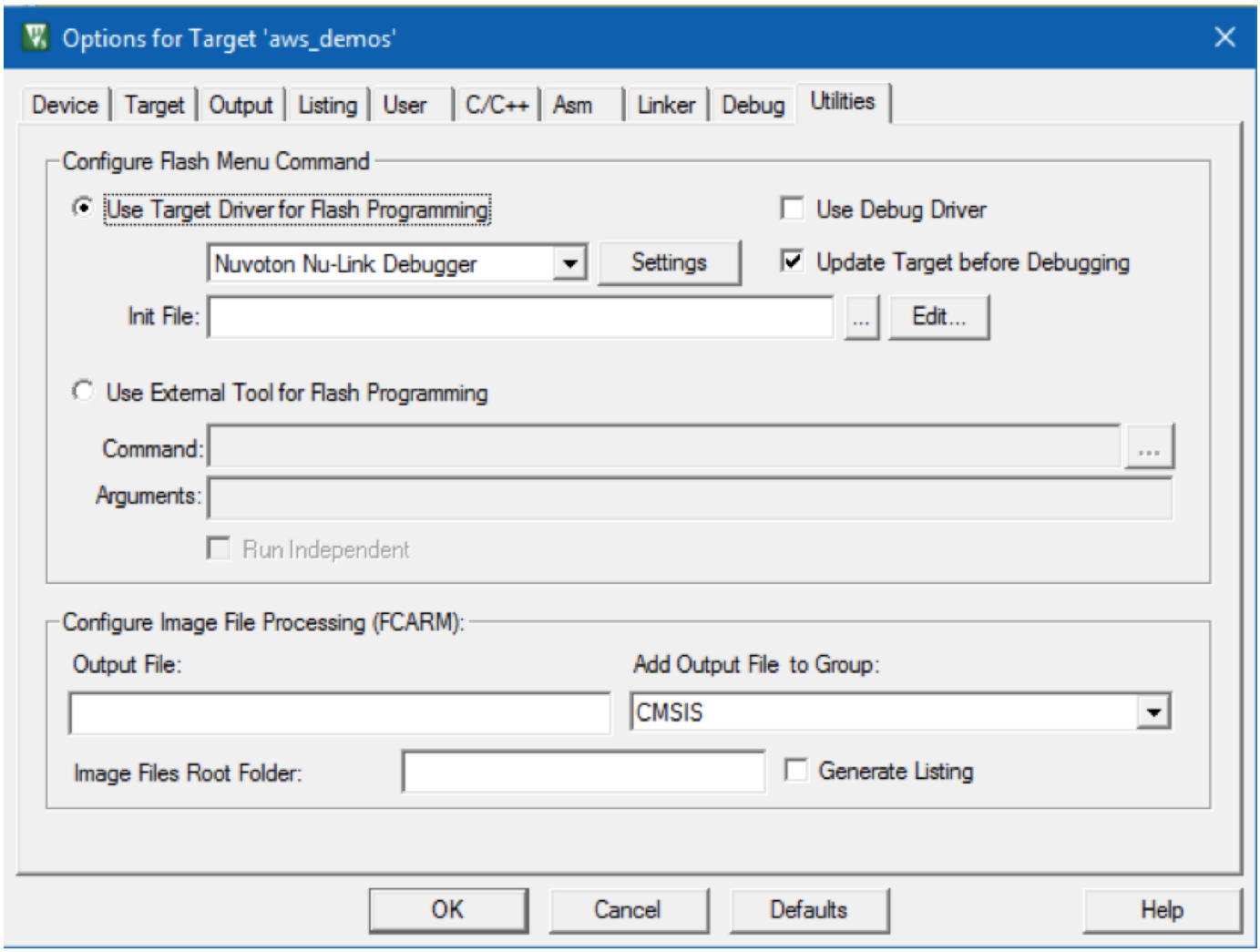
Pemecahan masalah pengaturan debug µVision

Jika Anda mengalami masalah saat men-debug aplikasi, periksa apakah pengaturan debug Anda diatur dengan benar di Keil µVision.

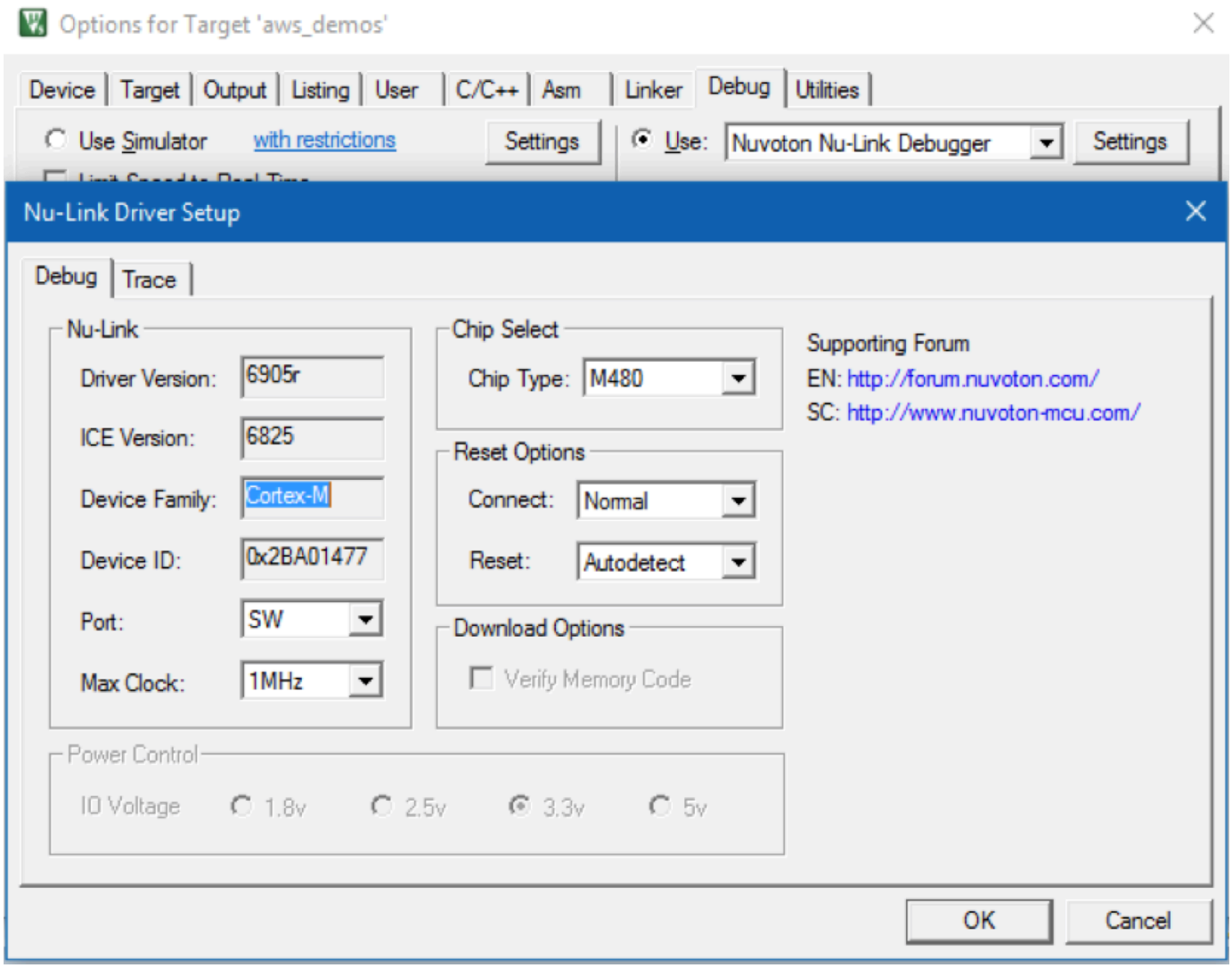
Untuk memverifikasi bahwa pengaturan debug µVision sudah benar

1. Buka Keil µVision.
2. Klik kanan `aws_demo` proyek di IDE, lalu pilih Options.

3. Pada tab Utilities, verifikasi bahwa Use Target Driver for Flash Programming dipilih, dan Nuvoton Nu-Link Debugger ditetapkan sebagai driver target.



4. Pada tab Debug, di sebelah Nuvoton Nu-Link Debugger, pilih Pengaturan.



5. Verifikasi bahwa Jenis Chip diatur ke M480.

Memulai dengan Modul LPC54 IoT NXP 018

> **Important**
>
> Integrasi referensi ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-Freertos yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Tutorial ini memberikan instruksi untuk memulai dengan Modul LPC54 IoT NXP 018. Jika Anda tidak memiliki Modul LPC54 IoT NXP 018, kunjungi Katalog Perangkat AWS Mitra untuk membelinya dari mitra kami. Gunakan kabel USB untuk menghubungkan Modul LPC54 IoT NXP 018 Anda ke komputer Anda.

Sebelum memulai, Anda harus mengonfigurasi AWS IoT dan mengunduh FreeRTOS Anda untuk menghubungkan perangkat Anda ke Cloud. AWS Lihat Langkah pertama untuk instruksi. Dalam tutorial ini, jalur ke direktori unduhan FreerTOS disebut sebagai. *freertos*

Gambaran Umum

Tutorial ini berisi petunjuk untuk langkah-langkah memulai berikut:

1. Menghubungkan papan Anda ke mesin host.
2. Menginstal perangkat lunak pada mesin host untuk mengembangkan dan men-debug aplikasi tertanam untuk papan mikrokontroler Anda.
3. Menyusun silang aplikasi demo FreeRTOS ke gambar biner.
4. Memuat gambar biner aplikasi ke papan Anda, dan kemudian menjalankan aplikasi.

Siapkan perangkat keras NXP

Untuk mengatur NXP 018 LPC54

- Hubungkan komputer Anda ke port USB pada NXP LPC54 018.

Untuk mengatur debugger JTAG

Anda memerlukan debugger JTAG untuk meluncurkan dan men-debug kode Anda yang berjalan di papan NXP 018. LPC54 FreeRTOS diuji menggunakan Modul IoT OM4 0006. Untuk informasi selengkapnya tentang debugger yang didukung, lihat Panduan Pengguna untuk Modul LPC54 IoT NXP 018 yang tersedia dari halaman produk Modul IoT 0007 OM4 018. LPC54

1. Jika Anda menggunakan debugger Modul IoT OM4 0006, gunakan kabel konverter untuk menghubungkan konektor 20-pin dari debugger ke konektor 10-pin pada modul NXP IoT.
2. Hubungkan NXP LPC54 018 dan OM4 0006 IoT Module Debugger ke port USB di komputer Anda menggunakan kabel mini-USB ke USB.

Siapkan lingkungan pengembangan Anda

FreeRTOS mendukung IDEs dua untuk Modul IoT NXP LPC54 018: IAR Embedded Workbench dan. MCUXpresso

Sebelum Anda mulai, instal salah satunya IDEs.

Untuk menginstal IAR Embedded Workbench untuk ARM

1. Jelajahi IAR Embedded Workbench untuk ARM dan unduh perangkat lunaknya.

   > **Note**
   >
   > IAR Embedded Workbench untuk ARM membutuhkan Microsoft Windows.

2. Jalankan penginstal dan ikuti petunjuknya.
3. Di Wisaya Lisensi, pilih Daftar dengan Sistem IAR untuk mendapatkan lisensi evaluasi.
4. Letakkan bootloader di perangkat sebelum mencoba menjalankan demo apa pun.

Untuk menginstal MCUXpresso dari NXP

1. Unduh dan jalankan MCUXpresso penginstal dari NXP.

   > **Note**
   >
   > Versi 10.3.x dan yang lebih baru didukung.

2. Jelajahi MCUXpresso SDK dan pilih Build your SDK.

   > **Note**
   >
   > Versi 2.5 dan yang lebih baru didukung.

3. Pilih Pilih Papan Pengembangan.
4. Di bawah Pilih Papan Pengembangan, di Cari berdasarkan Nama, masukkan**`LPC54018-IoT-Module`**.
5. Di bawah Papan, pilih LPC54018-IOT-Modul.
6. Verifikasi detail perangkat keras, lalu pilih Build MCUXepresso SDK.

7. SDK untuk Windows menggunakan MCUXpresso IDE sudah dibangun. Pilih Unduh SDK. Jika Anda menggunakan sistem operasi lain, di bawah Host OS, pilih sistem operasi Anda, lalu pilih Unduh SDK.
8. Mulai MCUXpresso IDE, dan pilih SDKs tab Terinstal.
9. Seret dan lepas file arsip SDK yang diunduh ke SDKs jendela Terinstal.

Jika mengalami masalah saat penginstalan, lihat NXP Support atau NXP Developer Resources.

Memantau pesan MQTT di cloud

Sebelum menjalankan proyek demo FreeRTOS, Anda dapat mengatur klien MQTT di konsol untuk memantau pesan AWS IoT yang dikirim perangkat Anda ke Cloud. AWS

Untuk berlangganan topik MQTT dengan klien MQTT AWS IoT

1. Masuk ke konsol AWS IoT tersebut.
2. Di panel navigasi, pilih Uji, lalu pilih klien pengujian MQTT untuk membuka klien MQTT.
3. Dalam Subscription topic, masukkan***your-thing-name*****/example/topic**, lalu pilih Subscribe to topic.

Ketika proyek demo berhasil berjalan di perangkat Anda, Anda melihat “Hello World!” dikirim beberapa kali ke topik yang Anda berlangganan.

Bangun dan jalankan proyek Demo FreeRTOS

Impor demo FreeRTOS ke IDE Anda

Untuk mengimpor kode sampel FreeRTOS ke IAR Embedded Workbench IDE

1. Buka IAR Embedded Workbench, dan dari menu File, pilih Open Workspace.
2. Di kotak teks direktori pencarian, masukkan**projects/nxp/lpc54018iotmodule/iar/aws_demos**, dan pilih aws_demos.eww.
3. Dari menu Project, pilih Rebuild All.

Untuk mengimpor kode sampel FreeRTOS ke IDE MCUXpresso

1. Buka MCUXpresso, dan dari menu File, pilih Buka Proyek Dari Sistem File.

2. Di kotak teks Direktori, masukkan`projects/nxp/lpc54018iotmodule/mcuxpresso/aws_demos`, dan pilih Selesai
3. Dari menu Project, pilih Build All.

Jalankan proyek demo FreeRTOS

Untuk menjalankan proyek demo FreeRTOS dengan IAR Embedded Workbench IDE

1. Di IDE Anda, dari menu Project, pilih Make.
2. Dari menu Project, pilih Download dan Debug.
3. Dari menu Debug, pilih Mulai Debugging.
4. Ketika debugger berhenti di breakpoint**`main`**, dari menu Debug, pilih Go.

> **Note**
>
> Jika kotak dialog Pemilihan Perangkat J-Link terbuka, pilih OK untuk melanjutkan. Dalam Pengaturan Perangkat Target kotak dialog, pilih Tidak ditentukan, pilih Cortex-M4, lalu pilih OK. Anda hanya perlu melakukan ini sekali.

Untuk menjalankan proyek demo FreeRTOS dengan IDE MCUxpresso

1. Di IDE Anda, dari menu Project, pilih Build.
2. Jika ini adalah pertama kalinya Anda melakukan debug, pilih `aws_demos` proyek dan dari toolbar Debug, pilih tombol debug biru.
3. Setiap probe debug yang terdeteksi ditampilkan. Pilih probe yang ingin Anda gunakan, lalu pilih OK untuk memulai debugging.

> **Note**
>
> Saat debugger berhenti di breakpoint`main()`, tekan tombol debug restart
>
>
>
> sekali untuk mengatur ulang sesi debugging. (Ini diperlukan karena bug dengan MCUXpresso debugger untuk modul NXP54 018-IoT).

4. Ketika debugger berhenti di breakpoint**`main()`**, dari menu Debug, pilih Go.

Pemecahan Masalah

Untuk informasi pemecahan masalah umum tentang Memulai FreeRTOS, lihat. Pemecahan masalah memulai

Memulai dengan Renesas Starter Kit+untuk N-2MB RX65

> **Important**
>
> Integrasi referensi ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-FreerTOS yang sekarang sudah tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Tutorial ini memberikan instruksi untuk memulai dengan Renesas Starter Kit+untuk N-2MB. RX65 Jika Anda tidak memiliki Renesas RSK+untuk RX65 N-2MB, kunjungi Katalog Perangkat AWS Mitra, dan beli satu dari mitra kami.

Sebelum memulai, Anda harus mengonfigurasi AWS IoT dan mengunduh FreeRTOS Anda untuk menghubungkan perangkat Anda ke Cloud. AWS Lihat Langkah pertama untuk instruksi. Dalam tutorial ini, jalur ke direktori unduhan FreeRTOS disebut sebagai. *`freertos`*

Gambaran Umum

Tutorial ini berisi petunjuk untuk langkah-langkah memulai berikut:

1. Menghubungkan papan Anda ke mesin host.
2. Menginstal perangkat lunak pada mesin host untuk mengembangkan dan men-debug aplikasi tertanam untuk papan mikrokontroler Anda.
3. Menyusun silang aplikasi demo FreeRTOS ke gambar biner.
4. Memuat gambar biner aplikasi ke papan Anda, dan kemudian menjalankan aplikasi.

Siapkan perangkat keras Renesas

Untuk mengatur RSK+untuk N-2MB RX65

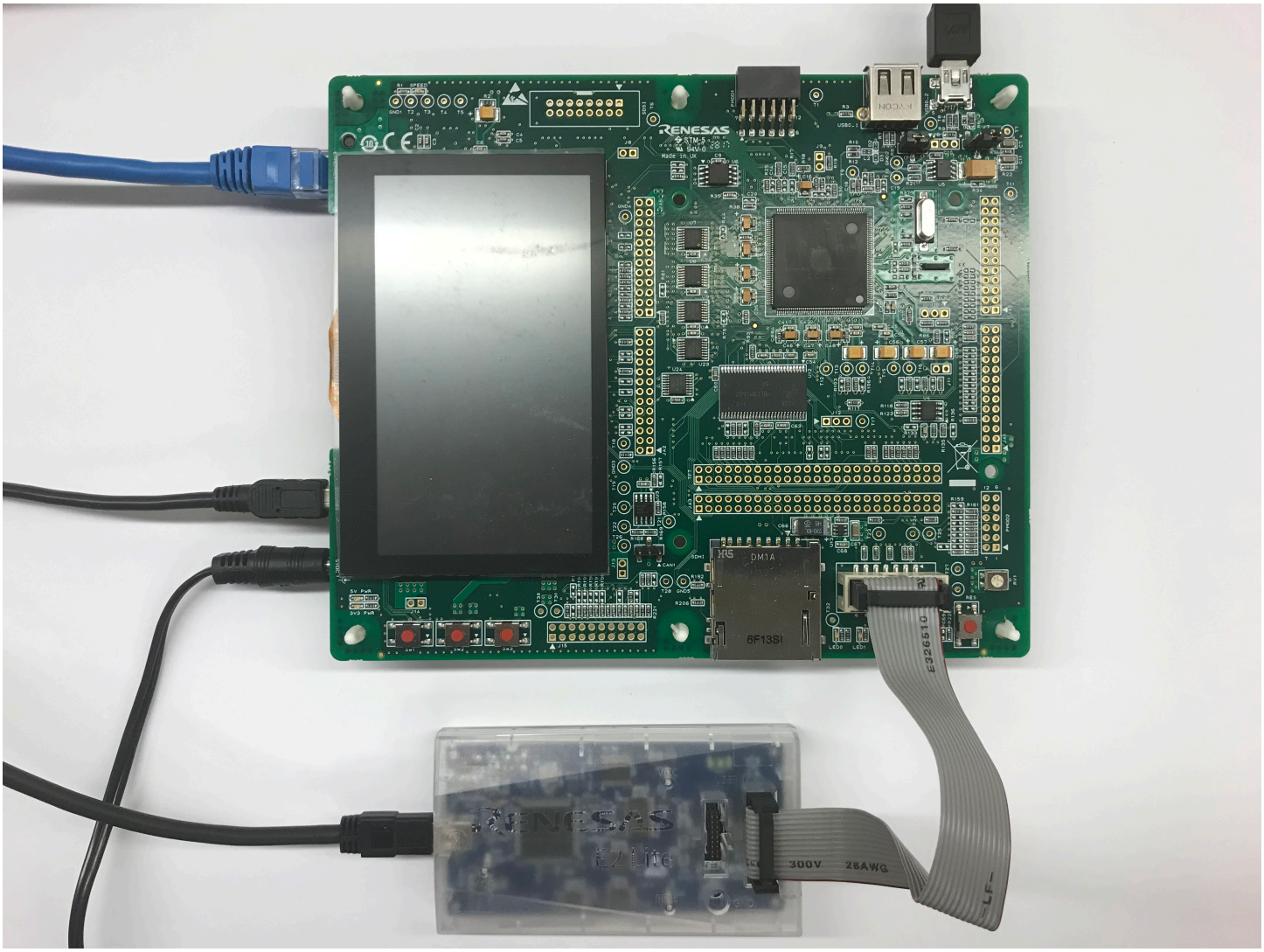
1. Hubungkan adaptor daya+5V positif ke konektor PWR pada RSK+untuk N-2MB. RX65

2. Hubungkan komputer Anda ke USB2 port.0 FS pada RSK+untuk N-2MB. RX65
3. Hubungkan komputer Anda ke USB-to-serial port pada RSK+untuk RX65 N-2MB.
4. Hubungkan router atau port Ethernet yang terhubung ke internet ke port Ethernet pada RSK +untuk N-2MB. RX65

Untuk mengatur modul E2 Lite Debugger

1. Gunakan kabel pita 14-pin untuk menghubungkan modul E2 Lite Debugger ke port 'E1/E2 Lite' pada RSK+untuk N-2MB. RX65
2. Gunakan kabel USB untuk menghubungkan modul debugger E2 Lite ke mesin host Anda. Ketika debugger E2 Lite terhubung ke papan dan komputer Anda, LED 'ACT' hijau pada debugger berkedip.
3. Setelah debugger terhubung ke mesin host Anda dan RSK+untuk RX65 N-2MB, driver debugger E2 Lite mulai menginstal.

   Perhatikan bahwa hak administrator diperlukan untuk menginstal driver.



Siapkan lingkungan pengembangan Anda

Untuk mengatur konfigurasi FreeRTOS untuk RX65 RSK+untuk N-2MB, gunakan kompiler Renesas e 2 studio IDE dan CC-RX.

> **Note**
>
> Kompiler Renesas e 2 studio IDE dan CC-RX hanya didukung pada sistem operasi Windows 7, 8, dan 10.

Untuk mengunduh dan menginstal e 2 studio

1. Buka halaman unduhan penginstal studio Renesas e 2, dan unduh penginstal offline.
2. Anda diarahkan ke halaman Login Renesas.

Jika Anda memiliki akun dengan Renesas, masukkan kredensyal masuk Anda, lalu pilih Login.

Jika Anda tidak memiliki akun, pilih Daftar sekarang, dan ikuti langkah pendaftaran pertama. Anda harus menerima email dengan tautan untuk mengaktifkan akun Renesas Anda. Ikuti tautan ini untuk menyelesaikan pendaftaran Anda dengan Renesas, lalu masuk ke Renesas.

3. Setelah Anda masuk, unduh penginstal studio e ^{2} ke komputer Anda.
4. Buka installer dan ikuti langkah-langkah untuk menyelesaikannya.

Untuk informasi lebih lanjut, lihat studio e ^{2} di situs web Renesas.

Untuk men-download dan menginstal RX Family C/C++ Compiler Package

1. Buka halaman download RX Family C/C++ Compiler Package, dan unduh paket V3.00.00.
2. Buka executable dan instal compiler.

Untuk informasi selengkapnya, lihat C/C++ Compiler Package for RX Family di situs web Renesas.

> **Note**
>
> Kompiler tersedia gratis untuk versi evaluasi saja dan berlaku selama 60 hari. Pada hari ke-61, Anda perlu mendapatkan Kunci Lisensi. Untuk informasi selengkapnya, lihat Alat Perangkat Lunak Evaluasi.

Bangun dan jalankan sampel FreeRTOS

Sekarang setelah Anda mengkonfigurasi proyek demo, Anda siap untuk membangun dan menjalankan proyek di papan Anda.

Membangun FreeRTOS Demo di e 2 studio

Untuk mengimpor dan membangun demo di e ^{2} studio

1. Luncurkan e ^{2} studio dari menu Start.
2. Pada jendela Pilih direktori sebagai ruang kerja, telusuri ke folder tempat Anda ingin bekerja, dan pilih Luncurkan.

3. Pertama kali Anda membuka e ^{2} studio, jendela Toolchain Registry terbuka. Pilih Renesas Toolchains, dan konfirmasikan bahwa itu **`CC-RX v3.00.00`** dipilih. Pilih Daftar, lalu pilih OK.
4. Jika Anda membuka e ^{2} studio untuk pertama kalinya, jendela Registrasi Code Generator muncul. Pilih OKE.
5. Jendela register komponen Code Generator COM muncul. Di bawah Silahkan restart e ^{2} studio untuk menggunakan Code Generator, pilih OK.
6. Jendela Restart e ^{2} studio muncul. Pilih OKE.
7. e ^{2} studio restart. Pada jendela Select a directory as a workspace, pilih Launch.
8. Pada layar sambutan e ^{2} studio, pilih ikon panah Go to the e ^{2} studio workbench.
9. Klik kanan jendela Project Explorer, dan pilih Impor.
10. Di wizard impor, pilih General, Existing Projects into Workspace, lalu pilih Next.
11. Pilih Browse, cari direktori`projects/renesas/rx65n-rsk/e2studio/aws_demos`, lalu pilih Selesai.
12. Dari menu Project, pilih Project, Build All.

    Konsol build mengeluarkan pesan peringatan bahwa License Manager tidak diinstal. Anda dapat mengabaikan pesan ini kecuali Anda memiliki kunci lisensi untuk kompiler CC-RX. Untuk menginstal License Manager, lihat halaman unduhan License Manager.

Memantau pesan MQTT di cloud

Sebelum menjalankan proyek demo FreeRTOS, Anda dapat mengatur klien MQTT di konsol untuk memantau pesan AWS IoT yang dikirim perangkat Anda ke Cloud. AWS

Untuk berlangganan topik MQTT dengan klien MQTT AWS IoT

1. Masuk ke konsol AWS IoT tersebut.
2. Di panel navigasi, pilih Uji, lalu pilih klien pengujian MQTT untuk membuka klien MQTT.
3. Dalam Subscription topic, masukkan***`your-thing-name`***`/example/topic`, lalu pilih Subscribe to topic.

Ketika proyek demo berhasil berjalan di perangkat Anda, Anda melihat “Hello World!” dikirim beberapa kali ke topik yang Anda berlangganan.

Jalankan proyek FreeRTOS

Untuk menjalankan proyek di e ^{2} studio

1. Konfirmasikan bahwa Anda telah menghubungkan modul E2 Lite Debugger ke RSK+Anda untuk N-2MB RX65
2. Dari menu atas, pilih Run, Debug Configuration.
3. Perluas Renesas GDB Hardware Debugging, dan pilih aws_demos. HardwareDebug
4. Pilih tab Debugger, lalu pilih tab Pengaturan Koneksi. Konfirmasikan bahwa pengaturan koneksi Anda sudah benar.
5. Pilih Debug untuk mengunduh kode ke papan Anda dan mulai men-debug.

   Anda mungkin diminta oleh peringatan firewall untuk`e2-server-gdb.exe`. Periksa Jaringan pribadi, seperti jaringan rumah atau kantor saya, lalu pilih Izinkan akses.
6. e ^{2} studio mungkin meminta untuk mengubah ke Renesas Debug Perspective. Pilih Ya.

   LED 'ACT' hijau pada E2 Lite Debugger menyala.
7. Setelah kode diunduh ke papan tulis, pilih Lanjutkan untuk menjalankan kode hingga baris pertama fungsi utama. Pilih Lanjutkan lagi untuk menjalankan sisa kode.

Untuk proyek terbaru yang dirilis oleh Renesas, lihat `renesas-rx` fork `amazon-freertos` repositori di. GitHub

Pemecahan Masalah

Untuk informasi pemecahan masalah umum tentang Memulai FreeRTOS, lihat. Pemecahan masalah memulai

Memulai dengan STMicroelectronics STM32 L4 Discovery Kit IoT Node

> **Important**
>
> Integrasi referensi ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-Freertos yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Tutorial ini memberikan instruksi untuk memulai dengan STMicroelectronics STM32 L4 Discovery Kit IoT Node. Jika Anda belum memiliki STMicroelectronics STM32 L4 Discovery Kit IoT Node, kunjungi Katalog AWS Perangkat Mitra untuk membelinya dari mitra kami.

Pastikan Anda telah menginstal firmware Wi-Fi terbaru. Untuk mengunduh firmware Wi-Fi terbaru, lihat STM32L4 Discovery kit IoT node, nirkabel berdaya rendah, Bluetooth Low Energy, NFC, Sub, Wi-Fi. GHz Di bawah Sumber Daya Biner, pilih pembaruan firmware modul Wi-Fi Inventek ISM 43362 (baca file readme untuk instruksi).

Sebelum memulai, Anda harus mengonfigurasi AWS IoT, unduhan FreeRTOS Anda, dan Wi-Fi untuk menghubungkan perangkat Anda ke Cloud. AWS Lihat Langkah pertama untuk instruksi. Dalam tutorial ini, jalur ke direktori unduhan FreeRTOS disebut sebagai. *`freertos`*

Gambaran Umum

Tutorial ini berisi petunjuk untuk langkah-langkah memulai berikut:

1. Menginstal perangkat lunak pada mesin host untuk mengembangkan dan men-debug aplikasi tertanam untuk papan mikrokontroler Anda.
2. Menyusun silang aplikasi demo FreeRTOS ke gambar biner.
3. Memuat gambar biner aplikasi ke papan Anda, dan kemudian menjalankan aplikasi.

Siapkan lingkungan pengembangan Anda

Instal Meja Kerja Sistem untuk STM32

1. Jelajahi Open STM32 .org.
2. Daftar di STM32 halaman web Terbuka. Anda harus masuk untuk mengunduh System Workbench.
3. Browse ke System Workbench untuk STM32 installer untuk men-download dan menginstal System Workbench.

Jika Anda mengalami masalah selama instalasi, lihat FAQs di situs web System Workbench.

Bangun dan jalankan proyek demo FreeRTOS

Impor demo FreeRTOS ke Meja Kerja Sistem STM32

1. Buka STM32 System Workbench dan masukkan nama untuk ruang kerja baru.

2. Dari menu File, pilih Impor. Expand General, pilih Existing Projects into Workspace, lalu pilih Next.
3. Di Pilih Direktori Root, masukkan`projects/st/stm32l475_discovery/ac6/aws_demos`.
4. Proyek `aws_demos` harus dipilih secara default.
5. Pilih Selesai untuk mengimpor proyek ke STM32 System Workbench.
6. Dari menu Project, pilih Build All. Konfirmasikan proyek dikompilasi tanpa kesalahan.

Memantau pesan MQTT di cloud

Sebelum menjalankan proyek demo FreeRTOS, Anda dapat mengatur klien MQTT di konsol untuk memantau pesan AWS IoT yang dikirim perangkat Anda ke Cloud. AWS

Untuk berlangganan topik MQTT dengan klien MQTT AWS IoT

1. Masuk ke konsol AWS IoT tersebut.
2. Di panel navigasi, pilih Uji, lalu pilih klien pengujian MQTT untuk membuka klien MQTT.
3. Dalam Subscription topic, masukkan***your-thing-name*****/example/topic**, lalu pilih Subscribe to topic.

Ketika proyek demo berhasil berjalan di perangkat Anda, Anda melihat “Hello World!” dikirim beberapa kali ke topik yang Anda berlangganan.

Jalankan proyek demo FreeRTOS

1. Gunakan kabel USB untuk menghubungkan STMicroelectronics STM32 L4 Discovery Kit IoT Node Anda ke komputer Anda. (Periksa dokumentasi pabrikan yang disertakan dengan papan Anda untuk port USB yang benar untuk digunakan.)
2. Dari Project Explorer, klik kanan`aws_demos`, pilih Debug As, dan kemudian pilih Ac6 STM32 C/C++ Application.

   Jika terjadi kesalahan debug saat pertama kali sesi debug diluncurkan, ikuti langkah-langkah berikut:

   1. Di STM32 System Workbench, dari menu Run, pilih Konfigurasi Debug.
   2. Pilih aws_demos Debug. (Anda mungkin perlu memperluas STM32 Debugging Ac6.)
   3. Pilih tab Debugger.

4. Di Skrip Konfigurasi, pilih Tampilkan Opsi Generator.
5. Dalam Pengaturan Mode, atur Reset Mode ke Reset Sistem Perangkat Lunak. Pilih Terapkan, lalu pilih Debug.

3. Ketika debugger berhenti di breakpoint di**`main()`**, dari menu Run, pilih Lanjutkan.

Menggunakan CMake dengan FreeRTOS

Jika Anda memilih untuk tidak menggunakan IDE untuk pengembangan FreeRTOS, Anda dapat CMake menggunakan alternatif untuk membangun dan menjalankan aplikasi demo atau aplikasi yang telah Anda kembangkan menggunakan editor kode pihak ketiga dan alat debugging.

Pertama buat folder untuk berisi file build yang dihasilkan (*`build-folder`*).

Gunakan perintah berikut untuk menghasilkan file build:

```
cmake -DVENDOR=st -DBOARD=stm32l475_discovery -DCOMPILER=arm-gcc -S freertos -B build-
folder
```

Jika tidak `arm-none-eabi-gcc` ada di jalur shell Anda, Anda juga perlu mengatur `AFR_TOOLCHAIN_PATH` CMake variabel. Sebagai contoh:

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

Untuk informasi lebih lanjut tentang menggunakan CMake dengan FreeRTOS, lihat. Menggunakan CMake dengan FreeRTOS

Pemecahan Masalah

Jika Anda melihat yang berikut dalam output UART dari aplikasi demo, Anda perlu memperbarui firmware modul Wi-Fi:

```
[Tmr Svc] WiFi firmware version is: xxxxxxxxxxxx
[Tmr Svc] [WARN] WiFi firmware needs to be updated.
```

Untuk mengunduh firmware Wi-Fi terbaru, lihat STM32L4 Discovery kit IoT node, nirkabel berdaya rendah, Bluetooth Low Energy, NFC, Sub, Wi-Fi. GHz Di Sumber Daya Biner, pilih tautan unduhan untuk pembaruan firmware modul Wi-Fi Inventek ISM 43362.

Untuk informasi pemecahan masalah umum tentang Memulai FreeRTOS, lihat. Pemecahan masalah memulai

Memulai dengan Texas Instruments CC322 0SF-LAUNCHXL

> **Important**
>
> Integrasi referensi ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-Freertos yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Tutorial ini memberikan petunjuk untuk memulai dengan Texas Instruments CC322 0SF-LAUNCHXL. Jika Anda tidak memiliki Kit Pengembangan Texas Instruments (TI) CC322 0SF-LAUNCHXL, kunjungi Katalog Perangkat AWS Mitra untuk membelinya dari mitra kami.

Sebelum memulai, Anda harus mengonfigurasi AWS IoT dan mengunduh FreeRTOS Anda untuk menghubungkan perangkat Anda ke Cloud. AWS Lihat Langkah pertama untuk instruksi. Dalam tutorial ini, jalur ke direktori unduhan FreeRTOS disebut sebagai. *`freertos`*

Gambaran Umum

Tutorial ini berisi petunjuk untuk langkah-langkah memulai berikut:

1. Menginstal perangkat lunak pada mesin host untuk mengembangkan dan men-debug aplikasi tertanam untuk papan mikrokontroler Anda.
2. Menyusun silang aplikasi demo FreeRTOS ke gambar biner.
3. Memuat gambar biner aplikasi ke papan Anda, dan kemudian menjalankan aplikasi.

Siapkan lingkungan pengembangan Anda

Ikuti langkah-langkah di bawah ini untuk mengatur lingkungan pengembangan Anda untuk memulai dengan FreeRTOS.

Perhatikan bahwa FreeRTOS mendukung IDEs dua untuk Kit Pengembangan CC322 TI 0SF-LAUNCHXL: Code Composer Studio dan IAR Embedded Workbench versi 8.32. Anda dapat menggunakan salah satu IDE untuk memulai.

Instal Code Composer Studio

1. Jelajahi TI Code Composer Studio.
2. Unduh penginstal offline untuk platform mesin host Anda (Windows, macOS, atau Linux 64-bit).
3. Buka zip dan jalankan penginstal offline. Ikuti petunjuknya.
4. Untuk Keluarga Produk untuk Diinstal, pilih SimpleLink Wi-Fi CC32xx Wireless MCUs.
5. Di halaman berikutnya, terima pengaturan default untuk probe debugging, lalu pilih Selesai.

Jika Anda mengalami masalah saat menginstal Code Composer Studio, lihat TI Development Tools Support, Code Composer Studio FAQs, dan Troubleshooting CCS.

Instal Meja Kerja Tertanam IAR

1. Unduh dan jalankan penginstal Windows untuk versi 8.32 dari IAR Embedded Workbench untuk ARM. Di driver probe Debug, pastikan TI XDS dipilih.
2. Selesaikan instalasi dan luncurkan program. Pada halaman Wisaya Lisensi, pilih Daftar dengan Sistem IAR untuk mendapatkan lisensi evaluasi, atau gunakan lisensi IAR Anda sendiri.

Instal SimpleLink CC322 0 SDK

Instal SimpleLink CC3220 SDK. SimpleLink Wi-Fi CC322 0 SDK berisi driver untuk MCU CC322 0SF yang dapat diprogram, lebih dari 40 aplikasi sampel, dan dokumentasi yang diperlukan untuk menggunakan sampel.

Instal Uniflash

Instal Uniflash. CCS Uniflash adalah alat mandiri yang digunakan untuk memprogram memori flash on-chip pada TI. MCUs Uniflash memiliki GUI, baris perintah, dan antarmuka scripting.

Instal paket layanan terbaru

1. Pada TI CC322 0SF-LAUNCHXL Anda, letakkan jumper SOP di set tengah pin (posisi = 1) dan setel ulang papan.
2. Mulai Uniflash. Jika LaunchPad papan CC322 0SF Anda muncul di bawah Perangkat Terdeteksi, pilih Mulai. Jika papan Anda tidak terdeteksi, pilih CC3220SF-LAUNCHXL dari daftar papan di bawah Konfigurasi Baru, lalu pilih Mulai Pembuat Gambar.
3. Pilih Proyek Baru.

4. Pada halaman Mulai proyek baru, masukkan nama untuk proyek Anda. Untuk Jenis Perangkat, pilih CC3220SF. Untuk Mode Perangkat, pilih Kembangkan, lalu pilih Buat Proyek.
5. Di sisi kanan jendela aplikasi Uniflash, pilih Connect.
6. Dari kolom kiri, pilih Advanced, Files, dan kemudian Service Pack.
7. Pilih Browse, lalu navigasikan ke tempat Anda menginstal SDK CC322 0SF SimpleLink . Paket layanan terletak di`ti/simplelink_cc32xx_sdk_`*`VERSION`*`/tools/cc32xx_tools/servicepack-cc3x20/sp_`*`VERSION`*`.bin`.
8. Pilih tombol Burn

   ( ),

   lalu pilih Program Image (Create & Program) untuk menginstal paket layanan. Ingatlah untuk mengganti jumper SOP kembali ke posisi 0 dan mengatur ulang papan.

Konfigurasikan penyediaan Wi-Fi

Untuk mengonfigurasi pengaturan Wi-Fi untuk papan Anda, lakukan salah satu hal berikut:

- Konfigurasikan aplikasi demo FreeRTOS yang dijelaskan dalam. Mengkonfigurasi demo FreeRTOS
- Gunakan SmartConfigdari Texas Instruments.

Bangun dan jalankan proyek demo FreeRTOS

Bangun dan jalankan proyek demo FreeRTOS di TI Code Composer

Untuk mengimpor demo FreeRTOS ke TI Code Composer

1. Buka TI Code Composer, dan pilih OK untuk menerima nama workspace default.
2. Pada halaman Memulai, pilih Impor Proyek.
3. Di Pilih direktori pencarian, masukkan. `projects/ti/cc3220_launchpad/ccs/aws_demos` Proyek `aws_demos` harus dipilih secara default. Untuk mengimpor proyek ke TI Code Composer, pilih Finish.
4. Di Project Explorer, klik dua kali aws_demos untuk membuat proyek aktif.
5. Dari Project, pilih Build Project untuk memastikan proyek dikompilasi tanpa kesalahan atau peringatan.

Untuk menjalankan demo FreeRTOS di TI Code Composer

1. Pastikan jumper Sense On Power (SOP) pada Texas Instruments CC322 0SF-LAUNCHXL Anda berada di posisi 0. Untuk informasi selengkapnya, lihat SimpleLink Wi-Fi CC3x2 0, Panduan Pengguna Prosesor CC3x3x Jaringan.
2. Gunakan kabel USB untuk menyambungkan Texas Instruments CC322 0SF-LAUNCHXL ke komputer.
3. Di project explorer, pastikan yang `CC3220SF.ccxml` dipilih sebagai konfigurasi target aktif. Untuk membuatnya aktif, klik kanan file dan pilih Set as active target configuration.
4. Di TI Code Composer, dari Run, pilih Debug.
5. Ketika debugger berhenti di breakpoint**`main()`**, buka menu Run, dan pilih Lanjutkan.

Memantau pesan MQTT di cloud

Sebelum menjalankan proyek demo FreeRTOS, Anda dapat mengatur klien MQTT di konsol untuk memantau pesan AWS IoT yang dikirim perangkat Anda ke Cloud. AWS

Untuk berlangganan topik MQTT dengan klien MQTT AWS IoT

1. Masuk ke konsol AWS IoT tersebut.
2. Di panel navigasi, pilih Uji, lalu pilih klien pengujian MQTT untuk membuka klien MQTT.
3. Dalam Subscription topic, masukkan***`your-thing-name`***`/example/topic`, lalu pilih Subscribe to topic.

Ketika proyek demo berhasil berjalan di perangkat Anda, Anda melihat “Hello World!” dikirim beberapa kali ke topik yang Anda berlangganan.

Bangun dan jalankan proyek demo FreeRTOS di IAR Embedded Workbench

Untuk mengimpor demo FreeRTOS ke IAR Embedded Workbench

1. Buka IAR Embedded Workbench, pilih File, lalu pilih Open Workspace.
2. Arahkan ke**`projects/ti/cc3220_launchpad/iar/aws_demos`**, pilih aws_demos.eww, lalu pilih OK.
3. Klik kanan nama proyek (`aws_demos`), lalu pilih Make.

Untuk menjalankan demo FreeRTOS di IAR Embedded Workbench

1. Pastikan jumper Sense On Power (SOP) pada Texas Instruments CC322 0SF-LAUNCHXL Anda berada di posisi 0. Untuk informasi selengkapnya, lihat SimpleLink Wi-Fi CC3x2 0, Panduan Pengguna Prosesor CC3x3x Jaringan.
2. Gunakan kabel USB untuk menyambungkan Texas Instruments CC322 0SF-LAUNCHXL ke komputer.
3. Membangun kembali proyek Anda.

   Untuk membangun kembali proyek, dari menu Project, pilih Make.
4. Dari menu Project, pilih Download dan Debug. Anda dapat mengabaikan “Peringatan: Gagal menginisialisasi EnergyTrace,” jika ditampilkan. Untuk informasi selengkapnya EnergyTrace, lihat EnergyTrace Teknologi MSP.
5. Ketika debugger berhenti di breakpoint**`main()`**, buka menu Debug, dan pilih Go.

Menggunakan CMake dengan FreeRTOS

Jika Anda memilih untuk tidak menggunakan IDE untuk pengembangan FreeRTOS, Anda dapat CMake menggunakan alternatif untuk membangun dan menjalankan aplikasi demo atau aplikasi yang telah Anda kembangkan menggunakan editor kode pihak ketiga dan alat debugging.

Untuk membangun demo FreeRTOS dengan CMake

1. Buat folder untuk berisi file build yang dihasilkan (*`build-folder`*).
2. Pastikan jalur pencarian Anda (variabel lingkungan $PATH) berisi folder tempat biner kompiler TI CGT berada (misalnya). `C:\ti\ccs910\ccs\tools\compiler\ti-cgt-arm_18.12.2.LTS\bin`

   Jika Anda menggunakan kompiler TI ARM dengan papan TI Anda, gunakan perintah berikut untuk menghasilkan file build dari kode sumber:

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

Untuk informasi selengkapnya, lihat Menggunakan CMake dengan FreeRTOS.

Pemecahan Masalah

Jika Anda tidak melihat pesan di klien MQTT AWS IoT konsol, Anda mungkin perlu mengonfigurasi pengaturan debug untuk papan.

Untuk mengonfigurasi pengaturan debug untuk papan TI

1. Di Code Composer, di Project Explorer, pilih aws_demos.
2. Dari menu Jalankan, pilih Konfigurasi Debug.
3. Di panel navigasi, pilih aws_demos.
4. Pada tab Target, di bawah Opsi Koneksi, pilih Setel ulang target pada sambungan.
5. Pilih Terapkan, lalu pilih Tutup.

Jika langkah-langkah ini tidak berhasil, lihat output program di terminal serial. Anda akan melihat beberapa teks yang menunjukkan sumber masalah.

Untuk informasi pemecahan masalah umum tentang Memulai FreeRTOS, lihat. Pemecahan masalah memulai

Memulai dengan Simulator Perangkat Windows

Tutorial ini memberikan instruksi untuk memulai dengan FreeRTOS Windows Device Simulator.

Sebelum memulai, Anda harus mengonfigurasi AWS IoT dan mengunduh FreeRTOS Anda untuk menghubungkan perangkat Anda ke Cloud. AWS Lihat Langkah pertama untuk instruksi. Dalam tutorial ini, jalur ke direktori unduhan FreeRTOS disebut sebagai. *`freertos`*

FreeRTOS dirilis sebagai file zip yang berisi pustaka FreeRTOS dan contoh aplikasi untuk platform yang Anda tentukan. Untuk menjalankan sampel pada mesin Windows, unduh pustaka dan sampel yang di-porting untuk dijalankan di Windows. Kumpulan file ini disebut sebagai simulator FreeRTOS untuk Windows.

> **Note**
>
> Tutorial ini tidak dapat berhasil dijalankan pada instance Amazon EC2 Windows.

Siapkan lingkungan pengembangan Anda

1. Instal Npcap versi terbaru. Pilih "Mode yang WinPcap kompatibel dengan API” selama instalasi.

2. Instal Microsoft Visual Studio.

   Visual Studio versi 2017 dan 2019 diketahui berfungsi. Semua edisi versi Visual Studio ini didukung (Komunitas, Profesional, atau Perusahaan).

   Selain IDE, instal pengembangan Desktop dengan komponen C++.

   Instal SDK Windows 10 terbaru. Anda dapat memilih ini di bawah bagian Opsional pengembangan Desktop dengan komponen C ++.
3. Pastikan Anda memiliki koneksi Ethernet terprogram aktif.
4. (Opsional) Jika Anda ingin menggunakan sistem build CMake berbasis untuk membangun proyek FreeRTOS Anda, instal versi terbaru. CMake FreeRTOS CMake membutuhkan versi 3.13 atau yang lebih baru.

Memantau pesan MQTT di cloud

Sebelum menjalankan proyek demo FreeRTOS, Anda dapat mengatur klien MQTT di konsol untuk memantau pesan AWS IoT yang dikirim perangkat Anda ke Cloud. AWS

Untuk berlangganan topik MQTT dengan klien MQTT AWS IoT

1. Masuk ke konsol AWS IoT tersebut.
2. Di panel navigasi, pilih Uji, lalu pilih klien pengujian MQTT untuk membuka klien MQTT.
3. Dalam Subscription topic, masukkan***your-thing-name*****/example/topic**, lalu pilih Subscribe to topic.

Ketika proyek demo berhasil berjalan di perangkat Anda, Anda melihat “Hello World!” dikirim beberapa kali ke topik yang Anda berlangganan.

Bangun dan jalankan proyek demo FreeRTOS

Anda dapat menggunakan Visual Studio atau CMake untuk membangun proyek FreeRTOS.

Membangun dan menjalankan proyek demo FreeRTOS dengan Visual Studio IDE

1. Muat proyek ke Visual Studio.

   Di Visual Studio, dari menu File, pilih Buka. Pilih File/Solusi, navigasikan ke **projects/pc/windows/visual_studio/aws_demos/aws_demos.sln** file, lalu pilih Buka.

2. Targetkan ulang proyek demo.

   Proyek demo yang disediakan tergantung pada Windows SDK, tetapi tidak memiliki versi Windows SDK yang ditentukan. Secara default, IDE mungkin mencoba membangun demo dengan versi SDK yang tidak ada di mesin Anda. Untuk mengatur versi Windows SDK, klik kanan `aws_demos` dan kemudian pilih Retarget Projects. Ini membuka jendela Review Solution Actions. Pilih versi Windows SDK yang ada di mesin Anda (nilai awal di dropdown baik-baik saja), lalu pilih OK.
3. Membangun dan menjalankan proyek.

   Dari menu Build, pilih Build Solution, dan pastikan solusi dibuat tanpa kesalahan atau peringatan. Pilih Debug, Mulai Debug untuk menjalankan proyek. Pada proses pertama, Anda harus memilih antarmuka jaringan.

Membangun dan menjalankan proyek demo FreeRTOS dengan CMake

Kami menyarankan Anda menggunakan CMake GUI alih-alih alat baris CMake perintah untuk membangun proyek demo untuk Windows Simulator.

Setelah Anda menginstal CMake, buka CMake GUI. Di Windows, Anda dapat menemukan ini dari menu Start di bawah CMake, CMake (cmake-gui).

1. Atur direktori kode sumber FreeRTOS.

   Di GUI, atur direktori kode sumber FreeRTOS *`freertos`* () untuk Di mana kode sumbernya.

   Tetapkan *`freertos`*`/build` untuk Dimana membangun binari.
2. Konfigurasikan CMake Proyek.

   Di CMake GUI, pilih Add Entry, dan pada jendela Add Cache Entry, atur nilai berikut:

   Nama

   AFR_BOARD

   Tipe

   STRING

   Nilai

   pc.windows

Deskripsi

(Opsional)

3. Pilih Konfigurasikan Jika CMake meminta Anda untuk membuat direktori build, pilih Ya, lalu pilih generator di bawah Tentukan generator untuk proyek ini. Kami merekomendasikan menggunakan Visual Studio sebagai generator, tetapi Ninja juga didukung. (Perhatikan bahwa saat menggunakan Visual Studio 2019, platform harus disetel ke Win32 alih-alih pengaturan defaultnya.) Jaga agar opsi generator lainnya tidak berubah dan pilih Selesai.
4. Menghasilkan dan Membuka CMake Proyek.

   Setelah Anda mengonfigurasi proyek, CMake GUI menunjukkan semua opsi yang tersedia untuk proyek yang dihasilkan. Untuk keperluan tutorial ini, Anda dapat meninggalkan opsi pada nilai defaultnya.

   Pilih Generate untuk membuat solusi Visual Studio, lalu pilih Open Project untuk membuka proyek di Visual Studio.

   Di Visual Studio, klik kanan `aws_demos` proyek dan pilih Set as StartUp Project. Ini memungkinkan Anda untuk membangun dan menjalankan proyek. Pada proses pertama, Anda harus memilih antarmuka jaringan.

Untuk informasi lebih lanjut tentang menggunakan CMake dengan FreeRTOS, lihat. Menggunakan CMake dengan FreeRTOS

Konfigurasikan antarmuka jaringan Anda

Pada proses pertama proyek demo, Anda harus memilih antarmuka jaringan yang akan digunakan. Program ini menghitung antarmuka jaringan Anda. Temukan nomor untuk antarmuka Ethernet terprogram Anda. Outputnya akan terlihat seperti ini:

```
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.
```

Setelah Anda mengidentifikasi nomor untuk antarmuka Ethernet terprogram Anda, tutup jendela aplikasi. Pada contoh sebelumnya, nomor yang akan digunakan adalah1.

Buka `FreeRTOSConfig.h` dan atur `configNETWORK_INTERFACE_TO_USE` ke nomor yang sesuai dengan antarmuka jaringan terprogram Anda.

> **Important**
>
> Hanya antarmuka Ethernet yang didukung. Wi-Fi tidak didukung.

Pemecahan Masalah

Memecahkan masalah umum pada Windows

Anda mungkin mengalami kesalahan berikut saat mencoba membangun proyek demo dengan Visual Studio:

```
Error "The Windows SDK version X.Y was not found" when building the provided Visual
 Studio solution.
```

Proyek harus ditargetkan ke versi Windows SDK yang ada di komputer Anda.

Untuk informasi pemecahan masalah umum tentang memulai dengan FreeRTOS, lihat. Pemecahan masalah memulai

Memulai dengan Xilinx Avnet MicroZed Industrial IoT Kit

> **Important**
>
> Integrasi referensi ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-Freertos yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Tutorial ini memberikan instruksi untuk memulai dengan Xilinx Avnet MicroZed Industrial IoT Kit. Jika Anda tidak memiliki Kit IoT MicroZed Industri Xilinx Avnet, kunjungi Katalog Perangkat Mitra untuk AWS membelinya dari mitra kami.

Sebelum memulai, Anda harus mengonfigurasi AWS IoT dan mengunduh FreeRTOS Anda untuk menghubungkan perangkat Anda ke Cloud. AWS Lihat Langkah pertama untuk instruksi. Dalam tutorial ini, jalur ke direktori unduhan FreeRTOS disebut sebagai. *freertos*

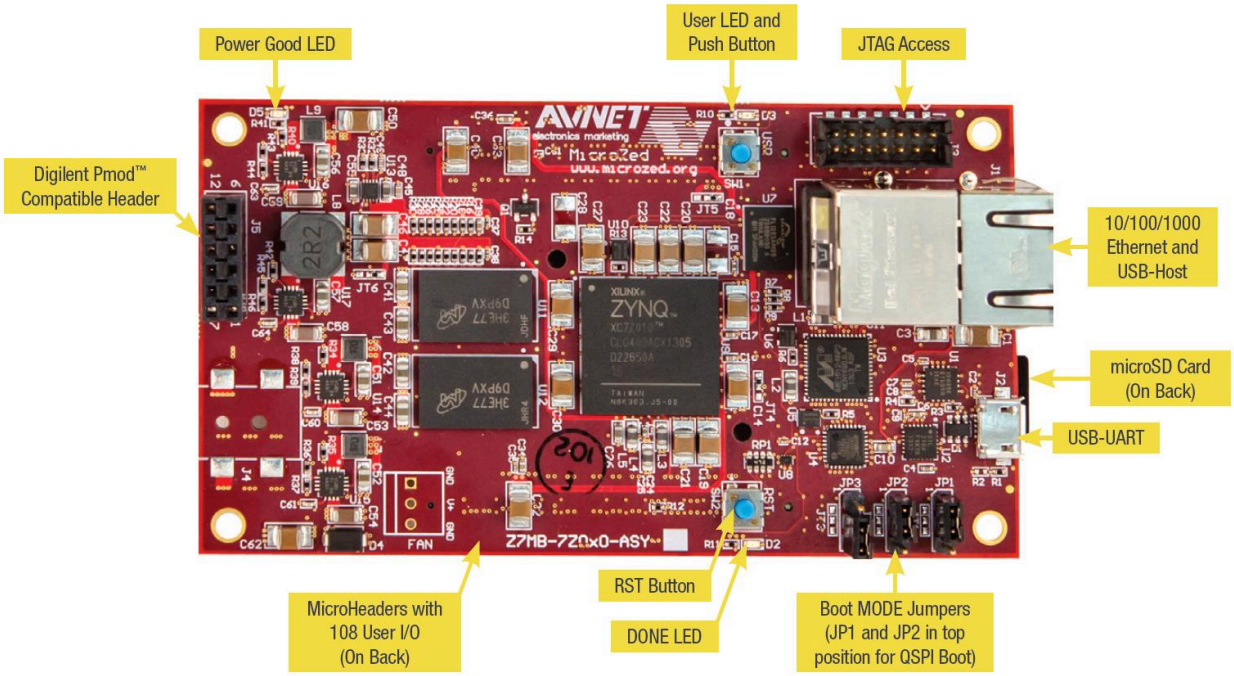
Gambaran Umum

Tutorial ini berisi petunjuk untuk langkah-langkah memulai berikut:

1. Menghubungkan papan Anda ke mesin host.
2. Menginstal perangkat lunak pada mesin host untuk mengembangkan dan men-debug aplikasi tertanam untuk papan mikrokontroler Anda.
3. Menyusun silang aplikasi demo FreeRTOS ke gambar biner.
4. Memuat gambar biner aplikasi ke papan Anda, dan kemudian menjalankan aplikasi.

Siapkan perangkat MicroZed keras

Diagram berikut mungkin berguna ketika Anda mengatur perangkat MicroZed keras:



Untuk mengatur MicroZed papan

1. Hubungkan komputer Anda ke port USB-UART di papan Anda. MicroZed
2. Hubungkan komputer Anda ke port JTAG Access di MicroZed papan Anda.
3. Hubungkan router atau port Ethernet yang terhubung ke internet ke port Ethernet dan USB-host di papan Anda. MicroZed

Siapkan lingkungan pengembangan Anda

Untuk mengatur konfigurasi FreeRTOS untuk MicroZed kit, Anda harus menggunakan Xilinx Software Development Kit (XSDK). XSDK didukung pada Windows dan Linux.

Unduh dan instal XSDK

Untuk menginstal perangkat lunak Xilinx, Anda memerlukan akun Xilinx gratis.

Untuk mengunduh XSDK

1. Buka halaman unduhan WebInstall Klien Standalone Kit Pengembangan Perangkat Lunak.
2. Pilih opsi yang sesuai untuk sistem operasi Anda.
3. Anda diarahkan ke halaman masuk Xilinx.

   Jika Anda memiliki akun dengan Xilinx, masukkan kredensyal masuk Anda, lalu pilih Masuk.

   Jika Anda tidak memiliki akun, pilih Buat akun Anda. Setelah mendaftar, Anda akan menerima email dengan tautan untuk mengaktifkan akun Xilinx Anda.
4. Pada halaman Verifikasi Nama dan Alamat, masukkan informasi Anda, lalu pilih Berikutnya. Unduhan harus siap untuk dimulai.
5. Simpan file `Xilinx_SDK_version_os`.

Untuk menginstal XSDK

1. Buka file `Xilinx_SDK_version_os`.
2. Di Select Edition to Install, pilih Xilinx Software Development Kit (XSDK) dan kemudian pilih Berikutnya.
3. Pada halaman berikut dari panduan instalasi, di bawah Opsi Instalasi, pilih Instal Driver Kabel dan kemudian pilih Berikutnya.

Jika komputer Anda tidak mendeteksi koneksi USB-UART, instal driver CP21 0x USB-to-UART Bridge VCP secara manual. MicroZed Untuk petunjuk, lihat Panduan USB-to-UART Instalasi Silicon Labs CP21 0x.

Untuk informasi selengkapnya tentang XSDK, lihat Memulai dengan Xilinx SDK di situs web Xilink.

Memantau pesan MQTT di cloud

Sebelum menjalankan proyek demo FreeRTOS, Anda dapat mengatur klien MQTT di konsol untuk memantau pesan AWS IoT yang dikirim perangkat Anda ke Cloud. AWS

Untuk berlangganan topik MQTT dengan klien MQTT AWS IoT

1. Masuk ke konsol AWS IoT tersebut.
2. Di panel navigasi, pilih Uji, lalu pilih klien pengujian MQTT untuk membuka klien MQTT.
3. Dalam Subscription topic, masukkan***your-thing-name*****/example/topic**, lalu pilih Subscribe to topic.

Bangun dan jalankan proyek demo FreeRTOS

Buka demo FreeRTOS di IDE XSDK

1. Luncurkan IDE XSDK dengan direktori ruang kerja yang disetel ke. *freertos*`/projects/xilinx/microzed/xsdk`
2. Tutup halaman selamat datang. Dari menu, pilih Project, lalu hapus Build Automatically.
3. Dari menu, pilih File, lalu pilih Impor.
4. Pada halaman Pilih, perluas Umum, pilih Proyek yang Ada ke Ruang Kerja, lalu pilih Berikutnya.
5. Pada halaman Impor Proyek, pilih Pilih direktori root, lalu masukkan direktori root proyek demo Anda:*freertos*`/projects/xilinx/microzed/xsdk/aws_demos`. Untuk menelusuri direktori, pilih Browse.

   Setelah Anda menentukan direktori root, proyek di direktori tersebut akan muncul di halaman Impor Proyek. Semua proyek yang tersedia dipilih secara default.

   > **Note**
   >
   > Jika Anda melihat peringatan di bagian atas halaman Impor Proyek (“Beberapa proyek tidak dapat diimpor karena sudah ada di ruang kerja.”) Anda dapat mengabaikannya.

6. Dengan semua proyek yang dipilih, pilih Selesai.
7. Jika Anda tidak melihat`aws_bsp`,`fsbl`, dan `MicroZed_hw_platform_0` proyek di panel proyek, ulangi langkah sebelumnya mulai dari #3 tetapi dengan direktori root disetel ke*`freertos`*`/vendors/xilinx`, dan import`aws_bsp`,`fsbl`, dan`MicroZed_hw_platform_0`.
8. Dari menu, pilih Jendela, lalu pilih Preferensi.
9. Di panel navigasi, perluas Run/Debug, pilih String Substitution, dan kemudian pilih New.
10. Dalam Variabel Substitusi String Baru, untuk Nama, masukkan**`AFR_ROOT`**. Untuk Nilai, masukkan jalur root dari file*`freertos`*`/projects/xilinx/microzed/xsdk/aws_demos`. Pilih OK, lalu pilih OK untuk menyimpan variabel dan menutup Preferensi.

Bangun proyek demo FreeRTOS

1. Di XSDK IDE, dari menu, pilih Project, lalu pilih Clean.
2. Di Bersih, biarkan opsi pada nilai defaultnya, lalu pilih OK. XSDK membersihkan dan membangun semua proyek, dan kemudian menghasilkan file. `.elf`

> **Note**
>
> Untuk membangun semua proyek tanpa membersihkannya, pilih Project, lalu pilih Build All.
> Untuk membangun proyek individual, pilih proyek yang ingin Anda bangun, pilih Project, lalu pilih Build Project.

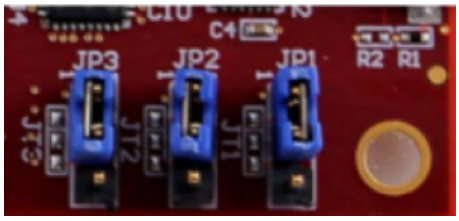
Hasilkan image boot untuk proyek demo FreeRTOS

1. Di XSDK IDE, klik kanan aws_demos, lalu pilih Create Boot Image.
2. Di Buat Gambar Boot, pilih Buat file BIF baru.
3. Di sebelah jalur file Output BIF, pilih Browse, lalu pilih `aws_demos.bif` located at*`<freertos>`*`/vendors/xilinx/microzed/aws_demos/aws_demos.bif`.
4. Pilih Tambahkan.
5. Pada Tambahkan partisi gambar boot baru, di sebelah jalur File, pilih Browse, lalu pilih`fsbl.elf`, terletak di`vendors/xilinx/fsbl/Debug/fsbl.elf`.
6. Untuk jenis partisi, pilih bootloader, dan kemudian pilih OK.

7. Pada Create Boot Image, pilih Create Image. Pada Override Files, pilih OK untuk menimpa yang ada `aws_demos.bif` dan menghasilkan `BOOT.bin` file di. `projects/xilinx/microzed/xsdk/aws_demos/BOOT.bin`

Debugging JTAG

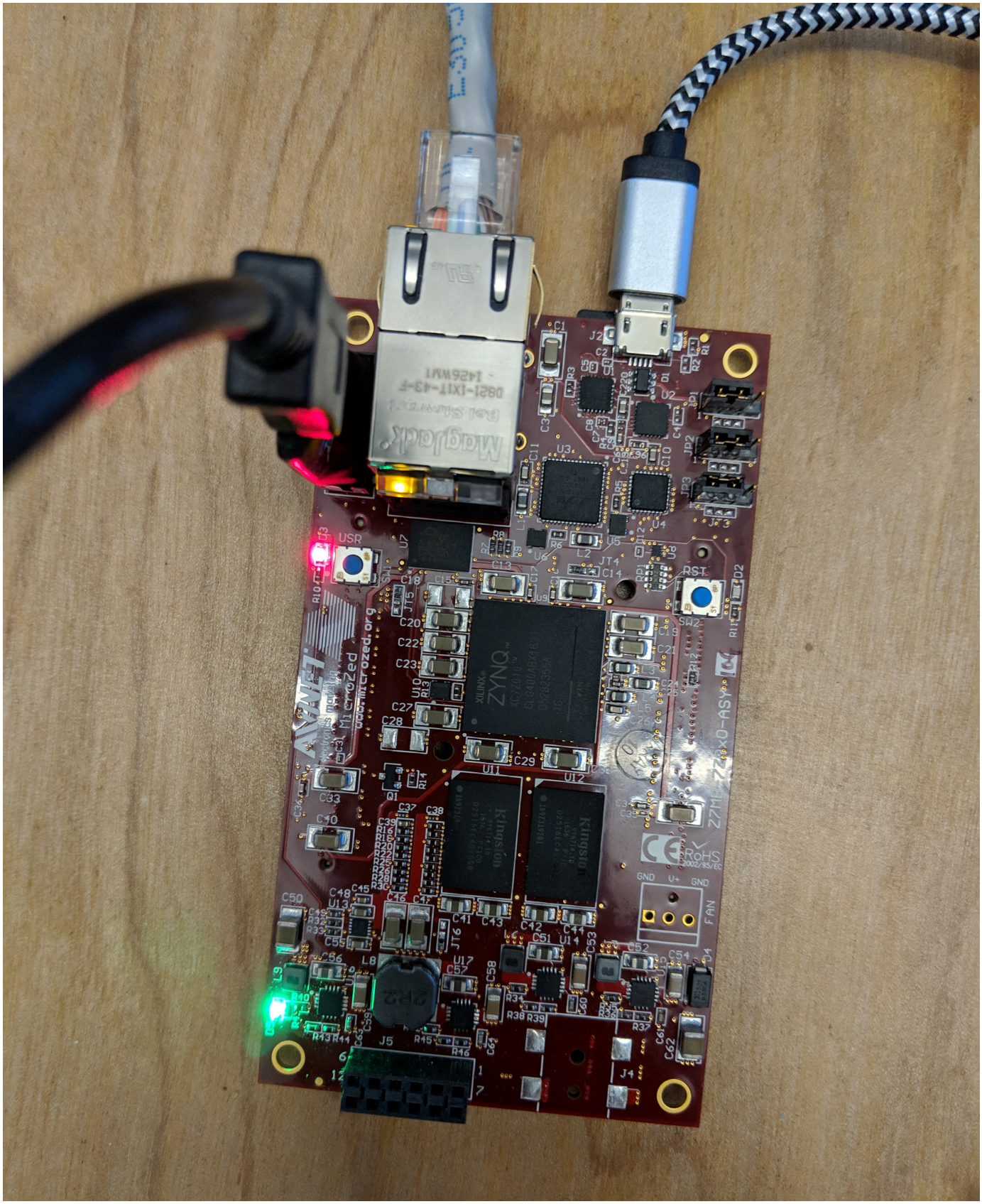
1. Atur jumper mode boot MicroZed papan Anda ke mode boot JTAG.



2. Masukkan kartu microSD Anda ke dalam slot kartu microSD yang terletak tepat di bawah port USB-UART.

> **Note**
>
> Sebelum Anda men-debug, pastikan untuk mencadangkan konten apa pun yang Anda miliki di kartu microSD.

Papan Anda akan terlihat mirip dengan yang berikut ini:


USR
RST
ZYNQ
XILINX
Kingston
GND
V+
GND
FAN

3. Di XSDK IDE, klik kanan aws_demos, pilih Debug As, lalu pilih 1 Launch on System Hardware (System Debugger).
4. Ketika debugger berhenti di breakpoint**`main()`**, dari menu, pilih Jalankan, lalu pilih Lanjutkan.

> **Note**
>
> Pertama kali Anda menjalankan aplikasi, pasangan kunci sertifikat baru diimpor ke memori non-volatile. Untuk proses selanjutnya, Anda dapat berkomentar `vDevModeKeyProvisioning()` di `main.c` file sebelum Anda membangun kembali gambar dan `BOOT.bin` file. Ini mencegah penyalinan sertifikat dan kunci penyimpanan di setiap proses.

Anda dapat memilih untuk mem-boot MicroZed papan Anda dari kartu microSD atau dari flash QSPI untuk menjalankan proyek demo FreeRTOS. Lihat petunjuknya di Hasilkan image boot untuk proyek demo FreeRTOS dan Jalankan proyek demo FreeRTOS.

Jalankan proyek demo FreeRTOS

Untuk menjalankan proyek demo FreeRTOS, Anda dapat mem-boot MicroZed papan Anda dari kartu microSD atau dari flash QSPI.

Saat Anda mengatur MicroZed papan Anda untuk menjalankan proyek demo FreeRTOS, lihat diagram di. Siapkan perangkat MicroZed keras Pastikan Anda telah menghubungkan MicroZed papan Anda ke komputer Anda.

Boot proyek FreeRTOS dari kartu microSD

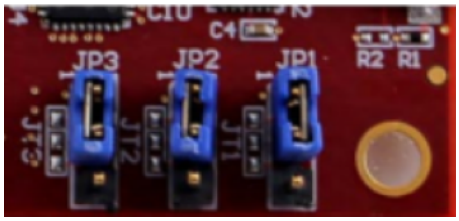
Format kartu microSD yang disediakan dengan Xilinx MicroZed Industrial IoT Kit.

1. Salin `BOOT.bin` file ke kartu microSD.
2. Masukkan kartu ke slot kartu microSD langsung di bawah port USB-UART.
3. Atur jumper mode MicroZed boot ke mode boot SD.

4. Tekan tombol RST untuk mengatur ulang perangkat dan mulai mem-boot aplikasi. Anda juga dapat mencabut kabel USB-UART dari port USB-UART, lalu memasukkan kembali kabel.

Boot proyek demo FreeRTOS dari flash QSPI

1. Atur jumper mode boot MicroZed papan Anda ke mode boot JTAG.



2. Verifikasi bahwa komputer Anda terhubung ke port Akses USB-UART dan JTAG. Lampu LED Power Good hijau harus diterangi.
3. Di XSDK IDE, dari menu, pilih Xilinx, dan kemudian pilih Program Flash.
4. Dalam Program Flash Memory, platform perangkat keras harus diisi secara otomatis. Untuk Koneksi, pilih server MicroZed perangkat keras Anda untuk menghubungkan papan Anda dengan komputer host Anda.

> **Note**
>
> Jika Anda menggunakan kabel Xilinx Smart Lync JTAG, Anda harus membuat server perangkat keras di XSDK IDE. Pilih Baru, lalu tentukan server Anda.

5. Di File Gambar, masukkan jalur direktori ke file `BOOT.bin` gambar Anda. Pilih Browse untuk menelusuri file sebagai gantinya.
6. Di Offset, masukkan**`0x0`**.
7. Di File FSBL, masukkan jalur direktori ke file Anda`fsbl.elf`. Pilih Browse untuk menelusuri file sebagai gantinya.
8. Pilih Program untuk memprogram papan Anda.
9. Setelah pemrograman QSPI selesai, lepaskan kabel USB-UART untuk mematikan papan.
10. Atur jumper mode boot MicroZed papan Anda ke mode boot QSPI.
11. Masukkan kartu Anda ke slot kartu microSD yang terletak tepat di bawah port USB-UART.

> **Note**
>
> Pastikan untuk mencadangkan konten apa pun yang Anda miliki di kartu microSD.

12. Tekan tombol RST untuk mengatur ulang perangkat dan mulai mem-boot aplikasi. Anda juga dapat mencabut kabel USB-UART dari port USB-UART, lalu memasukkan kembali kabel.

Pemecahan Masalah

Jika Anda menemukan kesalahan build yang terkait dengan jalur yang salah, cobalah untuk membersihkan dan membangun kembali proyek, seperti yang dijelaskan dalamBangun proyek demo FreeRTOS.

Jika Anda menggunakan Windows, pastikan Anda menggunakan garis miring maju saat Anda mengatur variabel substitusi string di Windows XSDK IDE.

Untuk informasi pemecahan masalah umum tentang Memulai FreeRTOS, lihat. Pemecahan masalah memulai

Langkah selanjutnya dengan FreeRTOS

> **Important**
>
> Halaman ini mengacu pada repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-Freertos yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Setelah Anda membangun, mem-flash, dan menjalankan proyek demo FreerTOS untuk papan Anda, Anda dapat mengunjungi situs web Freertos.org untuk mempelajari lebih lanjut tentang Membuat Proyek FreeRTOS Baru. Ada juga demo untuk banyak perpustakaan FreeRTOS yang menunjukkan bagaimana melakukan tugas-tugas penting, berinteraksi AWS IoT dengan layanan, dan kemampuan khusus papan program (seperti modem seluler). Untuk informasi selengkapnya, lihat halaman Kategori Perpustakaan FreeRTOS.

Situs web Freertos.org juga memiliki informasi mendalam tentang Freertos Kernel serta konsep dasar sistem operasi real-time. Untuk informasi selengkapnya, lihat halaman Dokumen Pengembang Kernel FreerTOS dan FreerTOS Kernel Secondary Docs.

Pembaruan FreeRTOS Over-the-Air

> **Note**
>
> Lihat AWS IoT Over-the-Air (OTA) di situs web FreeRTOS untuk informasi terbaru tentang Over-the-air melakukan pembaruan (OTA).

Over-the-air Pembaruan (OTA) memungkinkan Anda untuk menyebarkan pembaruan firmware ke satu atau beberapa perangkat di armada Anda. Meskipun pembaruan OTA dirancang untuk memperbarui firmware perangkat, Anda dapat menggunakannya untuk mengirim file apa pun ke satu atau beberapa perangkat yang terdaftar AWS IoT. Saat Anda mengirim pembaruan melalui udara, kami sarankan Anda menandatanganinya secara digital sehingga perangkat yang menerima file dapat memverifikasi bahwa mereka belum dirusak dalam perjalanan.

Anda dapat menggunakan Penandatanganan Kode AWS IoT untuk menandatangani file Anda, atau Anda dapat menandatangani file Anda dengan alat penandatanganan kode Anda sendiri.

Saat Anda membuat pembaruan OTA, akan Layanan Manajer Pembaruan OTA membuat AWS IoT pekerjaan untuk memberi tahu perangkat Anda bahwa pembaruan tersedia. Aplikasi demo OTA berjalan di perangkat Anda dan membuat tugas FreeRTOS yang berlangganan topik pemberitahuan AWS IoT untuk pekerjaan dan mendengarkan pesan pembaruan. Ketika pembaruan tersedia, Agen OTA menerbitkan permintaan ke AWS IoT dan menerima pembaruan menggunakan protokol HTTP atau MQTT, tergantung pada pengaturan yang Anda pilih. Agen OTA memeriksa tanda tangan digital dari file yang diunduh dan, jika file tersebut valid, instal pembaruan firmware. Jika Anda tidak menggunakan aplikasi demo FreerTOS OTA Update, Anda harus mengintegrasikan AWS IoT Perpustakaan Over the Air (OTA) ke dalam aplikasi Anda sendiri untuk mendapatkan kemampuan pembaruan firmware.

Pembaruan over-the-air FreeRTOS memungkinkan Anda untuk:

- Tanda tangani firmware secara digital sebelum penerapan.
- Terapkan gambar firmware baru ke satu perangkat, sekelompok perangkat, atau seluruh armada Anda.

- Terapkan firmware ke perangkat saat ditambahkan ke grup, reset, atau reprovisioned.
- Verifikasi keaslian dan integritas firmware baru setelah dikerahkan ke perangkat.
- Pantau kemajuan penerapan.
- Debug penerapan yang gagal.

Menandai sumber daya OTA

Untuk membantu mengelola sumber daya OTA, Anda dapat secara opsional menetapkan metadata Anda sendiri ke pembaruan dan aliran dalam bentuk tag. Tag memungkinkan Anda untuk mengkategorikan AWS IoT sumber daya Anda dengan cara yang berbeda (misalnya, berdasarkan tujuan, pemilik, atau lingkungan). Hal ini berguna jika Anda memiliki banyak sumber daya dengan jenis yang sama. Anda dapat dengan cepat mengidentifikasi sumber daya berdasarkan tag yang telah Anda tetapkan padanya.

Untuk informasi selengkapnya, lihat Menandai Sumber Daya AWS IoT Anda.

Prasyarat pembaruan OTA

Untuk menggunakan pembaruan over-the-air (OTA), lakukan hal berikut:

- Periksa Prasyarat untuk pembaruan OTA menggunakan HTTP atauPrasyarat untuk pembaruan OTA menggunakan MQTT.
- Buat bucket Amazon S3 untuk menyimpan pembaruan Anda.
- Buat peran layanan Pembaruan OTA.
- Buat kebijakan pengguna OTA.
- Buat sertifikat penandatanganan kode.
- Jika Anda menggunakan Penandatanganan Kode untuk AWS IoT,Berikan akses ke penandatanganan kode untuk AWS IoT.
- Unduh FreeRTOS dengan perpustakaan OTA.

Buat bucket Amazon S3 untuk menyimpan pembaruan Anda

File pembaruan OTA disimpan dalam ember Amazon S3.

Jika Anda menggunakan Penandatanganan Kode untuk AWS IoT, perintah yang Anda gunakan untuk membuat pekerjaan penandatanganan kode membutuhkan bucket sumber (tempat gambar

firmware yang tidak ditandatangani berada) dan bucket tujuan (tempat gambar firmware yang ditandatangani ditulis). Anda dapat menentukan bucket yang sama untuk sumber dan tujuan. Nama file diubah menjadi GUIDs sehingga file asli tidak ditimpa.

Untuk membuat bucket Amazon S3

1. Masuk ke konsol Amazon S3 di. https://console.aws.amazon.com/s3/
2. Pilih Buat bucket.
3. Masukkan nama ember.
4. Di bawah Pengaturan Bucket untuk Blokir Akses Publik tetap Blokir semua akses publik yang dipilih untuk menerima izin default.
5. Di bawah Bucket Versioning, pilih Aktifkan untuk menyimpan semua versi dalam bucket yang sama.
6. Pilih Buat bucket.

Untuk informasi selengkapnya tentang Amazon S3, lihat Panduan Pengguna Layanan Penyimpanan Sederhana Amazon.

Buat peran layanan Pembaruan OTA

Layanan Pembaruan OTA mengasumsikan peran ini untuk membuat dan mengelola pekerjaan pembaruan OTA atas nama Anda.

Untuk membuat peran layanan OTA

1. Masuk ke https://console.aws.amazon.com/iam/.
2. Dari panel navigasi, pilih Peran.
3. Pilih Buat peran.
4. Di Pilih jenis entitas tepercaya, pilih Layanan AWS .
5. Pilih IoT dari daftar layanan. AWS
6. Di bawah Pilih kasus penggunaan Anda, pilih IoT.
7. Pilih Berikutnya: Izin.
8. Pilih Berikutnya: Tanda.
9. Pilih Berikutnya: Tinjau.
10. Masukkan nama peran dan deskripsi, lalu pilih Buat peran.

Untuk informasi selengkapnya tentang peran IAM, lihat Peran IAM.

> **Important**
>
> Untuk mengatasi masalah keamanan wakil yang membingungkan, Anda harus mengikuti instruksi dalam AWS IoT Corepanduan ini.

Untuk menambahkan izin pembaruan OTA ke peran layanan OTA Anda

1. Di kotak pencarian di halaman konsol IAM, masukkan nama peran Anda, lalu pilih dari daftar.
2. Pilih Lampirkan kebijakan.
3. Di kotak Pencarian, masukkan "AmazonFreeRTOSOTAUpdate“, pilih AmazonFreeRTOSOTAUpdatedari daftar kebijakan yang difilter, lalu pilih Lampirkan kebijakan untuk melampirkan kebijakan ke peran layanan Anda.

Untuk menambahkan izin IAM yang diperlukan ke peran layanan OTA Anda

1. Di kotak pencarian di halaman konsol IAM, masukkan nama peran Anda, lalu pilih dari daftar.
2. Pilih Tambahkan kebijakan inline.
3. Pilih tab JSON.
4. Salin dan tempel dokumen kebijakan berikut ke dalam kotak teks:

```
{
    "Version": "2012-10-17",
    "Statement": [
      {
            "Effect": "Allow",
            "Action": [
                "iam:GetRole",
                "iam:PassRole"
            ],
            "Resource": "arn:aws:iam::your_account_id:role/your_role_name"
      }
    ]
}
```

Pastikan Anda mengganti *your_account_id* dengan ID AWS akun Anda, dan *your_role_name* dengan nama peran layanan OTA.

5. Pilih Tinjau kebijakan.
6. Masukkan nama untuk kebijakan tersebut, lalu pilih Buat kebijakan.

> **Note**
>
> Prosedur berikut tidak diperlukan jika nama bucket Amazon S3 Anda dimulai dengan “afr-ota”. Jika ya, kebijakan AWS terkelola `AmazonFreeRTOSOTAUpdate` sudah menyertakan izin yang diperlukan.

Untuk menambahkan izin Amazon S3 yang diperlukan ke peran layanan OTA Anda

1. Di kotak pencarian di halaman konsol IAM, masukkan nama peran Anda, lalu pilih dari daftar.
2. Pilih Tambahkan kebijakan inline.
3. Pilih tab JSON.
4. Salin dan tempel dokumen kebijakan berikut ke dalam kotak.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObjectVersion",
                "s3:GetObject",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::example-bucket/*"
            ]
        }
    ]
}
```

Kebijakan ini memberikan izin peran layanan OTA Anda untuk membaca objek Amazon S3. Pastikan Anda mengganti *example-bucket* dengan nama ember Anda.

5. Pilih Tinjau kebijakan.
6. Masukkan nama untuk kebijakan tersebut, lalu pilih Buat kebijakan.

Buat kebijakan pengguna OTA

Anda harus memberikan izin kepada pengguna Anda untuk melakukan over-the-air pembaruan. Pengguna Anda harus memiliki izin untuk:

- Akses bucket S3 tempat pembaruan firmware Anda disimpan.
- Akses sertifikat yang disimpan di AWS Certificate Manager.
- Akses fitur pengiriman AWS IoT file berbasis MQTT.
- Akses pembaruan FreerTOS OTA.
- Akses AWS IoT pekerjaan.
- Akses IAM.
- Penandatanganan Kode Akses untuk AWS IoT. Lihat Berikan akses ke penandatanganan kode untuk AWS IoT.
- Daftar platform perangkat keras FreeRTOS.
- Tag dan untag AWS IoT sumber daya.

Untuk memberikan izin yang diperlukan kepada pengguna, lihat Kebijakan IAM. Lihat juga Mengotorisasi pengguna dan layanan cloud untuk menggunakan AWS IoT Pekerjaan.

Untuk memberikan akses dan menambahkan izin bagi pengguna, grup, atau peran Anda:

- Pengguna dan grup di AWS IAM Identity Center:

  Buat rangkaian izin. Ikuti instruksi di Buat rangkaian izin di Panduan Pengguna AWS IAM Identity Center .
- Pengguna yang dikelola di IAM melalui penyedia identitas:

  Buat peran untuk federasi identitas. Ikuti instruksi dalam Buat peran untuk penyedia identitas pihak ketiga (federasi) dalam Panduan Pengguna IAM.
- Pengguna IAM:

- Buat peran yang dapat diambil pengguna Anda. Ikuti instruksi dalam Buat peran untuk pengguna IAM dalam Panduan Pengguna IAM.
- (Tidak disarankan) Lampirkan kebijakan langsung ke pengguna atau tambahkan pengguna ke grup pengguna. Ikuti petunjuk dalam Menambahkan izin ke pengguna (konsol) dalam Panduan Pengguna IAM.

Buat sertifikat penandatanganan kode

Untuk menandatangani gambar firmware secara digital, Anda memerlukan sertifikat penandatanganan kode dan kunci pribadi. Untuk tujuan pengujian, Anda dapat membuat sertifikat yang ditandatangani sendiri dan kunci pribadi. Untuk lingkungan produksi, beli sertifikat melalui otoritas sertifikat (CA) yang terkenal.

Platform yang berbeda memerlukan berbagai jenis sertifikat penandatanganan kode. Bagian berikut menjelaskan cara membuat sertifikat penandatanganan kode untuk berbagai platform yang memenuhi syarat Freertos.

Topik

- Membuat sertifikat penandatanganan kode untuk Texas Instruments 0SF-LAUNCHXL CC322
- Membuat sertifikat penandatanganan kode untuk Espressif ESP32
- Membuat sertifikat penandatanganan kode untuk Nordic nrf52840-dk
- Membuat sertifikat penandatanganan kode untuk simulator Windows FreeRTOS
- Membuat sertifikat penandatanganan kode untuk perangkat keras khusus

Membuat sertifikat penandatanganan kode untuk Texas Instruments 0SF-LAUNCHXL CC322

> **Important**
>
> Integrasi referensi ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-Freertos yang sekarang sudah tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Kit Pengembangan Launchpad Mikrokontroler Nirkabel SimpleLink Wi-Fi CC322 0SF mendukung dua rantai sertifikat untuk penandatanganan kode firmware:

- Produksi (sertifikat-katalog)

  Untuk menggunakan rantai sertifikat produksi, Anda harus membeli sertifikat penandatanganan kode komersial dan menggunakan alat TI Uniflash untuk mengatur papan ke mode produksi.

- Pengujian dan pengembangan (sertifikat-taman bermain)

  Rantai sertifikat taman bermain memungkinkan Anda untuk mencoba pembaruan OTA dengan sertifikat penandatanganan kode yang ditandatangani sendiri.

Gunakan AWS Command Line Interface untuk mengimpor sertifikat penandatanganan kode, kunci pribadi, dan rantai sertifikat Anda. AWS Certificate Manager Untuk informasi selengkapnya lihat Menginstal AWS CLI di Panduan AWS Command Line Interface Pengguna.

Unduh dan instal versi terbaru SimpleLink CC3220 SDK. Secara default, file yang Anda butuhkan berada di sini:

`C:\ti\simplelink_cc32xx_sdk_`*`version`*`\tools\cc32xx_tools\certificate-playground`(Jendela)

`/Applications/Ti/simplelink_cc32xx_`*`version`*`/tools/cc32xx_tools/certificate-playground` (macOS)

Sertifikat dalam SimpleLink CC322 0 SDK dalam format DER. Untuk membuat sertifikat penandatanganan kode yang ditandatangani sendiri, Anda harus mengonversinya ke format PEM.

Ikuti langkah-langkah ini untuk membuat sertifikat penandatanganan kode yang ditautkan ke hierarki sertifikat taman bermain Texas Instruments AWS Certificate Manager dan memenuhi serta Penandatanganan Kode untuk kriteria. AWS IoT

> **Note**
>
> Untuk membuat sertifikat penandatanganan kode, instal OpenSSL di mesin Anda. Setelah Anda menginstal OpenSSL, pastikan `openssl` bahwa ditetapkan ke OpenSSL executable di command prompt atau lingkungan terminal Anda.

Untuk membuat sertifikat penandatanganan kode yang ditandatangani sendiri

1. Buka command prompt atau terminal dengan izin administrator.

2. Di direktori kerja Anda, gunakan teks berikut untuk membuat file bernama`cert_config.txt`. Ganti *`test_signer@amazon.com`* dengan alamat email Anda.

```
[ req ]
prompt             = no
distinguished_name = my dn

[ my dn ]
commonName = test_signer@amazon.com

[ my_exts ]
keyUsage         = digitalSignature
extendedKeyUsage = codeSigning
```

3. Buat kunci pribadi dan permintaan penandatanganan sertifikat (CSR):

```
openssl req -config cert_config.txt -extensions my_exts -nodes -days 365 -newkey
 rsa:2048 -keyout tisigner.key -out tisigner.csr
```

4. Konversikan kunci privat CA root taman bermain Texas Instruments dari format DER ke format PEM.

   Kunci pribadi CA root TI playground terletak di sini:

   `C:\ti\simplelink_cc32xx_sdk_`*`version`*`\tools\cc32xx_tools\certificate-playground\dummy-root-ca-cert-key`(Jendela)

   `/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. Konversikan sertifikat CA root taman bermain Texas Instruments dari format DER ke format PEM.

   Sertifikat root taman bermain TI terletak di sini:

   `C:\ti\simplelink_cc32xx_sdk_`*`version`*`\tools\cc32xx_tools\certificate-playground/dummy-root-ca-cert`(Jendela)

   `/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. Tanda tangani CSR dengan root CA 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. Konversikan sertifikat penandatanganan kode (`tisigner.crt.pem`) Anda ke format DER:

```
openssl x509 -in tisigner.crt.pem -out tisigner.crt.der -outform DER
```

> **Note**
>
> Anda menulis `tisigner.crt.der` sertifikat ke papan pengembangan TI nanti.

8. Impor sertifikat penandatanganan kode, kunci pribadi, dan rantai sertifikat ke: 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
```

Perintah ini menampilkan ARN untuk sertifikat Anda. Anda memerlukan ARN ini saat Anda membuat pekerjaan pembaruan OTA.

> **Note**
>
> Langkah ini ditulis dengan asumsi bahwa Anda akan menggunakan Penandatanganan Kode AWS IoT untuk menandatangani gambar firmware Anda. Meskipun penggunaan Code Signing for AWS IoT direkomendasikan, Anda dapat menandatangani gambar firmware Anda secara manual.

Membuat sertifikat penandatanganan kode untuk Espressif ESP32

> **Important**
>
> Integrasi referensi ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-Freertos yang sekarang sudah tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

ESP32 Papan Espressif mendukung SHA-256 yang ditandatangani sendiri dengan sertifikat penandatanganan kode ECDSA.

> **Note**
>
> Untuk membuat sertifikat penandatanganan kode, instal OpenSSL di mesin Anda. Setelah Anda menginstal OpenSSL, pastikan `openssl` bahwa ditetapkan ke OpenSSL executable di command prompt atau lingkungan terminal Anda.
> Gunakan AWS Command Line Interface untuk mengimpor sertifikat penandatanganan kode, kunci pribadi, dan rantai sertifikat Anda. AWS Certificate Manager Untuk informasi tentang menginstal AWS CLI, lihat Menginstal AWS CLI.

1. Di direktori kerja Anda, gunakan teks berikut untuk membuat file bernama`cert_config.txt`. Ganti *`test_signer@amazon.com`* dengan alamat email Anda:

```
[ req ]
prompt             = no
distinguished_name = my_dn

[ my_dn ]
commonName = test_signer@amazon.com

[ my_exts ]
keyUsage         = digitalSignature
extendedKeyUsage = codeSigning
```

2. Buat kunci pribadi penandatanganan kode ECDSA:

```
openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-256 -pkeyopt
 ec_param_enc:named_curve -outform PEM -out ecdsasigner.key
```

3. Buat sertifikat penandatanganan kode ECDSA:

```
openssl req -new -x509 -config cert_config.txt -extensions my_exts -nodes -days 365
 -key ecdsasigner.key -out ecdsasigner.crt
```

4. Impor sertifikat penandatanganan kode, kunci pribadi, dan rantai sertifikat ke: AWS Certificate Manager

```
aws acm import-certificate --certificate fileb://ecdsasigner.crt --private-key
 fileb://ecdsasigner.key
```

   Perintah ini menampilkan ARN untuk sertifikat Anda. Anda memerlukan ARN ini saat Anda membuat pekerjaan pembaruan OTA.

> **Note**
>
> Langkah ini ditulis dengan asumsi bahwa Anda akan menggunakan Penandatanganan Kode AWS IoT untuk menandatangani gambar firmware Anda. Meskipun penggunaan Code Signing for AWS IoT direkomendasikan, Anda dapat menandatangani gambar firmware Anda secara manual.

Membuat sertifikat penandatanganan kode untuk Nordic nrf52840-dk

> **Important**
>
> Integrasi referensi ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-Freertos yang sekarang sudah tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Nordic nrf52840-dk mendukung sertifikat penandatanganan kode ECDSA yang ditandatangani sendiri. SHA256

> **Note**
>
> Untuk membuat sertifikat penandatanganan kode, instal OpenSSL di mesin Anda. Setelah Anda menginstal OpenSSL, pastikan `openssl` bahwa ditetapkan ke OpenSSL executable di command prompt atau lingkungan terminal Anda.
> Gunakan AWS Command Line Interface untuk mengimpor sertifikat penandatanganan kode, kunci pribadi, dan rantai sertifikat Anda. AWS Certificate Manager Untuk informasi tentang menginstal AWS CLI, lihat Menginstal AWS CLI.

1. Di direktori kerja Anda, gunakan teks berikut untuk membuat file bernama`cert_config.txt`. Ganti *`test_signer@amazon.com`* dengan alamat email Anda:

```
[ req ]
prompt             = no
distinguished_name = my_dn

[ my_dn ]
commonName = test_signer@amazon.com

[ my_exts ]
keyUsage         = digitalSignature
extendedKeyUsage = codeSigning
```

2. Buat kunci pribadi penandatanganan kode ECDSA:

```
openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-256 -pkeyopt
 ec_param_enc:named_curve -outform PEM -out ecdsasigner.key
```

3. Buat sertifikat penandatanganan kode ECDSA:

```
openssl req -new -x509 -config cert_config.txt -extensions my_exts -nodes -days 365
 -key ecdsasigner.key -out ecdsasigner.crt
```

4. Impor sertifikat penandatanganan kode, kunci pribadi, dan rantai sertifikat ke: AWS Certificate Manager

```
aws acm import-certificate --certificate fileb://ecdsasigner.crt --private-key
 fileb://ecdsasigner.key
```

Perintah ini menampilkan ARN untuk sertifikat Anda. Anda memerlukan ARN ini saat Anda membuat pekerjaan pembaruan OTA.

> **Note**
>
> Langkah ini ditulis dengan asumsi bahwa Anda akan menggunakan Penandatanganan Kode AWS IoT untuk menandatangani gambar firmware Anda. Meskipun penggunaan Code Signing for AWS IoT direkomendasikan, Anda dapat menandatangani gambar firmware Anda secara manual.

Membuat sertifikat penandatanganan kode untuk simulator Windows FreeRTOS

Simulator Windows FreeRTOS memerlukan sertifikat penandatanganan kode dengan kunci ECDSA P-256 dan hash SHA-256 untuk melakukan pembaruan OTA. Jika Anda tidak memiliki sertifikat penandatanganan kode, ikuti langkah-langkah berikut untuk membuatnya.

> **Note**
>
> Untuk membuat sertifikat penandatanganan kode, instal OpenSSL di mesin Anda. Setelah Anda menginstal OpenSSL, pastikan `openssl` bahwa ditetapkan ke OpenSSL executable di command prompt atau lingkungan terminal Anda.
> Gunakan AWS Command Line Interface untuk mengimpor sertifikat penandatanganan kode, kunci pribadi, dan rantai sertifikat Anda. AWS Certificate Manager Untuk informasi tentang menginstal AWS CLI, lihat Menginstal AWS CLI.

1. Di direktori kerja Anda, gunakan teks berikut untuk membuat file bernama`cert_config.txt`. Ganti *`test_signer@amazon.com`* dengan alamat email Anda:

```
[ req ]
prompt             = no
distinguished_name = my_dn

[ my_dn ]
commonName = test_signer@amazon.com

[ my_exts ]
keyUsage         = digitalSignature
```

```
extendedKeyUsage = codeSigning
```

2. Buat kunci pribadi penandatanganan kode ECDSA:

```
openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-256 -pkeyopt
 ec_param_enc:named_curve -outform PEM -out ecdsasigner.key
```

3. Buat sertifikat penandatanganan kode ECDSA:

```
openssl req -new -x509 -config cert_config.txt -extensions my_exts -nodes -days 365
 -key ecdsasigner.key -out ecdsasigner.crt
```

4. Impor sertifikat penandatanganan kode, kunci pribadi, dan rantai sertifikat ke: AWS Certificate Manager

```
aws acm import-certificate --certificate fileb://ecdsasigner.crt --private-key
 fileb://ecdsasigner.key
```

   Perintah ini menampilkan ARN untuk sertifikat Anda. Anda memerlukan ARN ini saat Anda membuat pekerjaan pembaruan OTA.

   > **Note**
   >
   > Langkah ini ditulis dengan asumsi bahwa Anda akan menggunakan Penandatanganan Kode AWS IoT untuk menandatangani gambar firmware Anda. Meskipun penggunaan Code Signing for AWS IoT direkomendasikan, Anda dapat menandatangani gambar firmware Anda secara manual.

Membuat sertifikat penandatanganan kode untuk perangkat keras khusus

Menggunakan toolset yang sesuai, buat sertifikat yang ditandatangani sendiri dan kunci pribadi untuk perangkat keras Anda.

Gunakan AWS Command Line Interface untuk mengimpor sertifikat penandatanganan kode, kunci pribadi, dan rantai sertifikat Anda. AWS Certificate Manager Untuk informasi tentang menginstal AWS CLI, lihat Menginstal AWS CLI.

Setelah membuat sertifikat penandatanganan kode, Anda dapat menggunakan sertifikat AWS CLI untuk mengimpornya ke ACM:

```
aws acm import-certificate --certificate fileb://code-sign.crt --private-key fileb://
code-sign.key
```

Output dari perintah ini menampilkan ARN untuk sertifikat Anda. Anda memerlukan ARN ini saat Anda membuat pekerjaan pembaruan OTA.

ACM membutuhkan sertifikat untuk menggunakan algoritma dan ukuran kunci tertentu. Untuk informasi selengkapnya, lihat Prasyarat untuk Mengimpor Sertifikat. Untuk informasi selengkapnya tentang ACM, lihat Mengimpor Sertifikat ke dalam. AWS Certificate Manager

Anda harus menyalin, menempel, dan memformat konten sertifikat penandatanganan kode Anda ke dalam `vendors/vendor/boards/board/aws_demos/config_files/ota_demo_config.h` file yang merupakan bagian dari kode FreeRTOS yang Anda unduh nanti.

Berikan akses ke penandatanganan kode untuk AWS IoT

Untuk memberikan akses dan menambahkan izin bagi pengguna, grup, atau peran Anda:

- Pengguna dan grup di AWS IAM Identity Center:

  Buat rangkaian izin. Ikuti instruksi di Buat rangkaian izin di Panduan Pengguna AWS IAM Identity Center .
- Pengguna yang dikelola di IAM melalui penyedia identitas:

  Buat peran untuk federasi identitas. Ikuti instruksi dalam Buat peran untuk penyedia identitas pihak ketiga (federasi) dalam Panduan Pengguna IAM.
- Pengguna IAM:
  - Buat peran yang dapat diambil pengguna Anda. Ikuti instruksi dalam Buat peran untuk pengguna IAM dalam Panduan Pengguna IAM.
  - (Tidak disarankan) Lampirkan kebijakan langsung ke pengguna atau tambahkan pengguna ke grup pengguna. Ikuti petunjuk dalam Menambahkan izin ke pengguna (konsol) dalam Panduan Pengguna IAM.

Unduh FreeRTOS dengan perpustakaan OTA

Anda dapat mengkloning atau mengunduh FreeRTOS dari. GitHub Lihat file README.md untuk instruksi.

Untuk informasi tentang menyiapkan dan menjalankan aplikasi demo OTA, lihatOver-the-air memperbarui aplikasi demo.

⚠ Important

- Dalam topik ini, jalur ke direktori unduhan FreerTOS disebut sebagai. *freertos*
- Karakter ruang di *freertos* jalur dapat menyebabkan kegagalan build. Saat Anda mengkloning atau menyalin repositori, pastikan jalur yang Anda buat tidak berisi karakter spasi.
- Panjang maksimum jalur file di Microsoft Windows adalah 260 karakter. Jalur direktori unduhan FreeRTOS yang panjang dapat menyebabkan kegagalan build.
- Karena kode sumber mungkin berisi tautan simbolis, jika Anda menggunakan Windows untuk mengekstrak arsip, Anda mungkin harus:
  - Aktifkan Mode Pengembang atau,
  - Gunakan konsol yang ditinggikan sebagai administrator.

  Dengan cara ini, Windows dapat membuat tautan simbolis dengan benar saat mengekstrak arsip. Jika tidak, tautan simbolis akan ditulis sebagai file normal yang berisi jalur tautan simbolis sebagai teks atau kosong. Untuk informasi lebih lanjut, lihat entri blog Symlinks di Windows 10! .

  Jika Anda menggunakan Git di bawah Windows, Anda harus mengaktifkan Mode Pengembang atau Anda harus:

  - Setel `core.symlinks` ke true dengan perintah berikut:

    ```
    git config --global core.symlinks true
    ```

  - Gunakan konsol yang ditinggikan sebagai administrator setiap kali Anda menggunakan perintah git yang menulis ke sistem (misalnya,git pull,git clone, dangit submodule update --init --recursive).

Prasyarat untuk pembaruan OTA menggunakan MQTT

Bagian ini menjelaskan persyaratan umum untuk menggunakan MQTT untuk melakukan over-the-air (pembaruan OTA).

Persyaratan minimum

- Firmware perangkat harus menyertakan pustaka FreerTOS yang diperlukan (Agen CoreMQTT, pembaruan OTA, dan dependensinya).
- FreeRTOS versi 1.4.0 atau yang lebih baru diperlukan. Namun, kami menyarankan Anda menggunakan versi terbaru jika memungkinkan.

Konfigurasi

Dimulai dengan versi 201912.00, FreerTOS OTA dapat menggunakan protokol HTTP atau MQTT untuk mentransfer gambar pembaruan firmware dari ke perangkat. AWS IoT Jika Anda menentukan kedua protokol saat membuat pembaruan OTA di FreeRTOS, setiap perangkat akan menentukan protokol yang digunakan untuk mentransfer gambar. Untuk informasi selengkapnya, lihat Prasyarat untuk pembaruan OTA menggunakan HTTP.

Secara default, konfigurasi protokol OTA `ota_config.h`adalah dengan menggunakan protokol MQTT.

Konfigurasi khusus perangkat

Tidak ada.

Penggunaan memori

Ketika MQTT digunakan untuk transfer data, tidak ada memori tambahan yang diperlukan untuk koneksi MQTT karena itu dibagi antara kontrol dan operasi data.

Kebijakan perangkat

Setiap perangkat yang menerima pembaruan OTA menggunakan MQTT harus terdaftar sebagai sesuatu AWS IoT dan benda tersebut harus memiliki kebijakan terlampir seperti yang tercantum di sini. Anda dapat menemukan informasi selengkapnya tentang item dalam `"Action"` dan `"Resource"` objek di AWS IoT Core Policy Actions dan AWS IoT Core Action Resources.

```
{
    "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/*"
            ]
        }
    ]
}
```

Catatan

- `iot:Connect`Izin memungkinkan perangkat Anda terhubung AWS IoT melalui MQTT.
- `iot:Publish`Izin `iot:Subscribe` dan pada topik AWS IoT pekerjaan (`.../jobs/*`) memungkinkan perangkat yang terhubung untuk menerima pemberitahuan pekerjaan dan dokumen pekerjaan, dan untuk mempublikasikan status penyelesaian pelaksanaan pekerjaan.
- `iot:Publish`Izin `iot:Subscribe` dan tentang topik aliran AWS IoT OTA (`.../streams/*`) memungkinkan perangkat yang terhubung untuk mengambil data pembaruan OTA dari. AWS IoT Izin ini diperlukan untuk melakukan pembaruan firmware melalui MQTT.
- `iot:Receive`Izin memungkinkan AWS IoT Core untuk mempublikasikan pesan tentang topik tersebut ke perangkat yang terhubung. Izin ini diperiksa pada setiap pengiriman pesan MQTT. Anda dapat menggunakan izin ini untuk mencabut akses ke klien yang saat ini berlangganan topik.

Prasyarat untuk pembaruan OTA menggunakan HTTP

Bagian ini menjelaskan persyaratan umum untuk menggunakan pembaruan HTTP to perform over-the-air (OTA). Dimulai dengan versi 201912.00, FreerTOS OTA dapat menggunakan protokol HTTP atau MQTT untuk mentransfer gambar pembaruan firmware dari ke perangkat. AWS IoT

> **Note**
>
> - Meskipun protokol HTTP dapat digunakan untuk mentransfer gambar firmware, pustaka Agen CoreMQTT masih diperlukan karena interaksi lain dengan AWS IoT Core menggunakan pustaka Agen CoreMQTT, termasuk mengirim atau menerima pemberitahuan eksekusi pekerjaan, dokumen pekerjaan, dan pembaruan status eksekusi.
> - Saat Anda menentukan protokol MQTT dan HTTP untuk pekerjaan pembaruan OTA, penyiapan perangkat lunak Agen OTA pada setiap perangkat menentukan protokol yang digunakan untuk mentransfer gambar firmware. Untuk mengubah Agen OTA dari metode protokol MQTT default ke protokol HTTP, Anda dapat memodifikasi file header yang digunakan untuk mengkompilasi kode sumber FreeRTOS untuk perangkat.

Persyaratan minimum

- Firmware perangkat harus menyertakan pustaka FreerTOS yang diperlukan (Agen CoreMQTT, HTTP, Agen OTA, dan dependensinya).
- FreeRTOS versi 201912.00 atau yang lebih baru diperlukan untuk mengubah konfigurasi protokol OTA untuk mengaktifkan transfer data OTA melalui HTTP.

Konfigurasi

Lihat konfigurasi protokol OTA berikut dalam file. `\vendors\boards\board\aws_demos\config_files\ota_config.h`

Untuk mengaktifkan transfer data OTA melalui HTTP

1. Ubah `configENABLED_DATA_PROTOCOLS` ke `OTA_DATA_OVER_HTTP`.
2. Saat OTA diperbarui, Anda dapat menentukan kedua protokol sehingga protokol MQTT atau HTTP dapat digunakan., Anda dapat mengatur protokol utama yang digunakan oleh perangkat ke HTTP dengan mengubah ke. `configOTA_PRIMARY_DATA_PROTOCOL OTA_DATA_OVER_HTTP`

> **Note**
>
> HTTP hanya didukung untuk operasi data OTA. Untuk operasi kontrol, Anda harus menggunakan MQTT.

Konfigurasi khusus perangkat

ESP32

Karena jumlah RAM yang terbatas, Anda harus mematikan BLE ketika Anda mengaktifkan HTTP sebagai protokol data OTA. Dalam vendors/espressif/boards/esp32/aws_demos/config_files/aws_iot_network_config.hfile, ubah `configENABLED_NETWORKS` menjadi `AWSIOT_NETWORK_TYPE_WIFI` saja.

```
/**
     * @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 )
```

Penggunaan memori

Ketika MQTT digunakan untuk transfer data, tidak ada memori heap tambahan yang diperlukan untuk koneksi MQTT karena itu dibagi antara kontrol dan operasi data. Namun, mengaktifkan data melalui HTTP membutuhkan memori heap tambahan. Berikut ini adalah data penggunaan memori heap untuk semua platform yang didukung, dihitung menggunakan FreeRTOS API`xPortGetFreeHeapSize`. Anda harus memastikan ada cukup RAM untuk menggunakan perpustakaan OTA.

Texas Instruments CC322 0SF-LAUNCHXL

Operasi kontrol (MQTT): 12 KB

Operasi data (HTTP): 10 KB

> **Note**
>
> TI menggunakan RAM yang jauh lebih sedikit karena menggunakan SSL pada perangkat keras, sehingga tidak menggunakan perpustakaan mbedtls.

Keingintahuan PIC32 Microchip MZEF

Operasi kontrol (MQTT): 65 KB

Operasi data (HTTP): 43 KB

Espressif ESP32

Operasi kontrol (MQTT): 65 KB

Operasi data (HTTP): 45 KB

> **Note**
>
> BLE ESP32 membutuhkan sekitar 87 KB RAM. Tidak ada cukup RAM untuk mengaktifkan semuanya, yang disebutkan dalam konfigurasi khusus perangkat di atas.

Simulator Windows

Operasi kontrol (MQTT): 82 KB

Operasi data (HTTP): 63 KB

Nordik nrf52840-dk

HTTP tidak didukung.

Kebijakan perangkat

Kebijakan ini memungkinkan Anda untuk menggunakan MQTT atau HTTP untuk pembaruan OTA.

Setiap perangkat yang menerima pembaruan OTA menggunakan HTTP harus terdaftar sebagai sesuatu AWS IoT dan benda tersebut harus memiliki kebijakan terlampir seperti yang tercantum di sini. Anda dapat menemukan informasi selengkapnya tentang item dalam `"Action"` dan `"Resource"` objek di AWS IoT Core Policy Actions dan AWS IoT Core Action Resources.

```
{
    "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/*"
            ]
        }
    ]
}
```

Catatan

- `iot:Connect`Izin memungkinkan perangkat Anda terhubung AWS IoT melalui MQTT.
- `iot:Publish`Izin `iot:Subscribe` dan pada topik AWS IoT pekerjaan (`.../jobs/*`) memungkinkan perangkat yang terhubung untuk menerima pemberitahuan pekerjaan dan dokumen pekerjaan, dan untuk mempublikasikan status penyelesaian pelaksanaan pekerjaan.

- `iot:Receive`Izin memungkinkan AWS IoT Core untuk mempublikasikan pesan tentang topik tersebut ke perangkat yang terhubung saat ini. Izin ini diperiksa pada setiap pengiriman pesan MQTT. Anda dapat menggunakan izin ini untuk mencabut akses ke klien yang saat ini berlangganan topik.

Tutorial OTA

Bagian ini berisi tutorial untuk memperbarui firmware pada perangkat yang menjalankan FreeRTOS menggunakan pembaruan OTA. Selain gambar firmware, Anda dapat menggunakan pembaruan OTA untuk mengirim semua jenis file ke perangkat yang terhubung AWS IoT.

Anda dapat menggunakan AWS IoT konsol atau AWS CLI untuk membuat pembaruan OTA. Konsol adalah cara termudah untuk memulai dengan OTA karena ia melakukan banyak pekerjaan untuk Anda. AWS CLI Ini berguna saat Anda mengotomatiskan pekerjaan pembaruan OTA, bekerja dengan sejumlah besar perangkat, atau menggunakan perangkat yang belum memenuhi syarat untuk FreeRTOS. Untuk informasi selengkapnya tentang perangkat yang memenuhi syarat untuk FreeRTOS, lihat situs web FreeRTOS Partners.

Untuk membuat pembaruan OTA

1. Menerapkan versi awal firmware Anda ke satu atau beberapa perangkat.
2. Verifikasi bahwa firmware berjalan dengan benar.
3. Ketika pembaruan firmware diperlukan, buat perubahan kode dan buat gambar baru.
4. Jika Anda menandatangani firmware secara manual, tandatangani lalu unggah gambar firmware yang ditandatangani ke bucket Amazon S3 Anda. Jika Anda menggunakan Penandatanganan Kode untuk AWS IoT, unggah image firmware yang tidak ditandatangani ke bucket Amazon S3.
5. Buat pembaruan OTA.

Saat Anda membuat pembaruan OTA, Anda menentukan protokol pengiriman gambar (MQTT atau HTTP) atau menentukan keduanya untuk memungkinkan perangkat memilih. Agen FreeRTOS OTA pada perangkat menerima gambar firmware yang diperbarui dan memverifikasi tanda tangan digital, checksum, dan nomor versi gambar baru. Jika pembaruan firmware diverifikasi, perangkat diatur ulang dan, berdasarkan logika yang ditentukan aplikasi, melakukan pembaruan. Jika perangkat Anda tidak menjalankan FreeRTOS, Anda harus menerapkan agen OTA yang berjalan di perangkat Anda.

Menginstal firmware awal

Untuk memperbarui firmware, Anda harus menginstal versi awal firmware yang menggunakan perpustakaan Agen OTA untuk mendengarkan pekerjaan pembaruan OTA. Jika Anda tidak menjalankan FreeRTOS, lewati langkah ini. Anda harus menyalin implementasi Agen OTA Anda ke perangkat Anda sebagai gantinya.

Topik

- Instal versi awal firmware pada Texas Instruments CC322 0SF-LAUNCHXL
- Instal versi awal firmware pada Espressif ESP32
- Instal versi awal firmware pada Nordic n RF5284 0 DK
- Firmware awal pada simulator Windows
- Instal versi awal firmware pada papan khusus

Instal versi awal firmware pada Texas Instruments CC322 0SF-LAUNCHXL

> **Important**
>
> Integrasi referensi ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-Freertos yang sekarang sudah tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Langkah-langkah ini ditulis dengan asumsi bahwa Anda telah membangun `aws_demos` proyek, seperti yang dijelaskan dalamUnduh, buat, flash, dan jalankan demo FreerTOS OTA di Texas Instruments 0SF-LAUNCHXL CC322.

1. Pada Texas Instruments CC322 0SF-LAUNCHXL Anda, letakkan jumper SOP di set tengah pin (posisi = 1) dan atur ulang papan.
2. Unduh dan instal alat TI Uniflash.
3. Mulai Uniflash. Dari daftar konfigurasi, pilih CC3220SF-LAUNCHXL, lalu pilih Start Image Creator.
4. Pilih Proyek Baru.
5. Pada halaman Mulai proyek baru, masukkan nama untuk proyek Anda. Untuk Jenis Perangkat, pilih CC3220SF. Untuk Mode Perangkat, pilih Kembangkan. Pilih Buat Proyek.

6. Putuskan sambungan emulator terminal Anda.
7. Di sisi kanan jendela aplikasi Uniflash, pilih Connect.
8. Di bawah Advanced, Files, pilih User Files.
9. Di panel Pemilih file, pilih ikon Tambah File.

10. Jelajahi `/Applications/Ti/simplelink_cc32xx_sdk_`*`version`*`/tools/cc32xx_tools/certificate-playground` direktori, pilih`dummy-root-ca-cert`, pilih Buka, lalu pilih Tulis.
11. Di panel Pemilih file, pilih ikon Tambah File.

12. Jelajahi direktori kerja tempat Anda membuat sertifikat penandatanganan kode dan kunci pribadi, pilih, pilih Buka **`tisigner.crt.der`**, lalu pilih Tulis.
13. Dari daftar drop-down Tindakan, pilih Pilih Gambar MCU, lalu pilih Browse untuk memilih gambar firmware yang akan digunakan tulis ke perangkat Anda (aws_demos.bin). File ini terletak di *`freertos`*`/vendors/ti/boards/cc3220_launchpad/aws_demos/ccs/Debug` direktori. Pilih Buka.

    a. Di kotak dialog file, konfirmasikan nama file diatur ke`mcuflashimg.bin`.

    b. Pilih kotak centang Vendor.

    c. Di bawah Token File, ketik**`1952007250`**.

    d. Di bawah Nama File Kunci Pribadi, pilih Jelajahi, lalu pilih `tisigner.key` dari direktori kerja tempat Anda membuat sertifikat penandatanganan kode dan kunci pribadi.

    e. Di bawah Nama File Sertifikasi, pilih`tisigner.crt.der`.

    f. Pilih Tulis.

14. Di panel kiri, di bawah File, pilih Paket Layanan.
15. Di bawah Nama File Paket Layanan, pilih Jelajahi, jelajahi `simplelink_cc32x_sdk_`*`version`*`/tools/cc32xx_tools/servicepack-cc3x20sp_3.7.0.1_2.0.0.0_2.2.0.6.bin`, pilih, lalu pilih Buka.
16. Di panel kiri, di bawah File, pilih Katalog Root-Certificate Tepercaya.
17. Kosongkan kotak centang Gunakan Katalog Root-Certificate Tepercaya default.
18. Di bawah File Sumber, pilih Browse, pilih ***`version`***simplelink_cc32xx_sdk_/20160911.lst, lalu pilih Buka. tools/cc32xx_tools/certificate-playground/certcatalogPlayGround

19. Di bawah File Sumber Tanda Tangan, pilih Browse, pilih simplelink_cc32xx_sdk_/20160911.lst.signed_3220.bin, lalu ***version*** pilih Buka. tools/cc32xx_tools/certificate-playground/certcatalogPlayGround
20. Pilih

    tombol untuk menyimpan proyek Anda.
21. Pilih

    tombolnya.
22. Pilih Gambar Program (Buat dan Program).
23. Setelah proses pemrograman selesai, letakkan jumper SOP ke set pin pertama (posisi = 0), atur ulang papan, dan sambungkan kembali emulator terminal Anda untuk memastikan outputnya sama seperti saat Anda men-debug demo dengan Code Composer Studio. Catat nomor versi aplikasi di output terminal. Anda menggunakan nomor versi ini nanti untuk memverifikasi bahwa firmware Anda telah diperbarui oleh pembaruan OTA.

    Terminal harus menampilkan output seperti berikut ini.

```
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
```

Instal versi awal firmware pada Espressif ESP32

> **Important**
>
> Integrasi referensi ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-Freertos yang sekarang sudah tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Panduan ini ditulis dengan asumsi bahwa Anda telah melakukan langkah-langkah dalam Memulai dengan Espressif ESP32 - DevKit C dan Prasyarat ESP-WROVER-KIT dan Over-the-Air Perbarui. Sebelum Anda mencoba pembaruan OTA, Anda mungkin ingin menjalankan proyek demo MQTT yang dijelaskan dalam Memulai dengan FreeRTOS untuk memastikan bahwa papan dan rantai alat Anda diatur dengan benar.

Untuk mem-flash gambar pabrik awal ke papan

1. Buka*freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h, beri komentar#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED, dan tentukan CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED atauCONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED.
2. Salin sertifikat penandatanganan kode berformat SHA-256/ECDSA PEM yang Anda buat di to. Prasyarat pembaruan OTA vendors/*vendor*/boards/*board*/aws_demos/config_files/ota_demo_config.h Itu harus diformat dengan cara berikut.

```
#define otapalconfigCODE_SIGNING_CERTIFICATE  \
"-----BEGIN CERTIFICATE-----\n" \
```

```
"...base64 data...\n" \
"-----END CERTIFICATE-----\n";
```

3. Dengan demo Pembaruan OTA yang dipilih, ikuti langkah-langkah yang sama yang diuraikan dalam Memulai ESP32 untuk membangun dan mem-flash gambar. Jika sebelumnya Anda telah membangun dan mem-flash proyek, Anda mungkin perlu menjalankannya `make clean` terlebih dahulu. Setelah Anda berlari`make flash monitor`, Anda akan melihat sesuatu seperti berikut ini. Urutan beberapa pesan mungkin berbeda, karena aplikasi demo menjalankan banyak tugas sekaligus.

```
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. ESP32 Papan sekarang mendengarkan pembaruan OTA. Monitor ESP-IDF diluncurkan oleh perintah. `make flash monitor` Anda dapat menekan Ctrl+] untuk berhenti. Anda juga dapat menggunakan program terminal TTY favorit Anda (misalnya, PuTTY, Tera Term, atau Layar GNU) untuk mendengarkan output serial papan. Ketahuilah bahwa menghubungkan ke port serial papan dapat menyebabkannya reboot.

Instal versi awal firmware pada Nordic n RF5284 0 DK

> **Important**
>
> Integrasi referensi ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-Freertos yang sekarang sudah tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Panduan ini ditulis dengan asumsi bahwa Anda telah melakukan langkah-langkah Memulai dengan Nordic n 0-DK RF5284 dan Over-the-Air Perbarui Prasyarat. Sebelum Anda mencoba pembaruan OTA, Anda mungkin ingin menjalankan proyek demo MQTT yang dijelaskan dalam Memulai dengan FreeRTOS untuk memastikan bahwa papan dan rantai alat Anda diatur dengan benar.

Untuk mem-flash gambar pabrik awal ke papan

1. Buka *freertos*`/vendors/nordic/boards/nrf52840-dk/aws_demos/config_files/aws_demo_config.h`.
2. Ganti `#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED` dengan `CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED` atau`CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED`.
3. Dengan demo Pembaruan OTA yang dipilih, ikuti langkah-langkah yang sama yang diuraikan Memulai dengan Nordic n 0-DK RF5284 untuk membangun dan mem-flash gambar.

   Anda akan melihat output seperti yang berikut ini.

```
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
```

Papan Anda sekarang mendengarkan pembaruan OTA.

Firmware awal pada simulator Windows

Saat Anda menggunakan simulator Windows, tidak perlu mem-flash versi awal firmware. Simulator Windows adalah bagian dari `aws_demos` aplikasi, yang juga mencakup firmware.

Instal versi awal firmware pada papan khusus

Menggunakan IDE Anda, bangun `aws_demos` proyek, pastikan untuk menyertakan perpustakaan OTA. Untuk informasi lebih lanjut tentang struktur kode sumber FreeRTOS, lihat. Demo FreeRTOS

Pastikan untuk menyertakan sertifikat penandatanganan kode, kunci pribadi, dan rantai kepercayaan sertifikat baik di proyek FreeRTOS atau di perangkat Anda.

Dengan menggunakan alat yang sesuai, bakar aplikasi ke papan Anda dan pastikan itu berjalan dengan benar.

Perbarui versi firmware Anda

Agen OTA yang disertakan dengan FreeRTOS memeriksa versi pembaruan apa pun dan menginstalnya hanya jika lebih baru daripada versi firmware yang ada. Langkah-langkah berikut menunjukkan kepada Anda cara menambah versi firmware dari aplikasi demo OTA.

1. Buka `aws_demos` proyek di IDE Anda.
2. Temukan file `/vendors/vendor/boards/board/aws_demos/config_files/ota_demo_config.h` dan naikkan nilai APP_VERSION_BUILD.
3. Untuk menjadwalkan pembaruan ke platform Renesas rx65n dengan tipe file selain 0 (file non-firmware), Anda harus menandatangani file dengan alat Renesas Secure Flash Programmer, jika

tidak maka akan gagal pemeriksaan tanda tangan pada perangkat. Alat ini membuat paket file yang ditandatangani dengan ekstensi yang `.rsu` merupakan jenis file berpemilik untuk Renesas. Alat ini dapat ditemukan di Github. Anda dapat menggunakan perintah contoh berikut untuk menghasilkan gambar:

```
"Renesas Secure Flash Programmer.exe" CUI Update "RX65N(ROM 2MB)/Secure
 Bootloader=256KB" "sig-sha256-ecdsa" 1 "file_name" "output_file_name.rsu"
```

4. Membangun kembali proyek.

Anda harus menyalin pembaruan firmware ke bucket Amazon S3 yang Anda buat seperti yang dijelaskan di. Buat bucket Amazon S3 untuk menyimpan pembaruan Anda Nama file yang perlu Anda salin ke Amazon S3 tergantung pada platform perangkat keras yang Anda gunakan:

- Texas Instruments CC322 0SF-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`

Membuat pembaruan OTA (AWS IoT konsol)

1. Di panel navigasi AWS IoT konsol, di bawah Kelola pilih Tindakan jarak jauh, lalu pilih Pekerjaan.
2. Pilih Buat tugas.
3. Di bawah Jenis pekerjaan pilih Buat pekerjaan pembaruan FreerTOS OTA, lalu pilih Berikutnya.
4. Di properti Job, masukkan nama Job dan (opsional) masukkan Deskripsi pekerjaan, lalu pilih Berikutnya.
5. Anda dapat menerapkan pembaruan OTA ke satu perangkat atau grup perangkat. Di bawah Perangkat yang akan diperbarui, pilih satu atau beberapa grup hal atau hal dari menu tarik-turun.
6. Di bawah Pilih protokol untuk transfer file, pilih HTTP atau MQTT, atau pilih keduanya untuk mengizinkan setiap perangkat menentukan protokol yang akan digunakan.
7. Di bawah Masuk dan pilih file Anda, pilih Tanda tangani file baru untuk saya.
8. Di bawah Profil penandatanganan kode, pilih Buat profil baru.
9. Di Buat profil penandatanganan kode, masukkan nama untuk profil penandatanganan kode Anda.

    a. Di bawah Platform perangkat keras perangkat, pilih platform perangkat keras Anda.

> **Note**
>
> Hanya platform perangkat keras yang telah memenuhi syarat untuk FreeRTOS yang ditampilkan dalam daftar ini. Jika Anda menguji platform yang tidak memenuhi syarat, dan Anda menggunakan ciphersuite ECDSA P-256 SHA-256 untuk penandatanganan, Anda dapat memilih profil penandatanganan kode Simulator Windows untuk menghasilkan tanda tangan yang kompatibel. Jika Anda menggunakan platform yang tidak memenuhi syarat, dan Anda menggunakan ciphersuite selain ECDSA P-256 SHA-256 untuk penandatanganan, Anda dapat menggunakan Penandatanganan Kode untuk AWS IoT, atau Anda dapat menandatangani pembaruan firmware Anda sendiri. Untuk informasi selengkapnya, lihat Menandatangani pembaruan firmware Anda secara digital.

b. Di bawah Sertifikat penandatanganan kode, pilih Pilih sertifikat yang ada lalu pilih sertifikat yang diimpor sebelumnya, atau pilih Impor sertifikat penandatanganan kode baru, pilih file Anda dan pilih Impor untuk mengimpor sertifikat baru.

c. Di bawah Nama jalur sertifikat penandatanganan kode di perangkat, masukkan nama jalur yang sepenuhnya memenuhi syarat ke sertifikat penandatanganan kode di perangkat Anda. Untuk sebagian besar perangkat, Anda dapat membiarkan bidang ini kosong. Untuk simulator Windows dan untuk perangkat yang menempatkan sertifikat di lokasi file tertentu, masukkan nama jalur di sini.

> **Important**
>
> Pada Texas Instruments CC322 0SF-LAUNCHXL, jangan sertakan garis miring depan (**/**) di depan nama file jika sertifikat penandatanganan kode Anda ada di root sistem file. Jika tidak, pembaruan OTA gagal selama otentikasi dengan `file not found` kesalahan.

d. Pilih Buat.

10. Di bawah File pilih Pilih file yang ada lalu pilih Browse S3. Daftar bucket Amazon S3 Anda ditampilkan. Pilih bucket yang berisi pembaruan firmware Anda, lalu pilih pembaruan firmware Anda di bucket.

> **Note**
>
> Proyek demo Microchip Curiosity PIC32 MZEF menghasilkan dua gambar biner dengan nama default dan. `mplab.production.bin` `mplab.production.ota.bin` Gunakan file kedua saat Anda mengunggah gambar untuk pembaruan OTA.

11. Di bawah Pathname file di perangkat, masukkan nama jalur yang sepenuhnya memenuhi syarat ke lokasi di perangkat Anda tempat pekerjaan OTA akan menyalin gambar firmware. Lokasi ini bervariasi menurut platform.

> **Important**
>
> Pada Texas Instruments CC322 0SF-LAUNCHXL, karena pembatasan keamanan, nama jalur gambar firmware harus. `/sys/mcuflashimg.bin`

12. Buka File Type dan masukkan nilai integer dalam kisaran 0-255. Jenis file yang Anda masukkan akan ditambahkan ke dokumen Job yang dikirimkan ke MCU. Pengembang firmware/perangkat lunak MCU memiliki kepemilikan penuh atas apa yang harus dilakukan dengan nilai ini. Skenario yang mungkin termasuk MCU yang memiliki prosesor sekunder yang firmwarenya dapat diperbarui secara independen dari prosesor utama. Saat perangkat menerima pekerjaan pembaruan OTA, perangkat dapat menggunakan Jenis File untuk mengidentifikasi prosesor mana pembaruan tersebut.
13. Di bawah peran IAM, pilih peran sesuai dengan instruksi di[Buat peran layanan Pembaruan OTA](#).
14. Pilih Berikutnya.
15. Masukkan ID dan deskripsi untuk pekerjaan pembaruan OTA Anda.
16. Di bawah Jenis Job, pilih Pekerjaan Anda akan selesai setelah menerapkan ke perangkat/grup yang dipilih (snapshot).
17. Pilih konfigurasi opsional yang sesuai untuk pekerjaan Anda (Peluncuran eksekusi Job, Job abort, batas waktu eksekusi Job, dan Tag).
18. Pilih Buat.

Untuk menggunakan image firmware yang ditandatangani sebelumnya

1. Di bawah Pilih dan tandatangani gambar firmware Anda, pilih Pilih gambar firmware yang ditandatangani sebelumnya.

2. Di bawah Nama jalur gambar firmware di perangkat, masukkan nama jalur yang sepenuhnya memenuhi syarat ke lokasi di perangkat Anda tempat pekerjaan OTA akan menyalin gambar firmware. Lokasi ini bervariasi menurut platform.
3. Di bawah Pekerjaan penandatanganan kode sebelumnya, pilih Pilih, lalu pilih pekerjaan penandatanganan kode sebelumnya yang digunakan untuk menandatangani gambar firmware yang Anda gunakan untuk pembaruan OTA.

Menggunakan gambar firmware yang ditandatangani khusus

1. Di bawah Pilih dan tandatangani gambar firmware Anda, pilih Gunakan gambar firmware bertanda tangan kustom saya.
2. Di bawah Nama jalur sertifikat penandatanganan kode di perangkat, masukkan nama jalur yang sepenuhnya memenuhi syarat ke sertifikat penandatanganan kode di perangkat Anda. Untuk sebagian besar perangkat, Anda dapat membiarkan bidang ini kosong. Untuk simulator Windows dan untuk perangkat yang menempatkan sertifikat di lokasi file tertentu, masukkan nama jalur di sini.
3. Di bawah Nama jalur gambar firmware di perangkat, masukkan nama jalur yang sepenuhnya memenuhi syarat ke lokasi di perangkat Anda tempat pekerjaan OTA akan menyalin gambar firmware. Lokasi ini bervariasi menurut platform.
4. Di bawah Tanda Tangan, tempel tanda tangan format PEM Anda.
5. Di bawah algoritma hash asli, pilih algoritma hash yang digunakan saat Anda membuat tanda tangan file Anda.
6. Di bawah Algoritma enkripsi asli, pilih algoritme yang digunakan saat Anda membuat tanda tangan file.
7. Di bawah Pilih gambar firmware Anda di Amazon S3, pilih bucket Amazon S3 dan gambar firmware yang ditandatangani di bucket Amazon S3.

Setelah Anda menentukan informasi penandatanganan kode, tentukan jenis pekerjaan pembaruan OTA, peran layanan, dan ID untuk pembaruan Anda.

> **Note**
>
> Jangan gunakan informasi identitas pribadi apa pun di ID pekerjaan untuk pembaruan OTA Anda. Contoh informasi yang dapat diidentifikasi secara pribadi meliputi:
>
> - Nama.

- Alamat IP.
- Alamat email.
- Lokasi.
- Rincian bank.
- Informasi medis.

1. Di bawah Jenis Job, pilih Pekerjaan Anda akan selesai setelah menerapkan ke perangkat/grup yang dipilih (snapshot).
2. Di bawah peran IAM untuk pekerjaan pembaruan OTA, pilih peran layanan OTA Anda.
3. Masukkan ID alfanumerik untuk pekerjaan Anda, lalu pilih Buat.

Pekerjaan muncul di AWS IoT konsol dengan status IN PROGRESS.

Note

- AWS IoT Konsol tidak memperbarui status pekerjaan secara otomatis. Segarkan browser Anda untuk melihat pembaruan.

Hubungkan terminal UART serial Anda ke perangkat Anda. Anda akan melihat output yang menunjukkan perangkat sedang mengunduh firmware yang diperbarui.

Setelah perangkat mengunduh firmware yang diperbarui, itu restart dan kemudian menginstal firmware. Anda dapat melihat apa yang terjadi di terminal UART.

Untuk tutorial yang menunjukkan cara menggunakan konsol untuk membuat pembaruan OTA, lihatOver-the-air memperbarui aplikasi demo.

Membuat pembaruan OTA dengan AWS CLI

Saat Anda menggunakan AWS CLI untuk membuat pembaruan OTA, Anda:

1. Tanda tangani gambar firmware Anda secara digital.
2. Buat aliran gambar firmware yang ditandatangani secara digital.
3. Mulai pekerjaan pembaruan OTA.

Menandatangani pembaruan firmware Anda secara digital

Ketika Anda menggunakan AWS CLI untuk melakukan pembaruan OTA, Anda dapat menggunakan Penandatanganan Kode untuk AWS IoT, atau Anda dapat menandatangani pembaruan firmware Anda sendiri. Untuk daftar algoritma penandatanganan kriptografi dan hashing yang didukung oleh Penandatanganan Kode, lihat. AWS IoT SigningConfigurationOverrides Jika Anda ingin menggunakan algoritma kriptografi yang tidak didukung oleh Penandatanganan Kode untuk AWS IoT, Anda harus menandatangani biner firmware Anda sebelum mengunggahnya ke Amazon S3.

Menandatangani gambar firmware Anda dengan Penandatanganan Kode untuk AWS IoT

Untuk menandatangani gambar firmware Anda menggunakan Penandatanganan Kode untuk AWS IoT, Anda dapat menggunakan salah satu alat AWS SDKs atau baris perintah. Untuk informasi selengkapnya tentang Penandatanganan Kode AWS IoT, lihat Penandatanganan Kode untuk AWS IoT.

Setelah menginstal dan mengonfigurasi alat penandatanganan kode, salin gambar firmware yang tidak ditandatangani ke bucket Amazon S3 dan mulai pekerjaan penandatanganan kode dengan perintah berikut. AWS CLI put-signing-profilePerintah membuat profil penandatanganan kode yang dapat digunakan kembali. start-signing-jobPerintah memulai pekerjaan penandatanganan.

```
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*dan *your-destination-bucket-name* bisa menjadi bucket Amazon S3 yang sama.

Ini adalah parameter untuk start-signing-job perintah put-signing-profile dan:

`source`

Menentukan lokasi firmware yang tidak ditandatangani dalam bucket S3.

- `bucketName`: Nama ember S3 Anda.
- `key`: Kunci (nama file) firmware Anda di bucket S3 Anda.
- `version`: Versi S3 firmware Anda di bucket S3 Anda. Ini berbeda dengan versi firmware Anda. Anda dapat menemukannya dengan menjelajah ke konsol Amazon S3, memilih bucket Anda, dan di bagian atas halaman, di sebelah Versi, memilih Tampilkan.

`destination`

Tujuan pada perangkat tempat firmware yang ditandatangani di ember S3 akan disalin. Format parameter ini sama dengan `source` parameter.

`signing-material`

ARN sertifikat penandatanganan kode Anda. ARN ini dihasilkan saat Anda mengimpor sertifikat Anda ke ACM.

`signing-parameters`

Peta pasangan kunci-nilai untuk penandatanganan. Ini dapat mencakup informasi apa pun yang ingin Anda gunakan selama penandatanganan.

> **Note**
>
> Parameter ini diperlukan saat Anda membuat profil penandatanganan kode untuk menandatangani pembaruan OTA dengan Penandatanganan Kode untuk. AWS IoT

`platform`

Platform perangkat keras tempat Anda mendistribusikan pembaruan OTA. `platformId`

Untuk mengembalikan daftar platform yang tersedia dan `platformId` nilainya, gunakan `aws signer list-signing-platforms` perintah.

Pekerjaan penandatanganan dimulai dan menulis gambar firmware yang ditandatangani ke bucket Amazon S3 tujuan. Nama file untuk image firmware yang ditandatangani adalah GUID. Anda memerlukan nama file ini saat Anda membuat aliran. Anda dapat menemukan nama file dengan

menjelajah ke konsol Amazon S3 dan memilih bucket Anda. Jika Anda tidak melihat file dengan nama file GUID, segarkan browser Anda.

Perintah menampilkan ARN pekerjaan dan ID pekerjaan. Anda membutuhkan nilai-nilai ini nanti. Untuk informasi selengkapnya tentang Penandatanganan Kode AWS IoT, lihat Penandatanganan Kode untuk AWS IoT.

Menandatangani gambar firmware Anda secara manual

Tanda tangani gambar firmware Anda secara digital dan unggah gambar firmware yang ditandatangani ke dalam bucket Amazon S3 Anda.

Membuat aliran pembaruan firmware Anda

Stream adalah antarmuka abstrak ke data yang dapat dikonsumsi oleh perangkat. Streaming dapat menyembunyikan kerumitan mengakses data yang disimpan di lokasi yang berbeda atau layanan berbasis cloud yang berbeda. Layanan OTA Update Manager memungkinkan Anda untuk menggunakan beberapa bagian data, disimpan di berbagai lokasi di Amazon S3, untuk melakukan Pembaruan OTA.

Saat membuat Pembaruan AWS IoT OTA, Anda juga dapat membuat aliran yang berisi pembaruan firmware yang ditandatangani. Buat file JSON (`stream.json`) yang mengidentifikasi image firmware yang ditandatangani. File JSON harus berisi yang berikut ini.

```
[
  {
    "fileId":"your_file_id",
    "s3Location":{
      "bucket":"your_bucket_name",
      "key":"your_s3_object_key"
    }
  }
]
```

Ini adalah atribut dalam file JSON:

`fileId`

Integer arbitrer antara 0-255 yang mengidentifikasi image firmware Anda.

`s3Location`

Bucket dan kunci untuk firmware untuk streaming.

`bucket`

Bucket Amazon S3 tempat penyimpanan image firmware yang tidak ditandatangani.

`key`

Nama file gambar firmware Anda yang ditandatangani di bucket Amazon S3. Anda dapat menemukan nilai ini di konsol Amazon S3 dengan melihat isi bucket Anda.

Jika Anda menggunakan Penandatanganan Kode untuk AWS IoT, nama file adalah GUID yang dihasilkan oleh Penandatanganan Kode untuk AWS IoT.

Gunakan perintah create-stream AWS CLI untuk membuat aliran.

```
aws iot create-stream \
    --stream-id your_stream_id \
    --description your_description \
    --files file://stream.json \
    --role-arn your_role_arn
```

Ini adalah argumen untuk create-stream AWS CLI perintah:

`stream-id`

String arbitrer untuk mengidentifikasi aliran.

`description`

Deskripsi opsional aliran.

`files`

Satu atau lebih referensi ke file JSON yang berisi data tentang gambar firmware untuk streaming. File JSON harus berisi atribut berikut:

`fileId`

ID file arbitrer.

`s3Location`

Nama bucket tempat image firmware yang ditandatangani disimpan dan kunci (nama file) dari image firmware yang ditandatangani.

`bucket`

Bucket Amazon S3 tempat gambar firmware yang ditandatangani disimpan.

`key`

Kunci (nama file) dari gambar firmware yang ditandatangani.

Saat Anda menggunakan Penandatanganan Kode untuk AWS IoT, kunci ini adalah GUID.

Berikut ini adalah contoh `stream.json` file.

```
[
    {
        "fileId":123,
        "s3Location": {
            "bucket":"codesign-ota-bucket",
            "key":"48c67f3c-63bb-4f92-a98a-4ee0fbc2bef6"
        }
    }
]
```

`role-arn`

Peran layanan OTA yang juga memberikan akses ke bucket Amazon S3 tempat image firmware disimpan.

Untuk menemukan kunci objek Amazon S3 dari gambar firmware yang ditandatangani, gunakan aws signer describe-signing-job --job-id ***`my-job-id`*** perintah di mana `my-job-id` ID pekerjaan ditampilkan oleh perintah. create-signing-job AWS CLI Output dari describe-signing-job perintah berisi kunci gambar firmware yang ditandatangani.

```
... text deleted for brevity ...
  "signedObject": {
    "s3": {
      "bucketName": "ota-bucket",
      "key": "7309da2c-9111-48ac-8ee4-5a4262af4429"
    }
  }
... text deleted for brevity ...
```

Membuat pembaruan OTA

Gunakan create-ota-update AWS CLI perintah untuk membuat pekerjaan pembaruan OTA.

> **Note**
>
> Jangan gunakan informasi identitas pribadi (PII) apa pun di ID pekerjaan pembaruan OTA Anda. Contoh informasi yang dapat diidentifikasi secara pribadi meliputi:
>
> - Nama.
> - Alamat IP.
> - Alamat email.
> - Lokasi.
> - Rincian bank.
> - Informasi medis.

```
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`format

```
{
  "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`bidang

Nama	Tipe	Deskripsi
`otaUpdateId`	string (maks: 128 menit: 1)	ID pembaruan OTA yang akan dibuat.
`description`	string (maks: 2028)	Deskripsi pembaruan OTA.
`targets`	daftar	Perangkat ditargetkan untuk menerima pembaruan OTA.
`protocols`	daftar	Protokol yang digunakan untuk mentransfer gambar pembaruan OTA. Nilai yang valid adalah [HTTP], [MQTT], [HTTP, MQTT]. Ketika HTTP dan MQTT ditentuka n, perangkat target dapat memilih protokol.
`targetSelection`	string	Menentukan apakah update akan terus berjalan (CONTINUOUS), atau akan selesai setelah semua hal yang ditentukan sebagai target telah menyelesaikan update (SNAPSHOT). Jika terus menerus, pembaruan juga dapat dijalankan pada suatu hal ketika perubahan

Nama	Tipe	Deskripsi
		terdeteksi di target. Misalnya, pembaruan akan berjalan pada sesuatu ketika benda itu ditambahkan ke grup target, bahkan setelah pembaruan selesai oleh semua hal yang awalnya ada di grup. Nilai yang valid: CONTINUOUS | SNAPSHOT. enum: TERUS MENERUS | SNAPSHOT
`awsJobExecutionsRollo utConfig`		Konfigurasi untuk peluncuran pembaruan OTA.
`maximumPerMinute`	integer (maks: 1000 menit: 1)	Jumlah maksimum eksekusi pekerjaan pembaruan OTA dimulai per menit.
`exponentialRate`		Tingkat kenaikan untuk peluncuran pekerjaan. Parameter ini memungkin kan Anda untuk menentukan kenaikan tingkat eksponensial untuk peluncuran pekerjaan.
`baseRatePerMinute`	integer (maks: 1000 menit: 1)	Jumlah minimum hal yang akan diberitahukan tentang pekerjaan yang tertunda, per menit, pada awal peluncuran pekerjaan. Ini adalah tingkat awal peluncuran.

Nama	Tipe	Deskripsi
`rateIncreaseCriteria`		Kriteria untuk memulai peningkatan tingkat peluncuran untuk suatu pekerjaan. AWS IoT mendukung hingga satu digit setelah desimal (misalnya, 1,5, tetapi tidak 1,55).
`numberOfNotifiedThings`	integer (min: 1)	Ketika jumlah hal ini telah diberitahukan, itu akan memulai peningkatan tingkat peluncuran.
`numberOfSucceededThings`	integer (min: 1)	Ketika sejumlah hal ini telah berhasil dalam pelaksanaan pekerjaan mereka, itu akan memulai peningkatan tingkat peluncuran.
`awsJobPresignedUrlConfig`		Informasi konfigurasi untuk pra-ditandatangani URLs.
`expiresInSec`	long	Berapa lama (dalam detik) pra-ditandatangani URLs valid. Nilai yang valid adalah 60 - 3600, nilai default adalah 1800 detik. URLs Pra-ditandatangani dibuat ketika permintaan untuk dokumen pekerjaan diterima.
`awsJobAbortConfig`		Kriteria yang menentukan kapan dan bagaimana penghentian pekerjaan terjadi.

Nama	Tipe	Deskripsi
`abortCriteriaList`	daftar	Daftar kriteria yang menentukan kapan dan bagaimana menghentikan pekerjaan.
`failureType`	string	Jenis kegagalan eksekusi pekerjaan yang dapat memulai penghentian pekerjaan. enum: GAGAL | DITOLAK | TIMED_OUT | SEMUA
`action`	string	Jenis tindakan pekerjaan yang harus diambil untuk memulai penghentian pekerjaan. enum: BATALKAN
`minNumberOfExecutedThings`	integer (min: 1)	Jumlah minimum hal yang harus menerima pemberitahuan pelaksanaan pekerjaan sebelum pekerjaan dapat dihentikan.
`awsJobTimeoutConfig`		Menentukan jumlah waktu setiap perangkat harus menyelesaikan pelaksanaan pekerjaan. Timer dimulai saat status eksekusi pekerjaan disetel ke`IN_PROGRESS` . Jika status eksekusi pekerjaan tidak disetel ke status terminal lain sebelum timer kedaluwarsa, itu akan secara otomatis diatur ke`TIMED_OUT` .

Nama	Tipe	Deskripsi
`inProgressTimeoutInMinutes`	long	Menentukan jumlah waktu, dalam menit, perangkat ini harus menyelesaikan pelaksanaan pekerjaan ini. Interval batas waktu dapat berkisar antara 1 menit dan 7 hari (1 hingga 10080 menit). Timer yang sedang berlangsung tidak dapat diperbarui dan akan berlaku untuk semua eksekusi pekerjaan untuk pekerjaan itu. Setiap kali eksekusi pekerjaan tetap dalam status IN_PROGRESS lebih lama dari interval ini, eksekusi pekerjaan akan gagal dan beralih ke status terminal`TIMED_OUT` .
`files`	daftar	File yang akan dialirkan oleh pembaruan OTA.
`fileName`	string	Nama file.
`fileType`	integer rentang- maks: 255 menit: 0	Nilai integer yang dapat Anda sertakan dalam dokumen pekerjaan untuk memungkinkan perangkat Anda mengidentifikasi jenis file yang diterima dari cloud.
`fileVersion`	string	Versi file.
`fileLocation`		Lokasi firmware yang diperbarui.

Nama	Tipe	Deskripsi
stream		Aliran yang berisi pembaruan OTA.
streamId	string (maks: 128 menit: 1)	ID stream.
fileId	integer (maks: 255 menit: 0)	ID file yang terkait dengan aliran.
s3Location		Lokasi firmware yang diperbarui di S3.
bucket	string (min: 1)	Ember S3.
key	string (min: 1)	Tombol S3.
version	string	Versi bucket S3.
codeSigning		Metode penandatanganan kode file.
awsSignerJobId	string	ID dari AWSSigner Job yang dibuat untuk menandatangani file.
startSigningJobParameter		Menjelaskan pekerjaan penandatanganan kode.
signingProfileParameter		Menjelaskan profil penandatanganan kode.
certificateArn	string	Sertifikat ARN.

Nama	Tipe	Deskripsi
`platform`	string	Platform perangkat keras perangkat Anda.
`certificatePathOnDevice`	string	Lokasi sertifikat penandatanganan kode di perangkat Anda.
`signingProfileName`	string	Nama profil penandatanganan kode.
`destination`		Lokasi untuk menulis file yang ditandatangani kode.
`s3Destination`		Menjelaskan lokasi di S3 dari firmware yang diperbarui.
`bucket`	string (min: 1)	Bucket S3 yang berisi firmware yang diperbarui.
`prefix`	string	Awalan S3.
`customCodeSigning`		Metode khusus untuk penandatanganan kode file.
`signature`		Tanda tangan untuk file tersebut.
`inlineDocument`	blob	Representasi biner yang dikodekan base64 dari tanda tangan penandatanganan kode.
`certificateChain`		Rantai sertifikat.
`certificateName`	string	Nama sertifikat.

Nama	Tipe	Deskripsi
`inlineDocument`	string	Representasi biner yang dikodekan base64 dari rantai sertifikat penandatanganan kode.
`hashAlgorithm`	string	Algoritma hash yang digunakan untuk kode menandatangani file.
`signatureAlgorithm`	string	Algoritma tanda tangan yang digunakan untuk kode menandatangani file.
`attributes`	map	Daftar pasangan nama/atribut.
`roleArn`	string (maks: 2048 menit: 20)	Peran IAM yang memberikan AWS IoT akses ke Amazon S3 AWS IoT , pekerjaan, AWS dan sumber daya Penandatanganan Kode untuk membuat pekerjaan pembaruan OTA.
`additionalParameters`	map	Daftar parameter pembaruan OTA tambahan yang merupakan pasangan nama-nilai.
`tags`	daftar	Metadata yang dapat digunakan untuk mengelola pembaruan.
`Key`	string (maks: 128 menit: 1)	Kunci tag.

Nama	Tipe	Deskripsi
`Value`	string (maks: 256 menit: 1)	Nilai tag.

Output

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

AWS CLI bidang keluaran

Nama	Tipe	Deskripsi
`otaUpdateId`	string (maks: 128 menit: 1)	ID pembaruan OTA.
`awsIotJobId`	string	ID AWS IoT pekerjaan yang terkait dengan pembaruan OTA.
`otaUpdateArn`	string	Pembaruan OTA ARN.
`awsIotJobArn`	string	AWS IoT Pekerjaan ARN terkait dengan pembaruan OTA.
`otaUpdateStatus`	string	Status pembaruan OTA. enum: CREATE_PENDING | CREATE_IN_PROGRESS | CREATE_COMPLETE | CREATE_FAILED

Berikut ini adalah contoh file JSON yang diteruskan ke create-ota-update perintah yang menggunakan Code Signing for AWS IoT.

```
[
  {
    "fileName": "firmware.bin",
    "fileType": 1,
    "fileLocation": {
      "stream": {
        "streamId": "004",
        "fileId":123
      }
    },
    "codeSigning": {
      "awsSignerJobId": "48c67f3c-63bb-4f92-a98a-4ee0fbc2bef6"
    }
  }
]
```

Berikut ini adalah contoh file JSON yang diteruskan ke create-ota-update AWS CLI perintah yang menggunakan file inline untuk menyediakan materi penandatanganan kode khusus.

```
[
  {
    "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"
```

```
      }
    }
  }
]
```

Berikut ini adalah contoh file JSON yang diteruskan ke create-ota-update AWS CLI perintah yang memungkinkan FreerTOS OTA untuk memulai pekerjaan penandatanganan kode dan membuat profil dan streaming penandatanganan kode.

```
[
  {
    "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"
          }
        }
      }
    }
  }
]
```

Berikut ini adalah contoh file JSON yang diteruskan ke create-ota-update AWS CLI perintah yang membuat pembaruan OTA yang memulai pekerjaan penandatanganan kode dengan profil yang ada dan menggunakan aliran yang ditentukan.

```
[
  {
    "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"
          }
        }
      }
    }
  }
]
```

Berikut ini adalah contoh file JSON yang diteruskan ke create-ota-update AWS CLI perintah yang memungkinkan FreeRTOS OTA untuk membuat aliran dengan ID pekerjaan penandatanganan kode yang ada.

```
[
  {
    "fileName": "your_firmware_path_on_device",
    "fileType": 1,
    "fileVersion": "1",
    "codeSigning":{
      "awsSignerJobId": "your_signer_job_id"
    }
  }
]
```

Berikut ini adalah contoh file JSON yang diteruskan ke create-ota-update AWS CLI perintah yang membuat pembaruan OTA. Pembaruan membuat aliran dari objek S3 yang ditentukan dan menggunakan penandatanganan kode khusus.

```
[
  {
    "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"
      }
    }
  }
]
```

Daftar pembaruan OTA

Anda dapat menggunakan list-ota-updates AWS CLI perintah untuk mendapatkan daftar semua pembaruan OTA.

```
aws iot list-ota-updates
```

Output dari list-ota-updates perintah terlihat seperti ini.

```
{
```

```
  "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
    }
  ]
}
```

Mendapatkan informasi tentang pembaruan OTA

Anda dapat menggunakan get-ota-update AWS CLI perintah untuk mendapatkan status pembuatan atau penghapusan pembaruan OTA.

```
aws iot get-ota-update --ota-update-id your-ota-update-id
```

Output dari get-ota-update perintah terlihat seperti berikut ini.

```
{
    "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"
    }
}
```

Nilai yang dikembalikan untuk `otaUpdateStatus` meliputi yang berikut:

`CREATE_PENDING`

Pembuatan pembaruan OTA sedang menunggu keputusan.

`CREATE_IN_PROGRESS`

Pembaruan OTA sedang dibuat.

`CREATE_COMPLETE`

Pembaruan OTA telah dibuat.

`CREATE_FAILED`

Pembuatan pembaruan OTA gagal.

`DELETE_IN_PROGRESS`

Pembaruan OTA sedang dihapus.

`DELETE_FAILED`

Penghapusan pembaruan OTA gagal.

> **Note**
>
> Untuk mendapatkan status eksekusi pembaruan OTA setelah dibuat, Anda perlu menggunakan describe-job-execution perintah. Untuk informasi selengkapnya, lihat Menjelaskan Eksekusi Pekerjaan.

Menghapus data terkait OTA

Saat ini, Anda tidak dapat menggunakan AWS IoT konsol untuk menghapus aliran atau pembaruan OTA. Anda dapat menggunakan file AWS CLI untuk menghapus aliran, pembaruan OTA, dan AWS IoT pekerjaan yang dibuat selama pembaruan OTA.

Menghapus aliran OTA

Saat Anda membuat pembaruan OTA yang menggunakan MQTT, Anda dapat menggunakan baris perintah atau AWS IoT konsol untuk membuat aliran untuk memecah firmware menjadi beberapa bagian sehingga dapat dikirim melalui MQTT. Anda dapat menghapus aliran ini dengan delete-stream AWS CLI perintah, seperti yang ditunjukkan pada contoh berikut.

```
aws iot delete-stream --stream-id your_stream_id
```

Menghapus pembaruan OTA

Saat Anda membuat pembaruan OTA, berikut ini dibuat:

- Entri dalam database pekerjaan pembaruan OTA.
- AWS IoT Pekerjaan untuk melakukan pembaruan.
- Eksekusi AWS IoT pekerjaan untuk setiap perangkat yang diperbarui.

delete-ota-updatePerintah menghapus entri di database pekerjaan pembaruan OTA saja. Anda harus menggunakan delete-job perintah untuk menghapus AWS IoT pekerjaan.

Gunakan delete-ota-update perintah untuk menghapus pembaruan OTA.

```
aws iot delete-ota-update --ota-update-id your_ota_update_id
```

`ota-update-id`

ID pembaruan OTA untuk dihapus.

`delete-stream`

Menghapus aliran yang terkait dengan pembaruan OTA.

`force-delete-aws-job`

Menghapus AWS IoT pekerjaan yang terkait dengan pembaruan OTA. Jika bendera ini tidak disetel dan pekerjaan dalam `In_Progress` status, pekerjaan tidak dihapus.

Menghapus pekerjaan IoT yang dibuat untuk pembaruan OTA

FreeRTOS menciptakan pekerjaan saat Anda membuat AWS IoT pembaruan OTA. Eksekusi pekerjaan juga dibuat untuk setiap perangkat yang memproses pekerjaan. Anda dapat menggunakan delete-job AWS CLI perintah untuk menghapus pekerjaan dan eksekusi pekerjaan yang terkait.

```
aws iot delete-job --job-id your-job-id --no-force
```

`no-force`Parameter menentukan bahwa hanya pekerjaan yang berada dalam status terminal (SELESAI atau DIBATALKAN) yang dapat dihapus. Anda dapat menghapus pekerjaan yang berada dalam keadaan non-terminal dengan meneruskan `force` parameter. Untuk informasi selengkapnya, lihat DeleteJob API.

> **Note**
>
> Menghapus pekerjaan dengan status IN_PROGRESS akan mengganggu eksekusi pekerjaan apa pun yang IN_PROGRESS di perangkat Anda dan dapat mengakibatkan perangkat dibiarkan dalam status nondeterministik. Pastikan bahwa setiap perangkat yang menjalankan pekerjaan yang telah dihapus dapat pulih ke keadaan yang diketahui.

Bergantung pada jumlah eksekusi pekerjaan yang dibuat untuk pekerjaan itu dan faktor lainnya, perlu beberapa menit untuk menghapus pekerjaan. Saat pekerjaan sedang dihapus, statusnya adalah DELETION_IN_PROGRESS. Mencoba menghapus atau membatalkan pekerjaan yang statusnya sudah DELETION_IN_PROGRESS menghasilkan kesalahan.

Anda dapat menggunakan delete-job-execution untuk menghapus eksekusi pekerjaan. Anda mungkin ingin menghapus eksekusi pekerjaan ketika sejumlah kecil perangkat tidak dapat memproses pekerjaan. Ini menghapus eksekusi pekerjaan untuk satu perangkat, seperti yang ditunjukkan pada contoh berikut.

```
aws iot delete-job-execution --job-id your-job-id --thing-name
                    your-thing-name --execution-number your-job-execution-number --no-
force
```

Seperti halnya delete-job AWS CLI perintah, Anda dapat meneruskan `--force` parameter delete-job-execution ke untuk memaksa penghapusan eksekusi pekerjaan. Untuk informasi selengkapnya, lihat DeleteJobExecutionAPI.

> **Note**
>
> Menghapus eksekusi pekerjaan dengan status IN_PROGRESS akan mengganggu eksekusi pekerjaan apa pun yang IN_PROGRESS di perangkat Anda dan dapat mengakibatkan perangkat dibiarkan dalam status nondeterministik. Pastikan bahwa setiap perangkat yang menjalankan pekerjaan yang telah dihapus dapat pulih ke keadaan yang diketahui.

Untuk informasi selengkapnya tentang menggunakan aplikasi demo pembaruan OTA, lihatOver-the-air memperbarui aplikasi demo.

Layanan Manajer Pembaruan OTA

Layanan Update Manager over-the-air (OTA) menyediakan cara untuk:

- Buat pembaruan OTA dan sumber daya yang digunakannya, termasuk AWS IoT pekerjaan, AWS IoT aliran, dan penandatanganan kode.
- Dapatkan informasi tentang pembaruan OTA.
- Buat daftar semua pembaruan OTA yang terkait dengan AWS akun Anda.
- Hapus pembaruan OTA.

Pembaruan OTA adalah struktur data yang dikelola oleh layanan OTA Update Manager. Ini berisi:

- ID pembaruan OTA.
- Deskripsi pembaruan OTA opsional.
- Daftar perangkat yang akan diperbarui (target).
- Jenis pembaruan OTA: CONTINUOUS atau SNAPSHOT. Lihat bagian Pekerjaan di Panduan AWS IoT Pengembang untuk diskusi tentang jenis pembaruan yang Anda butuhkan.
- Protokol yang digunakan untuk melakukan pembaruan OTA: [MQTT], [HTTP] atau [MQTT, HTTP]. Saat Anda menentukan MQTT dan HTTP, pengaturan perangkat menentukan protokol yang digunakan.
- Daftar file untuk dikirim ke perangkat target.
- Peran IAM yang memberikan AWS IoT akses ke Amazon S3 AWS IoT , pekerjaan, AWS dan sumber daya Penandatanganan Kode untuk membuat pekerjaan pembaruan OTA.
- Daftar opsional pasangan nama-nilai yang ditentukan pengguna.

Pembaruan OTA dirancang untuk memperbarui firmware perangkat, tetapi Anda dapat menggunakannya untuk mengirim file apa pun yang Anda inginkan ke satu atau beberapa perangkat yang terdaftar AWS IoT. Saat Anda mengirim pembaruan firmware melalui udara, kami sarankan Anda menandatanganinya secara digital sehingga perangkat yang menerimanya dapat memverifikasi bahwa mereka belum dirusak dalam perjalanan.

Anda dapat mengirim gambar firmware yang diperbarui menggunakan protokol HTTP atau MQTT, tergantung pada pengaturan yang Anda pilih. Anda dapat menandatangani pembaruan firmware Anda dengan Penandatanganan Kode untuk FreeRTOS atau Anda dapat menggunakan alat penandatanganan kode Anda sendiri.

Untuk kontrol lebih lanjut atas proses, Anda dapat menggunakan CreateStreamAPI untuk membuat aliran saat mengirim pembaruan melalui MQTT. Dalam beberapa kasus, Anda dapat memodifikasi kode Agen FreeRTOS untuk menyesuaikan ukuran blok yang Anda kirim dan terima.

Saat Anda membuat pembaruan OTA, layanan Manajer OTA membuat AWS IoT pekerjaan untuk memberi tahu perangkat Anda bahwa pembaruan tersedia. Agen FreeRTOS OTA berjalan di perangkat Anda dan mendengarkan pesan pembaruan. Ketika pembaruan tersedia, ia meminta gambar pembaruan firmware melalui HTTP atau MQTT dan menyimpan file secara lokal. Ini memeriksa tanda tangan digital dari file yang diunduh dan, jika valid, menginstal pembaruan firmware. Jika Anda tidak menggunakan FreeRTOS, Anda harus menerapkan Agen OTA Anda sendiri untuk mendengarkan dan mengunduh pembaruan dan melakukan operasi instalasi apa pun.

Mengintegrasikan Agen OTA ke dalam aplikasi Anda

Agen over-the-air (OTA) dirancang untuk menyederhanakan jumlah kode yang harus Anda tulis untuk menambahkan fungsionalitas pembaruan OTA ke produk Anda. Beban integrasi itu terutama terdiri dari inisialisasi Agen OTA dan membuat fungsi panggilan balik khusus untuk menanggapi pesan peristiwa Agen OTA. Selama inisialisasi OS, MQTT, HTTP (jika HTTP digunakan untuk unduhan file) dan antarmuka implementasi khusus platform (PAL) diteruskan ke Agen OTA. Buffer juga dapat diinisialisasi dan diteruskan ke Agen OTA.

> **Note**
>
> Meskipun integrasi fitur pembaruan OTA ke dalam aplikasi Anda agak sederhana, sistem pembaruan OTA memerlukan pemahaman lebih dari sekadar integrasi kode perangkat. Untuk membiasakan diri dengan cara mengonfigurasi AWS akun Anda dengan AWS IoT berbagai hal, kredensi, sertifikat penandatanganan kode, perangkat penyediaan, dan pekerjaan pembaruan OTA, lihat Prasyarat FreeRTOS.

Manajemen koneksi

Agen OTA menggunakan protokol MQTT untuk semua operasi komunikasi kontrol yang melibatkan AWS IoT layanan, tetapi tidak mengelola koneksi MQTT. Untuk memastikan bahwa Agen OTA tidak mengganggu kebijakan manajemen koneksi aplikasi Anda, koneksi MQTT (termasuk pemutusan sambungan dan fungsionalitas penyambungan kembali apa pun) harus ditangani oleh aplikasi pengguna utama. File dapat diunduh melalui protokol MQTT atau HTTP. Anda dapat memilih protokol mana saat Anda membuat pekerjaan OTA. Jika Anda memilih MQTT, Agen OTA menggunakan koneksi yang sama untuk operasi kontrol dan untuk mengunduh file.

Demo OTA sederhana

Berikut ini adalah kutipan dari demo OTA sederhana yang menunjukkan kepada Anda bagaimana Agen terhubung ke broker MQTT dan menginisialisasi Agen OTA. Dalam contoh ini, kami mengonfigurasi demo untuk menggunakan callback aplikasi OTA default dan mengembalikan beberapa statistik sekali per detik. Untuk singkatnya, kami meninggalkan beberapa detail dari demo ini.

Demo OTA juga menunjukkan manajemen koneksi MQTT dengan memantau panggilan balik pemutusan dan membangun kembali koneksi. Ketika pemutusan terjadi, demo pertama-tama menangguhkan operasi Agen OTA dan kemudian mencoba membangun kembali koneksi MQTT. Upaya penyambungan ulang MQTT ditunda oleh waktu yang meningkat secara eksponensial hingga nilai maksimum dan jitter juga ditambahkan. Jika koneksi dibangun kembali, Agen OTA melanjutkan operasinya.

Untuk contoh kerja yang menggunakan broker AWS IoT MQTT, lihat kode demo OTA di direktori. `demos/ota`

Karena Agen OTA adalah tugasnya sendiri, penundaan satu detik yang disengaja dalam contoh ini hanya memengaruhi aplikasi ini. Ini tidak berdampak pada kinerja Agen.

```
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;
}
```

Berikut adalah aliran tingkat tinggi dari aplikasi demo ini:

- Buat konteks Agen MQTT.
- Connect ke AWS IoT endpoint Anda.
- Inisialisasi Agen OTA.
- Loop yang memungkinkan pekerjaan pembaruan OTA dan menghasilkan statistik sekali dalam satu detik.
- Jika MQTT terputus, hentikan operasi Agen OTA.
- Coba sambungkan lagi dengan penundaan eksponensial dan jitter.
- Jika terhubung kembali, lanjutkan operasi Agen OTA.
- Jika Agen berhenti, tunda satu detik, lalu coba sambungkan kembali.

Menggunakan panggilan balik aplikasi untuk acara Agen OTA

Contoh sebelumnya digunakan `prvOtaAppCallback` sebagai handler callback untuk event Agen OTA. (Lihat parameter keempat untuk panggilan `OTA_Init` API). Jika Anda ingin menerapkan penanganan khusus dari peristiwa penyelesaian, Anda harus mengubah penanganan default di demo/aplikasi OTA. Selama proses OTA, Agen OTA dapat mengirim salah satu enum acara berikut ke handler callback. Terserah pengembang aplikasi untuk memutuskan bagaimana dan kapan menangani peristiwa ini.

```
/**
 * @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;
```

Agen OTA dapat menerima pembaruan di latar belakang selama pemrosesan aktif aplikasi utama. Tujuan penyampaian acara ini adalah untuk memungkinkan aplikasi memutuskan apakah tindakan dapat diambil segera atau jika harus ditangguhkan sampai setelah selesainya beberapa pemrosesan khusus aplikasi lainnya. Ini mencegah gangguan yang tidak terduga pada perangkat Anda selama pemrosesan aktif (misalnya, menyedot debu) yang akan disebabkan oleh reset setelah pembaruan firmware. Berikut adalah peristiwa pekerjaan yang diterima oleh handler callback:

`OtaJobEventActivate`

Saat penangan callback menerima acara ini, Anda dapat segera mengatur ulang perangkat atau menjadwalkan panggilan untuk mengatur ulang perangkat nanti. Ini memungkinkan Anda untuk menunda fase reset perangkat dan uji mandiri, jika perlu.

`OtaJobEventFail`

Ketika handler callback menerima acara ini, pembaruan gagal. Anda tidak perlu melakukan apa pun dalam kasus ini. Anda mungkin ingin menampilkan pesan log atau melakukan sesuatu yang spesifik untuk aplikasi.

`OtaJobEventStartTest`

Fase self-test dimaksudkan untuk memungkinkan firmware yang baru diperbarui untuk mengeksekusi dan menguji dirinya sendiri sebelum menentukan apakah berfungsi dengan baik dan berkomitmen untuk menjadi gambar aplikasi permanen terbaru. Ketika pembaruan baru diterima dan diautentikasi dan perangkat telah disetel ulang, Agen OTA mengirimkan `OtaJobEventStartTest` acara ke fungsi panggilan balik saat siap untuk pengujian. Pengembang dapat menambahkan tes yang diperlukan untuk menentukan apakah firmware perangkat berfungsi dengan baik setelah pembaruan. Ketika firmware perangkat dianggap dapat diandalkan oleh pengujian mandiri, kode harus memasukkan firmware sebagai gambar permanen baru dengan memanggil `OTA_SetImageState( OtaImageStateAccepted )` fungsi.

`OtaJobEventProcessed`

Acara OTA yang diantri `OTA_SignalEvent` diproses, sehingga operasi pembersihan seperti membebaskan buffer OTA dapat dilakukan.

`OtaJobEventSelfTestFailed`

Tes mandiri OTA gagal untuk pekerjaan saat ini. Penanganan default untuk acara ini adalah mematikan Agen OTA dan memulai ulang sehingga perangkat memutar kembali ke gambar sebelumnya.

`OtaJobEventUpdateComplete`

Acara pemberitahuan untuk penyelesaian pembaruan pekerjaan OTA.

Keamanan OTA

Berikut ini adalah tiga aspek keamanan over-the-air (OTA):

Keamanan koneksi

Layanan OTA Update Manager bergantung pada mekanisme keamanan yang ada, seperti otentikasi timbal balik Transport Layer Security (TLS), yang digunakan oleh. AWS IoT Lalu lintas pembaruan OTA melewati gateway AWS IoT perangkat dan menggunakan mekanisme AWS IoT keamanan. Setiap pesan HTTP atau MQTT yang masuk dan keluar melalui gateway perangkat mengalami otentikasi dan otorisasi yang ketat.

Keaslian dan integritas pembaruan OTA

Firmware dapat ditandatangani secara digital sebelum pembaruan OTA untuk memastikan bahwa itu berasal dari sumber yang dapat diandalkan dan belum dirusak.

Layanan FreerTOS OTA Update Manager menggunakan Penandatanganan Kode AWS IoT untuk menandatangani firmware Anda secara otomatis. Untuk informasi selengkapnya, lihat Penandatanganan Kode untuk AWS IoT.

Agen OTA, yang berjalan di perangkat Anda, melakukan pemeriksaan integritas pada firmware saat tiba di perangkat.

Keamanan operator

Setiap panggilan API yang dilakukan melalui API bidang kontrol menjalani autentikasi dan otorisasi IAM Signature Version 4 standar. Untuk membuat penerapan, Anda harus memiliki izin untuk memanggil`CreateDeployment`,, `CreateJob` dan. `CreateStream` APIs Selain itu, dalam kebijakan bucket Amazon S3 atau ACL Anda, Anda harus memberikan izin baca kepada kepala AWS IoT layanan sehingga pembaruan firmware yang disimpan di Amazon S3 dapat diakses selama streaming.

Penandatanganan Kode untuk AWS IoT

AWS IoT Konsol menggunakan Penandatanganan Kode AWS IoT untuk secara otomatis menandatangani gambar firmware Anda untuk perangkat apa pun yang didukung oleh AWS IoT.

Penandatanganan Kode untuk AWS IoT menggunakan sertifikat dan kunci pribadi yang Anda impor ke ACM. Anda dapat menggunakan sertifikat yang ditandatangani sendiri untuk pengujian, tetapi kami menyarankan Anda mendapatkan sertifikat dari otoritas sertifikat komersial (CA) yang terkenal.

Sertifikat penandatanganan kode menggunakan X.509 versi 3 dan ekstensi. `Key Usage` `Extended Key Usage` `Key Usage`Ekstensi diatur ke `Digital Signature` dan `Extended Key Usage` ekstensi diatur ke`Code Signing`. Untuk informasi selengkapnya tentang menandatangani image kode, lihat Panduan Penandatanganan Kode untuk AWS IoT Developer dan Penandatanganan Kode untuk Referensi AWS IoT API.

> **Note**
>
> Anda dapat mengunduh Penandatanganan Kode untuk AWS IoT SDK dari Tools for Amazon Web Services.

Pemecahan masalah OTA

Bagian berikut berisi informasi untuk membantu Anda memecahkan masalah dengan pembaruan OTA.

Topik

- Siapkan CloudWatch Log untuk pembaruan OTA
- Log panggilan API AWS IoT OTA dengan AWS CloudTrail
- Dapatkan Buat detail OTAUpdate kegagalan menggunakan AWS CLI
- Dapatkan kode kegagalan OTA dengan AWS CLI
- Memecahkan masalah pembaruan OTA dari beberapa perangkat
- Memecahkan masalah pembaruan OTA dengan Launchpad Texas CC322 Instruments 0SF

Siapkan CloudWatch Log untuk pembaruan OTA

Layanan Pembaruan OTA mendukung pencatatan dengan Amazon CloudWatch. Anda dapat menggunakan AWS IoT konsol untuk mengaktifkan dan mengonfigurasi CloudWatch pencatatan Amazon untuk pembaruan OTA. Untuk informasi selengkapnya, lihat Log Cloudwatch.

Untuk mengaktifkan logging, Anda harus membuat peran IAM dan mengonfigurasi logging pembaruan OTA.

> **Note**
>
> Sebelum Anda mengaktifkan pencatatan pembaruan OTA, pastikan Anda memahami izin akses CloudWatch Log. Pengguna dengan akses ke CloudWatch Log dapat melihat informasi debugging Anda. Untuk selengkapnya, lihat Otentikasi dan Kontrol Akses untuk CloudWatch Log Amazon.

Buat peran logging dan aktifkan logging

Gunakan AWS IoT konsol untuk membuat peran logging dan mengaktifkan logging.

1. Dari panel navigasi, pilih Pengaturan.
2. Di bawah Log, pilih Edit.
3. Di bawah Tingkat verbositas, pilih Debug.
4. Di bawah Tetapkan peran, pilih Buat baru untuk membuat peran IAM untuk logging.
5. Di bawah Nama, masukkan nama unik untuk peran Anda. Peran Anda akan dibuat dengan semua izin yang diperlukan.
6. Pilih Perbarui.

Log pembaruan OTA

Layanan Pembaruan OTA menerbitkan log ke akun Anda ketika salah satu hal berikut terjadi:

- Pembaruan OTA dibuat.
- Pembaruan OTA selesai.
- Pekerjaan penandatanganan kode dibuat.

- Pekerjaan penandatanganan kode selesai.
- Sebuah AWS IoT pekerjaan dibuat.
- Sebuah AWS IoT pekerjaan selesai.
- Aliran dibuat.

Anda dapat melihat log Anda di CloudWatch konsol.

Untuk melihat pembaruan OTA di CloudWatch Log

1. Dari panel navigasi, pilih Log.
2. Di Grup Log, pilih AWSIoTLogsV2.

Log pembaruan OTA dapat berisi properti berikut:

accountId

ID AWS akun tempat log dibuat.

actionType

Tindakan yang menghasilkan log. Ini dapat diatur ke salah satu nilai berikut:

- `CreateOTAUpdate`: Pembaruan OTA telah dibuat.
- `DeleteOTAUpdate`: Pembaruan OTA telah dihapus.
- `StartCodeSigning`: Pekerjaan penandatanganan kode dimulai.
- `CreateAWSJob`: Sebuah AWS IoT pekerjaan telah dibuat.
- `CreateStream`: Aliran telah dibuat.
- `GetStream`: Permintaan untuk aliran dikirim ke fitur pengiriman file AWS IoT berbasis MQTT.
- `DescribeStream`: Permintaan informasi tentang aliran dikirim ke fitur pengiriman file AWS IoT berbasis MQTT.

awsJobId

ID AWS IoT pekerjaan yang menghasilkan log.

clientId

ID klien MQTT yang membuat permintaan yang menghasilkan log.

clientToken

Token klien terkait dengan permintaan yang menghasilkan log.

detail

Informasi lebih lanjut tentang operasi yang menghasilkan log.

logLevel

Tingkat logging log. Untuk log pembaruan OTA, ini selalu disetel keDEBUG.

otaUpdateId

ID pembaruan OTA yang menghasilkan log.

protokol

Protokol yang digunakan untuk membuat permintaan yang menghasilkan log.

status

Status operasi yang menghasilkan log. Nilai yang valid adalah:

- Berhasil
- Kegagalan

streamId

ID AWS IoT aliran yang menghasilkan log.

timestamp

Waktu ketika log dihasilkan.

topicName

Topik MQTT yang digunakan untuk membuat permintaan yang menghasilkan log.

Contoh log

Berikut ini adalah contoh log yang dihasilkan saat pekerjaan penandatanganan kode dimulai:

```
{
    "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."
}
```

Berikut ini adalah contoh log yang dihasilkan saat AWS IoT pekerjaan dibuat:

```
{
    "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."
}
```

Berikut ini adalah contoh log yang dihasilkan saat pembaruan OTA dibuat:

```
{
    "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."
}
```

Berikut ini adalah contoh log yang dihasilkan ketika aliran dibuat:

```
{
    "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."
}
```

Berikut ini adalah contoh log yang dihasilkan saat pembaruan OTA dihapus:

```
{
    "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."
}
```

Berikut ini adalah contoh log yang dihasilkan saat perangkat meminta aliran dari fitur pengiriman file berbasis 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."
}
```

Berikut ini adalah contoh log yang dihasilkan saat perangkat memanggil `DescribeStream` API:

```
{
    "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."
}
```

Log panggilan API AWS IoT OTA dengan AWS CloudTrail

FreeRTOS terintegrasi CloudTrail dengan, layanan yang AWS IoT menangkap panggilan API OTA dan mengirimkan file log ke bucket Amazon S3 yang Anda tentukan. CloudTrail menangkap panggilan API dari kode Anda ke AWS IoT OTA APIs. Dengan menggunakan informasi yang dikumpulkan oleh CloudTrail, Anda dapat menentukan permintaan yang dibuat untuk AWS IoT OTA, alamat IP sumber dari mana permintaan dibuat, siapa yang membuat permintaan, kapan dibuat, dan sebagainya.

Untuk informasi selengkapnya CloudTrail, termasuk cara mengonfigurasi dan mengaktifkannya, lihat Panduan AWS CloudTrail Pengguna.

Informasi FreeRTOS di CloudTrail

Saat CloudTrail logging diaktifkan di AWS akun Anda, panggilan API yang dilakukan ke tindakan AWS IoT OTA dilacak dalam file CloudTrail log tempat mereka ditulis dengan catatan AWS layanan lainnya. CloudTrail menentukan kapan harus membuat dan menulis ke file baru berdasarkan periode waktu dan ukuran file.

Tindakan bidang kontrol AWS IoT OTA berikut dicatat oleh CloudTrail:

- CreateStream
- DescribeStream
- ListStreams
- UpdateStream
- DeleteStream
- Buat OTAUpdate
- Dapatkan OTAUpdate
- Daftar OTAUpdates

- Hapus OTAUpdate

> **Note**
>
> AWS IoT Tindakan bidang data OTA (sisi perangkat) tidak dicatat oleh CloudTrail. Gunakan CloudWatch untuk memantau ini.

Setiap entri log berisi informasi tentang siapa yang membuat permintaan tersebut. Informasi identitas pengguna dalam entri log membantu Anda menentukan hal berikut:

- Apakah permintaan tersebut dibuat dengan kredensial root atau pengguna IAM.
- Apakah permintaan tersebut dibuat dengan kredensial keamanan sementara untuk satu peran atau pengguna terfederasi.
- Apakah permintaan itu dibuat oleh AWS layanan lain.

Untuk informasi selengkapnya, lihat Elemen CloudTrail UserIdentity. AWS IoT Tindakan OTA didokumentasikan dalam Referensi API AWS IoT OTA.

Anda dapat menyimpan berkas log dalam bucket Amazon S3 selama yang diinginkan, tetapi Anda juga dapat menentukan aturan siklus hidup Amazon S3 untuk mengarsipkan atau menghapus berkas log secara otomatis. Secara default, berkas log Anda dienkripsi dengan menggunakan enkripsi sisi server (SSE) Amazon S3.

Jika Anda ingin diberi tahu saat file log dikirimkan, Anda dapat mengonfigurasi CloudTrail untuk mempublikasikan notifikasi Amazon SNS. Untuk informasi selengkapnya, lihat Mengonfigurasi Notifikasi Amazon SNS untuk. CloudTrail

Anda juga dapat menggabungkan file log AWS IoT OTA dari beberapa AWS Wilayah dan beberapa AWS akun ke dalam satu bucket Amazon S3.

Untuk informasi selengkapnya, lihat Menerima File CloudTrail Log dari Beberapa Wilayah dan Menerima File CloudTrail Log dari Beberapa Akun.

Memahami entri file log FreeRTOS

CloudTrail file log dapat berisi satu atau lebih entri log. Setiap entri berisi beberapa peristiwa yang diformat JSON. Entri log mewakili satu permintaan dari sumber mana pun dan mencakup informasi

tentang tindakan yang diminta, tanggal dan waktu tindakan, parameter permintaan, dan sebagainya. Entri log bukan jejak tumpukan yang dipesan dari panggilan API publik, sehingga tidak muncul dalam urutan tertentu.

Contoh berikut menunjukkan entri CloudTrail log yang menunjukkan log dari ajakan untuk `CreateOTAUpdate` bertindak.


```
{
    "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"
}
```

Dapatkan Buat detail OTAUpdate kegagalan menggunakan AWS CLI

Jika proses pembuatan pekerjaan pembaruan OTA gagal, mungkin ada tindakan yang dapat Anda ambil untuk memperbaiki masalah tersebut. Saat Anda membuat pekerjaan pembaruan OTA, layanan manajer OTA membuat pekerjaan IoT dan menjadwalkannya untuk perangkat target, dan proses ini juga membuat atau menggunakan jenis AWS sumber daya lain di akun Anda (pekerjaan penandatanganan kode, aliran, AWS IoT objek Amazon S3). Setiap kesalahan yang ditemui dapat menyebabkan proses gagal tanpa membuat AWS IoT pekerjaan. Di bagian pemecahan masalah ini kami memberikan instruksi tentang cara mengambil detail kegagalan.

1. Instal dan konfigurasikan AWS CLI.
2. Jalankan `aws configure` dan masukkan informasi berikut.

   ```
   $ aws configure
   AWS Access Key ID [None]: AccessID
   AWS Secret Access Key [None]: AccessKey
   Default region name [None]: Region
   Default output format [None]: json
   ```

   Untuk informasi selengkapnya, lihat Konfigurasi cepat dengan `aws configure`.
3. Jalankan:

```
aws iot get-ota-update --ota-update-id ota_update_job_001
```

Di *ota_update_job_001* mana ID yang Anda berikan pembaruan OTA saat Anda membuatnya.

4. Outputnya akan terlihat seperti ini:

```
{
    "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)"
        }
    }
}
```

Jika create gagal, `otaUpdateStatus` bidang dalam output perintah akan berisi `CREATE_FAILED` dan `errorInfo` bidang akan berisi rincian kegagalan.

Dapatkan kode kegagalan OTA dengan AWS CLI

1. Instal dan konfigurasikan AWS CLI.
2. Jalankan `aws configure` dan masukkan informasi berikut.

```
$ aws configure
AWS Access Key ID [None]: AccessID
AWS Secret Access Key [None]: AccessKey
Default region name [None]: Region
Default output format [None]: json
```

Untuk informasi selengkapnya, lihat Konfigurasi cepat dengan `aws configure`.

3. Jalankan:

```
aws iot describe-job-execution --job-id JobID --thing-name ThingName
```

Di *`JobID`* mana string ID pekerjaan lengkap untuk pekerjaan yang statusnya ingin kami dapatkan (dikaitkan dengan pekerjaan pembaruan OTA saat dibuat) dan *`ThingName`* merupakan nama AWS IoT benda yang didaftarkan perangkat seperti di AWS IoT

4. Outputnya akan terlihat seperti ini:

```
{
    "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
    }
}
```

Dalam contoh keluaran ini, "" di "" `reason` `detailsmap` "memiliki dua bidang: bidang yang ditampilkan sebagai “0xEEEEEEEEEE” berisi kode kesalahan generik dari Agen OTA; bidang yang ditampilkan sebagai “0xffffffff” berisi sub-kode. Kode kesalahan generik tercantum dalam https://docs.aws.amazon.com/freertos/latest/lib-ref/html1/aws__ota__agent_8h.html. Lihat kode kesalahan dengan awalan "`kOTA_Err_`”. Sub-kode dapat berupa kode khusus platform atau memberikan rincian lebih lanjut tentang kesalahan umum.

Memecahkan masalah pembaruan OTA dari beberapa perangkat

Untuk melakukan OTAs pada beberapa perangkat (benda) menggunakan gambar firmware yang sama, terapkan fungsi (misalnya`getThingName()`) yang mengambil `clientcredentialIOT_THING_NAME` dari memori non-volatile. Pastikan fungsi ini membaca nama benda dari bagian memori non-volatile yang tidak ditimpa oleh OTA, dan bahwa nama benda disediakan sebelum menjalankan pekerjaan pertama. Jika Anda menggunakan aliran JITP, Anda dapat membaca nama benda dari nama umum sertifikat perangkat Anda.

Memecahkan masalah pembaruan OTA dengan Launchpad Texas CC322 Instruments 0SF

Platform CC322 0SF Launchpad menyediakan mekanisme deteksi gangguan perangkat lunak. Ini menggunakan penghitung peringatan keamanan yang bertambah setiap kali ada pelanggaran integritas. Perangkat terkunci saat penghitung peringatan keamanan mencapai

ambang batas yang telah ditentukan (defaultnya adalah 15) dan host menerima peristiwa `SL_ERROR_DEVICE_LOCKED_SECURITY_ALERT` asinkron. Perangkat yang terkunci kemudian memiliki aksesibilitas terbatas. Untuk memulihkan perangkat, Anda dapat memprogram ulang atau menggunakan restore-to-factory proses untuk kembali ke gambar pabrik. Anda harus memprogram perilaku yang diinginkan dengan memperbarui event handler asinkron di. `network_if.c`

Perpustakaan FreeRTOS

Pustaka FreeRTOS menyediakan fungsionalitas tambahan untuk kernel FreerTOS dan perpustakaan internalnya. Anda dapat menggunakan pustaka FreeRTOS untuk jaringan dan keamanan dalam aplikasi tertanam. Pustaka FreeRTOS juga memungkinkan aplikasi Anda berinteraksi dengan layanan. AWS IoT FreeRTOS mencakup pustaka yang memungkinkan untuk:

- Hubungkan perangkat dengan aman ke AWS IoT Cloud menggunakan MQTT dan bayangan perangkat.
- Temukan dan sambungkan ke AWS IoT Greengrass core.
- Kelola koneksi Wi-Fi.
- Dengarkan dan prosesPembaruan FreeRTOS Over-the-Air.

`libraries`Direktori berisi kode sumber pustaka FreeRTOS. Ada fungsi pembantu yang membantu dalam mengimplementasikan fungsionalitas perpustakaan. Kami tidak menyarankan Anda mengubah fungsi pembantu ini.

Pustaka porting FreeRTOS

Pustaka porting berikut disertakan dalam konfigurasi FreeRTOS yang tersedia untuk diunduh di konsol FreeRTOS. Pustaka ini bergantung pada platform. Konten mereka berubah sesuai dengan platform perangkat keras Anda. Untuk informasi tentang porting pustaka ini ke perangkat, lihat Panduan Porting FreeRTOS.

Pustaka porting FreeRTOS

Perpustakaan	Referensi API	Deskripsi
Bluetooth Energi Rendah	Referensi API Energi Rendah Bluetooth	Menggunakan perpustakaan FreeRTOS Bluetooth Low Energy, mikrokontroler Anda dapat berkomunikasi dengan AWS IoT

Perpustakaan	Referensi API	Deskripsi
		broker MQTT melalui perangkat gateway. Untuk informasi selengkapnya, lihat Perpustakaan Bluetooth Rendah Energi.
Over-the-Air Update	AWS IoT Over-the-air perbarui referensi API	Pustaka pembaruan AWS IoT Over-the-air FreeRTOS (OTA) memungkinkan Anda mengelola pemberitahuan pembaruan, mengunduh pembaruan, dan melakukan verifikasi kriptografi pembaruan firmware pada perangkat FreeRTOS Anda. Untuk informasi selengkapnya, lihat AWS IoT Perpustakaan Over the Air (OTA).
Freertos+POSIX	Referensi Freertos+POSIX API	Anda dapat menggunakan pustaka Freertos+POSIX untuk mem-port aplikasi yang sesuai dengan POSIX ke ekosistem FreeRTOS. Untuk informasi lebih lanjut, lihat Freertos+POSIX.
Soket Aman	Referensi API Soket Aman	Untuk informasi selengkapnya, lihat Perpustakaan Secure Sockets.
Freertos+TCP	Referensi Freertos+TCP API	FreerTos+TCP adalah tumpukan TCP/IP yang dapat diskalakan, open source, dan thread aman untuk FreeRTOS. Untuk informasi selengkapnya, lihat Freertos+TCP.

Perpustakaan	Referensi API	Deskripsi
Wi-Fi	Referensi API Wi-Fi	Pustaka Wi-Fi FreeRTOS memungkinkan Anda untuk berinteraksi dengan tumpukan nirkabel tingkat rendah mikrokont roler Anda. Untuk informasi selengkapnya, lihat Pustaka Wi-Fi.
inti PKCS11		PKCS11 Pustaka inti adalah implementasi referensi dari Standar Kriptografi Kunci Publik #11, untuk mendukung penyediaan dan otentikasi klien TLS. Untuk informasi selengkapnya, lihat PKCS11 perpustakaan inti.
TLS		Untuk informasi selengkapnya, lihat Keamanan Lapisan Transportasi.
I/O umum	Referensi API I/O Umum	Untuk informasi selengkapnya, lihat I/O umum.
Antarmuka Seluler	Referensi API Antarmuka Seluler	Pustaka Antarmuka Seluler memperlihatkan kemampuan beberapa modem seluler populer melalui API yang seragam. Untuk informasi selengkapnya, lihat Perpustakaan Antarmuka Seluler.

Pustaka aplikasi FreeRTOS

Anda dapat secara opsional menyertakan pustaka aplikasi mandiri berikut dalam konfigurasi FreerTOS Anda untuk berinteraksi dengan layanan di cloud. AWS IoT

> **Note**
>
> Beberapa pustaka aplikasi memiliki pustaka yang APIs sama dengan library di AWS IoT Device SDK for Embedded C. Untuk library ini, lihat Referensi API AWS IoT Device SDK C. Untuk informasi selengkapnya tentang AWS IoT Device SDK for Embedded C, lihatAWS IoT Perangkat SDK untuk Embedded C.

Pustaka aplikasi FreeRTOS

Perpustakaan	Referensi API	Deskripsi
AWS IoT Device Defender	Referensi Device Defender C SDK API	Pustaka FreeRTOS menghubungkan AWS IoT Device Defender perangkat FreeRTOS Anda ke. AWS IoT Device Defender Untuk informasi selengkapnya, lihat AWS IoT Device Defender perpustakaan.
AWS IoT Greengrass	Referensi API Greengrass	Pustaka FreeRTOS menghubungkan AWS IoT Greengrass perangkat FreeRTOS Anda ke. AWS IoT Greengrass Untuk informasi selengkapnya, lihat AWS IoT Greengrass Perpustakaan penemuan.
MQTT	Referensi API Perpustakaan MQTT (v1.xx) Referensi API Agen MQTT (v1) MQTT (v2.xx) C SDK API Referensi	Pustaka CoreMQTT menyediakan klien untuk perangkat FreeRTOS Anda untuk mempublikasikan dan berlangganan topik MQTT. MQTT adalah protokol yang digunakan perangkat untuk berinteraksi. AWS IoT

Perpustakaan	Referensi API	Deskripsi
		Untuk informasi selengkapnya tentang pustaka CoreMQTT versi 3.0.0, lihat. Perpustakaan CoreMQTT
Agen CoreMQTT	Referensi API Perpustakaan Agen CoreMQTT	Pustaka Agen CoreMQTT adalah API tingkat tinggi yang menambahkan keamanan utas ke pustaka CoreMQTT. Ini memungkinkan Anda membuat tugas agen MQTT khusus yang mengelola koneksi MQTT di latar belakang dan tidak memerlukan intervensi apa pun dari tugas lain. Pustaka menyediakan thread aman yang setara dengan CoreMQTT APIs, sehingga dapat digunakan di lingkungan multi-threaded. Untuk informasi selengkapnya tentang pustaka Agen CoreMQTT, lihat. Perpustakaan Agen CoreMQTT
AWS IoT Device Shadow	Referensi Device Shadow C SDK API	Pustaka AWS IoT Device Shadow memungkinkan perangkat FreeRTOS Anda berinteraksi dengan AWS IoT bayangan perangkat. Untuk informasi selengkapnya, lihat AWS IoT Pustaka Device Shadow.

Mengkonfigurasi pustaka FreeRTOS

Pengaturan konfigurasi untuk FreeRTOS dan AWS IoT Device SDK untuk Embedded C didefinisikan sebagai konstanta preprocessor C. Anda dapat mengatur pengaturan konfigurasi dengan file konfigurasi global, atau dengan menggunakan opsi kompiler seperti `-D` di`gcc`. Karena pengaturan konfigurasi didefinisikan sebagai konstanta waktu kompilasi, pustaka harus dibangun kembali jika pengaturan konfigurasi diubah.

Jika Anda ingin menggunakan file konfigurasi global untuk mengatur opsi konfigurasi, buat dan simpan file dengan nama`iot_config.h`, dan letakkan di jalur penyertaan Anda. Di dalam file, gunakan `#define` arahan untuk mengonfigurasi pustaka, demo, dan pengujian FreeRTOS.

Untuk informasi selengkapnya tentang opsi konfigurasi global yang didukung, lihat Referensi File Konfigurasi Global.

Pustaka BackOffAlgorithm

> **Note**
>
> Konten di halaman ini mungkin tidak up-to-date. Silakan merujuk ke halaman perpustakaan Freertos.org untuk pembaruan terbaru.

Pengantar

Pustaka BackOffAlgorithm adalah pustaka utilitas yang digunakan untuk spasi transmisi ulang berulang dari blok data yang sama, untuk menghindari kemacetan jaringan. Pustaka ini menghitung periode backoff untuk mencoba kembali operasi jaringan (seperti koneksi jaringan yang gagal dengan server) menggunakan backoff eksponensial dengan algoritma jitter.

Backoff eksponensial dengan jitter biasanya digunakan saat mencoba kembali koneksi yang gagal atau permintaan jaringan ke server yang disebabkan oleh kemacetan jaringan atau beban tinggi di server. Ini digunakan untuk menyebarkan waktu permintaan coba lagi yang dibuat oleh beberapa perangkat yang mencoba koneksi jaringan pada saat yang sama. Dalam lingkungan dengan konektivitas yang buruk, klien dapat terputus kapan saja; jadi strategi backoff juga membantu klien menghemat baterai dengan tidak berulang kali mencoba rekoneksi ketika mereka tidak mungkin berhasil.

Perpustakaan ditulis dalam C dan dirancang agar sesuai dengan ISO C90 dan MISRA C: 2012. Perpustakaan tidak memiliki dependensi pada pustaka tambahan selain pustaka C standar dan tidak

memiliki alokasi heap, sehingga cocok untuk mikrokontroler IoT, tetapi juga sepenuhnya portabel ke platform lain.

Pustaka ini dapat digunakan secara bebas dan didistribusikan di bawah lisensi open source MIT.

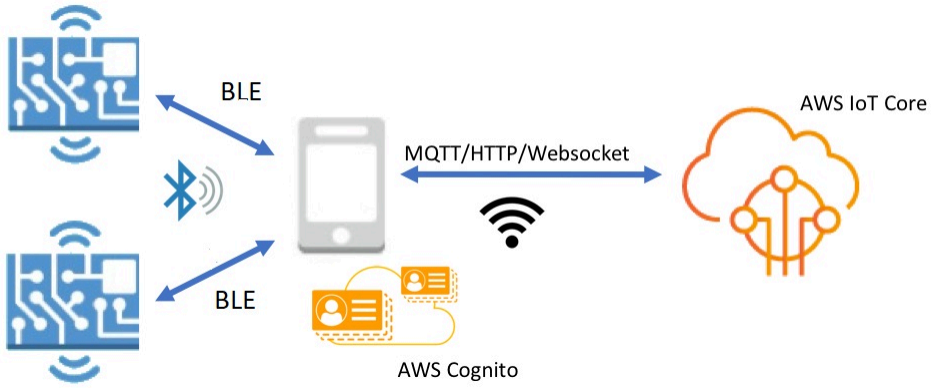
Ukuran Kode BackOffAlgorithm (contoh yang dihasilkan dengan GCC untuk ARM Cortex-M)		
File	Dengan Optimasi -O1	Dengan Optimasi -Os
backoff_algorithm.c	0,1K	0,1K
Total perkiraan	0,1K	0,1K

Perpustakaan Bluetooth Rendah Energi

> **Important**
>
> Pustaka ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-Freertos yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Gambaran Umum

FreeRTOS mendukung penerbitan dan berlangganan topik Message Queuing Telemetry Transport (MQTT) melalui Bluetooth Low Energy melalui perangkat proxy, seperti ponsel. Dengan perpustakaan FreeRTOS Bluetooth Low Energy (BLE), mikrokontroler Anda dapat berkomunikasi dengan aman dengan broker MQTT. AWS IoT

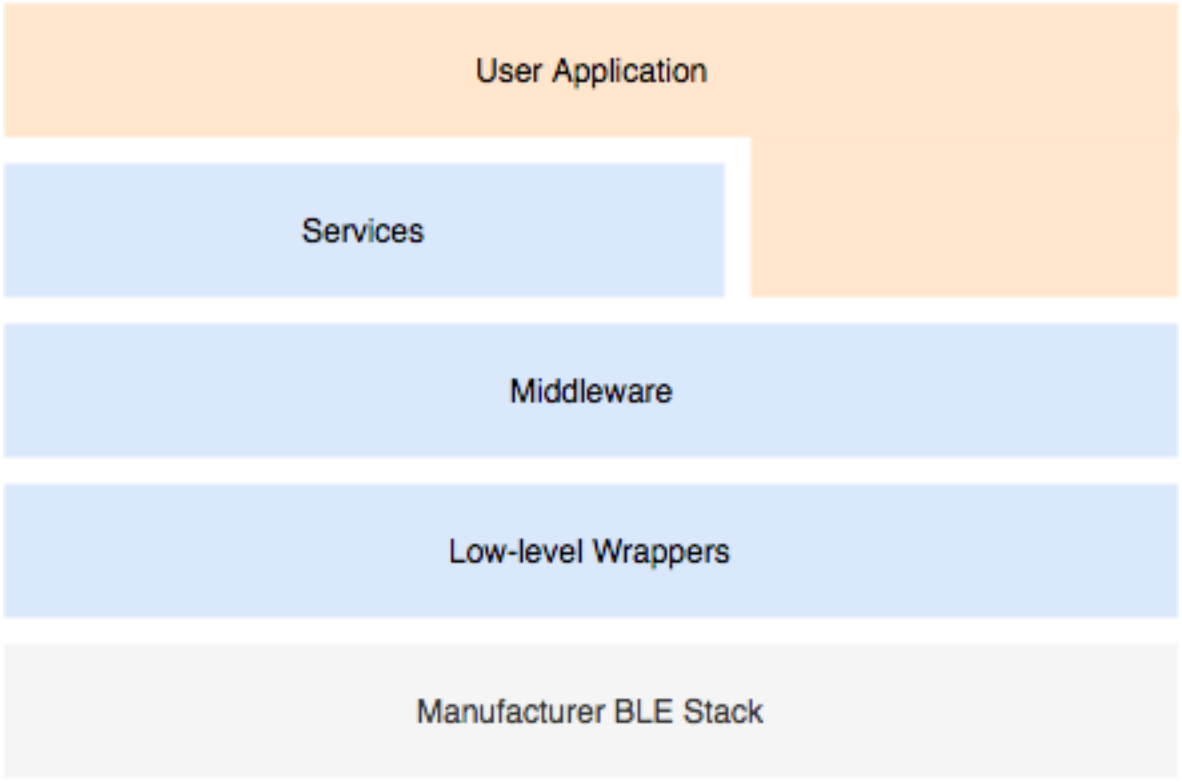


Menggunakan Ponsel SDKs untuk Perangkat Bluetooth FreeRTOS, Anda dapat menulis aplikasi seluler asli yang berkomunikasi dengan aplikasi yang disematkan pada mikrokontroler Anda melalui BLE. Untuk informasi selengkapnya tentang ponsel SDKs, lihatPonsel SDKs untuk perangkat Bluetooth FreeRTOS.

Pustaka FreeRTOS BLE mencakup layanan untuk mengkonfigurasi jaringan Wi-Fi, mentransfer sejumlah besar data, dan menyediakan abstraksi jaringan melalui BLE. Pustaka FreeRTOS BLE juga menyertakan middleware dan APIs level yang lebih rendah untuk kontrol lebih langsung atas tumpukan BLE Anda.

Arsitektur

Tiga lapisan membentuk perpustakaan FreerTOS BLE: layanan, middleware, dan pembungkus tingkat rendah.



Layanan

Lapisan layanan FreeRTOS BLE terdiri dari empat layanan Generic Attribute (GATT) yang memanfaatkan middleware: APIs

- Informasi perangkat
- Penyediaan Wi-Fi

- Abstraksi jaringan
- Transfer objek besar

Informasi perangkat

Layanan informasi Perangkat mengumpulkan detail tentang mikrokontroler Anda, termasuk:

- Versi FreeRTOS yang digunakan perangkat Anda.
- AWS IoT Titik akhir akun tempat perangkat terdaftar.
- Unit Transmisi Maksimum Energi Rendah Bluetooth (MTU).

Penyediaan Wi-Fi

Layanan penyediaan Wi-Fi memungkinkan mikrokontroler dengan kemampuan Wi-Fi untuk melakukan hal berikut:

- Daftar jaringan dalam jangkauan.
- Simpan jaringan dan kredenal jaringan ke memori flash.
- Tetapkan prioritas jaringan.
- Hapus jaringan dan kredenal jaringan dari memori flash.

Abstraksi jaringan

Layanan abstraksi jaringan mengabstraksi jenis koneksi jaringan untuk aplikasi. API umum berinteraksi dengan tumpukan perangkat keras Wi-Fi, Ethernet, dan Bluetooth Low Energy perangkat Anda, memungkinkan aplikasi kompatibel dengan beberapa jenis koneksi.

Transfer Objek Besar

Layanan Transfer Objek Besar mengirimkan data ke, dan menerima data dari, klien. Layanan lain, seperti penyediaan Wi-Fi dan abstraksi jaringan, menggunakan layanan Transfer Objek Besar untuk mengirim dan menerima data. Anda juga dapat menggunakan Large Object Transfer API untuk berinteraksi langsung dengan layanan.

MQTT di atas BLE

MQTT over BLE berisi profil GATT untuk membuat layanan proxy MQTT melalui BLE. Layanan proxy MQTT memungkinkan klien MQTT untuk berkomunikasi dengan broker AWS MQTT melalui

perangkat gateway. Misalnya, Anda dapat menggunakan layanan proxy untuk menghubungkan perangkat yang menjalankan FreeRTOS AWS ke MQTT melalui aplikasi ponsel cerdas. Perangkat BLE adalah server GATT dan mengekspos layanan dan karakteristik untuk perangkat gateway. Server GATT menggunakan layanan dan karakteristik yang terbuka ini untuk melakukan operasi MQTT dengan cloud untuk perangkat itu. Untuk detail selengkapnya, lihat Lampiran A: MQTT di atas profil BLE GATT .

Middleware

FreeRTOS Bluetooth Low Energy middleware adalah abstraksi dari level yang lebih rendah. APIs Middleware APIs membuat antarmuka yang lebih ramah pengguna ke tumpukan Bluetooth Low Energy.

Menggunakan middleware APIs, Anda dapat mendaftarkan beberapa callback, di beberapa lapisan, ke satu acara. Menginisialisasi middleware Bluetooth Low Energy juga menginisialisasi layanan dan mulai beriklan.

Langganan callback fleksibel

Misalkan perangkat keras Bluetooth Low Energy Anda terputus, dan layanan MQTT melalui Bluetooth Low Energy perlu mendeteksi pemutusan sambungan. Aplikasi yang Anda tulis mungkin juga perlu mendeteksi peristiwa pemutusan yang sama. Middleware Bluetooth Low Energy dapat merutekan acara ke berbagai bagian kode tempat Anda mendaftarkan callback, tanpa membuat lapisan yang lebih tinggi bersaing untuk sumber daya tingkat yang lebih rendah.

Pembungkus tingkat rendah

Pembungkus FreeRTOS Bluetooth Low Energy tingkat rendah adalah abstraksi dari tumpukan Bluetooth Low Energy pabrikan. Pembungkus tingkat rendah menawarkan satu set umum APIs untuk kontrol langsung atas perangkat keras. Tingkat rendah APIs mengoptimalkan penggunaan RAM, tetapi terbatas dalam fungsionalitas.

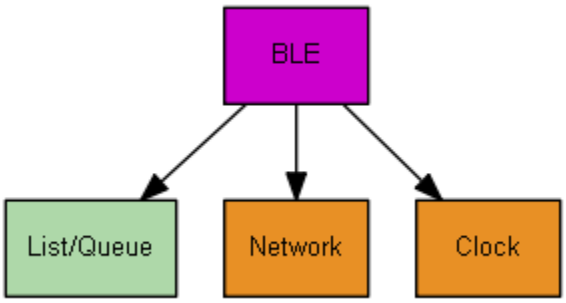
Gunakan layanan Bluetooth Low Energy APIs untuk berinteraksi dengan layanan Bluetooth Low Energy. Layanan APIs menuntut lebih banyak sumber daya daripada tingkat rendah APIs.

Ketergantungan dan persyaratan

Pustaka Bluetooth Low Energy memiliki dependensi langsung berikut:

- Perpustakaan Kontainer Linear

- Lapisan platform yang berinteraksi dengan sistem operasi untuk manajemen thread, timer, fungsi jam, dan akses jaringan.



Hanya layanan Penyediaan Wi-Fi yang memiliki dependensi pustaka FreeRTOS:

Layanan GATT	Dependensi
Penyediaan Wi-Fi	Pustaka Wi-Fi

Untuk berkomunikasi dengan broker AWS IoT MQTT, Anda harus memiliki AWS akun dan Anda harus mendaftarkan perangkat Anda sebagai barang. AWS IoT Untuk informasi selengkapnya tentang pengaturan, lihat Panduan AWS IoT Pengembang.

FreeRTOS Bluetooth Low Energy menggunakan Amazon Cognito untuk otentikasi pengguna di perangkat seluler Anda. Untuk menggunakan layanan proxy MQTT, Anda harus membuat identitas Amazon Cognito dan kumpulan pengguna. Setiap Identitas Amazon Cognito harus memiliki kebijakan yang sesuai yang dilampirkan padanya. Untuk informasi selengkapnya, lihat Panduan Developer Amazon Cognito .

File konfigurasi perpustakaan

Aplikasi yang menggunakan FreeRTOS MQTT melalui layanan Bluetooth Low Energy harus menyediakan file header, di `iot_ble_config.h` mana parameter konfigurasi ditentukan. Parameter konfigurasi yang tidak ditentukan mengambil nilai default yang ditentukan dalam`iot_ble_config_defaults.h`.

Beberapa parameter konfigurasi penting meliputi:

`IOT_BLE_ADD_CUSTOM_SERVICES`

Memungkinkan pengguna untuk membuat layanan mereka sendiri.

`IOT_BLE_SET_CUSTOM_ADVERTISEMENT_MSG`

Memungkinkan pengguna untuk menyesuaikan iklan dan memindai pesan respons.

Untuk informasi selengkapnya, lihat Referensi API Energi Rendah Bluetooth.

Pengoptimalan

Saat mengoptimalkan kinerja papan Anda, pertimbangkan hal berikut:

- Tingkat rendah APIs menggunakan lebih sedikit RAM, tetapi menawarkan fungsionalitas terbatas.
- Anda dapat mengatur `bleconfigMAX_NETWORK` parameter dalam file `iot_ble_config.h` header ke nilai yang lebih rendah untuk mengurangi jumlah tumpukan yang dikonsumsi.
- Anda dapat meningkatkan ukuran MTU ke nilai maksimumnya untuk membatasi buffering pesan, dan membuat kode berjalan lebih cepat dan mengkonsumsi lebih sedikit RAM.

Pembatasan penggunaan

Secara default, pustaka FreeRTOS Bluetooth Low Energy menyetel `eBTpropertySecureConnectionOnly` properti ke TRUE, yang menempatkan perangkat dalam mode Secure Connections Only. Seperti yang ditentukan oleh Spesifikasi Inti Bluetooth v5.0, Vol 3, Bagian C, 10.2.4, ketika perangkat berada dalam mode Secure Connections Only, mode keamanan LE tertinggi 1 level, level 4, diperlukan untuk akses ke atribut apa pun yang memiliki izin lebih tinggi dari level 1 mode keamanan LE terendah, level 1. Pada mode keamanan LE 1 level 4, perangkat harus memiliki kemampuan input dan output untuk perbandingan numerik.

Berikut adalah mode yang didukung, dan properti terkaitnya:

Mode 1, Level 1 (Tanpa keamanan)

```
/* 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 )
```

Mode 1, Level 2 (Pasangan yang tidak diautentikasi dengan enkripsi)

```
#define IOT_BLE_ENABLE_NUMERIC_COMPARISON       ( 0 )
```

```
#define IOT_BLE_ENABLE_SECURE_CONNECTION        ( 0 )
#define IOT_BLE_INPUT_OUTPUT                    ( eBTIONone )
```

Mode 1, Level 3 (Pasangan yang diautentikasi dengan enkripsi)

Mode ini tidak didukung.

Mode 1, Level 4 (Diautentikasi LE Secure Connections dipasangkan dengan enkripsi)

Mode ini didukung secara default.

Untuk informasi tentang mode keamanan LE, lihat Spesifikasi Inti Bluetooth v5.0, Vol 3, Bagian C, 10.2.1.

Inisialisasi

Jika aplikasi Anda berinteraksi dengan tumpukan Bluetooth Low Energy melalui middleware, Anda hanya perlu menginisialisasi middleware. Middleware menangani inisialisasi lapisan bawah tumpukan.

Middleware

Untuk menginisialisasi middleware

1. Inisialisasi driver perangkat keras Bluetooth Low Energy sebelum Anda memanggil API middleware Bluetooth Low Energy.
2. Aktifkan Bluetooth Energi Rendah.
3. Inisialisasi middleware dengan. `IotBLE_Init()`

> **Note**
>
> Langkah inisialisasi ini tidak diperlukan jika Anda menjalankan AWS demo. Inisialisasi demo ditangani oleh Network Manager, yang terletak di. *`freertos`*`/demos/network_manager`

Tingkat rendah APIs

Jika Anda tidak ingin menggunakan layanan FreeRTOS Bluetooth Low Energy GATT, Anda dapat melewati middleware dan berinteraksi langsung dengan level rendah untuk menghemat sumber daya. APIs

Untuk menginisialisasi tingkat rendah APIs

1. Inisialisasi driver perangkat keras Bluetooth Low Energy sebelum Anda menelepon. APIs Inisialisasi driver bukan bagian dari Bluetooth Low Energy APIs low-level.
2. Bluetooth Low Energy low-level API menyediakan panggilan aktifkan/nonaktifkan ke tumpukan Bluetooth Low Energy untuk mengoptimalkan daya dan sumber daya. Sebelum menelepon APIs, Anda harus mengaktifkan Bluetooth Low Energy.

   ```
   const BTInterface_t * pxIface = BTGetBluetoothInterface();
   xStatus = pxIface->pxEnable( 0 );
   ```

3. Manajer Bluetooth berisi APIs yang umum untuk Bluetooth Low Energy dan Bluetooth classic. Callback untuk manajer umum harus diinisialisasi kedua.

   ```
   xStatus = xBTInterface.pxBTInterface->pxBtManagerInit( &xBTManagerCb );
   ```

4. Adaptor Bluetooth Low Energy cocok di atas API umum. Anda harus menginisialisasi callback-nya seperti Anda menginisialisasi API umum.

   ```
   xBTInterface.pxBTLeAdapterInterface = ( BTBleAdapter_t * )
    xBTInterface.pxBTInterface->pxGetLeAdapter();
   xStatus = xBTInterface.pxBTLeAdapterInterface-
   >pxBleAdapterInit( &xBTBleAdapterCb );
   ```

5. Daftarkan aplikasi pengguna baru Anda.

   ```
   xBTInterface.pxBTLeAdapterInterface->pxRegisterBleApp( pxAppUuid );
   ```

6. Inisialisasi callback ke server GATT.

   ```
   xBTInterface.pxGattServerInterface = ( BTGattServerInterface_t * )
    xBTInterface.pxBTLeAdapterInterface->ppvGetGattServerInterface();
   xBTInterface.pxGattServerInterface->pxGattServerInit( &xBTGattServerCb );
   ```

   Setelah Anda menginisialisasi adaptor Bluetooth Low Energy, Anda dapat menambahkan server GATT. Anda hanya dapat mendaftarkan satu server GATT dalam satu waktu.

```
xStatus = xBTInterface.pxGattServerInterface->pxRegisterServer( pxAppUuid );
```

7. Atur properti aplikasi seperti koneksi aman saja dan ukuran MTU.

```
xStatus = xBTInterface.pxBTInterface-
>pxSetDeviceProperty( &pxProperty[ usIndex ] );
```

Referensi API

Untuk referensi API selengkapnya, lihat Referensi API Energi Rendah Bluetooth.

Contoh penggunaan

Contoh di bawah ini menunjukkan cara menggunakan pustaka Bluetooth Low Energy untuk mengiklankan dan membuat layanan baru. Untuk aplikasi demo FreeRTOS Bluetooth Low Energy lengkap, lihat Aplikasi Demo Energi Rendah Bluetooth.

Iklan

1. Dalam aplikasi Anda, atur UUID iklan:

```
static const BTUuid_t _advUUID =
{
    .uu.uu128 = IOT_BLE_ADVERTISING_UUID,
    .ucType   = eBTuuidType128
};
```

2. Kemudian tentukan fungsi `IotBle_SetCustomAdvCb` callback:

```
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;
}
```

Callback ini mengirimkan UUID dalam pesan iklan dan nama lengkap dalam respons pemindaian.

3. Bukavendors/*vendor*/boards/*board*/aws_demos/config_files/iot_ble_config.h, dan atur IOT_BLE_SET_CUSTOM_ADVERTISEMENT_MSG ke1. Ini memicu IotBle_SetCustomAdvCb panggilan balik.

Menambahkan layanan baru

Untuk contoh lengkap layanan, lihat*freertos*/.../ble/services.

1. Buat UUIDs untuk karakteristik dan deskriptor layanan:

```
#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. Buat buffer untuk mendaftarkan pegangan karakteristik dan deskriptor:

```
static uint16_t usHandlesBuffer[egattDemoNbAttributes];
```

3. Buat tabel atribut. Untuk menyimpan beberapa RAM, tentukan tabel sebagai file`const`.

> **Important**
>
> Selalu buat atribut secara berurutan, dengan layanan sebagai atribut pertama.

```
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. Buat array callback. Array callback ini harus mengikuti urutan yang sama seperti array tabel yang didefinisikan di atas.

Misalnya, jika `vReadCounter` dipicu saat `xCharCounterUUID_TYPE` diakses, dan `vWriteCommand` dipicu saat `xCharControlUUID_TYPE` diakses, tentukan array sebagai berikut:

```
static const IotBleAttributeEventCallback_t pxCallBackArray[egattDemoNbAttributes]
 =
    {
  NULL,
  vReadCounter,
  vEnableNotification,
  vWriteCommand
};
```

5. Buat layanan:

```
static const BTService_t xGattDemoService =
{
  .xNumberOfAttributes = egattDemoNbAttributes,
  .ucInstId = 0,
  .xType = eBTServiceTypePrimary,
  .pusHandlesBuffer = usHandlesBuffer,
  .pxBLEAttributes = (BTAttribute_t *)pxAttributeTable
};
```

6. Panggil API `IotBle_CreateService` dengan struktur yang Anda buat pada langkah sebelumnya. Middleware menyinkronkan pembuatan semua layanan, sehingga setiap layanan baru harus sudah ditentukan ketika `IotBle_AddCustomServicesCb` callback dipicu.

   a. Setel `IOT_BLE_ADD_CUSTOM_SERVICES` ke 1 dalam`vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/iot_ble_config.h`.

   b. Buat IotBle _ AddCustomServicesCb di aplikasi Anda:

```
void IotBle_AddCustomServicesCb(void)
{
    BTStatus_t xStatus;
    /* Select the handle buffer. */
    xStatus = IotBle_CreateService( (BTService_t *)&xGattDemoService,
 (IotBleAttributeEventCallback_t *)pxCallBackArray );
}
```

Porting

Input pengguna dan output periferal

Koneksi yang aman membutuhkan input dan output untuk perbandingan numerik. `eBLENumericComparisonCallback`Acara dapat didaftarkan menggunakan manajer acara:

```
xEventCb.pxNumericComparisonCb = &prvNumericComparisonCb;
xStatus = BLE_RegisterEventCb( eBLENumericComparisonCallback, xEventCb );
```

Periferal harus menampilkan passkey numerik dan mengambil hasil perbandingan sebagai input.

Implementasi API porting

Untuk mem-port FreeRTOS ke target baru, Anda harus menerapkan APIs beberapa untuk layanan Penyediaan Wi-Fi dan fungsionalitas Bluetooth Low Energy.

Bluetooth Energi Rendah APIs

Untuk menggunakan middleware FreeRTOS Bluetooth Low Energy, Anda harus menerapkan beberapa. APIs

APIs umum antara GAP untuk Bluetooth Classic dan GAP untuk Bluetooth Low Energy

- `pxBtManagerInit`
- `pxEnable`
- `pxDisable`
- `pxGetDeviceProperty`
- `pxSetDeviceProperty`(Semua opsi wajib diharapkan `eBTpropertyRemoteRssi` dan`eBTpropertyRemoteVersionInfo`)
- `pxPair`
- `pxRemoveBond`
- `pxGetConnectionState`
- `pxPinReply`
- `pxSspReply`
- `pxGetTxpower`
- `pxGetLeAdapter`
- `pxDeviceStateChangedCb`

- pxAdapterPropertiesCb
- pxSspRequestCb
- pxPairingStateChangedCb
- pxTxPowerCb

APIs khusus untuk GAP untuk Bluetooth Low Energy

- pxRegisterBleApp
- pxUnregisterBleApp
- pxBleAdapterInit
- pxStartAdv
- pxStopAdv
- pxSetAdvData
- pxConnParameterUpdateRequest
- pxRegisterBleAdapterCb
- pxAdvStartCb
- pxSetAdvDataCb
- pxConnParameterUpdateRequestCb
- pxCongestionCb

Peladen 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`

Untuk informasi selengkapnya tentang porting pustaka FreeRTOS Bluetooth Low Energy ke platform Anda, lihat Mem-porting Perpustakaan Energi Rendah Bluetooth di Panduan Porting FreerTOS.

Ponsel SDKs untuk perangkat Bluetooth FreeRTOS

> **Important**
>
> Pustaka ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-Freertos yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Anda dapat menggunakan Ponsel SDKs untuk Perangkat Bluetooth FreerTOS untuk membuat aplikasi seluler yang berinteraksi dengan mikrokontroler Anda melalui Bluetooth Low Energy. Ponsel juga SDKs dapat berkomunikasi dengan AWS layanan, menggunakan Amazon Cognito untuk otentikasi pengguna.

Android SDK untuk perangkat Bluetooth FreeRTOS

Gunakan SDK Android untuk Perangkat Bluetooth FreerTOS untuk membangun aplikasi seluler Android yang berinteraksi dengan mikrokontroler Anda melalui Bluetooth Low Energy. SDK tersedia di GitHub.

Untuk menginstal Android SDK untuk perangkat Bluetooth FreeRTOS, ikuti petunjuk untuk "Menyiapkan SDK" di file README.md proyek.

Untuk informasi tentang pengaturan dan menjalankan aplikasi mobile demo yang disertakan dengan SDK, lihat Prasyarat danAplikasi demo SDK Seluler Energi Rendah FreeRTOS Bluetooth.

SDK iOS untuk perangkat Bluetooth FreeRTOS

Gunakan SDK iOS untuk Perangkat Bluetooth FreeRTOS untuk membangun aplikasi seluler iOS yang berinteraksi dengan mikrokontroler Anda melalui Bluetooth Low Energy. SDK tersedia di GitHub.

Untuk menginstal SDK iOS

1. Instal CocoaPods:

```
$ gem install cocoapods
$ pod setup
```

> **Note**
>
> Anda mungkin perlu menggunakannya `sudo` untuk menginstal CocoaPods.

2. Instal SDK dengan CocoaPods (tambahkan ini ke podfile Anda):

```
$ pod 'FreeRTOS', :git => 'https://github.com/aws/amazon-freertos-ble-ios-sdk.git'
```

Untuk informasi tentang pengaturan dan menjalankan aplikasi mobile demo yang disertakan dengan SDK, lihat Prasyarat danAplikasi demo SDK Seluler Energi Rendah FreeRTOS Bluetooth.

Lampiran A: MQTT di atas profil BLE GATT

Detail Layanan GATT

MQTT melalui BLE menggunakan instance layanan GATT transfer data untuk mengirim pesan MQTT Concise Binary Object Representation (CBOR) antara perangkat FreeRTOS dan perangkat proxy. Layanan transfer data memperlihatkan karakteristik tertentu yang membantu mengirim dan menerima data mentah melalui protokol BLE GATT. Ini juga menangani fragmentasi dan perakitan muatan yang lebih besar dari ukuran BLE maximum transfer unit (MTU).

Layanan UUID

`A9D7-166A-D72E-40A9-A002-4804-4CC3-FF00`

Instans Layanan

Salah satu contoh layanan GATT dibuat untuk setiap sesi MQTT dengan broker. Setiap layanan memiliki UUID unik (dua byte) yang mengidentifikasi jenisnya. Setiap instance individu dibedakan oleh ID instance.

Setiap layanan dipakai sebagai layanan utama pada setiap perangkat server BLE. Anda dapat membuat beberapa instance layanan pada perangkat tertentu. Jenis layanan proxy MQTT memiliki UUID yang unik.

Karakteristik

Format konten karakteristik: CBOR

Ukuran nilai karakteristik maks: 512 byte

Karakteristik	Persyaratan	Properti Wajib	Properti opsional	Izin Keamanan	Deskripsi Singkat	UUID
Kontrol	M	Tulis	Tidak ada	Tulis Kebutuhan Enkripsi	Digunakan untuk memulai dan menghentikan proxy MQTT.	`A9D7-166A-D72E-40A9-A002-4804-4CC3-FF01`

Karakteristik	Persyaratan	Properti Wajib	Properti opsional	Izin Keamanan	Deskripsi Singkat	UUID
TXMessage	M	Baca, Pemberitahuan	Tidak ada	Baca Kebutuhan Enkripsi	Digunakan untuk mengirim notifikasi yang berisi pesan ke broker melalui proxy.	A9D7-166A-D72E-40A9-A002-4804-4CC3-FF02
RXMessage	M	Membaca, Menulis Tanpa Respon	Tidak ada	Baca, Tulis Kebutuhan Enkripsi	Digunakan untuk menerima pesan dari broker melalui proxy.	A9D7-166A-D72E-40A9-A002-4804-4CC3-FF03
TXLargePesan	M	Baca, Pemberitahuan	Tidak ada	Baca Kebutuhan Enkripsi	Digunakan untuk mengirim pesan besar (Message > BLE MTU Size) ke broker melalui proxy.	A9D7-166A-D72E-40A9-A002-4804-4CC3-FF04

Karakteristik	Persyaratan	Properti Wajib	Properti opsional	Izin Keamanan	Deskripsi Singkat	UUID
RXLargePesan	M	Membaca, Menulis Tanpa Respon	Tidak ada	Baca, Tulis Kebutuhan Enkripsi	Digunakan untuk menerima pesan besar (Message > BLE MTU Size) dari broker melalui proxy.	A9D7-166A-D72E-40A9-A002-4804-4CC3-FF05

Persyaratan Prosedur GATT

Baca Nilai Karakteristik	Wajib
Baca Nilai Karakteristik Panjang	Wajib
Tulis Nilai Karakteristik	Wajib
Tulis Nilai Karakteristik Panjang	Wajib
Baca deskriptor Karakteristik	Wajib
Tulis deskriptor Karakteristik	Wajib
Pemberitahuan	Wajib
Indikasi	Wajib

Jenis Pesan

Jenis pesan berikut dipertukarkan.

Tipe Pesan	Pesan	Peta dengan pasangan kunci/nilai ini
0x01	MENGHUBUNG	• Kunci = “w”, nilai = Tipe 0 Integer, Jenis pesan (1) • Kunci = “d”, nilai = Tipe 3, String Teks, Pengidentifikasi Klien untuk sesi • Key = “a”, value = Type 3, Text String, Broker endpoint untuk sesi • Kunci = “c”, Nilai = Jenis Nilai Sederhana Benar/Salah
0x02	CONNACK	• Kunci = “w, nilai = Tipe 0 Integer, Jenis pesan (2) • Kunci = “s”, Nilai = Tipe 0 Integer, Kode status
0x03	MENERBITKAN	• Kunci = “w”, nilai = Tipe 0 Integer, Jenis pesan (3) • Kunci = “u”, nilai = Tipe 3, String Teks, Topik untuk dipublikasikan • Kunci = “n”, nilai = Tipe 0, Integer, QoS untuk dipublikasikan • Key = “i”, value = Type 0, Integer, Message Identifier, Hanya untuk QoS 1 Publishes • Kunci = “k”, Nilai = Tipe 2, Byte String, Payload untuk dipublikasikan

Tipe Pesan	Pesan	Peta dengan pasangan kunci/nilai ini
0x04	KEMUNASAN	• Dikirim Hanya untuk pesan QoS 1. • Kunci = “w”, nilai = Tipe 0 Integer, Jenis pesan (4) • Kunci = “i”, nilai = Tipe 0, Integer, Pengidentifikasi Pesan
0x08	BERLANGGANAN	• Kunci = “w”, nilai = Tipe 0 Integer, Jenis pesan (8) • Kunci = “v”, nilai = Tipe 4, Array string teks, topik untuk berlangganan • Kunci = “o”, nilai = Tipe 4, Array Bilangan Bulat, QoS untuk berlangganan • Kunci = “i”, nilai = Tipe 0, Integer, Pengidentifikasi Pesan
0x09	SUBACK	• Kunci = “w”, nilai = Tipe 0 Integer, Jenis pesan (9) • Kunci = “i”, nilai = Tipe 0, Integer, Pengidentifikasi Pesan • Kunci = “s”, nilai = Tipe 0, Integer, Kode status untuk Berlangganan

Tipe Pesan	Pesan	Peta dengan pasangan kunci/nilai ini
0X0A	UNSUBSCRIBE (BERHENTI BERLANGGANAN)	• Kunci = “w”, nilai = Tipe 0 Integer, Jenis pesan (10) • Kunci = “v”, nilai = Tipe 4, Array string teks, topik untuk berhenti berlangganan • Kunci = “i”, nilai = Tipe 0, Integer, Pengidentifikasi Pesan
0x0B	UNSUBACK	• Kunci = “w”, nilai = Tipe 0 Integer, Jenis pesan (11) • Kunci = “i”, nilai = Tipe 0, Integer, Pengidentifikasi Pesan • Kunci = “s”, nilai = Tipe 0, Integer, Kode status untuk UnSubscription
0X0C	PINGREQ	• Kunci = “w”, nilai = Tipe 0 Integer, Jenis pesan (12)
0x0D	PINGRESP	• Kunci = “w”, nilai = Tipe 0 Integer, Jenis pesan (13)
0x0E	DISCONNNECT	• Kunci = “w”, nilai = Tipe 0 Integer, Jenis pesan (14)

Karakteristik Transfer Muatan Besar

TXLargePesan

TXLargePesan digunakan oleh perangkat untuk mengirim muatan besar yang lebih besar dari ukuran MTU yang dinegosiasikan untuk koneksi BLE.

- Perangkat mengirimkan byte MTU pertama dari muatan sebagai pemberitahuan melalui karakteristik.
- Proxy mengirimkan permintaan baca pada karakteristik ini untuk byte yang tersisa.
- Perangkat mengirimkan hingga ukuran MTU atau byte yang tersisa dari muatan, mana yang kurang. Setiap kali, itu meningkatkan pembacaan offset dengan ukuran muatan yang dikirim.
- Proxy akan terus membaca karakteristik hingga mendapat muatan panjang nol atau muatan kurang dari ukuran MTU.
- Jika perangkat tidak mendapatkan permintaan baca dalam batas waktu yang ditentukan, transfer gagal dan proxy serta gateway melepaskan buffer.
- Jika proxy tidak mendapatkan respons baca dalam batas waktu tertentu, transfer gagal dan proxy melepaskan buffer.

RXLargePesan

RXLargePesan digunakan oleh perangkat untuk menerima muatan besar yang lebih besar dari ukuran MTU yang dinegosiasikan untuk koneksi BLE.

- Proxy menulis pesan, hingga ukuran MTU, satu per satu, menggunakan tulis dengan respons pada karakteristik ini.
- Perangkat menyangga pesan hingga menerima permintaan tulis dengan panjang nol atau panjang kurang dari ukuran MTU.
- Jika perangkat tidak mendapatkan permintaan tulis dalam batas waktu yang ditentukan, transfer gagal dan perangkat melepaskan buffer.
- Jika proxy tidak mendapatkan respons tulis dalam batas waktu tertentu, transfer gagal dan proxy melepaskan buffer.

Perpustakaan Antarmuka Seluler

> **Note**
>
> Konten di halaman ini mungkin tidak up-to-date. Silakan merujuk ke halaman perpustakaan Freertos.org untuk pembaruan terbaru.

Pengantar

Pustaka Antarmuka Seluler mengimplementasikan API terpadu sederhana yang menyembunyikan kompleksitas perintah AT khusus modem seluler dan mengekspos antarmuka seperti soket ke pemrogram C.

Sebagian besar modem seluler menerapkan kurang lebih perintah AT yang ditentukan oleh standar 3GPP TS v27.007. Proyek ini menyediakan implementasi perintah AT standar tersebut dalam komponen umum yang dapat digunakan kembali. Tiga perpustakaan Antarmuka Seluler dalam proyek ini semuanya memanfaatkan kode umum itu. Pustaka untuk setiap modem hanya mengimplementasikan perintah AT khusus vendor, lalu mengekspos API perpustakaan Antarmuka Seluler yang lengkap.

Komponen umum yang mengimplementasikan standar 3GPP TS v27.007 telah ditulis sesuai dengan kriteria kualitas kode berikut:

- Skor Kompleksitas GNU tidak lebih dari 8
- Standar pengkodean MISRA C: 2012. Setiap penyimpangan dari standar didokumentasikan dalam komentar kode sumber yang ditandai dengan “coverity”.

Dependensi dan persyaratan

Tidak ada ketergantungan langsung untuk perpustakaan Antarmuka Seluler. Namun, Ethernet, Wi-Fi, dan seluler tidak dapat hidup berdampingan di tumpukan jaringan FreeRTOS. Pengembang harus memilih salah satu antarmuka jaringan untuk diintegrasikan dengan perpustakaan Secure Sockets.

Porting

Untuk informasi tentang porting pustaka Antarmuka Seluler ke platform Anda, lihat Mem-porting pustaka Antarmuka Seluler di Panduan Porting FreeRTOS.

Penggunaan memori

Ukuran Kode pustaka antarmuka seluler (contoh yang dihasilkan dengan GCC untuk ARM Cortex-M)		
File	Dengan Optimasi -O1	Dengan Optimasi -Os
selular_3gpp_api.c	6.3K	5.7K

Ukuran Kode pustaka antarmuka seluler (contoh yang dihasilkan dengan GCC untuk ARM Cortex-M)		
File	Dengan Optimasi -O1	Dengan Optimasi -Os
selular_3gpp_urc_handler.c	0.9K	0.8K
selular_at_core.c	1.4K	1.2K
selular_common_api.c	0.5K	0.5K
selular_common.c	1.6K	1.4K
selular_pkthandler.c	1.4K	1.2K
selular_pktio.c	1.8K	1.6K
Total perkiraan	13.9K	12,4K

Memulai

Unduh kode sumbernya

Kode sumber dapat diunduh sebagai bagian dari pustaka FreeRTOS atau dengan sendirinya.

Untuk mengkloning pustaka dari Github menggunakan HTTPS:

```
git clone https://github.com/FreeRTOS/FreeRTOS-Cellular-Interface.git
```

Menggunakan SSH:

```
git clone git@github.com:FreeRTOS/FreeRTOS-Cellular-Interface.git
```

Struktur folder

Di root repositori ini Anda akan melihat folder-folder ini:

- `source`: kode umum yang dapat digunakan kembali yang mengimplementasikan perintah AT standar yang ditentukan oleh 3GPP TS v27.007
- `doc`: dokumentasi

- `test`: uji unit dan cbmc
- `tools`: alat untuk analisis statis Coverity dan CMock

Konfigurasikan dan bangun Perpustakaan

Pustaka Antarmuka Seluler harus dibangun sebagai bagian dari aplikasi. Untuk melakukan ini, Anda harus menyediakan konfigurasi tertentu. Proyek FreerTos_Cellular_Interface_Windows_Simulator memberikan contoh cara mengonfigurasi build. Informasi lebih lanjut dapat ditemukan di Referensi API Seluler.

Silakan merujuk ke halaman Antarmuka Seluler untuk informasi lebih lanjut.

Integrasikan perpustakaan Antarmuka Seluler dengan platform MCU

Pustaka Antarmuka Seluler berjalan MCUs menggunakan antarmuka abstrak, Antarmuka Komunikasi, untuk berkomunikasi dengan modem seluler. Antarmuka Komunikasi harus diimplementasikan pada platform MCU juga. Implementasi yang paling umum dari Antarmuka Komunikasi berkomunikasi melalui perangkat keras UART, tetapi dapat diimplementasikan melalui antarmuka fisik lainnya, seperti SPI, juga. Dokumentasi untuk Antarmuka Komunikasi dapat ditemukan di Referensi API Perpustakaan Seluler. Contoh implementasi Comm Interface berikut tersedia:

- Antarmuka komunikasi simulator Windows FreeRTOS
- Antarmuka komunikasi FreeRTOS Common IO UART
- STM32 Antarmuka komunikasi papan penemuan L475
- Antarmuka komunikasi papan Sierra Sensor Hub

I/O umum

> **Important**
>
> Pustaka ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-Freertos yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Ikhtisar

Secara umum, driver perangkat independen dari sistem operasi yang mendasarinya dan khusus untuk konfigurasi perangkat keras tertentu. Lapisan abstraksi perangkat keras (HAL) menyediakan antarmuka umum antara driver dan kode aplikasi tingkat tinggi. HAL mengabstraksi detail tentang cara kerja driver tertentu dan menyediakan API yang seragam untuk mengontrol perangkat tersebut. Anda dapat menggunakan hal yang sama APIs untuk mengakses berbagai driver perangkat di beberapa papan referensi berbasis mikrokontroler (MCU).

FreeRTOS I/O umum bertindak sebagai lapisan abstraksi perangkat keras ini. Ini menyediakan seperangkat standar APIs untuk mengakses perangkat serial umum pada papan referensi yang didukung. Ini umum APIs berkomunikasi dan berinteraksi dengan periferal ini dan memungkinkan kode Anda berfungsi di seluruh platform. Tanpa I/O umum, menulis kode untuk bekerja dengan perangkat tingkat rendah adalah khusus vendor silikon.

Periferal yang didukung

- UART
- SPI
- I2C

Fitur yang didukung

- Baca/tulis sinkron - Fungsi tidak kembali sampai jumlah data yang diminta ditransfer.
- Baca/tulis asinkron - Fungsi segera kembali dan transfer data terjadi secara asinkron. Ketika tindakan selesai, callback pengguna terdaftar dipanggil.

Spesifik periferal

- I2C — Menggabungkan beberapa operasi menjadi satu transaksi. Digunakan untuk menulis kemudian membaca tindakan dalam satu transaksi.
- SPI — Transfer data antara primer dan sekunder, yang berarti menulis dan membaca terjadi secara bersamaan.

Porting

Untuk informasi lebih lanjut, lihat Panduan Porting FreerTOS.

AWS IoT Device Defender perpustakaan

> **Note**
>
> Konten di halaman ini mungkin tidak up-to-date. Silakan merujuk ke halaman perpustakaan Freertos.org untuk pembaruan terbaru.

Pengantar

Anda dapat menggunakan AWS IoT Device Defender pustaka untuk mengirim metrik keamanan dari perangkat IoT Anda ke. AWS IoT Device Defender Anda dapat menggunakan AWS IoT Device Defender untuk terus memantau metrik keamanan ini dari perangkat untuk penyimpangan dari apa yang telah Anda definisikan sebagai perilaku yang sesuai untuk setiap perangkat. Jika ada sesuatu yang tidak beres, AWS IoT Device Defender kirimkan peringatan sehingga Anda dapat mengambil tindakan untuk memperbaiki masalah tersebut. Interaksi dengan AWS IoT Device Defender menggunakan MQTT, protokol berlangganan publikasi ringan. Pustaka ini menyediakan API untuk menyusun dan mengenali string topik MQTT yang digunakan oleh. AWS IoT Device Defender

Untuk informasi lebih lanjut, lihat AWS IoT Device Defender dalam Panduan Pengembang AWS IoT .

Perpustakaan ditulis dalam C dan dirancang agar sesuai dengan ISO C90 dan MISRA C: 2012. Pustaka tidak memiliki dependensi pada pustaka tambahan apa pun selain pustaka C standar. Itu juga tidak memiliki dependensi platform apa pun, seperti threading atau sinkronisasi. Ini dapat digunakan dengan perpustakaan MQTT dan perpustakaan JSON atau CBOR apa pun. Perpustakaan memiliki bukti yang menunjukkan penggunaan memori yang aman dan tidak ada alokasi heap, sehingga cocok untuk mikrokontroler IoT, tetapi juga sepenuhnya portabel ke platform lain.

AWS IoT Device Defender Perpustakaan dapat digunakan secara bebas dan didistribusikan di bawah lisensi open source MIT.

Ukuran Kode AWS IoT Device Defender (contoh yang dihasilkan dengan GCC untuk ARM Cortex-M)		
File	Dengan Optimasi -O1	Dengan Optimasi -Os
pembela	1.1K	0.6K

Ukuran Kode AWS IoT Device Defender (contoh yang dihasilkan dengan GCC untuk ARM Cortex-M)		
File	Dengan Optimasi -O1	Dengan Optimasi -Os
Total perkiraan	1.1K	0.6K

AWS IoT Greengrass Perpustakaan penemuan

> **Note**
>
> Konten di halaman ini mungkin tidak up-to-date. Silakan merujuk ke halaman perpustakaan Freertos.org untuk pembaruan terbaru.

Gambaran Umum

Pustaka AWS IoT Greengrass Discovery digunakan oleh perangkat mikrokontroler Anda untuk menemukan inti Greengrass di jaringan Anda. Menggunakan AWS IoT Greengrass Discovery APIs, perangkat Anda dapat mengirim pesan ke inti Greengrass setelah menemukan titik akhir inti.

Dependensi dan persyaratan

Untuk menggunakan pustaka Greengrass Discovery, Anda harus membuat sesuatu, termasuk sertifikat dan AWS IoT kebijakan. Untuk informasi selengkapnya, lihat AWS IoT Memulai.

Anda harus menetapkan nilai untuk konstanta berikut dalam *`freertos`*`/demos/include/aws_clientcredential.h` file:

`clientcredentialMQTT_BROKER_ENDPOINT`

AWS IoT Titik akhir Anda.

`clientcredentialIOT_THING_NAME`

Nama benda IoT Anda.

`clientcredentialWIFI_SSID`

SSID untuk jaringan Wi-Fi Anda.

`clientcredentialWIFI_PASSWORD`

Kata sandi Wi-Fi Anda.

`clientcredentialWIFI_SECURITY`

Jenis keamanan yang digunakan oleh jaringan Wi-Fi Anda.

Anda juga harus menetapkan nilai untuk konstanta berikut dalam *`freertos`*`/demos/include/aws_clientcredential_keys.h` file:

`keyCLIENT_CERTIFICATE_PEM`

Sertifikat PEM terkait dengan barang Anda.

`keyCLIENT_PRIVATE_KEY_PEM`

Kunci pribadi PEM terkait dengan hal Anda.

Anda harus memiliki grup Greengrass dan perangkat inti yang disiapkan di konsol. Untuk informasi selengkapnya, lihat Memulai dengan AWS IoT Greengrass.

Meskipun pustaka CoreMQTT tidak diperlukan untuk konektivitas Greengrass, kami sangat menyarankan Anda menginstalnya. Perpustakaan dapat digunakan untuk berkomunikasi dengan inti Greengrass setelah ditemukan.

Referensi API

Untuk referensi API selengkapnya, lihat Referensi API Greengrass.

Contoh penggunaan

Alur kerja Greengrass

Perangkat MCU memulai proses penemuan dengan meminta dari file JSON AWS IoT yang berisi parameter konektivitas inti Greengrass. Ada dua metode untuk mengambil parameter konektivitas inti Greengrass dari file JSON:

- Seleksi otomatis berulang melalui semua inti Greengrass yang tercantum dalam file JSON dan terhubung ke yang pertama tersedia.
- Pemilihan manual menggunakan informasi `aws_ggd_config.h` untuk terhubung ke inti Greengrass yang ditentukan.

Cara menggunakan Greengrass API

Semua opsi konfigurasi default untuk Greengrass API didefinisikan dalam. `aws_ggd_config_defaults.h`

Jika hanya ada satu inti Greengrass, `GGD_GetGGCIPandCertificate` hubungi untuk meminta file JSON dengan informasi konektivitas inti Greengrass. `GGD_GetGGCIPandCertificate`Kapan dikembalikan, `pcBuffer` parameter berisi teks file JSON. `pxHostAddressData`Parameter berisi alamat IP dan port inti Greengrass yang dapat Anda sambungkan.

Untuk opsi penyesuaian lainnya, seperti mengalokasikan sertifikat secara dinamis, Anda harus memanggil yang berikut ini: APIs

`GGD_JSONRequestStart`

Membuat permintaan HTTP GET AWS IoT untuk memulai permintaan penemuan untuk menemukan inti Greengrass. `GD_SecureConnect_Send`digunakan untuk mengirim permintaan ke AWS IoT.

`GGD_JSONRequestGetSize`

Mendapat ukuran file JSON dari respons HTTP.

`GGD_JSONRequestGetFile`

Mendapat string objek JSON. `GGD_JSONRequestGetSize`dan `GGD_JSONRequestGetFile` gunakan `GGD_SecureConnect_Read` untuk mendapatkan data JSON dari soket. `GGD_JSONRequestStart`,`GGD_SecureConnect_Send`, `GGD_JSONRequestGetSize` harus dipanggil untuk menerima data JSON dari AWS IoT.

`GGD_GetIPandCertificateFromJSON`

Mengekstrak alamat IP dan sertifikat inti Greengrass dari data JSON. Anda dapat mengaktifkan pemilihan otomatis dengan mengatur `xAutoSelectFlag` ke`True`. Pemilihan otomatis menemukan perangkat inti pertama yang dapat dihubungkan oleh perangkat FreeRTOS Anda. Untuk terhubung ke inti Greengrass, panggil `GGD_SecureConnect_Connect` fungsi, meneruskan alamat IP, port, dan sertifikat perangkat inti. Untuk menggunakan pemilihan manual, atur bidang `HostParameters_t` parameter berikut:

`pcGroupName`

ID grup Greengrass tempat inti berada. Anda dapat menggunakan perintah `aws greengrass list-groups` CLI untuk menemukan ID grup Greengrass Anda.

`pcCoreAddress`

ARN dari inti Greengrass yang Anda hubungkan.

Pustaka CoreHTTP

> **Note**
>
> Konten di halaman ini mungkin tidak up-to-date. Silakan merujuk ke halaman perpustakaan Freertos.org untuk pembaruan terbaru.

Pustaka klien HTTP C untuk perangkat IoT kecil (MCU atau MPU kecil)

Pengantar

Pustaka CoreHTTP adalah implementasi klien dari subset standar HTTP/1.1. Standar HTTP menyediakan protokol stateless yang berjalan di atas TCP/IP dan sering digunakan dalam sistem informasi hiperteks terdistribusi, kolaboratif.

Pustaka CoreHTTP mengimplementasikan subset dari standar protokol HTTP/1.1. Pustaka ini telah dioptimalkan untuk footprint memori yang rendah. Pustaka menyediakan API yang sepenuhnya sinkron sehingga aplikasi dapat sepenuhnya mengelola konkurensi mereka. Ini hanya menggunakan buffer tetap, sehingga aplikasi memiliki kontrol penuh atas strategi alokasi memori mereka.

Perpustakaan ditulis dalam C dan dirancang agar sesuai dengan ISO C90 dan MISRA C: 2012. Satu-satunya dependensi perpustakaan adalah pustaka C standar dan versi LTS (v12.19.1) dari http-parser dari Node.js. Perpustakaan memiliki bukti yang menunjukkan penggunaan memori yang aman dan tidak ada alokasi heap, sehingga cocok untuk mikrokontroler IoT, tetapi juga sepenuhnya portabel ke platform lain.

Saat menggunakan koneksi HTTP dalam aplikasi IoT, kami menyarankan Anda menggunakan antarmuka transport yang aman, seperti yang menggunakan protokol TLS seperti yang ditunjukkan dalam file. Demo otentikasi timbal balik CoreHTTP

Pustaka ini dapat digunakan secara bebas dan didistribusikan di bawah lisensi open source MIT.

Ukuran Kode CoreHTTP (contoh yang dihasilkan dengan GCC untuk ARM Cortex-M)		
File	Dengan Optimasi -O1	Dengan Optimasi -Os
core_http_client.c	3.2K	2.6K
api.c (llhttp)	2.6K	2.0K
http.c (llhttp)	0.3K	0.3K
llhttp.c (llhttp)	17.9	15.9
Total perkiraan	23.9K	20.7K

Perpustakaan CoreJson

> **Note**
>
> Konten di halaman ini mungkin tidak up-to-date. Silakan merujuk ke halaman perpustakaan Freertos.org untuk pembaruan terbaru.

Pengantar

JSON (JavaScript Object Notation) adalah format serialisasi data yang dapat dibaca manusia. Ini banyak digunakan untuk bertukar data, seperti dengan layanan AWS IoT Device Shadow, dan merupakan bagian dari banyak APIs, seperti GitHub REST API. JSON dipertahankan sebagai standar oleh Ecma International.

Pustaka CoreJSON menyediakan parser yang mendukung pencarian kunci sambil secara ketat menegakkan sintaks Pertukaran Data JSON Standar ECMA-404. Perpustakaan ditulis dalam C dan dirancang untuk memenuhi ISO C90 dan MISRA C: 2012. Ini memiliki bukti yang menunjukkan penggunaan memori yang aman dan tidak ada alokasi tumpukan, sehingga cocok untuk mikrokontroler IoT, tetapi juga sepenuhnya portabel ke platform lain.

Penggunaan memori

Pustaka CoreJSON menggunakan tumpukan internal untuk melacak struktur bersarang dalam dokumen JSON. Tumpukan ada selama durasi panggilan fungsi tunggal; itu tidak dipertahankan.

Ukuran tumpukan dapat ditentukan dengan mendefinisikan makro,`JSON_MAX_DEPTH`, yang default ke 32 level. Setiap level mengkonsumsi satu byte.

Ukuran Kode CoreJson (contoh dihasilkan dengan GCC untuk ARM Cortex-M)		
File	Dengan Optimasi -O1	Dengan Optimasi -Os
core_json.c	2.9K	2.4K
Total perkiraan	2.9K	2.4K

Perpustakaan CoreMQTT

> **Note**
>
> Konten di halaman ini mungkin tidak up-to-date. Silakan merujuk ke halaman perpustakaan Freertos.org untuk pembaruan terbaru.

Pengantar

Pustaka CoreMQTT adalah implementasi klien dari standar MQTT (Message Queue Telemetry Transport). Standar MQTT menyediakan protokol pesan publish/subscribe (atau PubSub) ringan yang berjalan di atas TCP/IP dan sering digunakan dalam kasus penggunaan Machine to Machine (M2M) dan Internet of Things (IoT).

Pustaka CoreMQTT sesuai dengan standar protokol MQTT 3.1.1. Pustaka ini telah dioptimalkan untuk footprint memori yang rendah. Desain perpustakaan ini mencakup kasus penggunaan yang berbeda, mulai dari platform terbatas sumber daya yang hanya menggunakan pesan QoS 0 MQTT PUBLISH hingga platform kaya sumber daya menggunakan QoS 2 MQTT PUBLISH melalui koneksi TLS (Transport Layer Security). Library menyediakan menu fungsi composable, yang dapat dipilih dan digabungkan agar sesuai dengan kebutuhan kasus penggunaan tertentu.

Perpustakaan ditulis dalam C dan dirancang agar sesuai dengan ISO C90 dan MISRA C: 2012. Pustaka MQTT ini tidak memiliki dependensi pada pustaka tambahan apa pun kecuali untuk yang berikut ini:

- Pustaka C standar

- Antarmuka transportasi jaringan yang diimplementasikan pelanggan
- (Opsional) Fungsi waktu platform yang diterapkan pengguna

Perpustakaan dipisahkan dari driver jaringan yang mendasarinya melalui penyediaan spesifikasi antarmuka transportasi kirim dan terima yang sederhana. Penulis aplikasi dapat memilih antarmuka transportasi yang ada, atau mengimplementasikannya sendiri yang sesuai untuk aplikasi mereka.

Pustaka menyediakan API tingkat tinggi untuk terhubung ke broker MQTT, berlangganan/berhenti berlangganan topik, mempublikasikan pesan ke topik dan menerima pesan masuk. API ini mengambil antarmuka transport yang dijelaskan di atas sebagai parameter dan menggunakannya untuk mengirim dan menerima pesan ke dan dari broker MQTT.

Pustaka juga mengekspos serializer/deserializer API. This API can be used to build a simple IoT application consisting of only the required a subset of MQTT functionality, without any other overhead. The serializer/deserializer API tingkat rendah dapat digunakan bersama dengan API lapisan transport yang tersedia, seperti soket, untuk mengirim dan menerima pesan ke dan dari broker.

Saat menggunakan koneksi MQTT dalam aplikasi IoT, kami menyarankan Anda menggunakan antarmuka transport yang aman, seperti yang menggunakan protokol TLS.

Pustaka MQTT ini tidak memiliki dependensi platform, seperti threading atau sinkronisasi. Pustaka ini memang memiliki bukti yang menunjukkan penggunaan memori yang aman dan tidak ada alokasi heap, yang membuatnya cocok untuk mikrokontroler IoT, tetapi juga sepenuhnya portabel ke platform lain. Ini dapat digunakan secara bebas, dan didistribusikan di bawah lisensi open source MIT.

Ukuran Kode CoreMQTT (contoh yang dihasilkan dengan GCC untuk ARM Cortex-M)		
File	Dengan Optimasi -O1	Dengan Optimasi -Os
core_mqtt.c	4.0K	3.4K
core_mqtt_state.c	1.7K	1.3K
core_mqtt_serializer.c	2.8K	2.2K
Total perkiraan	8.5K	6.9K

Perpustakaan Agen CoreMQTT

> **Note**
>
> Konten di halaman ini mungkin tidak up-to-date. Silakan merujuk ke halaman perpustakaan Freertos.org untuk pembaruan terbaru.

Pengantar

Pustaka Agen CoreMQTT adalah API tingkat tinggi yang menambahkan keamanan utas ke file. Perpustakaan CoreMQTT Ini memungkinkan Anda membuat tugas agen MQTT khusus yang mengelola koneksi MQTT di latar belakang dan tidak memerlukan intervensi apa pun dari tugas lain. Pustaka menyediakan thread aman yang setara dengan CoreMQTT APIs, sehingga dapat digunakan di lingkungan multi-threaded.

Agen MQTT adalah tugas independen (atau utas eksekusi). Ini mencapai keamanan utas dengan menjadi satu-satunya tugas yang diizinkan untuk mengakses API pustaka MQTT. Ini membuat serial akses dengan mengisolasi semua panggilan API MQTT ke satu tugas, dan menghilangkan kebutuhan untuk semaphores atau primitif sinkronisasi lainnya.

Pustaka menggunakan antrian pesan aman utas (atau mekanisme komunikasi antar-proses lainnya) untuk membuat serial semua permintaan untuk memanggil MQTT. APIs Implementasi pesan dipisahkan dari perpustakaan melalui antarmuka pesan, yang memungkinkan perpustakaan untuk di-porting ke sistem operasi lain. Antarmuka pesan terdiri dari fungsi untuk mengirim dan menerima pointer ke struktur perintah agen, dan fungsi untuk mengalokasikan objek perintah ini, yang memungkinkan penulis aplikasi untuk memutuskan strategi alokasi memori yang sesuai untuk aplikasi mereka.

Perpustakaan ditulis dalam C dan dirancang agar sesuai dengan ISO C90 dan MISRA C: 2012. Pustaka tidak memiliki dependensi pada pustaka tambahan selain Perpustakaan CoreMQTT dan pustaka C standar. Perpustakaan memiliki bukti yang menunjukkan penggunaan memori yang aman dan tidak ada alokasi heap, sehingga dapat digunakan untuk mikrokontroler IoT, tetapi juga sepenuhnya portabel ke platform lain.

Pustaka ini dapat digunakan secara bebas dan didistribusikan di bawah lisensi open source MIT.

Ukuran Kode Agen CoreMQTT (contoh dihasilkan dengan GCC untuk ARM Cortex-M)		
File	Dengan Optimasi -O1	Dengan Optimasi -Os
core_mqtt_agent.c	1.7K	1,5K
core_mqtt_agent_command_functions.c	0.3K	0.2K
core_mqtt.c (CoreMQTT)	4.0K	3.4K
core_mqtt_state.c (CoreMQTT)	1.7K	1.3K
core_mqtt_serializer.c (CoreMQTT)	2.8K	2.2K
Total perkiraan	10.5K	8.6K

AWS IoT Perpustakaan Over the Air (OTA)

> **Note**
>
> Konten di halaman ini mungkin tidak up-to-date. Silakan merujuk ke halaman perpustakaan Freertos.org untuk pembaruan terbaru.

Pengantar

Pustaka pembaruan AWS IoT Over-the-air (OTA) memungkinkan Anda mengelola pemberitahuan, unduhan, dan verifikasi pembaruan firmware untuk perangkat FreeRTOS menggunakan HTTP atau MQTT sebagai protokol. Dengan menggunakan perpustakaan Agen OTA, Anda dapat secara logis memisahkan pembaruan firmware dan aplikasi yang berjalan di perangkat Anda. Agen OTA dapat berbagi koneksi jaringan dengan aplikasi. Dengan berbagi koneksi jaringan, Anda berpotensi menghemat sejumlah besar RAM. Selain itu, pustaka Agen OTA memungkinkan Anda menentukan logika khusus aplikasi untuk menguji, melakukan, atau memutar kembali pembaruan firmware.

Internet of Things (IoT) memperluas konektivitas internet ke perangkat tertanam yang secara tradisional tidak terhubung. Perangkat ini dapat diprogram untuk mengkomunikasikan data yang

dapat digunakan melalui internet, dan dapat dipantau dan dikendalikan dari jarak jauh. Dengan kemajuan teknologi, perangkat tertanam tradisional ini mendapatkan kemampuan internet di ruang konsumen, industri, dan perusahaan dengan cepat.

Perangkat IoT biasanya digunakan dalam jumlah besar dan seringkali di tempat-tempat yang sulit atau tidak praktis untuk diakses oleh operator manusia. Bayangkan sebuah skenario di mana kerentanan keamanan yang dapat mengekspos data ditemukan. Dalam skenario seperti itu, penting untuk memperbarui perangkat yang terpengaruh dengan perbaikan keamanan dengan cepat dan andal. Tanpa kemampuan untuk melakukan pembaruan OTA, mungkin juga sulit untuk memperbarui perangkat yang tersebar secara geografis. Memiliki teknisi memperbarui perangkat ini akan mahal, memakan waktu, dan sering kali tidak praktis. Waktu yang diperlukan untuk memperbarui perangkat ini membuat mereka terkena kerentanan keamanan untuk jangka waktu yang lebih lama. Mengingat perangkat ini untuk memperbarui juga akan mahal dan dapat menyebabkan gangguan signifikan bagi konsumen karena downtime.

Pembaruan Over the Air (OTA) memungkinkan untuk memperbarui firmware perangkat tanpa penarikan mahal atau kunjungan teknisi. Metode ini menambahkan manfaat berikut:

- Keamanan - Kemampuan untuk dengan cepat merespons kerentanan keamanan dan bug perangkat lunak yang ditemukan setelah perangkat digunakan di lapangan.
- Inovasi - Produk dapat sering diperbarui saat fitur baru dikembangkan, mendorong siklus inovasi. Pembaruan dapat berlaku dengan cepat dengan waktu henti minimum dibandingkan dengan metode pembaruan tradisional.
- Biaya - Pembaruan OTA dapat mengurangi biaya pemeliharaan secara signifikan dibandingkan dengan metode yang secara tradisional digunakan untuk memperbarui perangkat ini.

Menyediakan fungsionalitas OTA memerlukan pertimbangan desain berikut:

- Komunikasi Aman - Pembaruan harus menggunakan saluran komunikasi terenkripsi untuk mencegah unduhan dirusak selama transit.
- Pemulihan - Pembaruan dapat gagal karena hal-hal seperti konektivitas jaringan intermiten atau menerima pembaruan yang tidak valid. Dalam skenario ini, perangkat harus dapat kembali ke keadaan stabil dan menghindari menjadi bata.
- Verifikasi Penulis - Pembaruan harus diverifikasi agar berasal dari sumber tepercaya, bersama dengan validasi lain seperti memeriksa versi dan kompatibilitas.

Untuk informasi selengkapnya tentang menyiapkan pembaruan OTA dengan FreeRTOS, lihat. Pembaruan FreeRTOS Over-the-Air

AWS IoT Perpustakaan Over the Air (OTA)

Pustaka AWS IoT OTA memungkinkan Anda mengelola pemberitahuan pembaruan yang baru tersedia, mengunduhnya, dan melakukan verifikasi kriptografi pembaruan firmware. Menggunakan pustaka klien over-the-air (OTA), Anda dapat secara logis memisahkan mekanisme pembaruan firmware dari aplikasi yang berjalan di perangkat Anda. Pustaka klien over-the-air (OTA) dapat berbagi koneksi jaringan dengan aplikasi, menghemat memori di perangkat yang dibatasi sumber daya. Selain itu, pustaka klien over-the-air (OTA) memungkinkan Anda menentukan logika khusus aplikasi untuk menguji, melakukan, atau memutar kembali pembaruan firmware. Pustaka mendukung protokol aplikasi yang berbeda seperti Message Queuing Telemetry Transport (MQTT) dan Hypertext Transfer Protocol (HTTP) dan menyediakan berbagai opsi konfigurasi yang dapat Anda sesuaikan untuk jenis dan kondisi jaringan Anda.

Pustaka ini APIs menyediakan fungsi-fungsi utama ini:

- Mendaftar untuk pemberitahuan atau jajak pendapat untuk permintaan pembaruan baru yang tersedia.
- Menerima, mengurai, dan memvalidasi permintaan pembaruan.
- Unduh dan verifikasi file sesuai dengan informasi dalam permintaan pembaruan.
- Jalankan self-test sebelum mengaktifkan pembaruan yang diterima untuk memastikan validitas fungsional pembaruan.
- Perbarui status perangkat.

Pustaka ini menggunakan AWS layanan untuk mengelola berbagai fungsi terkait cloud seperti mengirim pembaruan firmware, memantau sejumlah besar perangkat di berbagai wilayah, mengurangi radius ledakan penyebaran yang salah, dan memverifikasi keamanan pembaruan. Pustaka ini dapat digunakan dengan pustaka MQTT atau HTTP apa pun.

Demo untuk pustaka ini menunjukkan over-the-air pembaruan lengkap menggunakan Perpustakaan dan Layanan CoreMQTT AWS pada perangkat FreeRTOS.

Fitur

Berikut adalah antarmuka Agen OTA yang lengkap:

OTA_Init

Menginisialisasi mesin OTA dengan memulai Agen OTA (“Tugas OTA”) dalam sistem. Hanya satu Agen OTA yang mungkin ada.

OTA_Shutdown

Sinyal ke Agen OTA untuk mematikan. Agen OTA secara opsional akan berhenti berlangganan dari semua topik pemberitahuan pekerjaan MQTT, berhenti dalam proses pekerjaan OTA, jika ada, dan menghapus semua sumber daya.

OTA_GetState

Mendapat status Agen OTA saat ini.

OTA_ActivateNewImage

Mengaktifkan gambar firmware mikrokontroler terbaru yang diterima melalui OTA. (Status pekerjaan terperinci sekarang harus self-test.)

OTA_SetImageState

Menetapkan status validasi gambar firmware mikrokontroler yang sedang berjalan (pengujian, diterima atau ditolak).

OTA_GetImageState

Mendapat status gambar firmware mikrokontroler yang sedang berjalan (pengujian, diterima atau ditolak).

OTA_CheckForUpdate

Meminta pembaruan OTA berikutnya yang tersedia dari layanan Pembaruan OTA.

OTA_Suspend

Tangguhkan semua operasi Agen OTA.

OTA_Resume

Lanjutkan operasi Agen OTA.

OTA_SignalEvent

Sinyal acara ke tugas Agen OTA.

OTA_EventProcessingTask

Loop pemrosesan acara agen OTA.

`OTA_GetStatistics`

Dapatkan statistik paket pesan OTA yang mencakup jumlah paket yang diterima, antri, diproses dan dijatuhkan.

`OTA_Err_strerror`

Kode kesalahan ke konversi string untuk kesalahan OTA.

`OTA_JobParse_strerror`

Mengkonversi kode kesalahan OTA Job Parsing menjadi string.

`OTA_PalStatus_strerror`

Kode status ke konversi string untuk status OTA PAL.

`OTA_OsStatus_strerror`

Kode status ke konversi string untuk status OS OTA.

Referensi API

Untuk informasi selengkapnya, lihat AWS IoT Over-the-air Update: Functions.

Contoh penggunaan

Aplikasi perangkat berkemampuan OTA yang khas menggunakan protokol MQTT menggerakkan Agen OTA dengan menggunakan urutan panggilan API berikut.

1. Connect ke Agen AWS IoT CoreMQTT. Untuk informasi selengkapnya, lihat Perpustakaan Agen CoreMQTT.
2. Inisialisasi Agen OTA dengan memanggil`OTA_Init`, termasuk buffer, antarmuka ota yang diperlukan, nama benda, dan panggilan balik aplikasi. Callback mengimplementasikan logika khusus aplikasi yang dijalankan setelah menyelesaikan pekerjaan pembaruan OTA.
3. Ketika pembaruan OTA selesai, FreeRTOS memanggil callback penyelesaian pekerjaan dengan salah satu peristiwa `accepted` berikut:,, atau. `rejected` `self test`
4. Jika gambar firmware baru telah ditolak (misalnya, karena kesalahan validasi), aplikasi biasanya dapat mengabaikan notifikasi dan menunggu pembaruan berikutnya.
5. Jika pembaruan valid dan telah ditandai sebagai diterima, hubungi `OTA_ActivateNewImage` untuk mengatur ulang perangkat dan mem-boot gambar firmware baru.

Porting

Untuk informasi tentang porting fungsionalitas OTA ke platform Anda, lihat Mem-porting Perpustakaan OTA di Panduan Porting FreerTOS.

Penggunaan memori

Ukuran Kode AWS IoT OTA (contoh yang dihasilkan dengan GCC untuk ARM Cortex-M)		
File	Dengan Optimasi -O1	Dengan Optimasi -Os
ota.c	8.3K	7.5K
ota_interface.c	0,1K	0,1K
ota_base64.c	0.6K	0.6K
ota_mqtt.c	2.4K	2.2K
ota_cbor.c	0.8K	0.6K
ota_http.c	0.3K	0.3K
Total perkiraan	12.5K	11.3K

PKCS11 perpustakaan inti

> **Note**
>
> Konten di halaman ini mungkin tidak up-to-date. Silakan merujuk ke halaman perpustakaan Freertos.org untuk pembaruan terbaru.

Gambaran Umum

Standar Kriptografi Kunci Publik #11 mendefinisikan API independen platform untuk mengelola dan menggunakan token kriptografi. PKCS #11 mengacu pada API yang ditentukan oleh standar dan standar itu sendiri. API kriptografi PKCS #11 mengabstraksi penyimpanan kunci, mendapatkan/mengatur properti untuk objek kriptografi, dan semantik sesi. Ini banyak digunakan

untuk memanipulasi objek kriptografi umum, dan ini penting karena fungsi yang ditentukannya memungkinkan perangkat lunak aplikasi untuk menggunakan, membuat, memodifikasi, dan menghapus objek kriptografi, tanpa pernah mengekspos objek tersebut ke memori aplikasi. Misalnya, integrasi referensi AWS FreeRTOS menggunakan subset kecil dari PKCS #11 API untuk mengakses kunci rahasia (pribadi) yang diperlukan untuk membuat koneksi jaringan yang diautentikasi dan diamankan oleh protokol Transport Layer Security (TLS) tanpa aplikasi pernah 'melihat' kunci.

PKCS11 Pustaka inti berisi implementasi tiruan berbasis perangkat lunak dari antarmuka (API) PKCS #11 yang menggunakan fungsionalitas kriptografi yang disediakan oleh Mbed TLS. Menggunakan tiruan perangkat lunak memungkinkan pengembangan dan fleksibilitas yang cepat, tetapi diharapkan Anda akan mengganti tiruan dengan implementasi khusus untuk penyimpanan kunci aman yang digunakan dalam perangkat produksi Anda. Umumnya, vendor untuk kriptoprosesor aman, seperti Trusted Platform Module (TPM), Hardware Security Module (HSM), Secure Element, atau jenis enklave perangkat keras aman lainnya, mendistribusikan implementasi PKCS #11 dengan perangkat keras. Oleh karena itu, tujuan dari pustaka tiruan PKCS11 perangkat lunak inti adalah untuk menyediakan implementasi PKCS #11 khusus non perangkat keras yang memungkinkan pembuatan prototipe dan pengembangan cepat sebelum beralih ke implementasi PKCS #11 khusus kriptoprosesor di perangkat produksi.

Hanya sebagian dari standar PKCS #11 yang diimplementasikan, dengan fokus pada operasi yang melibatkan kunci asimetris, pembuatan angka acak, dan hashing. Kasus penggunaan yang ditargetkan termasuk sertifikat dan manajemen kunci untuk otentikasi TLS, dan verifikasi tanda tangan kode, pada perangkat kecil yang disematkan. Lihat file `pkcs11.h` (diperoleh dari OASIS, badan standar) di repositori kode sumber FreeRTOS. Dalam implementasi referensi FreeRTOS, panggilan API PKCS #11 dilakukan oleh antarmuka pembantu TLS untuk melakukan otentikasi klien TLS selama. `SOCKETS_Connect` Panggilan API PKCS #11 juga dilakukan oleh alur kerja penyediaan pengembang satu kali kami untuk mengimpor sertifikat klien TLS dan kunci pribadi untuk otentikasi ke broker MQTT. AWS IoT Kedua kasus penggunaan tersebut, penyediaan dan otentikasi klien TLS, hanya memerlukan implementasi sebagian kecil dari standar antarmuka PKCS #11.

Fitur

Subset PKCS #11 berikut digunakan. Daftar ini kira-kira dalam urutan di mana rutinitas dipanggil untuk mendukung penyediaan, otentikasi klien TLS, dan pembersihan. Untuk penjelasan rinci tentang fungsi, lihat dokumentasi PKCS #11 yang disediakan oleh panitia standar.

Penyiapan umum dan merobohkan API

- `C_Initialize`

- C_Finalize
- C_GetFunctionList
- C_GetSlotList
- C_GetTokenInfo
- C_OpenSession
- C_CloseSession
- C_Login

Penyediaan API

- C_CreateObject CKO_PRIVATE_KEY(untuk kunci pribadi perangkat)
- C_CreateObject CKO_CERTIFICATE(untuk sertifikat perangkat dan sertifikat verifikasi kode)
- C_GenerateKeyPair
- C_DestroyObject

Autentikasi Klien

- C_GetAttributeValue
- C_FindObjectsInit
- C_FindObjects
- C_FindObjectsFinal
- C_GenerateRandom
- C_SignInit
- C_Sign
- C_VerifyInit
- C_Verify
- C_DigestInit
- C_DigestUpdate
- C_DigestFinal

Dukungan kriptosistem asimetris

Implementasi referensi FreeRTOS menggunakan PKCS #11 2048-bit RSA (hanya penandatanganan) dan ECDSA dengan kurva NIST P-256. Petunjuk berikut menjelaskan cara membuat AWS IoT sesuatu berdasarkan sertifikat klien P-256.

Pastikan Anda menggunakan versi OpenSSL dan OpenSSL berikut (atau yang AWS CLI lebih baru):

```
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
```

Prosedur berikut mengasumsikan bahwa Anda menggunakan `aws configure` perintah untuk mengkonfigurasi. AWS CLI Untuk informasi selengkapnya, lihat Konfigurasi cepat dengan `aws configure` di Panduan AWS Command Line Interface Pengguna.

Untuk membuat AWS IoT sesuatu berdasarkan sertifikat klien P-256

1. Buat AWS IoT sesuatu.

   ```
   aws iot create-thing --thing-name thing-name
   ```

2. Gunakan OpenSSL untuk membuat kunci 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. Buat permintaan pendaftaran sertifikat yang ditandatangani oleh kunci yang dibuat pada langkah 2.

   ```
   openssl req -new -nodes -days 365 -key thing-name.key -out thing-name.req
   ```

4. Kirim permintaan pendaftaran sertifikat ke. 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. Lampirkan sertifikat (direferensikan oleh output ARN dengan perintah sebelumnya) ke benda itu.

```
aws iot attach-thing-principal --thing-name thing-name \
  --principal "arn:aws:iot:us-
east-1:123456789012:cert/
86e41339a6d1bbc67abf31faf455092cdebf8f21ffbc67c4d238d1326c7de729"
```

6. Buat kebijakan. (Kebijakan ini terlalu permisif. Ini harus digunakan untuk tujuan pengembangan saja.)

```
aws iot create-policy --policy-name FullControl --policy-document file://
policy.json
```

Berikut ini adalah daftar file policy.json yang ditentukan dalam perintah. `create-policy` Anda dapat menghilangkan `greengrass:*` tindakan jika Anda tidak ingin menjalankan demo FreeRTOS untuk konektivitas dan penemuan Greengrass.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:*",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "greengrass:*",
      "Resource": "*"
    }
  ]
}
```

7. Lampirkan kepala sekolah (sertifikat) dan kebijakan pada benda itu.

```
aws iot attach-principal-policy --policy-name FullControl \
  --principal "arn:aws:iot:us-
east-1:123456789012:cert/
86e41339a6d1bbc67abf31faf455092cdebf8f21ffbc67c4d238d1326c7de729"
```

Sekarang, ikuti langkah-langkah di bagian AWS IoT Memulai panduan ini. Jangan lupa untuk menyalin sertifikat dan kunci pribadi yang Anda buat ke `aws_clientcredential_keys.h` file Anda. Salin nama benda Anda ke dalam`aws_clientcredential.h`.

> **Note**
>
> Sertifikat dan kunci pribadi dikodekan keras untuk tujuan demonstrasi saja. Aplikasi tingkat produksi harus menyimpan file-file ini di lokasi yang aman.

Porting

Untuk informasi tentang porting PKCS11 pustaka inti ke platform Anda, lihat Porting PKCS11 Perpustakaan inti di Panduan Porting FreerTOS.

Penggunaan memori

Ukuran Kode inti PKCS11 (contoh dihasilkan dengan GCC untuk ARM Cortex-M)		
File	Dengan Optimasi -O1	Dengan Optimasi -Os
core_pkcs11.c	0.8K	0.8K
core_pki_utils.c	0.5K	0.3K
core_pkcs11_mbedtls.c	8.9K	7.5K
Total perkiraan	10.2K	8.6K

Perpustakaan Secure Sockets

> **Important**
>
> Pustaka ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-Freertos yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Gambaran Umum

Anda dapat menggunakan pustaka FreeRTOS Secure Sockets untuk membuat aplikasi tertanam yang berkomunikasi dengan aman. Perpustakaan ini dirancang untuk memudahkan onboarding bagi pengembang perangkat lunak dari berbagai latar belakang pemrograman jaringan.

Pustaka FreeRTOS Secure Sockets didasarkan pada antarmuka soket Berkeley, dengan opsi komunikasi aman tambahan oleh protokol TLS. Untuk informasi tentang perbedaan antara pustaka FreeRTOS Secure Sockets dan antarmuka soket Berkeley, `SOCKETS_SetSockOpt` lihat di Referensi API Secure Sockets.

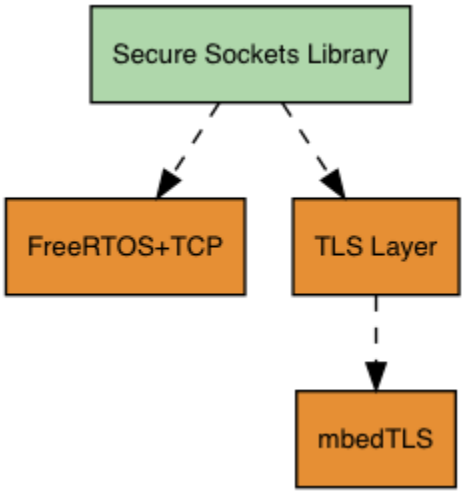
> **Note**
>
> Saat ini, hanya klien APIs, ditambah implementasi IP ringan (LWiP) dari API `Bind` sisi server, yang didukung untuk FreeRTOS Secure Sockets.

Ketergantungan dan persyaratan

Pustaka FreeRTOS Secure Sockets bergantung pada tumpukan TCP/IP dan implementasi TLS. Port untuk FreeRTOS memenuhi dependensi ini dalam salah satu dari tiga cara:

- Implementasi kustom dari TCP/IP dan TLS
- Implementasi kustom TCP/IP, dan lapisan FreeRTOS TLS dengan mbedTLS
- FreerTos+TCP dan lapisan FreeRTOS TLS dengan mbedTLS

Diagram ketergantungan di bawah ini menunjukkan implementasi referensi yang disertakan dengan pustaka FreeRTOS Secure Sockets. Implementasi referensi ini mendukung TLS dan TCP/IP melalui Ethernet dan Wi-Fi dengan Freertos+TCP dan mBedTLS sebagai dependensi. Untuk informasi selengkapnya tentang layer FreeRTOS TLS, lihat. Keamanan Lapisan Transportasi



Fitur

Fitur perpustakaan FreeRTOS Secure Sockets meliputi:

- Antarmuka standar berbasis soket Berkeley
- Thread-safe APIs untuk mengirim dan menerima data
- Easy-to-enable TLS

Pemecahan Masalah

Kode eror

Kode kesalahan yang dikembalikan oleh pustaka FreeRTOS Secure Sockets adalah nilai negatif. Untuk informasi selengkapnya tentang setiap kode kesalahan, lihat Kode Kesalahan Soket Aman di Referensi API Soket Aman.

> **Note**
>
> Jika FreeRTOS Secure Sockets API mengembalikan kode kesalahan, maka, yang bergantung pada Perpustakaan CoreMQTT pustaka Freertos Secure Sockets, mengembalikan kode kesalahan. `AWS_IOT_MQTT_SEND_ERROR`

Dukungan pengembang

Pustaka FreeRTOS Secure Sockets mencakup dua makro pembantu untuk menangani alamat IP:

`SOCKETS_inet_addr_quick`

Makro ini mengubah alamat IP yang dinyatakan sebagai empat oktet numerik terpisah menjadi alamat IP yang dinyatakan sebagai nomor 32-bit dalam urutan jaringan-byte.

`SOCKETS_inet_ntoa`

Makro ini mengubah alamat IP yang dinyatakan sebagai nomor 32-bit dalam urutan byte jaringan menjadi string dalam notasi desimal titik.

Pembatasan penggunaan

Hanya soket TCP yang didukung oleh perpustakaan FreeRTOS Secure Sockets. Soket UDP tidak didukung.

Server tidak APIs didukung oleh pustaka Freertos Secure Sockets, kecuali untuk implementasi IP ringan (LWiP) dari API sisi server. `Bind` APIs Klien didukung.

Inisialisasi

Untuk menggunakan pustaka FreeRTOS Secure Sockets, Anda perlu menginisialisasi perpustakaan dan dependensinya. Untuk menginisialisasi library Secure Sockets, gunakan kode berikut dalam aplikasi Anda:

```
BaseType_t xResult = pdPASS;
xResult = SOCKETS_Init();
```

Pustaka dependen harus diinisialisasi secara terpisah. Misalnya, jika Freertos+TCP adalah dependensi, Anda perlu memanggil dalam aplikasi Anda juga. `FreeRTOS_IPInit`

Referensi API

Untuk referensi API selengkapnya, lihat Referensi API Secure Sockets.

Contoh penggunaan

Kode berikut menghubungkan klien ke 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 );
}
```

Untuk contoh lengkap, lihatSoket Aman menggemakan demo klien.

Porting

FreeRTOS Secure Sockets bergantung pada tumpukan TCP/IP dan implementasi TLS. Bergantung pada tumpukan Anda, untuk mem-port pustaka Secure Sockets, Anda mungkin perlu mem-port beberapa hal berikut:

- Tumpukan Freertos+TCP TCP/IP
- Sebuah PKCS11 perpustakaan inti
- Sebuah Keamanan Lapisan Transportasi

Untuk informasi selengkapnya tentang porting, lihat Porting Perpustakaan Soket Aman di Panduan Porting FreerTOS.

AWS IoT Pustaka Device Shadow

> **Note**
>
> Konten di halaman ini mungkin tidak up-to-date. Silakan merujuk ke halaman perpustakaan Freertos.org untuk pembaruan terbaru.

Pengantar

Anda dapat menggunakan pustaka AWS IoT Device Shadow untuk menyimpan dan mengambil status saat ini (bayangan) dari setiap perangkat yang terdaftar. Bayangan perangkat adalah representasi virtual yang persisten dari perangkat Anda yang dapat berinteraksi dengan Anda dalam aplikasi web Anda bahkan jika perangkat sedang offline. Status perangkat ditangkap sebagai bayangannya dalam dokumen JSON. Anda dapat mengirim perintah ke layanan AWS IoT Device Shadow melalui MQTT atau HTTP untuk menanyakan status perangkat terbaru yang diketahui, atau untuk mengubah status. Setiap bayangan perangkat secara unik diidentifikasi dengan nama benda yang sesuai, representasi perangkat tertentu atau entitas logis di AWS Cloud. Untuk informasi selengkapnya, lihat Mengelola Perangkat dengan AWS IoT. Rincian lebih lanjut tentang bayangan dapat ditemukan di AWS IoT dokumentasi.

Pustaka AWS IoT Device Shadow tidak memiliki dependensi pada pustaka tambahan selain pustaka C standar. Itu juga tidak memiliki dependensi platform apa pun, seperti threading atau sinkronisasi. Ini dapat digunakan dengan perpustakaan MQTT dan perpustakaan JSON apa pun.

Pustaka ini dapat digunakan secara bebas dan didistribusikan di bawah lisensi open source MIT.

Ukuran Kode AWS IoT Device Shadow (contoh yang dihasilkan dengan GCC untuk ARM Cortex-M)		
File	Dengan Optimasi -O1	Dengan Optimasi -Os
bayangan.c	1.2K	0.9K
Total perkiraan	1.2K	0,9K

AWS IoT Perpustakaan Lowongan Kerja

> **Note**
>
> Konten di halaman ini mungkin tidak up-to-date. Silakan merujuk ke halaman perpustakaan Freertos.org untuk pembaruan terbaru.

Pengantar

AWS IoT Jobs adalah layanan yang memberi tahu satu atau lebih perangkat yang terhubung tentang pekerjaan yang tertunda. Anda dapat menggunakan pekerjaan untuk mengelola armada perangkat, memperbarui firmware dan sertifikat keamanan di perangkat Anda, atau melakukan tugas administratif seperti memulai ulang perangkat dan melakukan diagnostik. Untuk informasi selengkapnya, lihat Tugas di AWS IoT Panduan Developer. Interaksi dengan layanan AWS IoT Jobs menggunakan MQTT, protokol berlangganan publikasi ringan. Pustaka ini menyediakan API untuk menyusun dan mengenali string topik MQTT yang digunakan oleh layanan Jobs. AWS IoT

Perpustakaan AWS IoT Jobs ditulis dalam C dan dirancang agar sesuai dengan ISO C90 dan MISRA C: 2012. Pustaka tidak memiliki dependensi pada pustaka tambahan selain pustaka C standar. Ini dapat digunakan dengan perpustakaan MQTT dan perpustakaan JSON apa pun. Perpustakaan memiliki bukti yang menunjukkan penggunaan memori yang aman dan tidak ada alokasi heap, sehingga cocok untuk mikrokontroler IoT, tetapi juga sepenuhnya portabel ke platform lain.

Perpustakaan ini dapat digunakan secara bebas dan didistribusikan di bawah lisensi open source MIT.

Ukuran Kode AWS IoT Pekerjaan (contoh yang dihasilkan dengan GCC untuk ARM Cortex-M)		
File	Dengan Optimasi -O1	Dengan Optimasi -Os
jobs.c	1.9K	1.6K
Total perkiraan	1.9K	1.6K

Keamanan Lapisan Transportasi

> **Important**
>
> Pustaka ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-Freertos yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Antarmuka FreeRTOS Transport Layer Security (TLS) adalah pembungkus opsional tipis yang digunakan untuk mengabstraksi detail implementasi kriptografi dari antarmuka Secure Sockets Layer (SSL) di atasnya dalam tumpukan protokol. Tujuan dari antarmuka TLS adalah untuk membuat perpustakaan kripto perangkat lunak saat ini, mbed TLS, mudah diganti dengan implementasi alternatif untuk negosiasi protokol TLS dan primitif kriptografi. Antarmuka TLS dapat ditukar tanpa perubahan apa pun yang diperlukan pada antarmuka SSL. Lihat `iot_tls.h` di repositori kode sumber FreeRTOS.

Antarmuka TLS bersifat opsional karena Anda dapat memilih untuk berinteraksi langsung dari SSL ke perpustakaan kripto. Antarmuka tidak digunakan untuk solusi MCU yang mencakup implementasi offload full-stack TLS dan transportasi jaringan.

Untuk informasi selengkapnya tentang porting antarmuka TLS, lihat Porting Perpustakaan TLS di Panduan Porting FreeRTOS.

Pustaka Wi-Fi

> **Important**
>
> Pustaka ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-Freertos yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Gambaran Umum

Pustaka Wi-Fi FreeRTOS mengabstraksi implementasi Wi-Fi khusus port ke dalam API umum yang menyederhanakan pengembangan aplikasi dan porting untuk semua papan berkualifikasi Freertos dengan kemampuan Wi-Fi. Dengan menggunakan API umum ini, aplikasi dapat berkomunikasi dengan tumpukan nirkabel tingkat rendah mereka melalui antarmuka umum.

Dependensi dan persyaratan

Pustaka Wi-Fi FreeRTOS membutuhkan inti Freertos+TCP.

Fitur

Pustaka Wi-Fi mencakup fitur-fitur berikut:

- Support untuk WEP, WPA WPA2, dan otentikasi WPA3
- Pemindaian Titik Akses
- Manajemen daya
- Profil jaringan

Untuk informasi selengkapnya tentang fitur perpustakaan Wi-Fi, lihat di bawah.

Mode Wi-Fi

Perangkat Wi-Fi dapat berada dalam salah satu dari tiga mode: Station, Access Point, atau P2P. Anda bisa mendapatkan mode perangkat Wi-Fi saat ini dengan meneleponWIFI_GetMode. Anda dapat mengatur mode wi-fi perangkat dengan meneleponWIFI_SetMode. Beralih mode dengan memanggil WIFI_SetMode memutus perangkat, jika sudah terhubung ke jaringan.

Mode stasiun

Setel perangkat Anda ke mode Stasiun untuk menghubungkan papan ke titik akses yang ada.

Mode Titik Akses (AP)

Setel perangkat Anda ke mode AP untuk menjadikan perangkat sebagai titik akses bagi perangkat lain untuk terhubung. Saat perangkat Anda dalam mode AP, Anda dapat menghubungkan perangkat lain ke perangkat FreeRTOS Anda dan mengonfigurasi kredenal Wi-Fi baru. Untuk mengkonfigurasi mode AP, panggil`WIFI_ConfigureAP`. Untuk menempatkan perangkat Anda ke mode AP, hubungi`WIFI_StartAP`. Untuk mematikan mode AP, hubungi`WIFI_StopAP`.

> **Note**
>
> Pustaka FreeRTOS tidak menyediakan penyediaan Wi-Fi dalam mode AP. Anda harus menyediakan fungsionalitas tambahan, termasuk kemampuan server DHCP dan HTTP, untuk mencapai dukungan penuh mode AP.

Modus P2P

Atur perangkat Anda ke mode P2P untuk memungkinkan beberapa perangkat terhubung satu sama lain secara langsung, tanpa titik akses.

Keamanan

Wi-Fi API mendukung WEP, WPA WPA2, dan jenis WPA3 keamanan. Saat perangkat dalam mode Stasiun, Anda harus menentukan jenis keamanan jaringan saat memanggil `WIFI_ConnectAP` fungsi. Saat perangkat dalam mode AP, perangkat dapat dikonfigurasi untuk menggunakan salah satu jenis keamanan yang didukung:

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

Memindai dan menghubungkan

Untuk memindai titik akses terdekat, atur perangkat Anda ke mode Stasiun, dan panggil `WIFI_Scan` fungsinya. Jika Anda menemukan jaringan yang diinginkan dalam pemindaian, Anda dapat terhubung ke jaringan dengan menelepon `WIFI_ConnectAP` dan memberikan kredenal jaringan. Anda dapat memutuskan sambungan perangkat Wi-Fi dari jaringan dengan meneleponWIFI_Disconnect. Untuk informasi selengkapnya tentang pemindaian dan penghubung, lihat Contoh penggunaan danReferensi API.

Manajemen daya

Perangkat Wi-Fi yang berbeda memiliki kebutuhan daya yang berbeda, tergantung pada aplikasi dan sumber daya yang tersedia. Perangkat mungkin selalu dinyalakan untuk mengurangi latensi atau mungkin terhubung sebentar-sebentar dan beralih ke mode daya rendah saat Wi-Fi tidak diperlukan. API antarmuka mendukung berbagai mode manajemen daya seperti selalu aktif, daya rendah, dan mode normal. Anda mengatur mode daya untuk perangkat menggunakan `WIFI_SetPMMode` fungsi tersebut. Anda bisa mendapatkan mode daya perangkat saat ini dengan memanggil `WIFI_GetPMMode` fungsi tersebut.

Profil jaringan

Pustaka Wi-Fi memungkinkan Anda menyimpan profil jaringan di memori non-volatile perangkat Anda. Hal ini memungkinkan Anda untuk menyimpan pengaturan jaringan sehingga mereka dapat diambil ketika perangkat terhubung kembali ke jaringan Wi-Fi, menghapus kebutuhan untuk menyediakan perangkat lagi setelah mereka telah terhubung ke jaringan. `WIFI_NetworkAdd`menambahkan profil jaringan. `WIFI_NetworkGet`mengambil profil jaringan. `WIFI_NetworkDel`menghapus profil jaringan. Jumlah profil yang dapat Anda simpan tergantung pada platform.

Konfigurasi

Untuk menggunakan pustaka Wi-Fi, Anda perlu menentukan beberapa pengidentifikasi dalam file konfigurasi. Untuk informasi tentang pengidentifikasi ini, lihat. Referensi API

> **Note**
>
> Pustaka tidak menyertakan file konfigurasi yang diperlukan. Anda harus membuat satu. Saat membuat file konfigurasi Anda, pastikan untuk menyertakan pengidentifikasi konfigurasi khusus papan apa pun yang dibutuhkan papan Anda.

Inisialisasi

Sebelum Anda menggunakan pustaka Wi-Fi, Anda perlu menginisialisasi beberapa komponen khusus papan, selain komponen FreeRTOS. Menggunakan `vendors/vendor/boards/board/aws_demos/application_code/main.c` file sebagai template untuk inisialisasi, lakukan hal berikut:

1. Hapus contoh logika koneksi Wi-Fi `main.c` jika aplikasi Anda menangani koneksi Wi-Fi. Ganti panggilan `DEMO_RUNNER_RunDemos()` fungsi berikut:

   ```
   if( SYSTEM_Init() == pdPASS )
       {
       ...
           DEMO_RUNNER_RunDemos();
       ...
           }
   ```

   Dengan panggilan ke aplikasi Anda sendiri:

   ```
   if( SYSTEM_Init() == pdPASS )
       {
       ...
           // This function should create any tasks
           // that your application requires to run.
           YOUR_APP_FUNCTION();
       ...
           }
   ```

2. Panggilan `WIFI_On()` untuk menginisialisasi dan menyalakan chip Wi-Fi Anda.

   > **Note**
   >
   > Beberapa papan mungkin memerlukan inisialisasi perangkat keras tambahan.

3. Lewati `WIFINetworkParams_t` struktur yang dikonfigurasi `WIFI_ConnectAP()` untuk menghubungkan papan Anda ke jaringan Wi-Fi yang tersedia. Untuk informasi lebih lanjut tentang `WIFINetworkParams_t` struktur, lihat Contoh penggunaan danReferensi API.

Referensi API

Untuk referensi API selengkapnya, lihat Referensi API Wi-Fi.

Contoh penggunaan

Menghubungkan ke AP yang dikenal

```
#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
}
```

Memindai terdekat APs

```
WIFINetworkParams_t xNetworkParams;
WIFIReturnCode_t xWifiStatus;

configPRINT( ("Turning on wifi...\n") );
xWifiStatus = WIFI_On();

configPRINT( ("Checking status...\n") );
if( xWifiStatus == eWiFiSuccess )
{
    configPRINT( ("WiFi module initialized.\n") );
}
else
{
    configPRINTF( ("WiFi module failed to initialize.\n" ) );
    // Handle module init failure
}

WIFI_SetMode(eWiFiModeStation);

/* Some boards might require additional initialization steps to use the Wi-Fi library.
 */

while (1)
{
    configPRINT( ("Starting scan\n") );
    const uint8_t ucNumNetworks = 12; //Get 12 scan results
    WIFIScanResult_t xScanResults[ ucNumNetworks ];
    xWifiStatus = WIFI_Scan( xScanResults, ucNumNetworks ); // Initiate scan

    configPRINT( ("Scan started\n") );

    // For each scan result, print out the SSID and RSSI
    if ( xWifiStatus == eWiFiSuccess )
    {
        configPRINT( ("Scan success\n") );
        for ( uint8_t i=0; i<ucNumNetworks; i++ )
        {
```

```
            configPRINTF( ("%s : %d \n", xScanResults[i].cSSID,
 xScanResults[i].cRSSI) );
        }
    } else {
        configPRINTF( ("Scan failed, status code: %d\n", (int)xWifiStatus) );
    }

    vTaskDelay(200);
}
```

Porting

`iot_wifi.c`Implementasi perlu mengimplementasikan fungsi yang didefinisikan dalam`iot_wifi.h`. Paling tidak, implementasi perlu kembali `eWiFiNotSupported` untuk fungsi yang tidak penting atau tidak didukung.

Untuk informasi selengkapnya tentang porting pustaka Wi-Fi, lihat Mem-porting Perpustakaan Wi-Fi di Panduan Porting FreerTOS.

Demo FreeRTOS

FreerTOS menyertakan beberapa aplikasi demo di folder, di `demos` bawah direktori FreeRTOS utama. Semua contoh yang dapat dieksekusi oleh FreeRTOS muncul di folder, di `common` bawah. `demos` Ada juga folder untuk setiap platform yang memenuhi syarat Freertos di bawah folder. `demos`

Sebelum Anda mencoba aplikasi demo, kami sarankan Anda menyelesaikan tutorial diMemulai dengan FreeRTOS. Ini menunjukkan cara mengatur dan menjalankan demo Agen CoreMQTT.

Menjalankan demo FreeRTOS

Topik berikut menunjukkan cara mengatur dan menjalankan demo FreeRTOS:

- Aplikasi demo Bluetooth Energi Rendah
- Bootloader demo untuk Microchip Curiosity MZEF PIC32
- AWS IoT Device Defender demo
- AWS IoT Greengrass Aplikasi demo penemuan V1
- AWS IoT Greengrass V2
- Demo CoreHTTP

- AWS IoT Demo perpustakaan pekerjaan
- Demo CoreMQTT
- Over-the-air memperbarui aplikasi demo
- Soket Aman menggemakan demo klien
- AWS IoT Aplikasi demo Device Shadow

`DEMO_RUNNER_RunDemos`Fungsi, yang terletak di *`freertos`*`/demos/demo_runner/iot_demo_runner.c` file, menginisialisasi utas terpisah tempat aplikasi demo tunggal berjalan. Secara default, `DEMO_RUNNER_RunDemos` hanya memanggil dan memulai demo Agen CoreMQTT. Bergantung pada konfigurasi yang Anda pilih saat mengunduh FreeRTOS, dan tempat Anda mengunduh FreeRTOS, fungsi pelari contoh lainnya mungkin dimulai secara default. Untuk mengaktifkan aplikasi demo, buka *`freertos`*`/vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/aws_demo_config.h` file, dan tentukan demo yang ingin Anda jalankan.

> **Note**
>
> Tidak semua kombinasi contoh bekerja sama. Bergantung pada kombinasinya, perangkat lunak mungkin tidak berjalan pada target yang dipilih karena kendala memori. Kami menyarankan Anda menjalankan satu demo pada satu waktu.

Mengkonfigurasi demo

Demo telah dikonfigurasi untuk membantu Anda memulai dengan cepat. Anda mungkin ingin mengubah beberapa konfigurasi untuk proyek Anda untuk membuat versi yang berjalan di platform Anda. Anda dapat menemukan file konfigurasi di`vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files`.

Aplikasi demo Bluetooth Energi Rendah

> **Important**
>
> Demo ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-Freertos yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Gambaran Umum

FreeRTOS Bluetooth Low Energy mencakup tiga aplikasi demo:

- MQTT melalui Bluetooth Energi Rendahdemo

  Aplikasi ini menunjukkan cara menggunakan MQTT melalui layanan Bluetooth Low Energy.
- Penyediaan Wi-Fidemo

  Aplikasi ini menunjukkan cara menggunakan layanan Penyediaan Wi-Fi Energi Rendah Bluetooth.
- Server Atribut Generikdemo

  Aplikasi ini menunjukkan cara menggunakan APIs middleware FreeRTOS Bluetooth Low Energy untuk membuat server GATT sederhana.

> **Note**
>
> Untuk mengatur dan menjalankan demo FreeRTOS, ikuti langkah-langkahnya. Memulai dengan FreeRTOS

Prasyarat

Untuk mengikuti demo ini, Anda memerlukan mikrokontroler dengan kemampuan Bluetooth Low Energy. Anda juga membutuhkan SDK iOS untuk perangkat Bluetooth FreeRTOS atauAndroid SDK untuk perangkat Bluetooth FreeRTOS.

Siapkan AWS IoT dan Amazon Cognito untuk FreerTOS Bluetooth Low Energy

Untuk menghubungkan perangkat Anda ke AWS IoT seluruh MQTT, Anda perlu mengatur dan Amazon AWS IoT Cognito.

Untuk mengatur AWS IoT

1. Siapkan AWS akun di https://aws.amazon.com/.
2. Buka AWS IoT konsol, dan dari panel navigasi, pilih Kelola, lalu pilih Things.
3. Pilih Buat, lalu pilih Buat satu hal.
4. Masukkan nama untuk perangkat Anda, lalu pilih Berikutnya.

5. Jika Anda menghubungkan mikrokontroler Anda ke cloud melalui perangkat seluler, pilih Buat sesuatu tanpa sertifikat. Karena Ponsel SDKs menggunakan Amazon Cognito untuk otentikasi perangkat, Anda tidak perlu membuat sertifikat perangkat untuk demo yang menggunakan Bluetooth Low Energy.

   Jika Anda menghubungkan mikrokontroler Anda ke cloud secara langsung melalui Wi-Fi, pilih Buat sertifikat, pilih Aktifkan, lalu unduh sertifikat, kunci publik, dan kunci pribadi benda tersebut.
6. Pilih hal yang baru saja Anda buat dari daftar barang terdaftar, lalu pilih Interact dari halaman benda Anda. Catat titik akhir AWS IoT REST API.

Untuk informasi selengkapnya tentang pengaturan, lihat Memulai dengan AWS IoT.

Untuk membuat kumpulan pengguna Amazon Cognito

1. Buka konsol Amazon Cognito, dan pilih Kelola Kumpulan Pengguna.
2. Pilih Buat kolam pengguna.
3. Beri nama kumpulan pengguna, lalu pilih Tinjau default.
4. Dari panel navigasi, pilih Klien aplikasi, lalu pilih Tambahkan klien aplikasi.
5. Masukkan nama untuk klien aplikasi, lalu pilih Buat klien aplikasi.
6. Dari panel navigasi, pilih Tinjau, lalu pilih Buat kumpulan.

   Catat ID kumpulan yang muncul di halaman Pengaturan Umum kumpulan pengguna Anda.
7. Dari panel navigasi, pilih Klien aplikasi, lalu pilih Tampilkan detail. Catat ID klien aplikasi dan rahasia klien aplikasi.

Untuk membuat kumpulan identitas Amazon Cognito

1. Buka konsol Amazon Cognito, dan pilih Kelola Kolam Identitas.
2. Masukkan nama untuk kolam identitas Anda.
3. Perluas penyedia Autentikasi, pilih tab Cognito, lalu masukkan ID kumpulan pengguna dan ID klien aplikasi Anda.
4. Pilih Buat kolam.
5. Perluas Detail Tampilan, dan catat dua nama peran IAM. Pilih Izinkan untuk membuat peran IAM untuk identitas yang diautentikasi dan tidak diautentikasi untuk mengakses Amazon Cognito.

6. Pilih Edit kolam identitas. Catat ID kumpulan identitas. Itu harus dari bentuk`us-west-2:12345678-1234-1234-1234-123456789012`.

Untuk informasi selengkapnya tentang pengaturan Amazon Cognito, lihat Memulai dengan Amazon Cognito.

Untuk membuat dan melampirkan kebijakan IAM ke identitas yang diautentikasi

1. Buka konsol IAM, dan dari panel navigasi, pilih Peran.
2. Temukan dan pilih peran identitas yang diautentikasi, pilih Lampirkan kebijakan, lalu pilih Tambahkan kebijakan sebaris.
3. Pilih tab JSON, dan tempel JSON berikut:

```
{
   "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. Pilih Kebijakan tinjauan, masukkan nama untuk kebijakan tersebut, lalu pilih Buat kebijakan.

Simpan informasi Anda AWS IoT dan Amazon Cognito di tangan. Anda memerlukan titik akhir dan IDs untuk mengautentikasi aplikasi seluler Anda dengan Cloud. AWS

Siapkan lingkungan FreeRTOS Anda untuk Bluetooth Low Energy

Untuk mengatur lingkungan Anda, Anda perlu mengunduh FreeRTOS dengan Perpustakaan Bluetooth Rendah Energi mikrokontroler di Anda, dan mengunduh serta mengonfigurasi SDK Seluler untuk Perangkat Bluetooth FreeRTOS di perangkat seluler Anda.

Untuk mengatur lingkungan mikrokontroler Anda dengan FreeRTOS Bluetooth Low Energy

1. Unduh atau kloning FreeRTOS dari. GitHub Lihat file README.md untuk instruksi.
2. Siapkan FreeRTOS di mikrokontroler Anda.

   Untuk informasi tentang memulai dengan FreeRTOS pada mikrokontroler berkualifikasi FreeRTOS, lihat panduan untuk papan Anda di Memulai dengan FreeRTOS.

   > **Note**
   >
   > Anda dapat menjalankan demo pada mikrokontroler Bluetooth Low Energy dengan FreeRTOS dan porting FreeRTOS Bluetooth Low Energy library. Saat ini, proyek demo MQTT melalui Bluetooth Energi Rendah FreeRTOS sepenuhnya di-porting ke perangkat berkemampuan Bluetooth Low Energy berikut:
   >
   > - Espressif ESP32 - DevKit C dan ESP-WROVER-KIT
   > - Nordik n 0-DK RF5284

Komponen umum

Aplikasi demo FreeRTOS memiliki dua komponen umum:

- Network Manager
- Aplikasi demo SDK Seluler Energi Rendah Bluetooth

Network Manager

Network Manager mengelola koneksi jaringan mikrokontroler Anda. Itu terletak di direktori FreeRTOS Anda di. `demos/network_manager/aws_iot_network_manager.c` Jika Network Manager diaktifkan untuk Wi-Fi dan Bluetooth Low Energy, demo dimulai dengan Bluetooth Low Energy secara default. Jika koneksi Bluetooth Low Energy terganggu, dan papan Anda terganggu Wi-Fi-enabled,

Manajer Jaringan beralih ke koneksi Wi-Fi yang tersedia untuk mencegah Anda memutuskan sambungan dari jaringan.

Untuk mengaktifkan jenis koneksi jaringan dengan Network Manager, tambahkan jenis koneksi jaringan ke `configENABLED_NETWORKS` parameter di `vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/aws_iot_network_config.h` (di mana *`vendor`* adalah nama vendor dan *`board`* adalah nama papan yang Anda gunakan untuk menjalankan demo).

Misalnya, jika Anda mengaktifkan Bluetooth Low Energy dan Wi-Fi, baris yang dimulai dengan `#define configENABLED_NETWORKS` in `aws_iot_network_config.h` berbunyi sebagai berikut:

```
#define  configENABLED_NETWORKS  ( AWSIOT_NETWORK_TYPE_BLE | AWSIOT_NETWORK_TYPE_WIFI )
```

Untuk mendapatkan daftar jenis koneksi jaringan yang saat ini didukung, lihat baris yang dimulai dengan `#define AWSIOT_NETWORK_TYPE` in`aws_iot_network.h`.

Aplikasi demo SDK Seluler Energi Rendah FreeRTOS Bluetooth

Aplikasi demo FreeRTOS Bluetooth Low Energy Mobile SDK terletak GitHub di Android SDK untuk Perangkat Bluetooth FreeRTOS di bawah dan `amazon-freertos-ble-android-sdk/app` SDK iOS untuk Perangkat Bluetooth FreeRTOS di bawah. `amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo` Dalam contoh ini, kami menggunakan tangkapan layar dari versi iOS dari aplikasi seluler demo.

> **Note**
>
> Jika Anda menggunakan perangkat iOS, Anda memerlukan Xcode untuk membangun aplikasi seluler demo. Jika Anda menggunakan perangkat Android, Anda dapat menggunakan Android Studio untuk membuat aplikasi seluler demo.

Untuk mengonfigurasi aplikasi demo SDK iOS

Saat Anda menentukan variabel konfigurasi, gunakan format nilai placeholder yang disediakan dalam file konfigurasi.

1. Konfirmasikan bahwa SDK iOS untuk perangkat Bluetooth FreeRTOS sudah diinstal.
2. Keluarkan perintah berikut dari`amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo/`:

```
$ pod install
```

3. Buka `amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo/AmazonFreeRTOSDemo.xcworkspace` proyek dengan Xcode, dan ubah akun pengembang penandatanganan ke akun Anda.
4. Buat AWS IoT kebijakan di wilayah Anda (jika Anda belum melakukannya).

> **Note**
>
> Kebijakan ini berbeda dengan kebijakan IAM yang dibuat untuk identitas yang diautentikasi Amazon Cognito.

   a. Buka konsol AWS IoT.

   b. Di panel navigasi, pilih Aman, pilih Kebijakan, lalu pilih Buat. Masukkan nama untuk mengidentifikasi kebijakan Anda. Di bagian Tambahkan pernyataan, pilih Mode lanjutan. Salin dan tempel JSON berikut ke jendela editor kebijakan. Ganti *aws-region* dan *aws-account* dengan AWS Wilayah dan ID akun Anda.

```
{
    "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. Pilih Buat.

5. Bukaamazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo/AmazonFreeRTOSDemo/Amazon/AmazonConstants.swift, dan definisikan ulang variabel berikut:

   - region: AWS Wilayah Anda.
   - iotPolicyName: Nama AWS IoT kebijakan Anda.
   - mqttCustomTopic: Topik MQTT yang ingin Anda publikasikan.

6. Buka amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo/AmazonFreeRTOSDemo/Support/awsconfiguration.json.

   Di bawahCognitoIdentity, mendefinisikan ulang variabel berikut:

   - PoolId: ID kumpulan identitas Amazon Cognito Anda.
   - Region: AWS Wilayah Anda.

   Di bawahCognitoUserPool, mendefinisikan ulang variabel berikut:

   - PoolId: ID kumpulan pengguna Amazon Cognito Anda.
   - AppClientId: ID klien aplikasi Anda.
   - AppClientSecret: Rahasia klien aplikasi Anda.
   - Region: AWS Wilayah Anda.

Untuk mengonfigurasi aplikasi demo Android SDK

Saat Anda menentukan variabel konfigurasi, gunakan format nilai placeholder yang disediakan dalam file konfigurasi.

1. Konfirmasikan bahwa Android SDK untuk perangkat Bluetooth FreeRTOS sudah diinstal.
2. Buat AWS IoT kebijakan di wilayah Anda (jika Anda belum melakukannya).

> **Note**
>
> Kebijakan ini berbeda dengan kebijakan IAM yang dibuat untuk identitas yang diautentikasi Amazon Cognito.

a. Buka konsol AWS IoT.

b. Di panel navigasi, pilih Aman, pilih Kebijakan, lalu pilih Buat. Masukkan nama untuk mengidentifikasi kebijakan Anda. Di bagian Tambahkan pernyataan, pilih Mode lanjutan. Salin dan tempel JSON berikut ke jendela editor kebijakan. Ganti *aws-region* dan *aws-account* dengan AWS Wilayah dan ID akun Anda.

```
{
    "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. Pilih Buat.

3. Buka https://github.com/aws/amazon-freertos-ble-android- sdk/blob/master/app/src/main/java/software/amazon/freertos/demo/DemoConstants .java dan mendefinisikan ulang variabel berikut:

   - `AWS_IOT_POLICY_NAME`: Nama AWS IoT kebijakan Anda.
   - `AWS_IOT_REGION`: AWS Wilayah Anda.

4. Buka https://github.com/aws/amazon-freertos-ble-android- sdk/blob/master/app/src/main/res/raw/awsconfiguration .json.

   Di bawah`CognitoIdentity`, mendefinisikan ulang variabel berikut:

   - `PoolId`: ID kumpulan identitas Amazon Cognito Anda.
   - `Region`: AWS Wilayah Anda.

   Di bawah`CognitoUserPool`, mendefinisikan ulang variabel berikut:

   - `PoolId`: ID kumpulan pengguna Amazon Cognito Anda.
   - `AppClientId`: ID klien aplikasi Anda.
   - `AppClientSecret`: Rahasia klien aplikasi Anda.
   - `Region`: AWS Wilayah Anda.

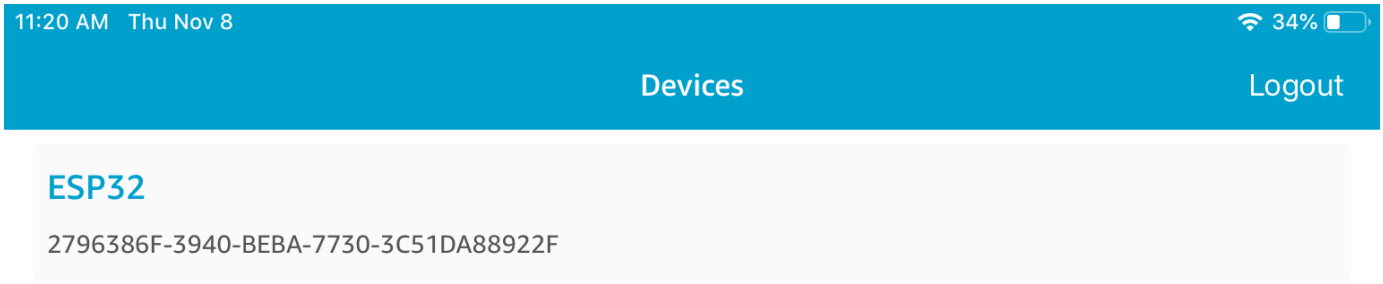
Untuk menemukan dan membangun koneksi aman dengan mikrokontroler Anda melalui Bluetooth Low Energy

1. Untuk memasangkan mikrokontroler dan perangkat seluler Anda dengan aman (langkah 6), Anda memerlukan emulator terminal serial dengan kemampuan input dan output (seperti TeraTerm). Konfigurasikan terminal untuk terhubung ke papan Anda dengan koneksi serial seperti yang diinstruksikan. Menginstal emulator terminal
2. Jalankan proyek demo Bluetooth Low Energy di mikrokontroler Anda.
3. Jalankan aplikasi demo Bluetooth Low Energy Mobile SDK di perangkat seluler Anda.

   Untuk memulai aplikasi demo di Android SDK dari baris perintah, jalankan perintah berikut:

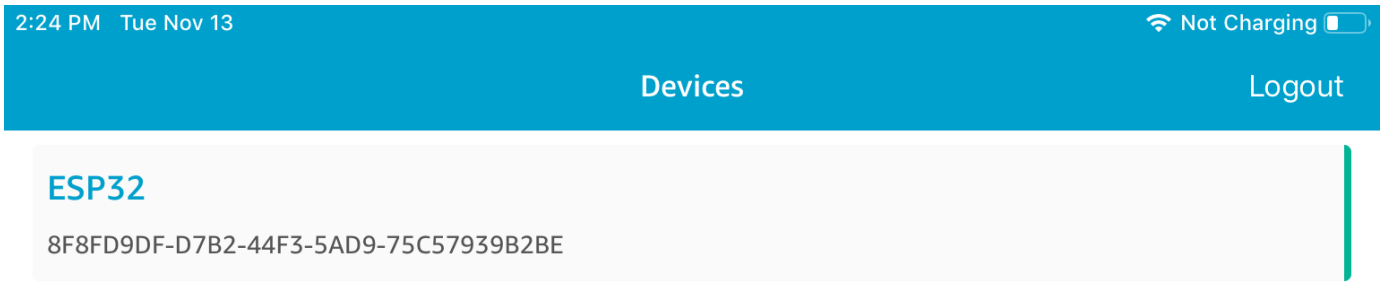
   ```
   $ ./gradlew installDebug
   ```

4. Konfirmasikan bahwa mikrokontroler Anda muncul di bawah Perangkat di aplikasi demo Bluetooth Low Energy Mobile SDK.

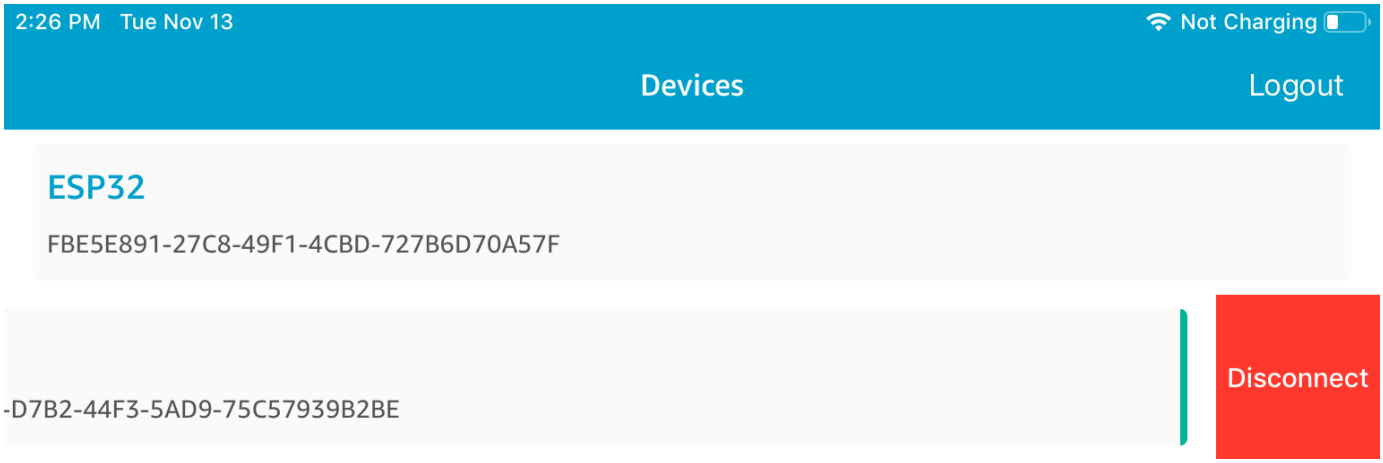


> **Note**
>
> Semua perangkat dengan FreeRTOS dan layanan informasi perangkat *freertos*`/.../device_information` () yang berada dalam jangkauan muncul dalam daftar.

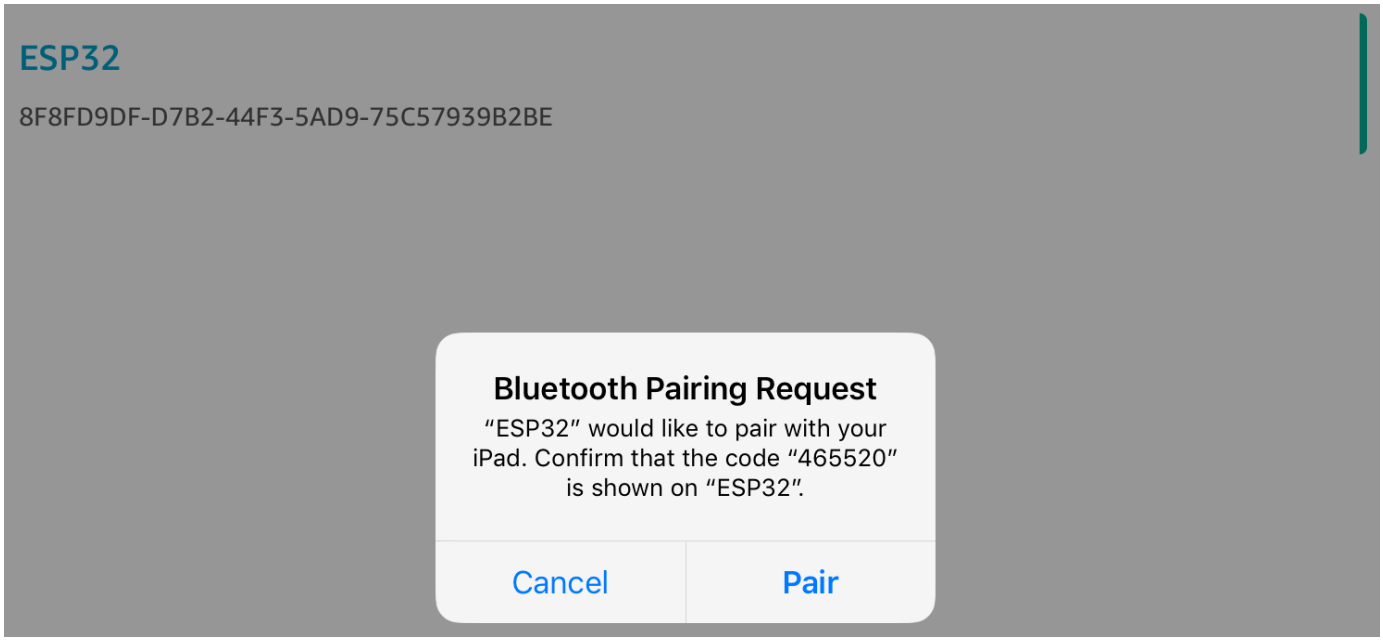
5. Pilih mikrokontroler Anda dari daftar perangkat. Aplikasi membuat koneksi dengan papan, dan garis hijau muncul di sebelah perangkat yang terhubung.



Anda dapat memutuskan sambungan dari mikrokontroler Anda dengan menyeret garis ke kiri.



6. Jika diminta, pasangkan mikrokontroler dan perangkat seluler Anda.



Jika kode untuk perbandingan numerik sama pada kedua perangkat, pasangkan perangkat.

> **Note**
>
> Aplikasi demo Bluetooth Low Energy Mobile SDK menggunakan Amazon Cognito untuk otentikasi pengguna. Pastikan Anda telah menyiapkan pengguna Amazon Cognito dan kumpulan identitas, dan Anda telah melampirkan kebijakan IAM ke identitas yang diautentikasi.

MQTT melalui Bluetooth Energi Rendah

Dalam demo MQTT melalui Bluetooth Low Energy, mikrokontroler Anda menerbitkan pesan ke AWS Cloud melalui proxy MQTT.

Untuk berlangganan topik demo MQTT

1. Masuk ke AWS IoT konsol.
2. Di panel navigasi, pilih Uji, lalu pilih klien pengujian MQTT untuk membuka klien MQTT.
3. Dalam Subscription topic, masukkan***thing-name*/example/topic1**, lalu pilih Subscribe to topic.

Jika Anda menggunakan Bluetooth Low Energy untuk memasangkan mikrokontroler dengan perangkat seluler Anda, pesan MQTT dirutekan melalui aplikasi demo Bluetooth Low Energy Mobile SDK di perangkat seluler Anda.

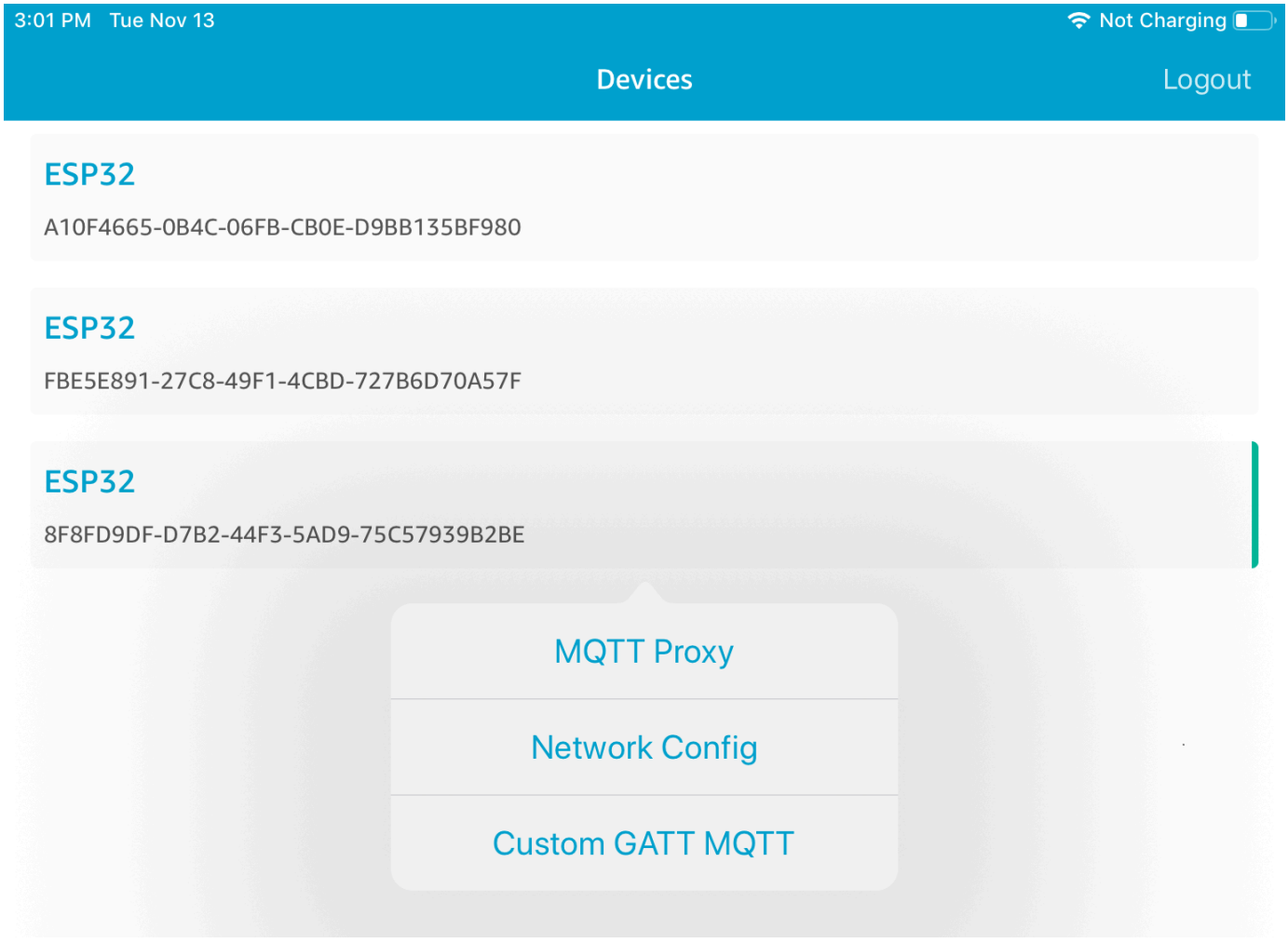
Untuk mengaktifkan demo melalui Bluetooth Low Energy

1. Bukavendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h, dan tentukanCONFIG_MQTT_BLE_TRANSPORT_DEMO_ENABLED.
2. Bukademos/include/aws_clientcredential.h, dan konfigurasikan clientcredentialMQTT_BROKER_ENDPOINT dengan titik akhir AWS IoT broker. Konfigurasikan clientcredentialIOT_THING_NAME dengan nama benda untuk perangkat pengontrol mikro BLE. Titik akhir AWS IoT broker dapat diperoleh dari AWS IoT konsol dengan memilih Pengaturan di panel navigasi kiri, atau melalui CLI dengan menjalankan perintah:. aws iot describe-endpoint --endpoint-type=iot:Data-ATS

> **Note**
>
> Titik akhir AWS IoT broker dan nama benda keduanya harus berada di wilayah yang sama di mana identitas cognito dan kumpulan pengguna dikonfigurasi.

Untuk menjalankan demo

1. Bangun dan jalankan proyek demo pada mikrokontroler Anda.
2. Pastikan Anda telah memasangkan papan dan perangkat seluler Anda menggunakan. Aplikasi demo SDK Seluler Energi Rendah FreeRTOS Bluetooth
3. Dari daftar Perangkat di aplikasi seluler demo, pilih mikrokontroler Anda, lalu pilih MQTT Proxy untuk membuka pengaturan proxy MQTT.



4. Setelah Anda mengaktifkan proxy MQTT, pesan MQTT muncul pada *thing-name*`/example/topic1` topik, dan data dicetak ke terminal UART.

Penyediaan Wi-Fi

Penyediaan Wi-Fi adalah layanan FreeRTOS Bluetooth Low Energy yang memungkinkan Anda mengirim kredenal jaringan Wi-Fi dengan aman dari perangkat seluler ke mikrokontroler melalui Bluetooth Low Energy. Kode sumber untuk layanan Penyediaan Wi-Fi dapat ditemukan di. *freertos*`/.../wifi_provisioning`

> **Note**
>
> Demo Penyediaan Wi-Fi saat ini didukung di ESP32 Espressif - C. DevKit

Untuk mengaktifkan demo

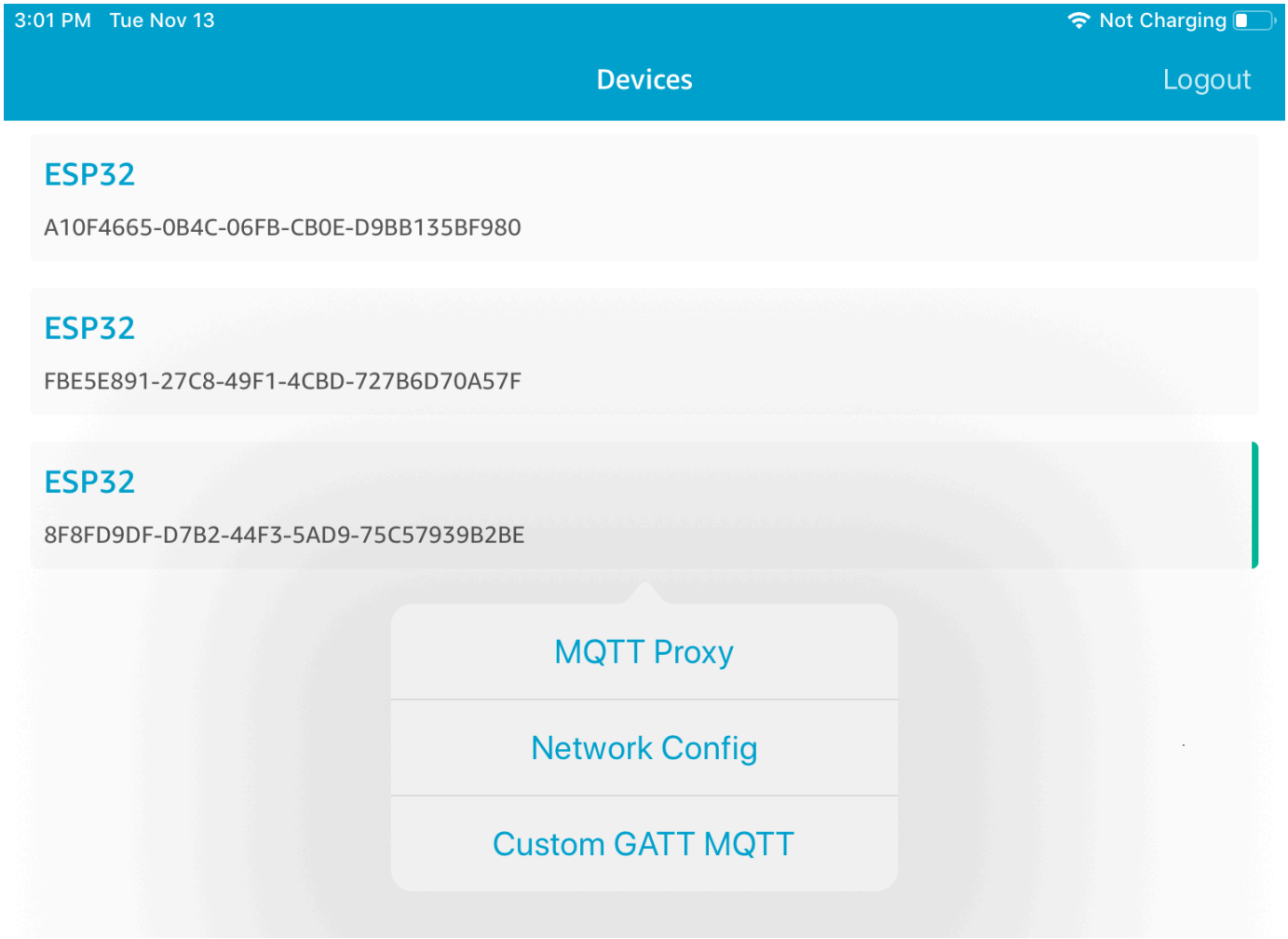
1. Aktifkan layanan Penyediaan Wi-Fi. Buka`vendors/`*vendor*`/boards/`*board*`/aws_demos/config_files/iot_ble_config.h`, dan atur `#define IOT_BLE_ENABLE_WIFI_PROVISIONING` ke `1` (di mana *vendor* adalah nama vendor dan *board* adalah nama papan yang Anda gunakan untuk menjalankan demo).

   > **Note**
   >
   > Layanan Penyediaan Wi-Fi dinonaktifkan secara default.

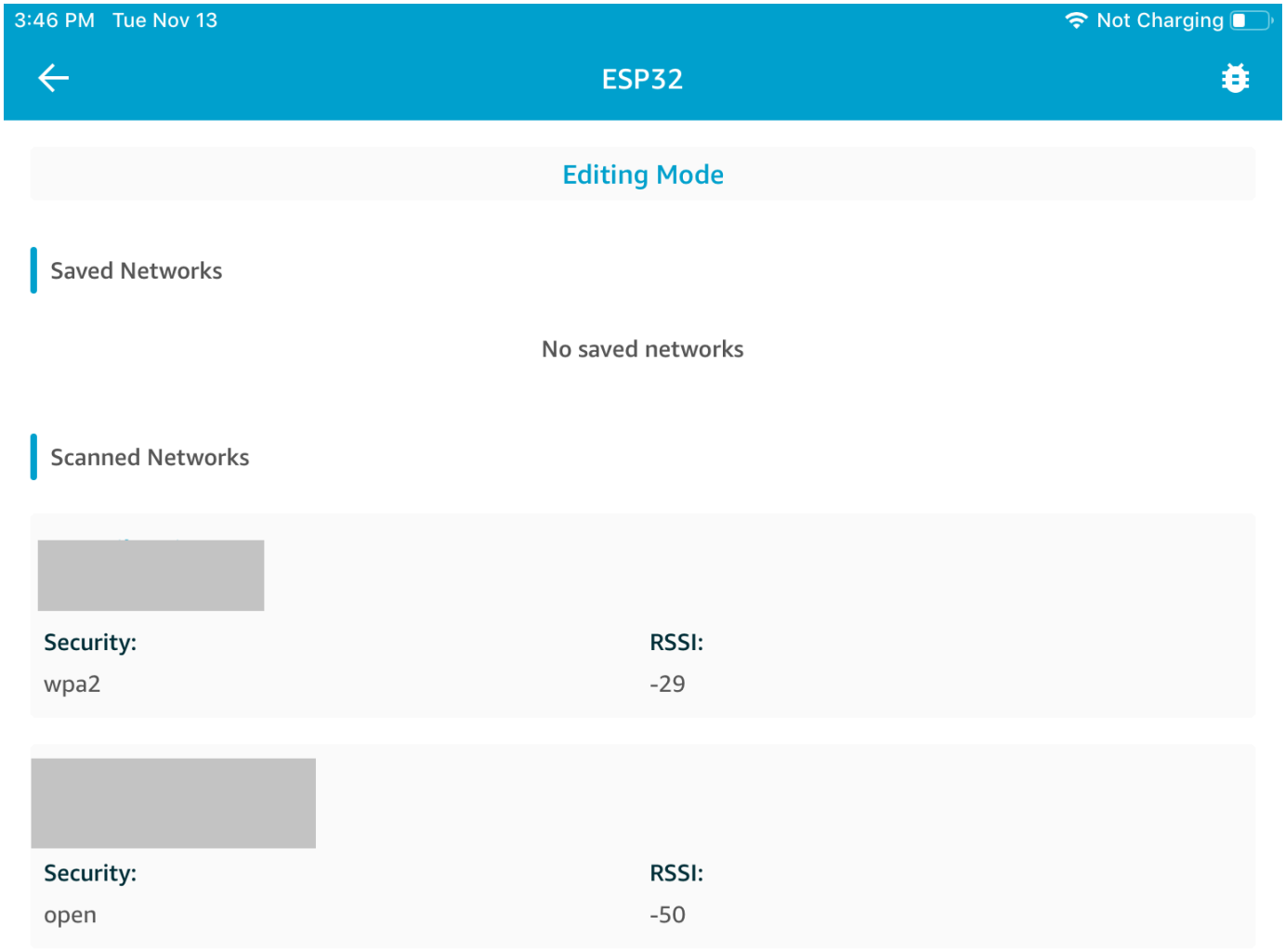
2. Konfigurasikan Network Manager untuk mengaktifkan Bluetooth Low Energy dan Wi-Fi.

Untuk menjalankan demo

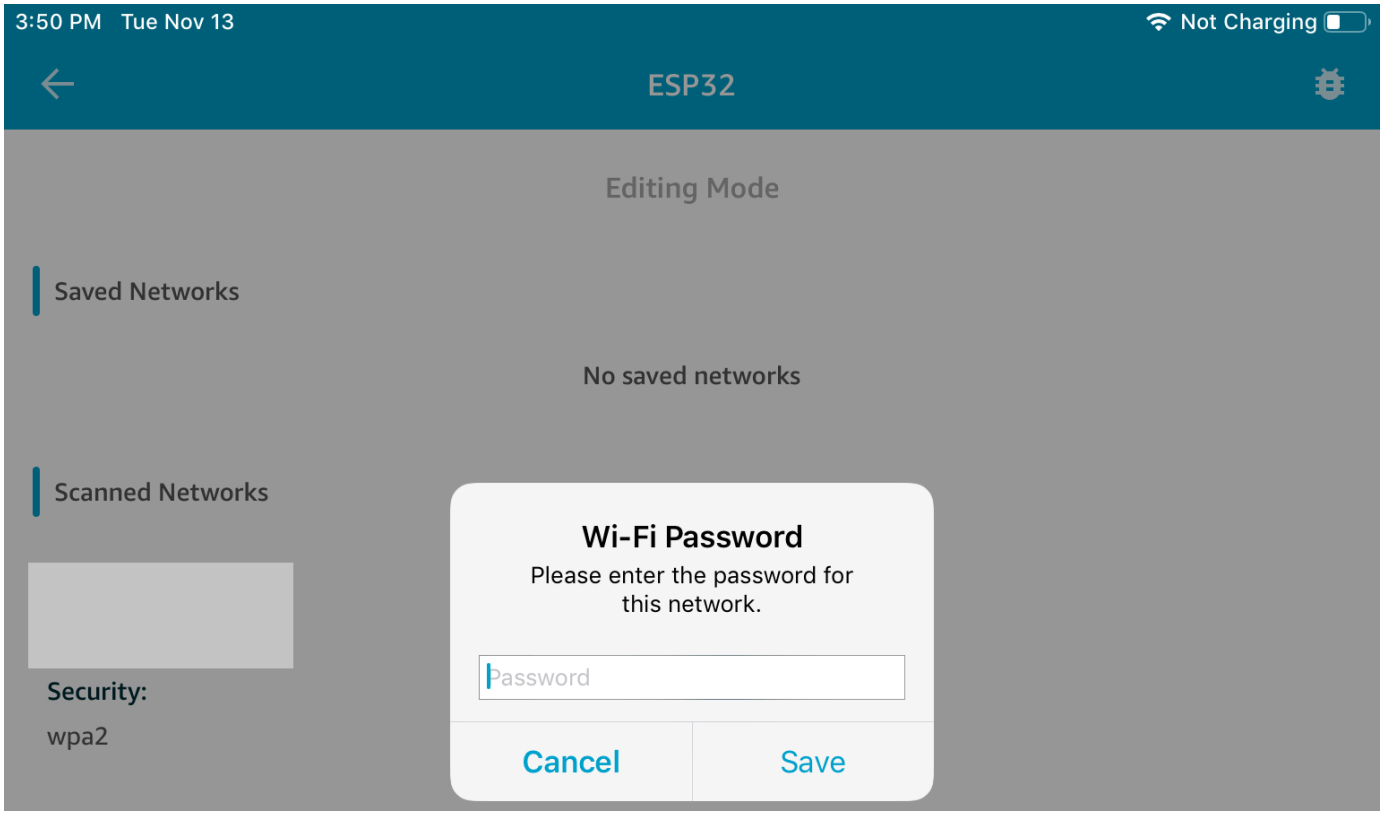
1. Bangun dan jalankan proyek demo pada mikrokontroler Anda.
2. Pastikan Anda telah memasangkan mikrokontroler dan perangkat seluler Anda menggunakan. Aplikasi demo SDK Seluler Energi Rendah FreeRTOS Bluetooth
3. Dari daftar Perangkat di aplikasi seluler demo, pilih mikrokontroler Anda, lalu pilih Network Config untuk membuka pengaturan konfigurasi jaringan.



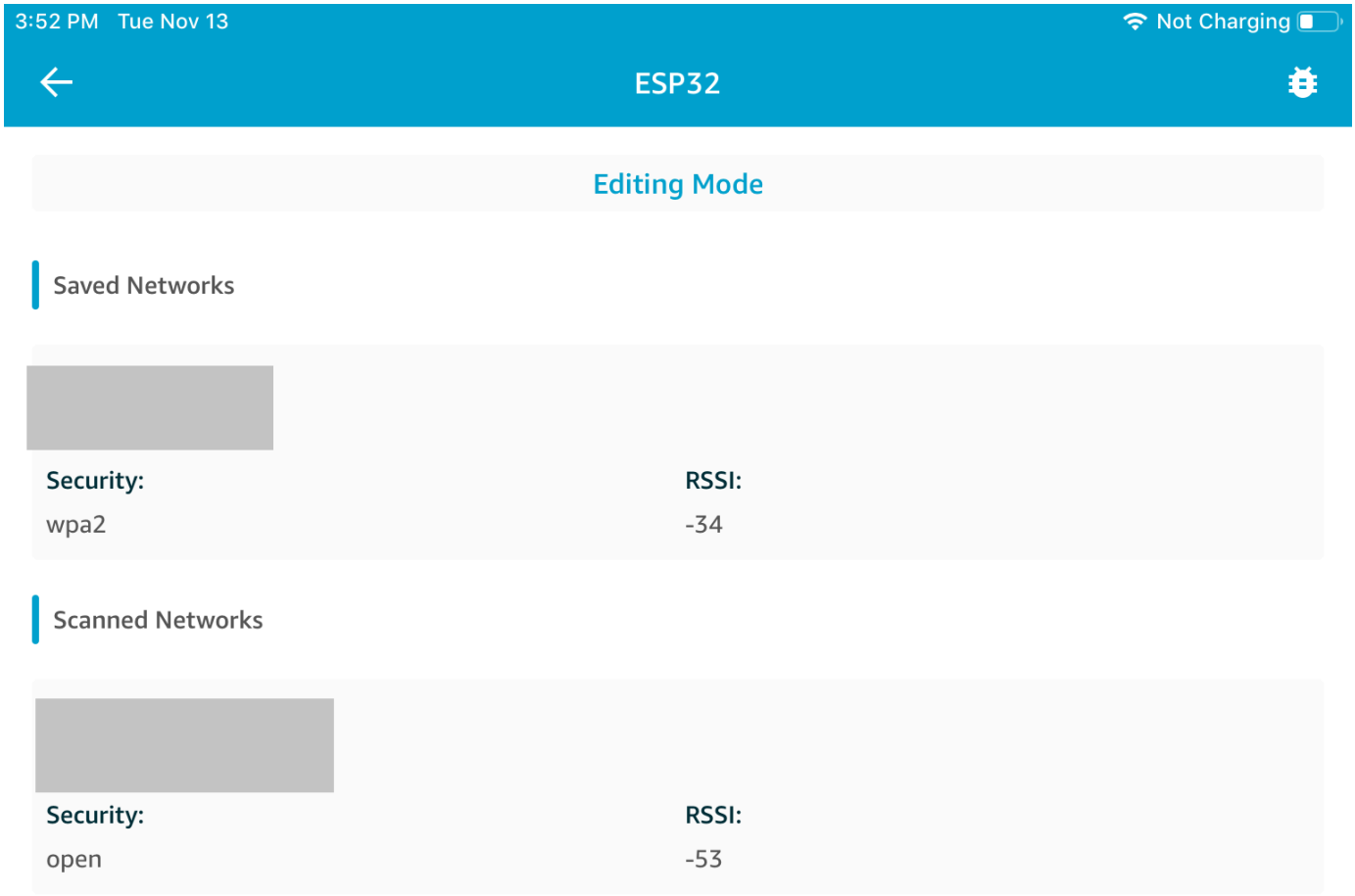
4. Setelah Anda memilih Network Config untuk papan Anda, mikrokontroler mengirimkan daftar jaringan di sekitarnya ke perangkat seluler. Jaringan Wi-Fi yang tersedia muncul dalam daftar di bawah Jaringan yang Dipindai.



Dari daftar Jaringan yang Dipindai, pilih jaringan Anda, lalu masukkan SSID dan kata sandi, jika diperlukan.

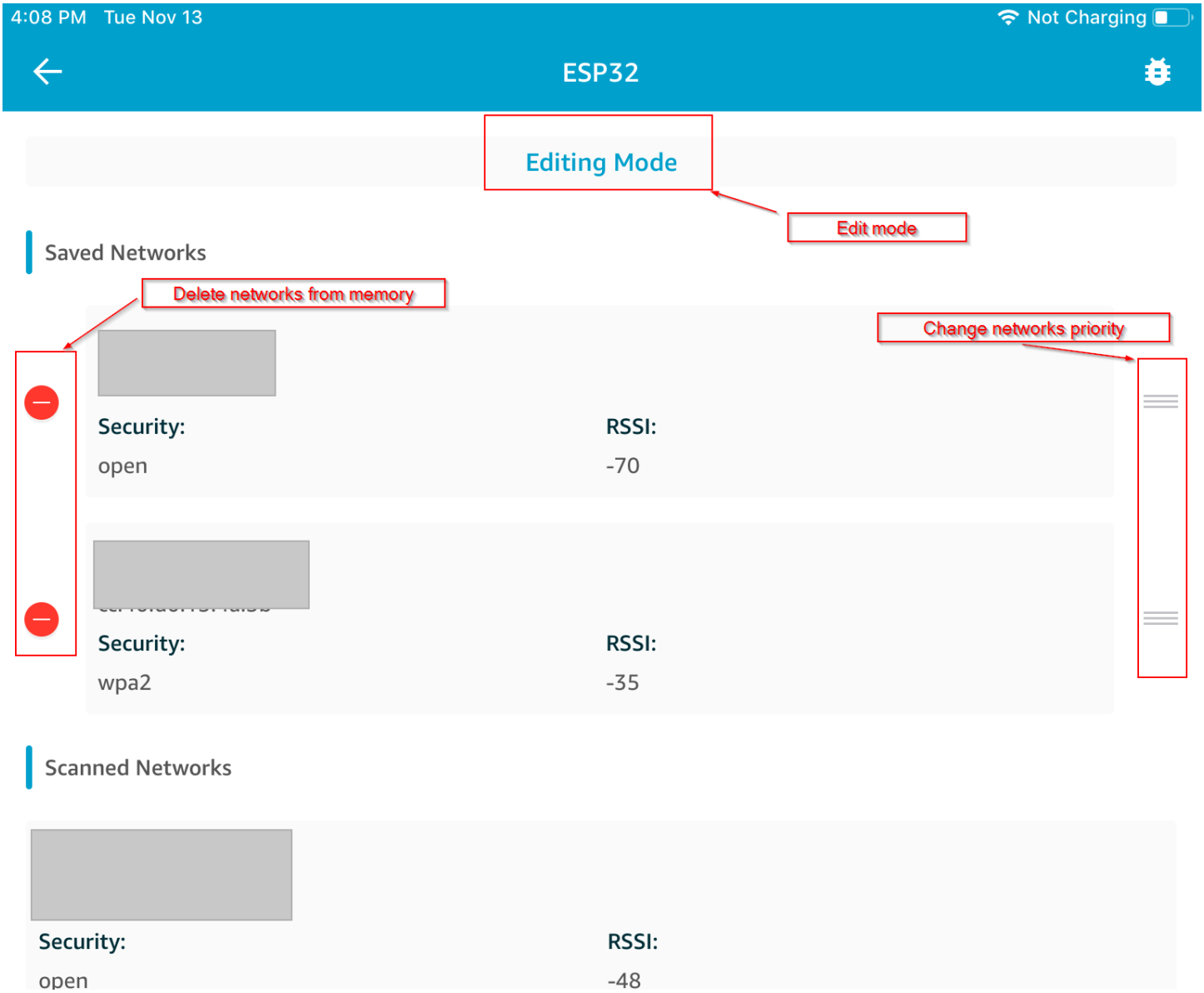


Mikrokontroler terhubung ke dan menyimpan jaringan. Jaringan muncul di bawah Jaringan Tersimpan.



Anda dapat menyimpan beberapa jaringan di aplikasi seluler demo. Saat Anda me-restart aplikasi dan demo, mikrokontroler terhubung ke jaringan tersimpan pertama yang tersedia, mulai dari bagian atas daftar Jaringan Tersimpan.

Untuk mengubah urutan prioritas jaringan atau menghapus jaringan, pada halaman Konfigurasi Jaringan, pilih Mode Pengeditan. Untuk mengubah urutan prioritas jaringan, pilih sisi kanan jaringan yang ingin Anda prioritaskan ulang, dan seret jaringan ke atas atau ke bawah. Untuk menghapus jaringan, pilih tombol merah di sisi kiri jaringan yang ingin Anda hapus.



Server Atribut Generik

Dalam contoh ini, aplikasi Server Atribut Generik (GATT) demo pada mikrokontroler Anda mengirimkan nilai penghitung sederhana ke file. Aplikasi demo SDK Seluler Energi Rendah FreeRTOS Bluetooth

Menggunakan Bluetooth Low Energy Mobile SDKs, Anda dapat membuat klien GATT Anda sendiri untuk perangkat seluler yang terhubung ke server GATT pada mikrokontroler Anda dan berjalan secara paralel dengan aplikasi seluler demo.

Untuk mengaktifkan demo

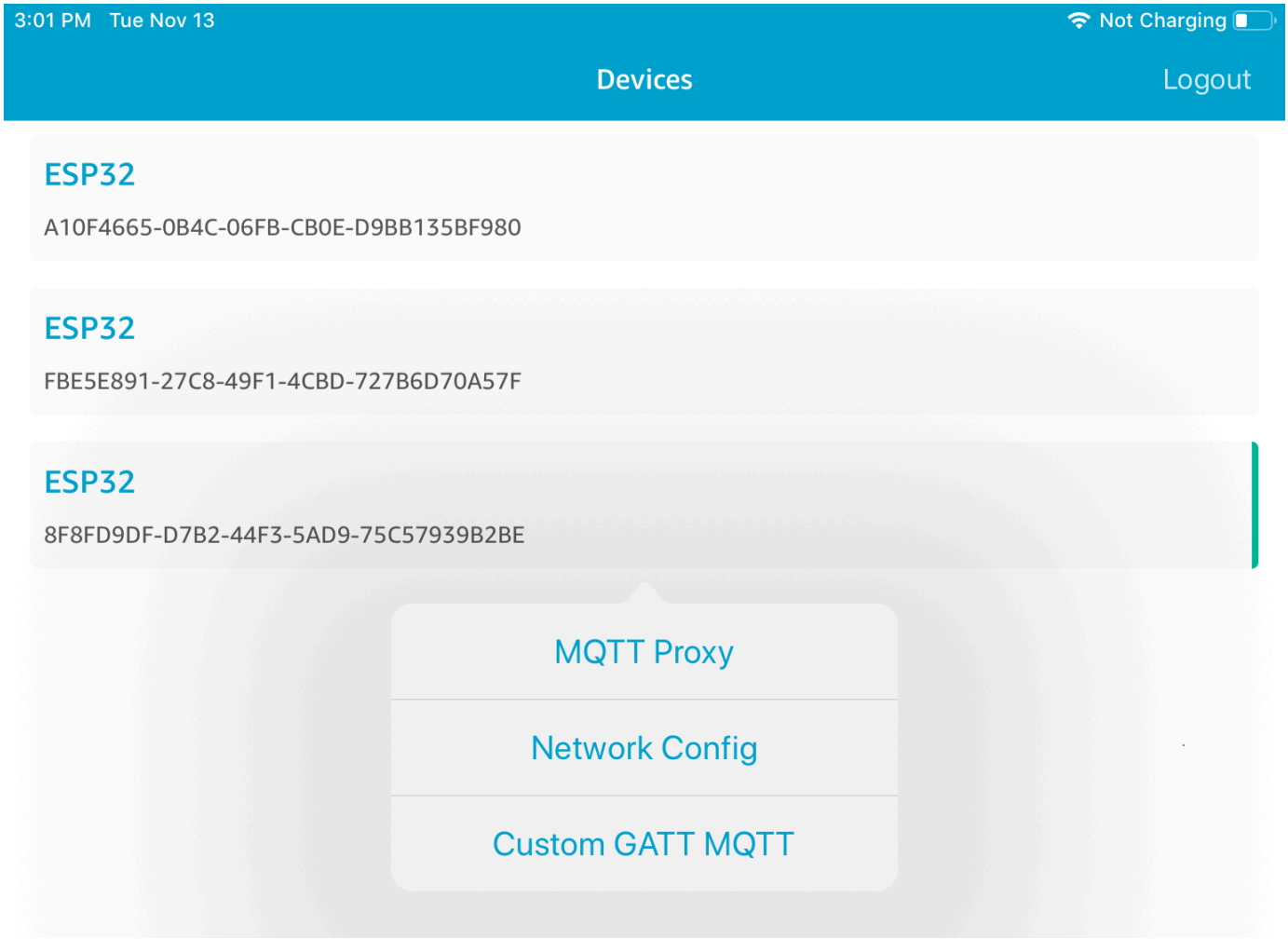
1. Aktifkan demo Bluetooth Low Energy GATT. Di `vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/iot_ble_config.h` (di mana *`vendor`* adalah nama vendor dan *`board`* adalah nama papan yang Anda gunakan untuk menjalankan demo), tambahkan `#define IOT_BLE_ADD_CUSTOM_SERVICES ( 1 )` ke daftar pernyataan define.

   > **Note**
   >
   > Demo Bluetooth Low Energy GATT dinonaktifkan secara default.

2. Buka*`freertos`*`/vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/aws_demo_config.h`, beri komentar`#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED`, dan tentukan`CONFIG_BLE_GATT_SERVER_DEMO_ENABLED`.

Untuk menjalankan demo

1. Bangun dan jalankan proyek demo pada mikrokontroler Anda.
2. Pastikan Anda telah memasangkan papan dan perangkat seluler Anda menggunakan. Aplikasi demo SDK Seluler Energi Rendah FreeRTOS Bluetooth
3. Dari daftar Perangkat di aplikasi, pilih papan Anda, lalu pilih MQTT Proxy untuk membuka opsi proxy MQTT.



4. Kembali ke daftar Perangkat, pilih papan Anda, lalu pilih Custom GATT MQTT untuk membuka opsi layanan GATT kustom.
5. Pilih Mulai Penghitung untuk mulai menerbitkan data ke topik ***your-thing-name*/example/topic** MQTT.

   Setelah Anda mengaktifkan proxy MQTT, Hello World dan pesan penghitung tambahan muncul pada topik. *your-thing-name*/example/topic

Bootloader demo untuk Microchip Curiosity MZEF PIC32

> **Important**
>
> Demo ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek

FreeRTOS yang sudah ada berdasarkan repositori Amazon-FreerTOS yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Note

Sesuai dengan Microchip, kami menghapus Curiosity PIC32 MZEF (DM320104) dari cabang utama repositori Freertos Reference Integration dan tidak akan lagi membawanya dalam rilis baru. Microchip telah mengeluarkan pemberitahuan resmi bahwa PIC32 MZEF (DM320104) tidak lagi direkomendasikan untuk desain baru. Proyek PIC32 MZEF dan kode sumber masih dapat diakses melalui tag rilis sebelumnya. Microchip merekomendasikan agar pelanggan menggunakan papan Pengembangan Curiosity PIC32 MZ-EF-2.0 (DM320209) untuk desain baru. PIC32MZv1 Platform ini masih dapat ditemukan di v202012.00 dari repositori Integrasi Referensi FreeRTOS. Namun, platform ini tidak lagi didukung oleh Referensi FreeRTOS v202107.00.

Bootloader demo ini mengimplementasikan pemeriksaan versi firmware, verifikasi tanda tangan kriptografi, dan pengujian mandiri aplikasi. Kemampuan ini mendukung pembaruan firmware over-the-air (OTA) untuk FreeRTOS.

Verifikasi firmware termasuk memverifikasi keaslian dan integritas firmware baru yang diterima melalui udara. Bootloader memverifikasi tanda tangan kriptografi aplikasi sebelum boot. Demo ini menggunakan algoritma tanda tangan digital kurva elips (ECDSA) di atas SHA-256. Utilitas yang disediakan dapat digunakan untuk menghasilkan aplikasi yang ditandatangani yang dapat di-flash pada perangkat.

Bootloader mendukung fitur-fitur berikut yang diperlukan untuk OTA:

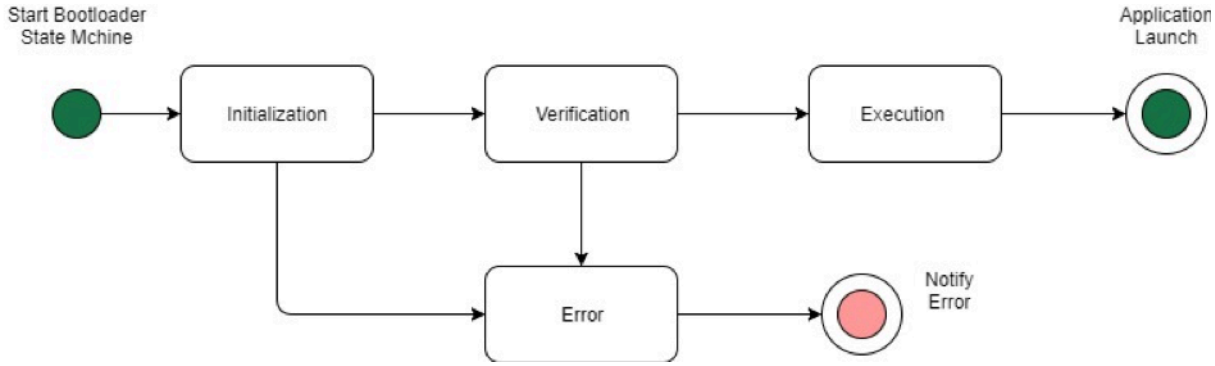
- Mempertahankan gambar aplikasi pada perangkat dan beralih di antara mereka.
- Memungkinkan eksekusi self-test dari gambar OTA yang diterima dan rollback pada kegagalan.
- Memeriksa tanda tangan dan versi gambar pembaruan OTA.

Note

Untuk mengatur dan menjalankan demo FreeRTOS, ikuti langkah-langkahnya. Memulai dengan FreeRTOS

Status bootloader

Proses bootloader ditampilkan di mesin status berikut.



Tabel berikut menjelaskan status bootloader.

Status Bootloader	Deskripsi
Inisialisasi	Bootloader dalam keadaan inisialisasi.
Verifikasi	Bootloader memverifikasi gambar yang ada di perangkat.
Jalankan Gambar	Bootloader meluncurkan gambar yang dipilih.
Jalankan Default	Bootloader meluncurkan gambar default.
Kesalahan	Bootloader dalam keadaan kesalahan.

Dalam diagram sebelumnya, keduanya `Execute Image` dan `Execute Default` ditampilkan sebagai negara. `Execution`

Status Eksekusi Bootloader

Bootloader dalam `Execution` keadaan dan siap meluncurkan gambar terverifikasi yang dipilih. Jika gambar yang akan diluncurkan ada di bank atas, bank ditukar sebelum mengeksekusi gambar, karena aplikasi selalu dibangun untuk bank bawah.

Status Eksekusi Default Bootloader

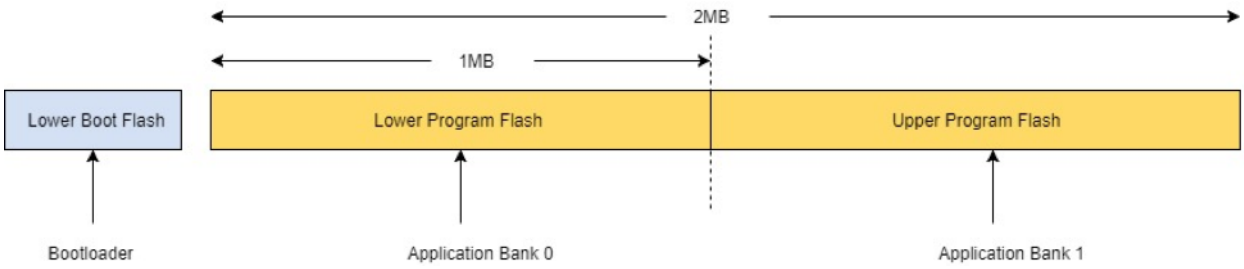
Jika opsi konfigurasi untuk meluncurkan gambar default diaktifkan, bootloader meluncurkan aplikasi dari alamat eksekusi default. Opsi ini harus dinonaktifkan kecuali saat debugging.

Status Kesalahan Bootloader

Bootloader dalam keadaan kesalahan dan tidak ada gambar yang valid di perangkat. Bootloader harus memberi tahu pengguna. Implementasi default mengirimkan pesan log ke konsol dan mengedipkan LED dengan cepat di papan tanpa batas waktu.

Perangkat flash

Platform Microchip Curiosity PIC32 MZEF berisi flash program internal dua megabyte (MB) yang dibagi menjadi dua bank. Ini mendukung pertukaran peta memori antara kedua bank ini dan pembaruan langsung. Bootloader demo diprogram di wilayah flash boot bawah yang terpisah.



Struktur gambar aplikasi

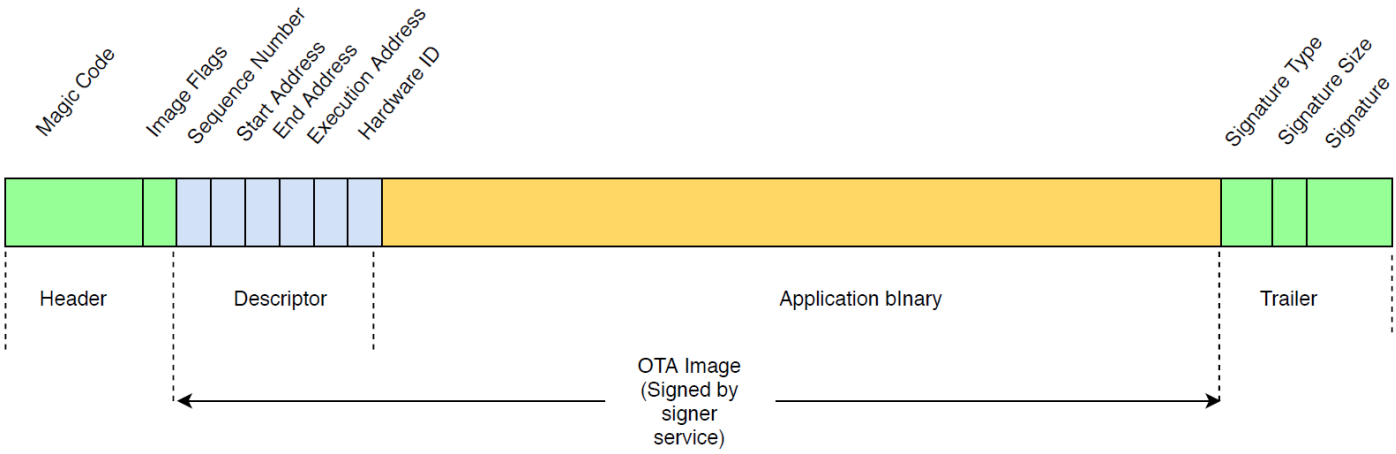


Diagram menunjukkan komponen utama dari gambar aplikasi yang disimpan di setiap bank perangkat.

Komponen	Ukuran (dalam byte)
Header gambar	8 byte

Komponen	Ukuran (dalam byte)
Deskriptor gambar	24 byte
Aplikasi biner	< 1 MB - (324)
Trailer	292 byte

Header gambar

Gambar aplikasi pada perangkat harus dimulai dengan header yang terdiri dari kode ajaib dan bendera gambar.

Bidang Header	Ukuran (dalam byte)
Kode ajaib	7 byte
Bendera gambar	1 byte

Kode ajaib

Gambar pada perangkat flash harus dimulai dengan kode ajaib. Kode ajaib default adalah`@AFRTOS`. Bootloader memeriksa apakah ada kode ajaib yang valid sebelum mem-boot gambar. Ini adalah langkah pertama verifikasi.

Bendera gambar

Bendera gambar digunakan untuk menyimpan status gambar aplikasi. Bendera digunakan dalam proses OTA. Bendera gambar kedua bank menentukan keadaan perangkat. Jika gambar yang dijalankan ditandai sebagai komit tertunda, itu berarti perangkat berada dalam fase uji mandiri OTA. Bahkan jika gambar pada perangkat ditandai valid, mereka melalui langkah verifikasi yang sama pada setiap boot. Jika gambar ditandai sebagai baru, bootloader menandainya sebagai komit tertunda dan meluncurkannya untuk self-test setelah verifikasi. Bootloader juga menginisialisasi dan memulai pengatur waktu pengawas sehingga jika gambar OTA baru gagal menguji sendiri, perangkat reboot dan bootloader menolak gambar dengan menghapusnya dan mengeksekusi gambar yang valid sebelumnya.

Perangkat hanya dapat memiliki satu gambar yang valid. Gambar lainnya dapat berupa gambar OTA baru atau komit yang tertunda (self-test). Setelah pembaruan OTA berhasil, gambar lama dihapus dari perangkat.

Status	Nilai	Deskripsi
Gambar baru	0xFF	Gambar aplikasi baru dan tidak pernah dieksekusi.
Komit tertunda	0xFE	Gambar aplikasi ditandai untuk eksekusi uji.
Valid	0xFC	Gambar aplikasi ditandai valid dan berkomitmen.
Tidak valid	0xF8	Gambar aplikasi ditandai tidak valid.

Deskriptor gambar

Gambar aplikasi pada perangkat flash harus berisi deskriptor gambar mengikuti header gambar. Deskriptor gambar dihasilkan oleh utilitas pasca-build yang menggunakan file konfigurasi (`ota-descriptor.config`) untuk menghasilkan deskriptor yang sesuai dan menambahnya ke biner aplikasi. Output dari langkah pasca-build ini adalah gambar biner yang dapat digunakan untuk OTA.

Bidang Deskriptor	Ukuran (dalam byte)
Nomor Urutan	4 byte
Alamat Mulai	4 byte
Alamat Akhir	4 byte
Alamat Eksekusi	4 byte
ID Perangkat Keras	4 byte
Dilindungi	4 byte

Nomor Urutan

Nomor urut harus ditambah sebelum membuat gambar OTA baru. Lihat `ota-descriptor.config` filenya. Bootloader menggunakan nomor ini untuk menentukan gambar yang akan di-boot. Nilai yang valid adalah dari 1 hingga 4294967295.

Alamat Mulai

Alamat awal gambar aplikasi pada perangkat. Karena deskriptor gambar ditambahkan ke biner aplikasi, alamat ini adalah awal dari deskriptor gambar.

Alamat Akhir

Alamat akhir gambar aplikasi pada perangkat, tidak termasuk trailer gambar.

Alamat Eksekusi

Alamat eksekusi gambar.

ID Perangkat Keras

ID perangkat keras unik yang digunakan oleh bootloader untuk memastikan gambar OTA dibuat untuk platform yang benar.

Dilindungi

Ini dicadangkan untuk penggunaan masa depan.

Cuplikan gambar

Cuplikan gambar ditambahkan ke biner aplikasi. Ini berisi string tipe tanda tangan, ukuran tanda tangan, dan tanda tangan gambar.

Bidang Trailer	Ukuran (dalam byte)
Jenis Tanda Tangan	32 byte
Ukuran Tanda Tangan	4 byte
Tanda tangan	256 byte

Jenis Tanda Tangan

Tipe tanda tangan adalah string yang mewakili algoritma kriptografi yang digunakan dan berfungsi sebagai penanda untuk trailer. Bootloader mendukung algoritme tanda tangan digital kurva elips (ECDSA). Defaultnya adalah sig-sha256-ecdsa.

Ukuran Tanda Tangan

Ukuran tanda tangan kriptografi, dalam byte.

Tanda tangan

Tanda tangan kriptografi dari biner aplikasi ditambahkan dengan deskriptor gambar.

Konfigurasi bootloader

Opsi konfigurasi bootloader dasar disediakan di*freertos*`/vendors/microchip/boards/curiosity_pic32mzef/bootloader/config_files/aws_boot_config.h`. Beberapa opsi disediakan hanya untuk tujuan debugging.

Aktifkan Mulai Default

Mengaktifkan eksekusi aplikasi dari alamat default dan harus diaktifkan untuk debugging saja. Gambar dieksekusi dari alamat default tanpa verifikasi apa pun.

Aktifkan Verifikasi Tanda Tangan Crypto

Mengaktifkan verifikasi tanda tangan kriptografi saat boot. Gambar yang gagal dihapus dari perangkat. Opsi ini disediakan hanya untuk tujuan debugging dan harus tetap diaktifkan dalam produksi.

Hapus Gambar Tidak Valid

Mengaktifkan penghapusan bank penuh jika verifikasi gambar di bank itu gagal. Opsi ini disediakan untuk debugging dan harus tetap diaktifkan dalam produksi.

Aktifkan Verifikasi ID Perangkat Keras

Mengaktifkan verifikasi ID perangkat keras di deskriptor gambar OTA dan ID perangkat keras yang diprogram di bootloader. Ini opsional dan dapat dinonaktifkan jika verifikasi ID perangkat keras tidak diperlukan.

Aktifkan Verifikasi Alamat

Memungkinkan verifikasi alamat awal, akhir, dan eksekusi di deskriptor gambar OTA. Kami menyarankan agar Anda tetap mengaktifkan opsi ini.

Membangun bootloader

Bootloader demo disertakan sebagai proyek yang dapat dimuat dalam proyek yang terletak *freertos*/vendors/microchip/boards/curiosity_pic32mzef/aws_demos/mplab/ di aws_demos repositori kode sumber FreeRTOS. Ketika aws_demos proyek dibangun, ia membangun bootloader terlebih dahulu, diikuti oleh aplikasi. Output akhir adalah gambar hex terpadu termasuk bootloader dan aplikasi. factory_image_generator.pyUtilitas disediakan untuk menghasilkan gambar hex terpadu dengan tanda tangan kriptografi. Skrip utilitas bootloader terletak di. *freertos*/demos/ota/bootloader/utility/

Langkah pra-pembuatan bootloader

Langkah pra-build ini mengeksekusi skrip utilitas yang disebut codesigner_cert_utility.py yang mengekstrak kunci publik dari sertifikat penandatanganan kode dan menghasilkan file header C yang berisi kunci publik dalam format yang disandikan Abstract Syntax Notation One (ASN.1). Header ini dikompilasi ke dalam proyek bootloader. Header yang dihasilkan berisi dua konstanta: array kunci publik dan panjang kunci. Proyek bootloader juga dapat dibangun tanpa aws_demos dan dapat di-debug sebagai aplikasi normal.

AWS IoT Device Defender demo

Important

Demo ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-Freertos yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Pengantar

Demo ini menunjukkan kepada Anda cara menggunakan pustaka AWS IoT Device Defender untuk AWS IoT Device Defenderterhubung. Demo menggunakan pustaka CoreMQTT untuk membuat

koneksi MQTT melalui TLS (otentikasi timbal balik) ke Broker AWS IoT MQTT dan pustaka CoreJson untuk memvalidasi dan mengurai tanggapan yang diterima dari layanan. AWS IoT Device Defender Demo menunjukkan cara membuat laporan berformat JSON menggunakan metrik yang dikumpulkan dari perangkat, dan cara mengirimkan laporan yang dibuat ke layanan. AWS IoT Device Defender Demo juga menunjukkan cara mendaftarkan fungsi callback dengan pustaka CoreMQTT untuk menangani respons dari AWS IoT Device Defender layanan guna mengonfirmasi apakah laporan terkirim diterima atau ditolak.

> **Note**
>
> Untuk mengatur dan menjalankan demo FreeRTOS, ikuti langkah-langkahnya. Memulai dengan FreeRTOS

Fungsionalitas

Demo ini membuat tugas aplikasi tunggal yang menunjukkan cara mengumpulkan metrik, membuat laporan pembela perangkat dalam format JSON, dan mengirimkannya ke AWS IoT Device Defender layanan melalui koneksi MQTT yang aman ke Broker MQTT. AWS IoT Demo ini mencakup metrik jaringan standar serta metrik khusus. Untuk metrik kustom, demo meliputi:

- Sebuah metrik bernama "`task_numbers`" yang merupakan daftar tugas FreeRTOS. IDs Jenis metrik ini adalah “daftar angka”.
- Metrik bernama "`stack_high_water_mark`" yang merupakan tanda air tumpukan tinggi untuk tugas aplikasi demo. Jenis metrik ini adalah “angka”.

Cara kami mengumpulkan metrik jaringan tergantung pada tumpukan TCP/IP yang digunakan. Untuk FreerTos+TCP dan konfigurasi LWiP yang didukung, kami menyediakan implementasi pengumpulan metrik yang mengumpulkan metrik nyata dari perangkat dan mengirimkannya dalam laporan. AWS IoT Device Defender Anda dapat menemukan implementasi untuk Freertos+TCP dan LWiP aktif. GitHub

Untuk papan yang menggunakan tumpukan TCP/IP lainnya, kami menyediakan definisi rintisan dari fungsi pengumpulan metrik yang mengembalikan nol untuk semua metrik jaringan. Menerapkan fungsi *`freertos`*`/demos/device_defender_for_aws/metrics_collector/stub/metrics_collector.c` untuk tumpukan jaringan Anda untuk mengirim metrik nyata. File ini juga tersedia di situs GitHubweb.

Untuk ESP32, konfigurasi LWiP default tidak menggunakan penguncian inti dan oleh karena itu demo akan menggunakan metrik stubbed. Jika Anda ingin menggunakan implementasi pengumpulan metrik LWiP referensi, tentukan makro berikut di: `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
```

Berikut ini adalah contoh output ketika Anda menjalankan 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-------
```

Jika papan Anda tidak menggunakan FreerTos+TCP atau konfigurasi LWiP yang didukung, output akan terlihat seperti berikut.

```
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-------
```

Kode sumber demo ada di unduhan Anda di *freertos*`/demos/device_defender_for_aws/` direktori atau di GitHubsitus web.

Berlangganan topik AWS IoT Device Defender

Fungsi subscribeToDefenderTopik berlangganan topik MQTT di mana tanggapan terhadap laporan Device Defender yang diterbitkan akan diterima. Ini menggunakan makro `DEFENDER_API_JSON_ACCEPTED` untuk membangun string topik di mana tanggapan untuk laporan pembela perangkat yang diterima diterima. Ini menggunakan makro `DEFENDER_API_JSON_REJECTED` untuk membangun string topik di mana tanggapan untuk laporan pembela perangkat yang ditolak akan diterima.

Mengumpulkan metrik perangkat

collectDeviceMetricsFungsi ini mengumpulkan metrik jaringan menggunakan fungsi yang didefinisikan dalam. `metrics_collector.h` Metrik yang dikumpulkan adalah jumlah byte dan paket yang dikirim dan diterima, port TCP terbuka, port UDP terbuka, dan koneksi TCP yang ditetapkan.

Menghasilkan AWS IoT Device Defender laporan

Fungsi generateDeviceMetricsLaporan menghasilkan laporan pembela perangkat menggunakan fungsi yang ditentukan dalam`report_builder.h`. Fungsi itu mengambil metrik jaringan dan buffer, membuat dokumen JSON dalam format seperti yang diharapkan AWS IoT Device Defender dan menulisnya ke buffer yang disediakan. Format dokumen JSON yang diharapkan oleh AWS IoT Device Defender ditentukan dalam metrik sisi perangkat di Panduan Pengembang.AWS IoT

Menerbitkan AWS IoT Device Defender laporan

AWS IoT Device Defender Laporan ini diterbitkan pada topik MQTT untuk menerbitkan laporan JSON. AWS IoT Device Defender Laporan dibuat menggunakan makro`DEFENDER_API_JSON_PUBLISH`, seperti yang ditunjukkan dalam cuplikan kode ini di situs web. GitHub

Callback untuk menangani tanggapan

Fungsi PublishCallback menangani pesan publikasi MQTT yang masuk. Ini menggunakan `Defender_MatchTopic` API dari AWS IoT Device Defender perpustakaan untuk memeriksa apakah pesan MQTT yang masuk berasal dari layanan. AWS IoT Device Defender Jika pesan berasal dari AWS IoT Device Defender layanan, itu mem-parsing respons JSON yang diterima dan mengekstrak ID laporan dalam respons. ID laporan kemudian diverifikasi sama dengan yang dikirim dalam laporan.

AWS IoT Greengrass Aplikasi demo penemuan V1

> **Important**
>
> Demo ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-Freertos yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Sebelum Anda menjalankan demo AWS IoT Greengrass Discovery untuk FreeRTOS, Anda perlu AWS mengatur AWS IoT Greengrass,, dan. AWS IoT Untuk mengatur AWS, ikuti instruksi diMenyiapkan AWS akun dan izin. Untuk mengatur AWS IoT Greengrass, Anda perlu membuat grup Greengrass dan kemudian menambahkan inti Greengrass. Untuk informasi selengkapnya tentang pengaturan AWS IoT Greengrass, lihat Memulai dengan AWS IoT Greengrass.

Setelah Anda mengatur AWS dan AWS IoT Greengrass, Anda perlu mengkonfigurasi beberapa izin tambahan untuk AWS IoT Greengrass.

Untuk mengatur AWS IoT Greengrass izin

1. Jelajahi konsol IAM.
2. Dari panel navigasi, pilih Peran, lalu temukan dan pilih ServiceRoleGreengrass_.
3. Pilih Lampirkan kebijakan, pilih AmazonS3 FullAccess dan AWSIoTFullAccess, lalu pilih Lampirkan kebijakan.
4. Jelajahi ke AWS IoT konsol.
5. Di panel navigasi, pilih Greengrass, pilih Grup, lalu pilih grup Greengrass yang sebelumnya Anda buat.
6. Pilih Pengaturan, lalu pilih Tambahkan peran.
7. Pilih Greengrass_ServiceRole, lalu pilih Simpan.

Hubungkan papan Anda ke AWS IoT dan konfigurasikan demo FreerTOS Anda.

1. Mendaftarkan papan MCU Anda dengan AWS IoT

   Setelah mendaftarkan papan, Anda perlu membuat dan melampirkan kebijakan Greengrass baru ke sertifikat perangkat.

Untuk membuat AWS IoT Greengrass kebijakan baru

1. Jelajahi ke AWS IoT konsol.
2. Di panel navigasi, pilih Aman, pilih Kebijakan, lalu pilih Buat.
3. Masukkan nama untuk mengidentifikasi kebijakan Anda.
4. Di bagian Tambahkan pernyataan, pilih Mode lanjutan. Salin dan tempel JSON berikut ke jendela editor kebijakan:

```
{
      "Effect": "Allow",
      "Action": [
        "greengrass:*"
      ],
      "Resource": "*"
}
```

   Kebijakan ini memberikan AWS IoT Greengrass izin ke semua sumber daya.
5. Pilih Buat.

Untuk melampirkan AWS IoT Greengrass kebijakan ke sertifikat perangkat

1. Jelajahi ke AWS IoT konsol.
2. Di panel navigasi, pilih Kelola, pilih Things, lalu pilih hal yang sebelumnya Anda buat.
3. Pilih Keamanan, lalu pilih sertifikat yang dilampirkan ke perangkat Anda.
4. Pilih Kebijakan, pilih Tindakan, lalu pilih Lampirkan Kebijakan.
5. Temukan dan pilih kebijakan Greengrass yang Anda buat sebelumnya, lalu pilih Lampirkan.

2. Mengunduh FreeRTOS

> **Note**
>
> Jika Anda mengunduh FreeRTOS dari konsol FreerTOS, pilih Connect to - bukan Connect to AWS IoT Greengrass -. ***`Platform`*** AWS IoT***`Platform`***

3. Mengkonfigurasi demo FreeRTOS.

Buka*freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h, beri komentar#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED, dan tentukanCONFIG_GREENGRASS_DISCOVERY_DEMO_ENABLED.

Setelah Anda mengatur AWS IoT dan AWS IoT Greengrass, dan setelah Anda mengunduh dan mengkonfigurasi FreeRTOS, Anda dapat membangun, mem-flash, dan menjalankan demo Greengrass di perangkat Anda. Untuk mengatur lingkungan pengembangan perangkat keras dan perangkat lunak papan Anda, ikuti instruksi diPanduan memulai khusus dewan.

Demo Greengrass menerbitkan serangkaian pesan ke inti Greengrass, dan ke klien MQTT. AWS IoT Untuk melihat pesan di klien AWS IoT MQTT, buka AWS IoT konsol, pilih Uji, pilih klien pengujian MQTT dan kemudian tambahkan langganan ke. `freertos/demos/ggd`

Di klien MQTT, Anda akan melihat string berikut:

```
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
```

Menggunakan EC2 instans Amazon

Jika Anda bekerja dengan EC2 instans Amazon

1. Temukan DNS Publik (IPv4) yang terkait dengan EC2 instans Amazon Anda— buka EC2 konsol Amazon, dan di panel navigasi kiri, pilih Instans. Pilih EC2 instans Amazon Anda, lalu pilih panel Deskripsi. Cari entri untuk DNS Publik (IPv4) dan catat itu.
2. Temukan entri untuk grup Keamanan dan pilih grup keamanan yang dilampirkan ke EC2 instans Amazon Anda.
3. Pilih tab Aturan masuk lalu pilih Edit aturan masuk dan tambahkan aturan berikut.

   Aturan-aturan ke dalam

Tipe	Protokol	Rentang port	Sumber	Deskripsi - opsional
HTTP	TCP	80	0.0.0.0/0	-

Tipe	Protokol	Rentang port	Sumber	Deskripsi - opsional
HTTP	TCP	80	::/0	-
SSH	TCP	22	0.0.0.0/0	-
TCP Kustom	TCP	8883	0.0.0.0/0	Komunikasi MQTT
TCP Kustom	TCP	8883	::/0	Komunikasi MQTT
HTTPS	TCP	443	0.0.0.0/0	-
HTTPS	TCP	443	::0/0	-
Semua ICMP - IPv4	ICMP	Semua	0.0.0.0/0	-
Semua ICMP - IPv4	ICMP	Semua	::0/0	-

4. Di AWS IoT konsol pilih Greengrass, lalu Grup, dan pilih grup Greengrass yang sebelumnya Anda buat. Pilih Pengaturan. Ubah Deteksi koneksi lokal menjadi Mengelola informasi koneksi secara manual.
5. Di panel navigasi, pilih Cores lalu pilih inti grup Anda.
6. Pilih Konektivitas dan pastikan Anda hanya memiliki satu titik akhir inti (hapus semua sisanya) dan itu bukan alamat IP (karena dapat berubah). Pilihan terbaik adalah menggunakan DNS Publik (IPv4) yang Anda catat di langkah pertama.
7. Tambahkan FreeRTOS IoT yang Anda buat ke grup GG.
    a. Pilih panah belakang untuk kembali ke halaman AWS IoT Greengrass grup. Di panel navigasi, pilih Perangkat lalu pilih Tambah Perangkat.
    b. Pilih Pilih IoT Thing. Pilih perangkat Anda lalu pilih Selesai.
8. Tambahkan langganan yang diperlukan— di halaman Grup Greengrass, pilih Langganan lalu pilih Tambahkan Langganan dan masukkan informasi seperti yang ditunjukkan di sini.

Langganan

Sumber	Target	Topik
TIGG1	IoT Cloud	freertos/demos/ggd

Di mana “Sumber” adalah nama yang diberikan untuk AWS IoT benda yang dibuat di AWS IoT konsol saat Anda mendaftarkan papan Anda - "TIGG1" dalam contoh yang diberikan di sini.

9. Mulai penyebaran AWS IoT Greengrass grup Anda dan pastikan bahwa penerapan berhasil. Anda sekarang harus berhasil menjalankan demo AWS IoT Greengrass penemuan.

AWS IoT Greengrass V2

Kompatibilitas dengan AWS IoT Greengrass V2 perangkat

AWS IoT Greengrass V2 dukungan untuk perangkat klien kompatibel dengan versi sebelumnya. AWS IoT Greengrass V1 Anda dapat menghubungkan perangkat klien FreeRTOS ke perangkat inti V2 tanpa mengubah kode aplikasi. Untuk mengaktifkan perangkat klien terhubung ke perangkat inti V2, lakukan hal berikut.

- Terapkan perangkat lunak Greengrass ke perangkat inti Greengrass. Lihat Connect perangkat klien ke perangkat inti untuk menghubungkan perangkat AWS IoT Greengrass V2.
- Untuk menyampaikan pesan (termasuk fungsi Lambda) antara perangkat klien, layanan AWS IoT Core cloud, dan komponen Greengrass, gunakan dan konfigurasikan komponen jembatan MQTT.
- Menyebarkan komponen detektor IP untuk mendeteksi informasi konektivitas secara otomatis, atau mengelola titik akhir secara manual.
- Lihat Berinteraksi dengan AWS IoT perangkat lokal untuk informasi selengkapnya.

Untuk detail selengkapnya, lihat AWS dokumentasi tentang menjalankan AWS IoT Greengrass V1 aplikasi di AWS IoT Greengrass V2.

Demo CoreHTTP

> **Important**
>
> Demo ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-Freertos yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Demo ini dapat membantu Anda mempelajari cara menggunakan pustaka CoreHTTP.

Topik

- Demo otentikasi timbal balik CoreHTTP
- Demo unggahan dasar CoreHTTP Amazon S3
- Demo unduhan dasar CoreHTTP S3
- CoreHTTP demo multithreaded dasar

Demo otentikasi timbal balik CoreHTTP

> **Important**
>
> Demo ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-Freertos yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Pengantar

Proyek demo CoreHTTP (Mutual Authentication) menunjukkan kepada Anda cara membuat koneksi ke server HTTP menggunakan TLS dengan otentikasi timbal balik antara klien dan server. Demo ini menggunakan implementasi antarmuka transport berbasis MBEDTLS untuk membuat koneksi TLS yang diautentikasi server dan klien, dan menunjukkan alur kerja respons permintaan di HTTP.

> **Note**
>
> Untuk mengatur dan menjalankan demo FreeRTOS, ikuti langkah-langkahnya. Memulai dengan FreeRTOS

Fungsionalitas

Demo ini membuat tugas aplikasi tunggal dengan contoh yang menunjukkan cara menyelesaikan yang berikut:

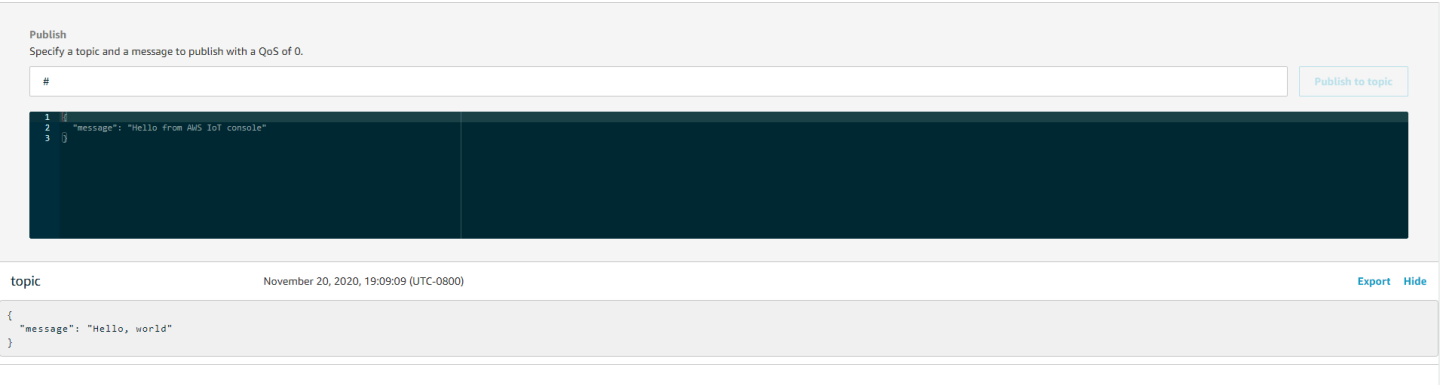
- Connect ke server HTTP pada AWS IoT endpoint.
- Kirim permintaan POST.
- Menerima tanggapannya.
- Putuskan sambungan dari server.

Setelah Anda menyelesaikan langkah-langkah ini, demo menghasilkan output yang mirip dengan tangkapan layar berikut.

```
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-------
```

AWS IoT Konsol menghasilkan output yang mirip dengan tangkapan layar berikut.



Organisasi kode sumber

File sumber demo diberi nama `http_demo_mutual_auth.c` dan dapat ditemukan di *`freertos`*`/demos/coreHTTP/` direktori dan di GitHubsitus web.

Menghubungkan ke server AWS IoT HTTP

connectToServerWithBackoffRetriesFungsi ini mencoba untuk membuat koneksi TLS yang saling diautentikasi ke server HTTP. AWS IoT Jika koneksi gagal, ia mencoba lagi setelah batas waktu. Nilai batas waktu meningkat secara eksponensial hingga jumlah upaya maksimum tercapai atau nilai batas waktu maksimum tercapai. `RetryUtils_BackoffAndSleep`Fungsi ini memberikan nilai batas waktu yang meningkat secara eksponensial dan pengembalian `RetryUtilsRetriesExhausted` ketika jumlah maksimum upaya telah tercapai. `connectToServerWithBackoffRetries`Fungsi mengembalikan status kegagalan jika koneksi TLS ke broker tidak dapat dibuat setelah jumlah upaya yang dikonfigurasi.

Mengirim permintaan HTTP dan menerima respons

Fungsi prvSendHttpRequest menunjukkan cara mengirim permintaan POST ke server AWS IoT HTTP. Untuk informasi selengkapnya tentang membuat permintaan ke REST API AWS IoT, lihat Protokol komunikasi perangkat - HTTPS. Respons diterima dengan panggilan API CoreHTTP yang sama,. `HTTPClient_Send`

Demo unggahan dasar CoreHTTP Amazon S3

> **Important**
>
> Demo ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-Freertos yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Pengantar

Contoh ini menunjukkan cara mengirim permintaan PUT ke server HTTP Amazon Simple Storage Service (Amazon S3) Simple Storage S3) dan mengunggah file kecil. Ini juga melakukan permintaan GET untuk memverifikasi ukuran file setelah unggahan. Contoh ini menggunakan antarmuka transportasi jaringan yang menggunakan mBEDTLS untuk membuat koneksi yang saling diautentikasi antara klien perangkat IoT yang menjalankan CoreHTTP dan server HTTP Amazon S3.

> **Note**
>
> Untuk mengatur dan menjalankan demo FreeRTOS, ikuti langkah-langkahnya. Memulai dengan FreeRTOS

Berulir tunggal versus multi ulir

Ada dua model penggunaan CoreHTTP, single threaded dan multithreaded (multitasking). Meskipun demo di bagian ini menjalankan perpustakaan HTTP di utas, ini sebenarnya menunjukkan cara menggunakan CoreHTTP dalam satu lingkungan berulir. Hanya satu tugas dalam demo ini yang menggunakan HTTP API. Meskipun aplikasi single threaded harus berulang kali memanggil perpustakaan HTTP, aplikasi multithreaded malah dapat mengirim permintaan HTTP di latar belakang dalam tugas agen (atau daemon).

Organisasi kode sumber

File sumber demo diberi nama `http_demo_s3_upload.c` dan dapat ditemukan di *`freertos`*`/demos/coreHTTP/` direktori dan di GitHubsitus web.

Mengkonfigurasi koneksi server HTTP Amazon S3

Demo ini menggunakan URL yang telah ditandatangani sebelumnya untuk terhubung ke server HTTP Amazon S3 dan mengotorisasi akses ke objek untuk diunduh. Koneksi TLS server HTTP Amazon S3 hanya menggunakan otentikasi server. Pada tingkat aplikasi, akses ke objek diautentikasi dengan parameter dalam kueri URL yang telah ditandatangani sebelumnya. Ikuti langkah-langkah di bawah ini untuk mengonfigurasi koneksi Anda AWS.

1. Siapkan AWS akun:

    a. Jika Anda belum melakukannya, buat AWS akun.

    b. Akun dan izin ditetapkan menggunakan AWS Identity and Access Management (IAM). Anda menggunakan IAM untuk mengelola izin untuk setiap pengguna di akun Anda. Secara default, pengguna tidak memiliki izin sampai diberikan oleh pemilik root.

        i. Untuk menambahkan pengguna ke AWS akun Anda, lihat Panduan Pengguna IAM.

        ii. Berikan izin ke AWS akun Anda untuk mengakses FreeRTOS AWS IoT dan dengan menambahkan kebijakan ini:

            - AmazonS3 FullAccess

2. Buat bucket di Amazon S3 dengan mengikuti langkah-langkah di Bagaimana cara membuat bucket S3? di Panduan Pengguna Layanan Penyimpanan Sederhana Amazon.
3. Unggah file ke Amazon S3 dengan mengikuti langkah-langkah di Bagaimana cara mengunggah file dan folder ke bucket S3? .
4. Hasilkan URL yang telah ditandatangani sebelumnya menggunakan skrip yang terletak di `FreeRTOS-Plus/Demo/coreHTTP_Windows_Simulator/Common/presigned_url_generator/presigned_urls_gen.py` file.

   Untuk petunjuk penggunaan, lihat `FreeRTOS-Plus/Demo/coreHTTP_Windows_Simulator/Common/presigned_url_generator/README.md` file.

Fungsionalitas

Demo pertama terhubung ke server HTTP Amazon S3 dengan otentikasi server TLS. Kemudian, itu membuat permintaan HTTP untuk mengunggah data yang ditentukan dalam`democonfigDEMO_HTTP_UPLOAD_DATA`. Setelah mengunggah file, ia memeriksa bahwa file berhasil diunggah dengan meminta ukuran file. Kode sumber untuk demo dapat ditemukan di situs GitHubweb.

Menghubungkan ke server HTTP Amazon S3

connectToServerWithBackoffRetriesFungsi ini mencoba untuk membuat koneksi TCP ke server HTTP. Jika koneksi gagal, ia mencoba lagi setelah batas waktu. Nilai batas waktu akan meningkat secara eksponensial hingga jumlah upaya maksimum tercapai atau nilai batas waktu maksimum tercapai. `connectToServerWithBackoffRetries`Fungsi mengembalikan status kegagalan jika koneksi TCP ke server tidak dapat dibuat setelah jumlah upaya yang dikonfigurasi.

`prvConnectToServer`Fungsi ini menunjukkan cara membuat koneksi ke server HTTP Amazon S3 dengan menggunakan otentikasi server saja. Ini menggunakan antarmuka transport berbasis MBEDTLS yang diimplementasikan dalam file. `FreeRTOS-Plus/Source/Application-Protocols/network_transport/freertos_plus_tcp/using_mbedtls/using_mbedtls.c` Definisi `prvConnectToServer` dapat ditemukan di situs GitHubweb.

Unggah data

`prvUploadS3ObjectFile`Fungsi ini menunjukkan cara membuat permintaan PUT dan menentukan file yang akan diunggah. Bucket Amazon S3 tempat file diunggah dan nama file yang akan diunggah ditentukan dalam URL yang telah ditandatangani sebelumnya. Untuk menyimpan

memori, buffer yang sama digunakan untuk header permintaan dan untuk menerima respons. Respons diterima secara sinkron menggunakan fungsi `HTTPClient_Send` API. Kode status `200 OK` respons diharapkan dari server HTTP Amazon S3. Kode status lainnya adalah kesalahan.

Kode sumber untuk `prvUploadS3ObjectFile()` dapat ditemukan di situs GitHubweb.

Memverifikasi unggahan

`prvVerifyS3ObjectFileSize`Fungsi ini memanggil `prvGetS3ObjectFileSize` untuk mengambil ukuran objek di bucket S3. Server HTTP Amazon S3 saat ini tidak mendukung permintaan HEAD menggunakan URL yang telah ditandatangani sebelumnya, jadi byte ke-0 diminta. Ukuran file terkandung dalam bidang `Content-Range` header respon. `206 Partial Content`Respons diharapkan dari server. Kode status respons lainnya adalah kesalahan.

Kode sumber untuk `prvGetS3ObjectFileSize()` dapat ditemukan di situs GitHubweb.

Demo unduhan dasar CoreHTTP S3

> **Important**
>
> Demo ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-Freertos yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Pengantar

Demo ini menunjukkan cara menggunakan permintaan rentang untuk mengunduh file dari server HTTP Amazon S3. Permintaan rentang didukung secara native di CoreHTTP API saat Anda gunakan `HTTPClient_AddRangeHeader` untuk membuat permintaan HTTP. Untuk lingkungan mikrokontroler, permintaan jangkauan sangat dianjurkan. Dengan mengunduh file besar dalam rentang terpisah, alih-alih dalam satu permintaan, setiap bagian file dapat diproses tanpa memblokir soket jaringan. Permintaan jangkauan menurunkan risiko paket yang jatuh, yang memerlukan transmisi ulang pada koneksi TCP, sehingga meningkatkan konsumsi daya perangkat.

Contoh ini menggunakan antarmuka transportasi jaringan yang menggunakan mBEDTLS untuk membuat koneksi yang saling diautentikasi antara klien perangkat IoT yang menjalankan CoreHTTP dan server HTTP Amazon S3.

> **Note**
>
> Untuk mengatur dan menjalankan demo FreeRTOS, ikuti langkah-langkahnya. Memulai dengan FreeRTOS

Berulir tunggal versus multi ulir

Ada dua model penggunaan CoreHTTP, single threaded dan multithreaded (multitasking). Meskipun demo di bagian ini menjalankan perpustakaan HTTP di utas, ini sebenarnya menunjukkan cara menggunakan CoreHTTP dalam satu lingkungan berulir (hanya satu tugas yang menggunakan API HTTP dalam demo). Meskipun aplikasi single threaded harus berulang kali memanggil perpustakaan HTTP, aplikasi multithreaded malah dapat mengirim permintaan HTTP di latar belakang dalam tugas agen (atau daemon).

Organisasi kode sumber

Proyek demo diberi nama `http_demo_s3_download.c` dan dapat ditemukan di *`freertos`*`/demos/coreHTTP/` direktori dan di GitHubsitus web.

Mengkonfigurasi koneksi server HTTP Amazon S3

Demo ini menggunakan URL yang telah ditandatangani sebelumnya untuk terhubung ke server HTTP Amazon S3 dan mengotorisasi akses ke objek untuk diunduh. Koneksi TLS server HTTP Amazon S3 hanya menggunakan otentikasi server. Pada tingkat aplikasi, akses ke objek diautentikasi dengan parameter dalam kueri URL yang telah ditandatangani sebelumnya. Ikuti langkah-langkah di bawah ini untuk mengonfigurasi koneksi Anda AWS.

1. Siapkan AWS akun:
   a. Jika Anda belum melakukannya, buat dan aktifkan AWS akun.
   b. Akun dan izin ditetapkan menggunakan AWS Identity and Access Management (IAM). IAM memungkinkan Anda mengelola izin untuk setiap pengguna di akun Anda. Secara default, pengguna tidak memiliki izin sampai diberikan oleh pemilik root.
      i. Untuk menambahkan pengguna ke AWS akun Anda, lihat Panduan Pengguna IAM.
      ii. Berikan izin ke AWS akun Anda untuk mengakses FreeRTOS AWS IoT dan dengan menambahkan kebijakan ini:
         - AmazonS3 FullAccess

2. Buat bucket di S3 dengan mengikuti langkah-langkah di Bagaimana cara membuat Bucket S3? di Panduan Pengguna Konsol Layanan Penyimpanan Sederhana Amazon.
3. Unggah file ke S3 dengan mengikuti langkah-langkah di Bagaimana cara mengunggah file dan folder ke bucket S3? .
4. Hasilkan URL yang telah ditandatangani sebelumnya menggunakan skrip yang terletak di`FreeRTOS-Plus/Demo/coreHTTP_Windows_Simulator/Common/presigned_url_generator/presigned_urls_gen.py`. Untuk petunjuk penggunaan, lihat`FreeRTOS-Plus/Demo/coreHTTP_Windows_Simulator/Common/presigned_url_generator/README.md`.

Fungsionalitas

Demo mengambil ukuran file terlebih dahulu. Kemudian ia meminta setiap rentang byte secara berurutan, dalam satu lingkaran, dengan ukuran rentang. `democonfigRANGE_REQUEST_LENGTH`

Kode sumber untuk demo dapat ditemukan di situs GitHubweb.

Menghubungkan ke server HTTP Amazon S3

Fungsi connectToServerWithBackoffRetries() mencoba untuk membuat koneksi TCP ke server HTTP. Jika koneksi gagal, ia mencoba lagi setelah batas waktu. Nilai batas waktu akan meningkat secara eksponensial hingga jumlah upaya maksimum tercapai atau nilai batas waktu maksimum tercapai. `connectToServerWithBackoffRetries()`mengembalikan status kegagalan jika koneksi TCP ke server tidak dapat dibuat setelah jumlah upaya yang dikonfigurasi.

Fungsi ini `prvConnectToServer()` menunjukkan cara membuat koneksi ke server HTTP Amazon S3 hanya menggunakan otentikasi server. Ini menggunakan antarmuka transport berbasis MBEDTLS yang diimplementasikan dalam file FreerTos- _mbedtls.c. Plus/Source/Application-Protocols/network_transport/freertos_plus_tcp/using_mbedtls/using

Kode sumber untuk `prvConnectToServer()` dapat ditemukan di GitHub.

Membuat permintaan rentang

Fungsi API `HTTPClient_AddRangeHeader()` mendukung serialisasi rentang byte ke header permintaan HTTP untuk membentuk permintaan rentang. Rentang permintaan digunakan dalam demo ini untuk mengambil ukuran file dan untuk meminta setiap bagian dari file.

Fungsi `prvGetS3ObjectFileSize()` mengambil ukuran file di bucket S3. `Connection: keep-alive`Header ditambahkan dalam permintaan pertama ini ke Amazon S3 untuk menjaga koneksi

tetap terbuka setelah respons dikirim. Server HTTP S3 saat ini tidak mendukung permintaan HEAD menggunakan URL yang telah ditandatangani sebelumnya, sehingga byte ke-0 diminta. Ukuran file terkandung dalam bidang `Content-Range` header respon. `206 Partial Content`Respons diharapkan dari server; kode status respons lain yang diterima adalah kesalahan.

Kode sumber untuk `prvGetS3ObjectFileSize()` dapat ditemukan di GitHub.

Setelah mengambil ukuran file, demo ini membuat permintaan rentang baru untuk setiap rentang byte file yang akan diunduh. Ini digunakan `HTTPClient_AddRangeHeader()` untuk setiap bagian file.

Mengirim permintaan jangkauan dan menerima tanggapan

Fungsi `prvDownloadS3ObjectFile()` mengirimkan permintaan rentang dalam satu lingkaran hingga seluruh file diunduh. Fungsi API `HTTPClient_Send()` mengirimkan permintaan dan menerima respons secara sinkron. Ketika fungsi kembali, respon diterima dalam file`xResponse`. Kode status kemudian diverifikasi menjadi `206 Partial Content` dan jumlah byte yang diunduh sejauh ini ditambah dengan nilai header. `Content-Length`

Kode sumber untuk `prvDownloadS3ObjectFile()` dapat ditemukan di GitHub.

CoreHTTP demo multithreaded dasar

> **Important**
>
> Demo ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-Freertos yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Pengantar

Demo ini menggunakan antrian aman thread FreeRTOS untuk menahan permintaan dan tanggapan yang menunggu untuk diproses. Dalam demo ini, ada tiga tugas yang perlu diperhatikan.

- Tugas utama menunggu permintaan muncul dalam antrian permintaan. Ini akan mengirim permintaan tersebut melalui jaringan, kemudian menempatkan respons ke dalam antrian respons.
- Tugas permintaan membuat objek permintaan perpustakaan HTTP untuk dikirim ke server dan menempatkannya ke dalam antrian permintaan. Setiap objek permintaan menentukan rentang byte dari file S3 yang telah dikonfigurasi aplikasi untuk diunduh.

- Tugas respons menunggu respons muncul dalam antrian respons. Ini mencatat setiap respons yang diterimanya.

Demo multithreaded dasar ini dikonfigurasi untuk menggunakan koneksi TLS dengan otentikasi server saja, ini diperlukan oleh server HTTP Amazon S3. Otentikasi lapisan aplikasi dilakukan dengan menggunakan parameter Signature Version 4 dalam kueri URL yang telah ditetapkan sebelumnya.

Organisasi kode sumber

Proyek demo diberi nama `http_demo_s3_download_multithreaded.c` dan dapat ditemukan di *`freertos`*`/demos/coreHTTP/` direktori dan situs GitHubweb.

Membangun proyek demo

Proyek demo menggunakan edisi komunitas gratis Visual Studio. Untuk membangun demo:

1. Buka file solusi `mqtt_multitask_demo.sln` Visual Studio dari dalam Visual Studio IDE.
2. Pilih Build Solution dari menu Build IDE.

> **Note**
>
> Jika Anda menggunakan Microsoft Visual Studio 2017 atau yang lebih lama, maka Anda harus memilih Platform Toolset yang kompatibel dengan versi Anda: Project -> RTOSDemos Properties -> Platform Toolset.

Mengkonfigurasi proyek demo

Demo menggunakan tumpukan Freertos+TCP TCP/IP, jadi ikuti instruksi yang diberikan untuk proyek starter TCP/IP untuk:

1. Instal komponen prasyarat (seperti WinPCap).
2. Secara opsional mengatur alamat IP statis atau dinamis, alamat gateway dan netmask.
3. Secara opsional mengatur alamat MAC.
4. Pilih antarmuka jaringan Ethernet di mesin host Anda.
5. Yang penting menguji koneksi jaringan Anda sebelum mencoba menjalankan demo HTTP.

Mengkonfigurasi koneksi server HTTP Amazon S3

Ikuti instruksi untuk Mengkonfigurasi koneksi server HTTP Amazon S3 di demo unduhan dasar CoreHTTP.

Fungsionalitas

Demo menciptakan tiga tugas secara total:

- Salah satu yang mengirim permintaan dan menerima tanggapan melalui jaringan.
- Salah satu yang membuat permintaan untuk dikirim.
- Salah satu yang memproses tanggapan yang diterima.

Dalam demo ini, tugas utama:

1. Membuat antrian permintaan dan respons.
2. Membuat koneksi ke server.
3. Membuat tugas permintaan dan respons.
4. Menunggu antrian permintaan untuk mengirim permintaan melalui jaringan.
5. Menempatkan tanggapan yang diterima melalui jaringan ke dalam antrian respons.

Tugas permintaan:

1. Menciptakan setiap permintaan rentang.

Tugas respons:

1. Memproses setiap tanggapan yang diterima.

Typedefs

Demo mendefinisikan struktur berikut untuk mendukung multithreading.

Permintaan item

Struktur berikut menentukan item permintaan untuk ditempatkan ke dalam antrian permintaan. Item permintaan disalin ke antrian setelah tugas permintaan membuat permintaan HTTP.

```
/**
 * @brief Data type for the request queue.
 *
 * Contains the request header struct and its corresponding buffer, to be
 * populated and enqueued by the request task, and read by the main task. The
 * buffer is included to avoid pointer inaccuracy during queue copy operations.
 */
typedef struct RequestItem
{
    HTTPRequestHeaders_t xRequestHeaders;
    uint8_t ucHeaderBuffer[ democonfigUSER_BUFFER_LENGTH ];
} RequestItem_t;
```

Item Respons

Struktur berikut menentukan item respons untuk ditempatkan ke dalam antrian respons. Item respons disalin ke dalam antrian setelah tugas HTTP utama menerima respons melalui jaringan.

```
/**
 * @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;
```

Tugas kirim HTTP utama

Tugas aplikasi utama:

1. Mem-parsing URL yang telah ditetapkan sebelumnya untuk alamat host untuk membuat koneksi dengan server HTTP Amazon S3.
2. Mem-parsing URL yang telah ditetapkan sebelumnya untuk jalur ke objek di bucket S3.
3. Terhubung ke server HTTP Amazon S3 menggunakan TLS dengan otentikasi server.

4. Membuat antrian permintaan dan respons.
5. Membuat tugas permintaan dan respons.

Fungsi `prvHTTPDemoTask()` melakukan pengaturan ini, dan memberikan status demo. Kode sumber untuk fungsi ini dapat ditemukan di Github.

Dalam fungsinya`prvDownloadLoop()`, tugas utama memblokir dan menunggu permintaan dari antrian permintaan. Ketika menerima permintaan, ia mengirimkannya menggunakan fungsi API`HTTPClient_Send()`. Jika fungsi API berhasil, maka ia menempatkan respons ke dalam antrian respons.

Kode sumber untuk `prvDownloadLoop()` dapat ditemukan di Github.

Tugas permintaan HTTP

Tugas permintaan ditentukan dalam fungsi`prvRequestTask`. Kode sumber untuk fungsi ini dapat ditemukan di Github.

Tugas permintaan mengambil ukuran file di bucket Amazon S3. Ini dilakukan dalam fungsi`prvGetS3ObjectFileSize`. Header “Connection: keep-alive” ditambahkan ke permintaan ini ke Amazon S3 agar koneksi tetap terbuka setelah respons dikirim. Server HTTP Amazon S3 saat ini tidak mendukung permintaan HEAD menggunakan URL yang telah ditetapkan sebelumnya, sehingga byte ke-0 diminta. Ukuran file terkandung dalam bidang `Content-Range` header respon. `206 Partial Content`Respons diharapkan dari server; kode status respons lain yang diterima adalah kesalahan.

Kode sumber untuk `prvGetS3ObjectFileSize` dapat ditemukan di Github.

Setelah mengambil ukuran file, tugas permintaan terus meminta setiap rentang file. Setiap permintaan rentang ditempatkan ke dalam antrian permintaan untuk tugas utama yang akan dikirim. Rentang file dikonfigurasi oleh pengguna demo di makro`democonfigRANGE_REQUEST_LENGTH`. Permintaan rentang didukung secara native di API pustaka klien HTTP menggunakan fungsi `HTTPClient_AddRangeHeader` tersebut. Fungsi ini `prvRequestS3ObjectRange` menunjukkan cara menggunakan`HTTPClient_AddRangeHeader()`.

Kode sumber untuk fungsi tersebut `prvRequestS3ObjectRange` dapat ditemukan di Github.

Tugas respons HTTP

Tugas respons menunggu pada antrian respons untuk tanggapan yang diterima melalui jaringan. Tugas utama mengisi antrian respons ketika berhasil menerima respons HTTP. Tugas ini memproses respons dengan mencatat kode status, header, dan badan. Aplikasi dunia nyata dapat memproses respons dengan menulis badan respons ke memori flash, misalnya. Jika kode status respons tidak`206 partial content`, maka tugas memberi tahu tugas utama bahwa demo harus gagal. Tugas respons ditentukan dalam fungsi`prvResponseTask`. Kode sumber untuk fungsi ini dapat ditemukan di Github.

AWS IoT Demo perpustakaan pekerjaan

> **Important**
>
> Demo ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-FreerTOS yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Pengantar

Demo pustaka AWS IoT Jobs menunjukkan kepada Anda cara menyambung ke layanan AWS IoT Jobs melalui koneksi MQTT, mengambil pekerjaan dari AWS IoT, dan memprosesnya di perangkat. Proyek demo AWS IoT Jobs menggunakan port Windows FreeRTOS, sehingga dapat dibangun dan dievaluasi dengan versi Komunitas Visual Studio di Windows. Tidak diperlukan perangkat keras mikrokontroler. Demo menetapkan koneksi aman ke broker AWS IoT MQTT menggunakan TLS dengan cara yang sama seperti. Demo otentikasi timbal balik CoreMQTT

> **Note**
>
> Untuk mengatur dan menjalankan demo FreeRTOS, ikuti langkah-langkahnya. Memulai dengan FreeRTOS

Organisasi kode sumber

Kode demo ada di `jobs_demo.c` file dan dapat ditemukan di GitHubsitus web atau di *`freertos`*`/demos/jobs_for_aws/` direktori.

Konfigurasikan koneksi AWS IoT broker MQTT

Dalam demo ini, Anda menggunakan koneksi MQTT ke broker MQTT. AWS IoT Koneksi ini dikonfigurasi dengan cara yang sama sepertiDemo otentikasi timbal balik CoreMQTT.

Fungsionalitas

Demo menunjukkan alur kerja yang digunakan untuk menerima pekerjaan dari AWS IoT dan memprosesnya di perangkat. Demo ini interaktif dan mengharuskan Anda untuk membuat pekerjaan dengan menggunakan AWS IoT konsol atau AWS Command Line Interface (AWS CLI). Untuk informasi selengkapnya tentang membuat pekerjaan, lihat create-job di Command Reference.AWS CLI Demo mengharuskan dokumen pekerjaan memiliki `action` kunci yang disetel `print` untuk mencetak pesan ke konsol.

Lihat format berikut untuk dokumen pekerjaan ini.

```
{
    "action": "print",
    "message": "ADD_MESSAGE_HERE"
}
```

Anda dapat menggunakan AWS CLI untuk membuat pekerjaan seperti pada perintah contoh berikut.

```
aws iot create-job \
    --job-id t12 \
    --targets arn:aws:iot:region:123456789012:thing/device1 \
    --document '{"action":"print","message":"hello world!"}'
```

Demo juga menggunakan dokumen pekerjaan yang memiliki `action` kunci yang disetel `publish` untuk menerbitkan ulang pesan ke suatu topik. Lihat format berikut untuk dokumen pekerjaan ini.

```
{
    "action": "publish",
    "message": "ADD_MESSAGE_HERE",
    "topic": "topic/name/here"
}
```

Demo loop hingga menerima dokumen pekerjaan dengan `action` kunci yang disetel `exit` untuk keluar dari demo. Format untuk dokumen pekerjaan adalah sebagai berikut.

```
{
    "action: "exit"
}
```

Titik masuk demo Jobs

Kode sumber untuk fungsi titik masuk demo Jobs dapat ditemukan di GitHub. Fungsi ini melakukan operasi berikut:

1. Buat koneksi MQTT menggunakan fungsi pembantu di. `mqtt_demo_helpers.c`
2. Berlangganan topik MQTT untuk `NextJobExecutionChanged` API, menggunakan fungsi pembantu di. `mqtt_demo_helpers.c` String topik dirakit sebelumnya, menggunakan makro yang ditentukan oleh pustaka AWS IoT Jobs.
3. Publikasikan ke topik MQTT untuk `StartNextPendingJobExecution` API, menggunakan fungsi pembantu di. `mqtt_demo_helpers.c` String topik dirakit sebelumnya, menggunakan makro yang ditentukan oleh pustaka AWS IoT Jobs.
4. Panggilan berulang kali `MQTT_ProcessLoop` untuk menerima pesan masuk yang diserahkan `prvEventCallback` untuk diproses.
5. Setelah demo menerima tindakan keluar, berhenti berlangganan dari topik MQTT dan putuskan sambungan, menggunakan fungsi pembantu dalam file. `mqtt_demo_helpers.c`

Callback untuk pesan MQTT yang diterima

prvEventCallbackFungsi panggilan `Jobs_MatchTopic` dari pustaka AWS IoT Jobs untuk mengklasifikasikan pesan MQTT yang masuk. Jika jenis pesan sesuai dengan pekerjaan baru, `prvNextJobHandler()` dipanggil.

Fungsi prvNextJobHandler, dan fungsi yang dipanggilnya, mengurai dokumen pekerjaan dari pesan berformat JSON, dan menjalankan tindakan yang ditentukan oleh pekerjaan. Yang menarik adalah `prvSendUpdateForJob` fungsinya.

Kirim pembaruan untuk pekerjaan yang sedang berjalan

Fungsi prvSendUpdateForJob() memanggil `Jobs_Update()` dari pustaka Jobs untuk mengisi string topik yang digunakan dalam operasi publikasi MQTT yang segera mengikuti.

Demo CoreMQTT

> **Important**
>
> Demo ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-Freertos yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Demo ini dapat membantu Anda mempelajari cara menggunakan pustaka CoreMQTT.

Topik

- Demo otentikasi timbal balik CoreMQTT
- Demo berbagi koneksi Agen CoreMQTT

Demo otentikasi timbal balik CoreMQTT

> **Important**
>
> Demo ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-Freertos yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Pengantar

Proyek demo otentikasi timbal balik CoreMQTT menunjukkan kepada Anda cara membuat koneksi ke broker MQTT menggunakan TLS dengan otentikasi timbal balik antara klien dan server. Demo ini menggunakan implementasi antarmuka transport berbasis MBEDTLS untuk membuat server dan koneksi TLS yang diautentikasi klien, dan menunjukkan alur kerja subscribe-publish MQTT pada tingkat QoS 1. Ini berlangganan filter topik, kemudian menerbitkan ke topik yang cocok dengan filter dan menunggu penerimaan pesan-pesan itu kembali dari server di tingkat QoS 1. Siklus penerbitan ke broker dan menerima pesan yang sama kembali dari broker diulang tanpa batas waktu. Pesan dalam demo ini dikirim di QoS 1, yang menjamin setidaknya satu pengiriman sesuai dengan spesifikasi MQTT.

> **Note**
>
> Untuk mengatur dan menjalankan demo FreeRTOS, ikuti langkah-langkahnya. Memulai dengan FreeRTOS

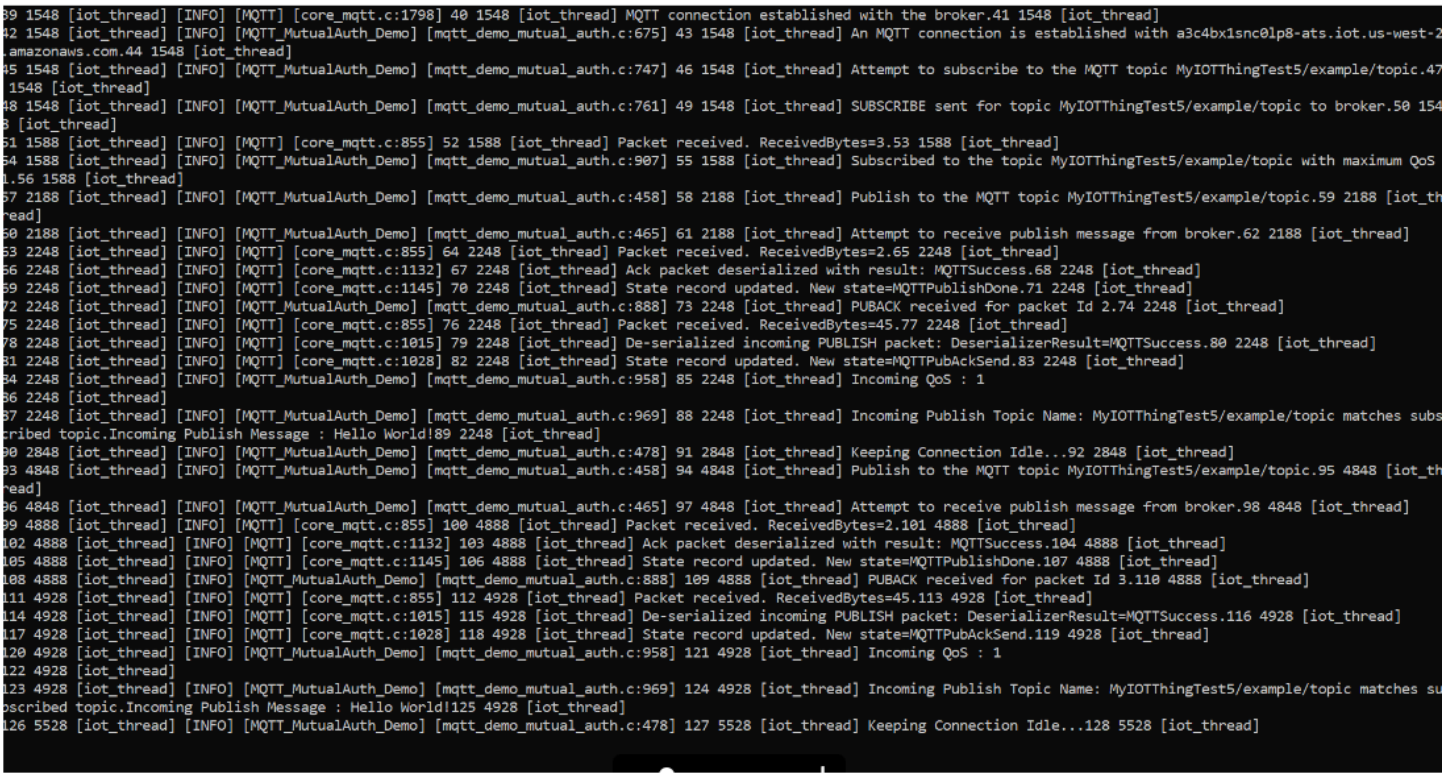
Kode sumber

File sumber demo diberi nama `mqtt_demo_mutual_auth.c` dan dapat ditemukan di *`freertos`*`/demos/coreMQTT/` direktori dan GitHubsitus web.

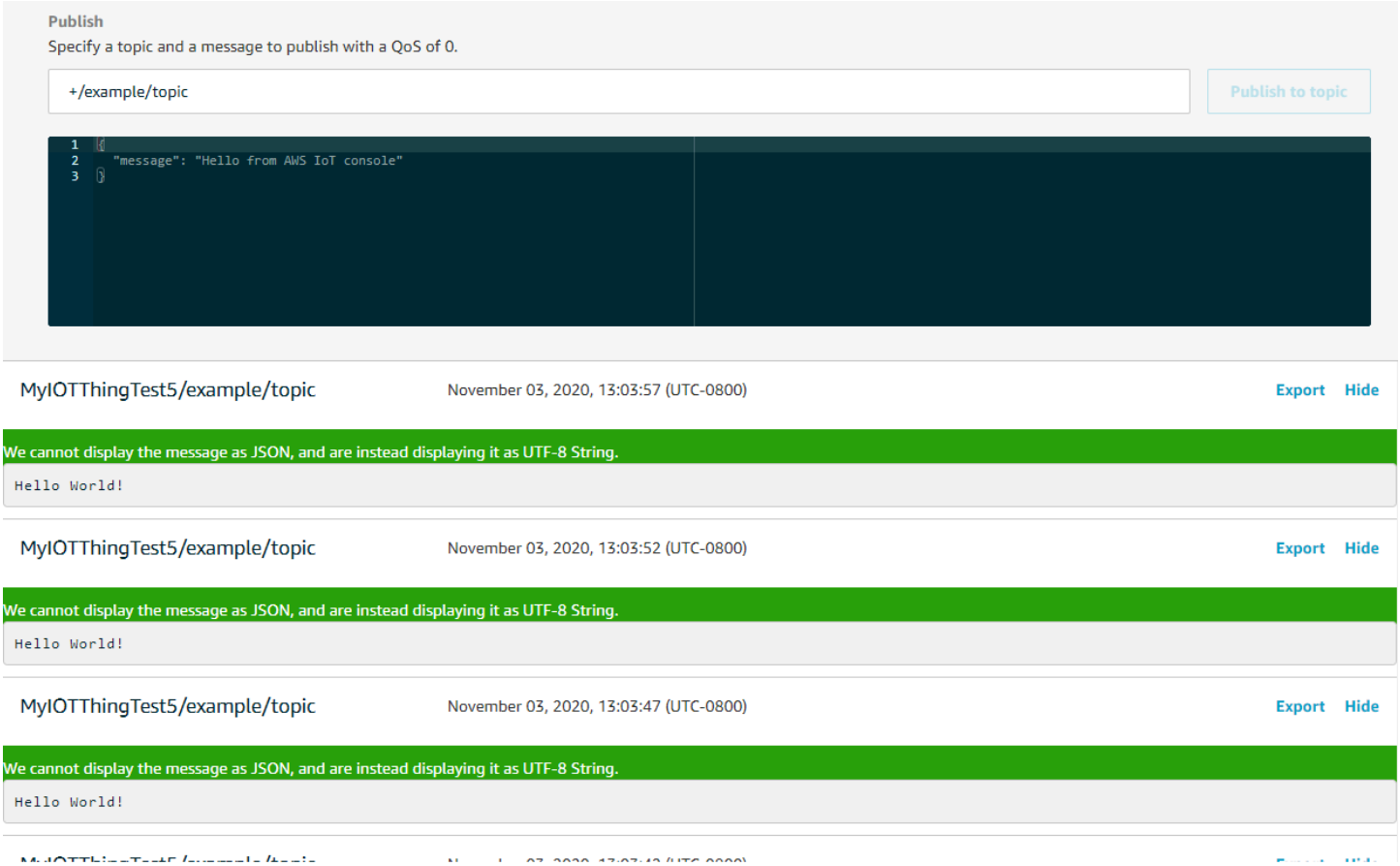
Fungsionalitas

Demo menciptakan tugas aplikasi tunggal yang berputar melalui serangkaian contoh yang menunjukkan cara terhubung ke broker, berlangganan topik di broker, mempublikasikan ke topik di broker, lalu akhirnya, putuskan sambungan dari broker. Aplikasi demo berlangganan dan menerbitkan topik yang sama. Setiap kali demo menerbitkan pesan ke broker MQTT, broker mengirim pesan yang sama kembali ke aplikasi demo.

Penyelesaian demo yang berhasil akan menghasilkan output yang mirip dengan gambar berikut.



AWS IoT Konsol akan menghasilkan output yang mirip dengan gambar berikut.



Coba lagi logika dengan backoff eksponensial dan jitter

Fungsi prvBackoffForCoba lagi menunjukkan bagaimana operasi jaringan yang gagal dengan server, misalnya, koneksi TLS atau permintaan berlangganan MQTT, dapat dicoba ulang dengan backoff eksponensial dan jitter. Fungsi menghitung periode backoff untuk upaya coba lagi berikutnya, dan melakukan penundaan backoff jika upaya coba lagi belum habis. Karena perhitungan periode backoff membutuhkan pembuatan angka acak, fungsi menggunakan PKCS11 modul untuk menghasilkan angka acak. Penggunaan PKCS11 modul memungkinkan akses ke True Random Number Generator (TRNG) jika platform vendor mendukungnya. Kami menyarankan Anda menyemai generator angka acak dengan sumber entropi khusus perangkat sehingga kemungkinan tabrakan dari perangkat selama percobaan ulang koneksi dikurangi.

Menghubungkan ke broker MQTT

prvConnectToServerWithBackoffRetriesFungsi ini mencoba untuk membuat koneksi TLS yang saling diautentikasi ke broker MQTT. Jika koneksi gagal, ia mencoba lagi setelah periode backoff. Periode backoff akan meningkat secara eksponensial sampai jumlah upaya maksimum tercapai atau periode backoff maksimum tercapai. `BackoffAlgorithm_GetNextBackoff`Fungsi

ini memberikan nilai backoff yang meningkat secara eksponensial dan pengembalian `RetryUtilsRetriesExhausted` ketika jumlah upaya maksimum telah tercapai. `prvConnectToServerWithBackoffRetries`Fungsi mengembalikan status kegagalan jika koneksi TLS ke broker tidak dapat dibuat setelah jumlah upaya yang dikonfigurasi.

MQTTConnectionWithBrokerFungsi PRVCreate menunjukkan cara membuat koneksi MQTT ke broker MQTT dengan sesi bersih. Ini menggunakan antarmuka transportasi TLS, yang diimplementasikan dalam `FreeRTOS-Plus/Source/Application-Protocols/platform/freertos/transport/src/tls_freertos.c` file. Ingatlah bahwa kami sedang mengatur detik keep-alive untuk broker. `xConnectInfo`

Fungsi selanjutnya menunjukkan bagaimana antarmuka transport TLS dan fungsi waktu diatur dalam konteks MQTT menggunakan fungsi tersebut. `MQTT_Init` Hal ini juga menunjukkan bagaimana sebuah event callback function pointer (`prvEventCallback`) diatur. Callback ini digunakan untuk melaporkan pesan masuk.

Berlangganan topik MQTT

MQTTSubscribeWithBackoffRetriesFungsi prv menunjukkan cara berlangganan filter topik pada broker MQTT. Contoh ini menunjukkan cara berlangganan satu filter topik, tetapi dimungkinkan untuk meneruskan daftar filter topik dalam panggilan API berlangganan yang sama untuk berlangganan lebih dari satu filter topik. Juga, jika broker MQTT menolak permintaan berlangganan, langganan akan mencoba lagi, dengan dukungan eksponensial, untuk. `RETRY_MAX_ATTEMPTS`

Menerbitkan ke topik

MQTTPublishToTopicFungsi prv menunjukkan cara mempublikasikan ke topik di broker MQTT.

Menerima pesan masuk

Aplikasi mendaftarkan fungsi callback peristiwa sebelum terhubung ke broker, seperti yang dijelaskan sebelumnya. `prvMQTTDemoTask`Fungsi ini memanggil `MQTT_ProcessLoop` fungsi untuk menerima pesan masuk. Ketika pesan MQTT masuk diterima, ia memanggil fungsi callback acara yang terdaftar oleh aplikasi. prvEventCallbackFungsi ini adalah contoh dari fungsi callback peristiwa tersebut. `prvEventCallback`memeriksa jenis paket yang masuk dan memanggil handler yang sesuai. Dalam contoh di bawah ini, fungsi panggilan `prvMQTTProcessIncomingPublish()` untuk menangani pesan publikasi masuk atau `prvMQTTProcessResponse()` untuk menangani pengakuan (ACK).

Memproses paket publikasi MQTT yang masuk

MQTTProcessIncomingPublishFungsi prv menunjukkan cara memproses paket publikasi dari broker MQTT.

Berhenti berlangganan dari suatu topik

Langkah terakhir dalam alur kerja adalah berhenti berlangganan dari topik sehingga broker tidak akan mengirim pesan yang dipublikasikan. `mqttexampleTOPIC` Berikut adalah definisi dari fungsi prv MQTTUnsubscribe FromTopic.

Mengubah root CA yang digunakan dalam demo

Secara default, demo FreeRTOS menggunakan sertifikat Amazon Root CA 1 (kunci RSA 2048 bit) untuk mengautentikasi dengan server. AWS IoT Core Dimungkinkan untuk menggunakan sertifikat CA lainnya untuk otentikasi server, termasuk sertifikat Amazon Root CA 3 (kunci ECC 256 bit). Untuk mengubah CA root untuk demo otentikasi timbal balik CoreMQTT:

1. Dalam editor teks, buka *freertos*`/vendors/`*vendor*`/boards/`*board*`/aws_demos/config_files/mqtt_demo_mutual_auth_config.h` file.
2. Dalam file, cari baris berikut.

   ```
   * #define democonfigROOT_CA_PEM    "...insert here..."
   ```

   Hapus komentar baris ini dan, jika perlu, pindahkan melewati akhir `*/` blok komentar.
3. Salin sertifikat CA yang ingin Anda gunakan lalu tempel di `"...insert here..."` teks. Hasilnya akan terlihat seperti contoh berikut ini.

   ```
   #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. (Opsional) Anda dapat mengubah CA root untuk demo lainnya. Ulangi langkah 1 hingga 3 untuk setiap *freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/*demo-name*_config.h file.

Demo berbagi koneksi Agen CoreMQTT

> **Important**
>
> Demo ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-Freertos yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Pengantar

Proyek demo berbagi koneksi CoreMQTT menunjukkan kepada Anda cara menggunakan aplikasi multithreaded untuk membuat koneksi ke broker AWS MQTT menggunakan TLS dengan otentikasi timbal balik antara klien dan server. Demo ini menggunakan implementasi antarmuka transport berbasis MBEDTLS untuk membuat server dan koneksi TLS yang diautentikasi klien, dan menunjukkan alur kerja subscribe-publish MQTT pada tingkat QoS 1. Demo berlangganan filter topik, menerbitkan topik yang cocok dengan filter, dan kemudian menunggu untuk menerima pesan-pesan itu kembali dari server di tingkat QoS 1. Siklus penerbitan ke broker dan menerima pesan yang sama kembali dari broker diulang beberapa kali untuk setiap tugas yang dibuat. Pesan dalam demo ini dikirim di QoS 1, yang menjamin setidaknya satu pengiriman sesuai dengan spesifikasi MQTT.

> **Note**
>
> Untuk mengatur dan menjalankan demo FreeRTOS, ikuti langkah-langkahnya. Memulai dengan FreeRTOS

Demo ini menggunakan antrian aman utas untuk menahan perintah untuk berinteraksi dengan MQTT API. Ada dua tugas yang perlu diperhatikan dalam demo ini.

- Tugas Agen MQTT (utama) memproses perintah dari antrian perintah sementara tugas lain mengantrekannya. Tugas ini memasuki loop, di mana ia memproses perintah dari antrian perintah. Jika perintah terminasi diterima, tugas ini akan keluar dari loop.

- Tugas subpub demo membuat langganan ke topik MQTT, lalu membuat operasi publikasi dan mendorongnya ke antrian perintah. Operasi publikasi ini kemudian dijalankan oleh tugas Agen MQTT. Tugas subpub demo menunggu publikasi selesai, ditunjukkan dengan eksekusi callback penyelesaian perintah, kemudian memasuki penundaan singkat sebelum memulai publikasi berikutnya. Tugas ini menunjukkan contoh bagaimana tugas aplikasi akan menggunakan CoreMQTT Agent API.

Untuk pesan publikasi yang masuk, Agen CoreMQTT memanggil fungsi panggilan balik tunggal. Demo ini juga mencakup pengelola langganan yang memungkinkan tugas menentukan panggilan balik untuk memanggil pesan publikasi masuk tentang topik yang telah mereka langgani. Panggilan balik publikasi masuk agen dalam demo ini memanggil manajer berlangganan untuk menyebarkan publikasi ke tugas apa pun yang telah mendaftarkan langganan.

Demo ini menggunakan koneksi TLS dengan otentikasi timbal balik untuk terhubung. AWS Jika jaringan tiba-tiba terputus selama demo, maka klien mencoba untuk menyambung kembali menggunakan logika backoff eksponensial. Jika klien berhasil terhubung kembali, tetapi broker tidak dapat melanjutkan sesi sebelumnya, maka klien akan berlangganan kembali ke topik yang sama dengan sesi sebelumnya.

Single threaded vs multithreaded

Ada dua model penggunaan CoreMQTT, single threaded dan multithreaded (multitasking). Model berulir tunggal menggunakan pustaka CoreMQTT hanya dari satu utas, dan mengharuskan Anda untuk membuat panggilan eksplisit berulang di pustaka MQTT. Kasus penggunaan multithreaded malah dapat menjalankan protokol MQTT di latar belakang dalam tugas agen (atau daemon), seperti yang ditunjukkan dalam demo yang didokumentasikan di sini. Saat Anda menjalankan protokol MQTT dalam tugas agen, Anda tidak perlu secara eksplisit mengelola status MQTT apa pun atau memanggil fungsi API. `MQTT_ProcessLoop` Selain itu, saat Anda menggunakan tugas agen, beberapa tugas aplikasi dapat berbagi satu koneksi MQTT tanpa perlu sinkronisasi primitif seperti mutex.

Kode sumber

File sumber demo diberi nama `mqtt_agent_task.c` dan `simple_sub_pub_demo.c` dan dapat ditemukan di *`freertos`*`/demos/coreMQTT_Agent/` direktori dan GitHubsitus web.

Fungsionalitas

Demo ini membuat setidaknya dua tugas: tugas utama yang memproses permintaan panggilan API MQTT, dan sejumlah subtugas yang dapat dikonfigurasi yang membuat permintaan tersebut. Dalam demo ini, tugas utama membuat subtugas, memanggil loop pemrosesan, dan membersihkan setelahnya. Tugas utama menciptakan koneksi MQTT tunggal ke broker yang dibagi di antara subtugas. Subtugas membuat langganan MQTT dengan broker dan kemudian mempublikasikan pesan ke sana. Setiap subtugas menggunakan topik unik untuk penerbitannya.

Tugas utama

Tugas aplikasi utama, RunCoreMQTTAgentDemo, menetapkan sesi MQTT, membuat subtugas, dan menjalankan loop pemrosesan MQTTAgent_ CommandLoop hingga perintah terminasi diterima. Jika jaringan tiba-tiba terputus, demo akan terhubung kembali ke broker di latar belakang, dan membangun kembali langganan dengan broker. Setelah loop pemrosesan berakhir, ia terputus dari broker.

Commands

Saat Anda menjalankan API Agen CoreMQTT, ia membuat perintah yang dikirim ke antrian tugas agen, yang diproses di. `MQTTAgent_CommandLoop()` Pada saat perintah dibuat, callback penyelesaian opsional dan parameter konteks dapat diteruskan. Setelah perintah yang sesuai selesai, callback penyelesaian akan dipanggil dengan konteks yang diteruskan dan nilai pengembalian apa pun yang dibuat sebagai hasil dari perintah. Tanda tangan untuk callback penyelesaian adalah sebagai berikut:

```
typedef void (* MQTTAgentCommandCallback_t )( void * pCmdCallbackContext,
                                              MQTTAgentReturnInfo_t * pReturnInfo );
```

Konteks penyelesaian perintah ditentukan pengguna; untuk demo ini, itu adalah: struct. MQTTAgent CommandContext

Perintah dianggap selesai ketika:

- Berlangganan, berhenti berlangganan, dan menerbitkan dengan QoS > 0: Setelah paket pengakuan yang sesuai telah diterima.
- Semua operasi lainnya: Setelah API CoreMQTT yang sesuai telah dipanggil.

Setiap struktur yang digunakan oleh perintah, termasuk mempublikasikan informasi, informasi berlangganan, dan konteks penyelesaian, harus tetap dalam lingkup sampai perintah selesai. Tugas panggilan tidak boleh menggunakan kembali struktur perintah apa pun sebelum pemanggilan callback penyelesaian. Perhatikan bahwa karena callback penyelesaian dipanggil oleh Agen MQTT, callback akan berjalan dengan konteks thread tugas agen, bukan tugas yang membuat perintah. Mekanisme komunikasi antar-proses, seperti pemberitahuan tugas atau antrian, dapat digunakan untuk memberi sinyal tugas panggilan penyelesaian perintah.

Menjalankan loop perintah

Perintah diproses terus menerus di`MQTTAgent_CommandLoop()`. Jika tidak ada perintah yang akan diproses, loop akan menunggu maksimum satu `MQTT_AGENT_MAX_EVENT_QUEUE_WAIT_TIME` untuk ditambahkan ke antrian, dan, jika tidak ada perintah yang ditambahkan, itu akan menjalankan satu iterasi. `MQTT_ProcessLoop()` Ini memastikan bahwa MQTT Keep-Alive dikelola, dan bahwa setiap publikasi yang masuk diterima bahkan ketika tidak ada perintah dalam antrian.

Fungsi loop perintah akan kembali karena alasan berikut:

- Perintah mengembalikan kode status apa pun selain itu`MQTTSuccess`. Status kesalahan dikembalikan oleh loop perintah, sehingga Anda dapat memutuskan bagaimana menanganinya. Dalam demo ini, koneksi TCP dibangun kembali, dan upaya penyambungan kembali dilakukan. Jika ada kesalahan, koneksi ulang dapat terjadi di latar belakang tanpa intervensi dari tugas lain menggunakan MQTT.
- Perintah putuskan sambungan (dari`MQTTAgent_Disconnect`) diproses. Loop perintah keluar sehingga TCP dapat terputus.
- Perintah terminate (from`MQTTAgent_Terminate`) diproses. Perintah ini juga menandai perintah apa pun yang masih dalam antrian atau menunggu paket pengakuan sebagai kesalahan, dengan kode pengembalian. `MQTTRecvFailed`

Manajer berlangganan

Karena demo menggunakan banyak topik, manajer berlangganan adalah cara mudah untuk mengaitkan topik berlangganan dengan panggilan balik atau tugas unik. Manajer berlangganan dalam demo ini adalah single-threaded, jadi tidak boleh digunakan oleh banyak tugas secara bersamaan. Dalam demo ini, fungsi pengelola langganan hanya dipanggil dari fungsi panggilan balik yang diteruskan ke agen MQTT, dan dijalankan hanya dengan konteks thread tugas agen.

Tugas berlangganan-terbitkan sederhana

Setiap instance prvSimpleSubscribePublishTaskmembuat langganan ke topik MQTT, dan membuat operasi publikasi untuk topik itu. Untuk mendemonstrasikan beberapa jenis publikasi, tugas bernomor genap menggunakan QoS 0 (yang selesai setelah paket publikasi dikirim) dan tugas ganjil menggunakan QoS 1 (yang selesai setelah menerima paket PUBACK).

Over-the-air memperbarui aplikasi demo

FreeRTOS menyertakan aplikasi demo yang menunjukkan fungsionalitas perpustakaan (OTA) over-the-air. Aplikasi demo OTA terletak di *freertos*`/demos/ota/ota_demo_core_mqtt/ota_demo_core_mqtt.c` atau *freertos*`/demos/ota/ota_demo_core_http/ota_demo_core_http.c` file.

Aplikasi demo OTA melakukan hal berikut:

1. Menginisialisasi tumpukan jaringan FreeRTOS dan kumpulan buffer MQTT.
2. Membuat tugas untuk menggunakan pustaka OTA menggunakan`vRunOTAUpdateDemo()`.
3. Membuat klien MQTT menggunakan. `_establishMqttConnection()`
4. Terhubung ke broker AWS IoT MQTT menggunakan `IotMqtt_Connect()` dan mendaftarkan panggilan balik pemutusan MQTT:. `prvNetworkDisconnectCallback`
5. Panggilan `OTA_AgentInit()` untuk membuat tugas OTA dan mendaftarkan panggilan balik yang akan digunakan saat tugas OTA selesai.
6. Menggunakan kembali koneksi MQTT dengan `xOTAConnectionCtx.pvControlClient = _mqttConnection;`
7. Jika MQTT terputus, aplikasi menangguhkan agen OTA, mencoba menyambung kembali menggunakan penundaan eksponensial dengan jitter, dan kemudian melanjutkan agen OTA.

Sebelum Anda dapat menggunakan pembaruan OTA, lengkapi semua prasyarat di Pembaruan FreeRTOS Over-the-Air

Setelah Anda menyelesaikan pengaturan untuk pembaruan OTA, unduh, buat, flash, dan jalankan demo FreeRTOS OTA pada platform yang mendukung fungsionalitas OTA. Instruksi demo khusus perangkat tersedia untuk perangkat berkualifikasi Freertos berikut:

- Texas Instruments CC322 0SF-LAUNCHXL
- Keingintahuan PIC32 Microchip MZEF

- Espressif ESP32
- Unduh, buat, flash, dan jalankan demo FreerTOS OTA di Renesas N RX65

Setelah Anda membangun, mem-flash, dan menjalankan aplikasi demo OTA di perangkat Anda, Anda dapat menggunakan AWS IoT konsol atau AWS CLI untuk membuat pekerjaan pembaruan OTA. Setelah Anda membuat pekerjaan pembaruan OTA, sambungkan emulator terminal untuk melihat kemajuan pembaruan OTA. Catat kesalahan apa pun yang dihasilkan selama proses berlangsung.

Pekerjaan pembaruan OTA yang berhasil menampilkan output seperti berikut ini. Beberapa baris dalam contoh ini telah dihapus dari daftar untuk singkatnya.

```
    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.
```

Over-the-air konfigurasi demo

Konfigurasi demo OTA adalah opsi konfigurasi khusus demo yang disediakan di. `aws_iot_ota_update_demo.c` Konfigurasi ini berbeda dari konfigurasi pustaka OTA yang disediakan dalam file konfigurasi pustaka OTA.

OTA_DEMO_KEEP_ALIVE_SECONDS

Untuk klien MQTT, konfigurasi ini adalah interval waktu maksimum yang dapat berlalu antara menyelesaikan transmisi satu paket kontrol dan mulai mengirim paket berikutnya. Dengan tidak adanya paket kontrol, PINGREQ dikirim. Pialang harus memutuskan koneksi klien yang tidak mengirim pesan atau paket PINGREQ dalam satu setengah kali interval tetap hidup ini. Konfigurasi ini harus disesuaikan berdasarkan persyaratan aplikasi.

OTA_DEMO_CONN_RETRY_BASE_INTERVAL_SECONDS

Interval dasar, dalam hitungan detik, sebelum mencoba kembali koneksi jaringan. Demo OTA akan mencoba menyambung kembali setelah interval waktu dasar ini. Interval digandakan setelah setiap upaya gagal. Penundaan acak, hingga maksimum penundaan dasar ini, juga ditambahkan ke interval.

OTA_DEMO_CONN_RETRY_MAX_INTERVAL_SECONDS

Interval maksimum, dalam hitungan detik, sebelum mencoba kembali koneksi jaringan. Penundaan koneksi kembali digandakan pada setiap upaya yang gagal, tetapi hanya dapat mencapai nilai maksimum ini, ditambah jitter dengan interval yang sama.

Unduh, buat, flash, dan jalankan demo FreerTOS OTA di Texas Instruments 0SF-LAUNCHXL CC322

> **Important**
>
> Integrasi referensi ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-FreerTOS yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Untuk mengunduh FreeRTOS dan kode demo OTA

- Anda dapat mengunduh kode sumber di GitHub situs di https://github.com/FreeRTOS/FreeRTOS.

Untuk membangun aplikasi demo

1. Ikuti petunjuk Memulai dengan FreeRTOS untuk mengimpor `aws_demos` proyek ke Code Composer Studio, konfigurasikan AWS IoT titik akhir Anda, SSID Wi-Fi dan kata sandi Anda, serta kunci pribadi dan sertifikat untuk papan Anda.
2. Buka*freertos*`/vendors/`*vendor*`/boards/`*board*`/aws_demos/config_files/aws_demo_config.h`, beri komentar`#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED`, dan tentukan `CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED` atau`CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED`.
3. Bangun solusinya dan pastikan itu dibangun tanpa kesalahan.
4. Mulai emulator terminal dan gunakan pengaturan berikut untuk terhubung ke papan Anda:
   - Tingkat baud: 115200
   - Bit data: 8
   - Paritas: Tidak ada
   - Hentikan bit: 1
5. Jalankan proyek di papan Anda untuk mengonfirmasi bahwa proyek tersebut dapat terhubung ke Wi-Fi dan broker AWS IoT pesan MQTT.

Saat dijalankan, emulator terminal akan menampilkan teks seperti berikut:

```
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
```

Unduh, buat, flash, dan jalankan demo FreerTOS OTA di Microchip Curiosity MZEF PIC32

Important

Integrasi referensi ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-FreerTOS yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Note

Sesuai dengan Microchip, kami menghapus Curiosity PIC32 MZEF (DM320104) dari cabang utama repositori Freertos Reference Integration dan tidak akan lagi membawanya dalam rilis baru. Microchip telah mengeluarkan pemberitahuan resmi bahwa PIC32 MZEF (DM320104) tidak lagi direkomendasikan untuk desain baru. Proyek PIC32 MZEF dan kode sumber masih dapat diakses melalui tag rilis sebelumnya. Microchip merekomendasikan agar pelanggan

menggunakan papan Pengembangan Curiosity PIC32 MZ-EF-2.0 (DM320209) untuk desain baru. PIC32MZv1 Platform ini masih dapat ditemukan di v202012.00 dari repositori Integrasi Referensi FreeRTOS. Namun, platform ini tidak lagi didukung oleh Referensi FreeRTOS v202107.00.

Untuk mengunduh kode demo FreeRTOS OTA

- Anda dapat mengunduh kode sumber di GitHub situs di https://github.com/FreeRTOS/FreeRTOS.

Untuk membangun aplikasi demo pembaruan OTA

1. Ikuti petunjuk Memulai dengan FreeRTOS untuk mengimpor `aws_demos` proyek ke MPLAB X IDE, konfigurasikan AWS IoT titik akhir Anda, SSID Wi-Fi dan kata sandi Anda, serta kunci pribadi dan sertifikat untuk papan Anda.
2. Buka `vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/ota_demo_config.h` file, dan masukkan sertifikat Anda.

   ```
   [] = "your-certificate-key";
   ```

3. Tempel konten sertifikat penandatanganan kode Anda di sini:

   ```
   #define otapalconfigCODE_SIGNING_CERTIFICATE [] = "your-certificate-key";
   ```

   Ikuti format yang sama dengan `aws_clientcredential_keys.h` -- setiap baris harus diakhiri dengan karakter baris baru ('\n') dan diapit tanda kutip.

   Misalnya, sertifikat Anda akan terlihat mirip dengan yang berikut ini:

   ```
   "-----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. Instal Python 3 atau yang lebih baru.
5. Instal pyOpenSSL dengan menjalankan`pip install pyopenssl`.
6. Salin sertifikat penandatanganan kode Anda dalam format.pem di jalur. `demos/ota/bootloader/utility/codesigner_cert_utility/` Ganti nama file `aws_ota_codesigner_certificate.pem` sertifikat.
7. Buka*`freertos`*`/vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/aws_demo_config.h`, beri komentar`#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED`, dan tentukan `CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED` atau`CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED`.
8. Bangun solusinya dan pastikan itu dibangun tanpa kesalahan.
9. Mulai emulator terminal dan gunakan pengaturan berikut untuk terhubung ke papan Anda:
   - Tingkat baud: 115200
   - Bit data: 8
   - Paritas: Tidak ada
   - Hentikan bit: 1
10. Cabut debugger dari papan Anda dan jalankan proyek di papan Anda untuk mengonfirmasi bahwa debugger dapat terhubung ke Wi-Fi dan broker pesan AWS IoT MQTT.

Ketika Anda menjalankan proyek, MPLAB X IDE harus membuka jendela output. Pastikan ICD4tab dipilih. Anda akan melihat output berikut.

```
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
```

Emulator terminal harus menampilkan teks seperti berikut:

```
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
```

Output ini menunjukkan Microchip Curiosity PIC32 MZEF dapat terhubung AWS IoT dan berlangganan topik MQTT yang diperlukan untuk pembaruan OTA. `Missing job parameter`Pesan diharapkan karena tidak ada pekerjaan pembaruan OTA yang tertunda.

Unduh, buat, flash, dan jalankan demo FreeRTOS OTA di Espressif ESP32

> **Important**
>
> Integrasi referensi ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-FreerTOS yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

1. Unduh sumber FreeRTOS dari. GitHub Lihat file README.md untuk instruksi. Buat proyek di IDE Anda yang mencakup semua sumber dan pustaka yang diperlukan.
2. Ikuti petunjuk di Memulai dengan Espressif untuk menyiapkan toolchain berbasis GCC yang diperlukan.

3. Buka*freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h, beri komentar#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED, dan tentukan CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED atauCONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED.
4. Bangun proyek demo dengan berjalan make di vendors/espressif/boards/esp32/aws_demos direktori. Anda dapat mem-flash program demo dan memverifikasi outputnya dengan menjalankanmake flash monitor, seperti yang dijelaskan dalam Memulai dengan Espressif.
5. Sebelum menjalankan demo Pembaruan OTA:
   - Buka*freertos*/vendors/*vendor*/boards/*board*/aws_demos/config_files/aws_demo_config.h, beri komentar#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED, dan tentukan CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED atauCONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED.
   - Bukavendors/*vendor*/boards/*board*/aws_demos/config_files/ota_demo_config.h, dan salin sertifikat penandatanganan kode SHA-256/ECDSA Anda di:

```
#define otapalconfigCODE_SIGNING_CERTIFICATE [] = "your-certificate-key";
```

Unduh, buat, flash, dan jalankan demo FreerTOS OTA di Renesas N RX65

> **Important**
>
> Integrasi referensi ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-Freertos yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Bab ini menunjukkan cara mengunduh, membangun, mem-flash, dan menjalankan aplikasi demo FreerTOS OTA di Renesas N. RX65

Topik

- Siapkan lingkungan operasi Anda

- Siapkan AWS sumber daya Anda
- Impor, konfigurasikan file header dan buat aws_demos dan boot_loader

Siapkan lingkungan operasi Anda

Prosedur di bagian ini menggunakan lingkungan berikut:

- IDE: ^{2} studio 7.8.0, dan ^{2} studio 2020-07
- Rantai Alat: CCRX Compiler v3.0.1
- Perangkat target: RSKRX65 N-2MB
- Debugger: E $^{2, E 2}$ Lite emulator
- Perangkat lunak: Renesas Flash Programmer, Renesas Secure Flash Programmer.exe, Tera Term

Untuk mengatur perangkat keras Anda

1. Hubungkan emulator E ^{2} Lite dan port serial USB ke papan RX65 N dan PC Anda.
2. Hubungkan sumber daya ke RX65 N.

Siapkan AWS sumber daya Anda

1. Untuk menjalankan demo FreeRTOS, Anda harus memiliki akun dengan pengguna IAM AWS yang memiliki izin untuk mengakses layanan. AWS IoT Jika Anda belum melakukannya, ikuti langkah-langkahnyaMenyiapkan AWS akun dan izin.
2. Untuk mengatur pembaruan OTA, ikuti langkah-langkahnyaPrasyarat pembaruan OTA. Secara khusus, ikuti langkah-langkahnyaPrasyarat untuk pembaruan OTA menggunakan MQTT.
3. Buka konsol AWS IoT.
4. Di panel navigasi kiri, pilih Kelola, lalu pilih Things to create a thing.

   Sesuatu adalah representasi dari perangkat atau entitas logis di AWS IoT. Ini bisa berupa perangkat fisik atau sensor (misalnya, bola lampu atau sakelar di dinding). Ini juga bisa menjadi entitas logis seperti contoh aplikasi atau entitas fisik yang tidak terhubung AWS IoT, tetapi terkait dengan perangkat yang melakukannya (misalnya, mobil yang memiliki sensor mesin atau panel kontrol). AWS IoT menyediakan registri sesuatu yang membantu Anda mengelola barang-barang Anda.

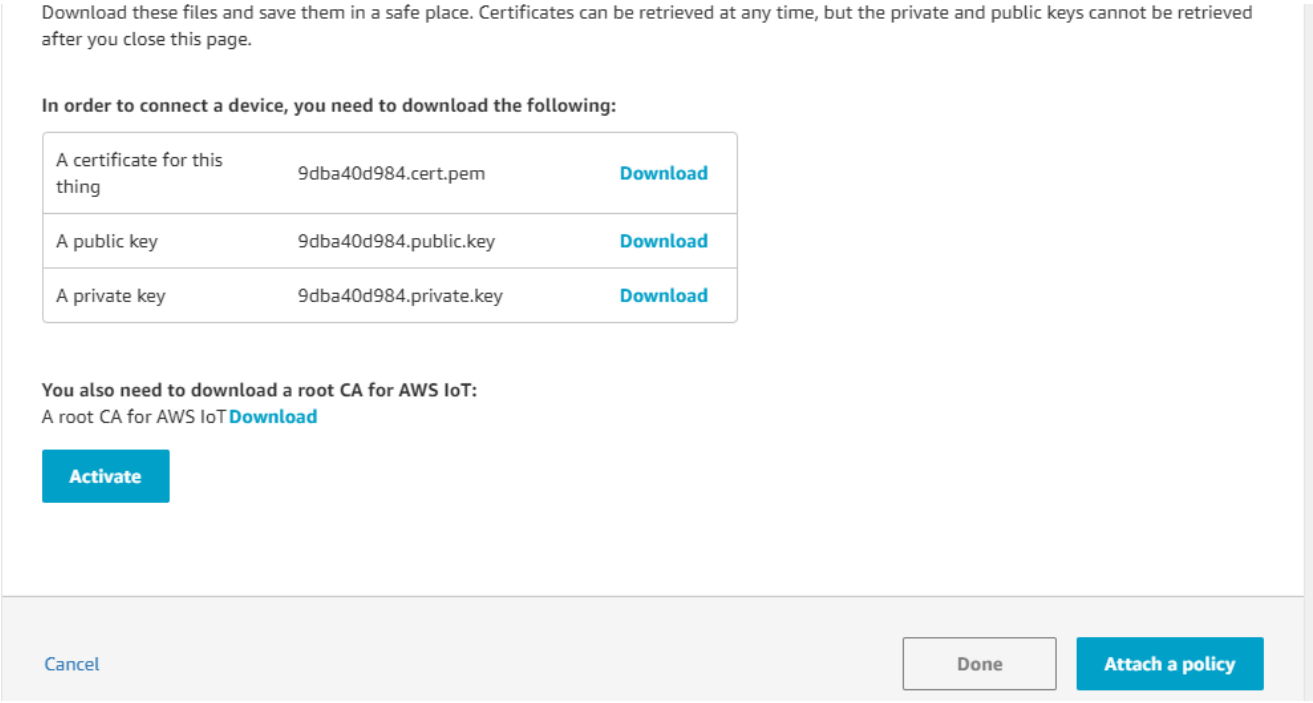
   a. Pilih Buat, lalu pilih Buat satu hal.

b. Masukkan Nama untuk barang Anda, lalu pilih Berikutnya.

c. Pilih Buat sertifikat.

d. Unduh tiga file yang dibuat dan kemudian pilih Aktifkan.

e. Pilih Lampirkan kebijakan.



f. Pilih kebijakan yang Anda buatKebijakan perangkat.

Setiap perangkat yang menerima pembaruan OTA menggunakan MQTT harus terdaftar sebagai sesuatu AWS IoT dan benda tersebut harus memiliki kebijakan terlampir seperti yang terdaftar. Anda dapat menemukan informasi selengkapnya tentang item dalam `"Action"` dan `"Resource"` objek di Tindakan Kebijakan AWS IoT Inti dan Sumber Daya Tindakan AWS IoT Inti.

Catatan

- `iot:Connect`Izin memungkinkan perangkat Anda terhubung AWS IoT melalui MQTT.
- `iot:Publish`Izin `iot:Subscribe` dan pada topik AWS IoT pekerjaan (`.../jobs/*`) memungkinkan perangkat yang terhubung untuk menerima pemberitahuan pekerjaan dan dokumen pekerjaan, dan untuk mempublikasikan status penyelesaian pelaksanaan pekerjaan.
- `iot:Publish`Izin `iot:Subscribe` dan tentang topik aliran AWS IoT OTA (`.../streams/*`) memungkinkan perangkat yang terhubung untuk mengambil data

pembaruan OTA dari. AWS IoT Izin ini diperlukan untuk melakukan pembaruan firmware melalui MQTT.

- `iot:Receive`Izin memungkinkan AWS IoT Core untuk mempublikasikan pesan tentang topik tersebut ke perangkat yang terhubung. Izin ini diperiksa pada setiap pengiriman pesan MQTT. Anda dapat menggunakan izin ini untuk mencabut akses ke klien yang saat ini berlangganan topik.

5. Untuk membuat profil penandatanganan kode dan mendaftarkan sertifikat penandatanganan kode pada. AWS

   a. Untuk membuat kunci dan sertifikasi, lihat bagian 7.3 “Menghasilkan ECDSA- Pasangan SHA256 Kunci dengan OpenSSL” di Kebijakan Desain Pembaruan Firmware Renesas MCU.

   b. Buka konsol AWS IoT. Di panel navigasi kiri, pilih Kelola, lalu Pekerjaan. Pilih Create a job lalu Create OTA update Job.

   c. Di bawah Pilih perangkat yang akan diperbarui pilih Pilih lalu pilih hal yang Anda buat sebelumnya. Pilih Selanjutnya.

   d. Pada halaman pekerjaan Create a FreeRTOS OTA update, lakukan hal berikut:

      i. Untuk Pilih protokol untuk transfer gambar firmware, pilih MQTT.

      ii. Untuk Pilih dan tandatangani gambar firmware Anda, pilih Tanda tangani gambar firmware baru untuk saya.

      iii. Untuk profil penandatanganan kode, pilih Buat.

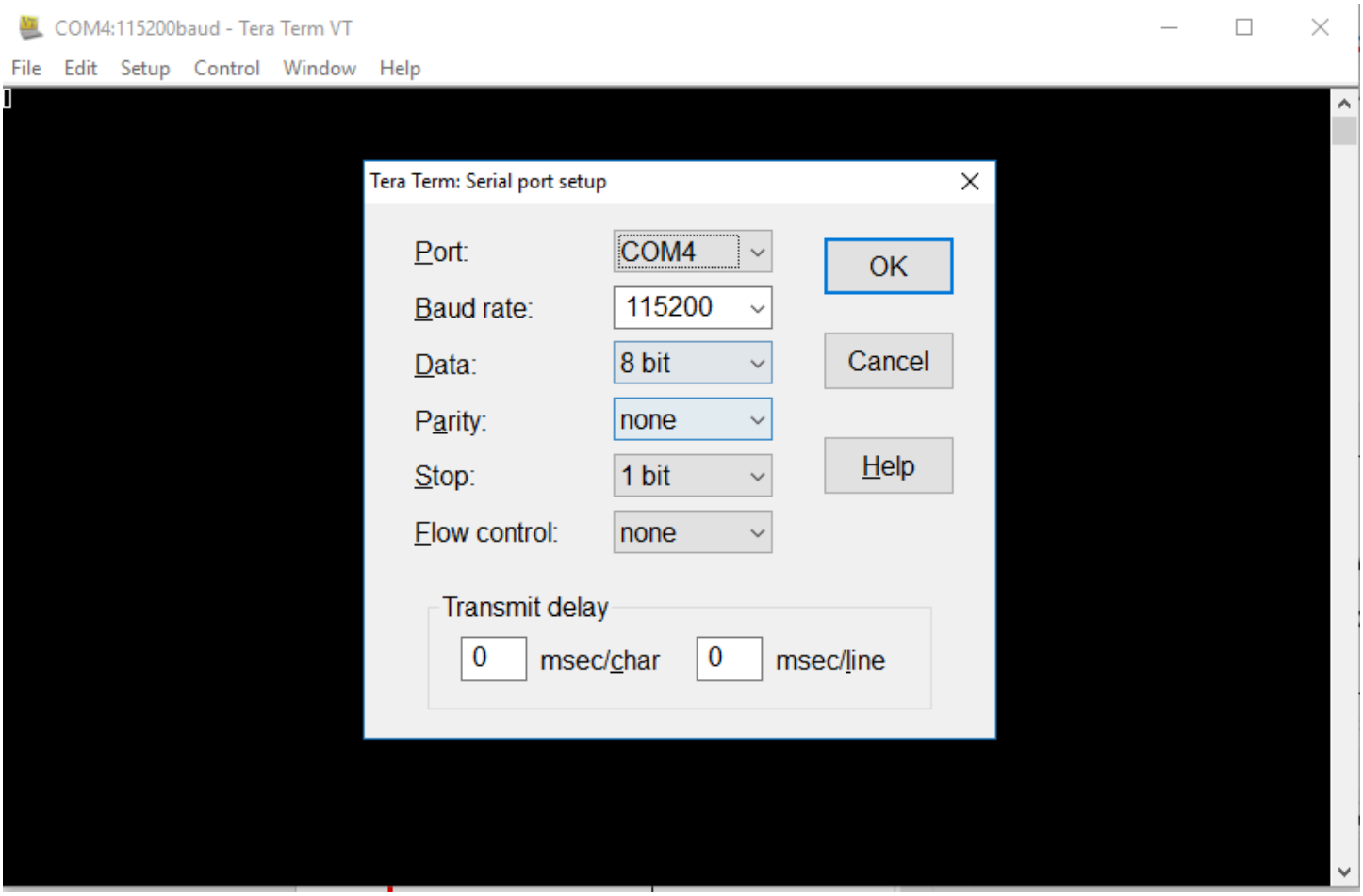
      iv. Di jendela Buat profil penandatanganan kode, masukkan nama Profil. Untuk platform perangkat keras perangkat pilih Windows Simulator. Untuk sertifikat penandatanganan kode pilih Impor.

      v. Telusuri untuk memilih sertifikat (`secp256r1.crt`), kunci privat sertifikat (`secp256r1.key`), dan rantai sertifikat (`ca.crt`).

      vi. Masukkan Pathname sertifikat penandatanganan kode di perangkat. Lalu pilih Buat.

6. Untuk memberikan akses ke penandatanganan kode AWS IoT, ikuti langkah-langkahnyaBerikan akses ke penandatanganan kode untuk AWS IoT.

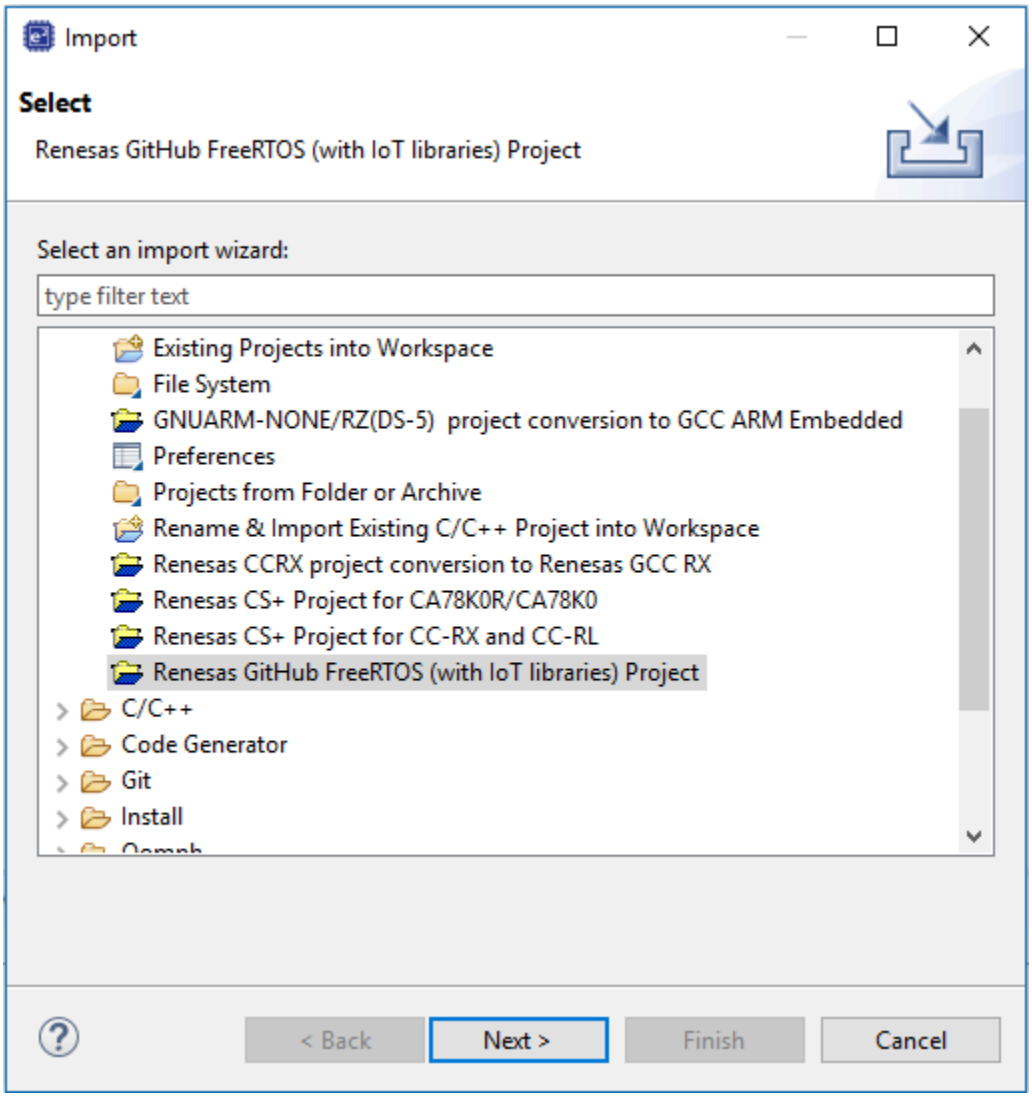
Jika Tera Term tidak diinstal pada PC Anda, Anda dapat mengunduhnya https://ttssh2.osdn.jp/index.htmldari.en dan mengaturnya seperti yang ditunjukkan di sini. Pastikan Anda mencolokkan port Serial USB dari perangkat Anda ke PC Anda.



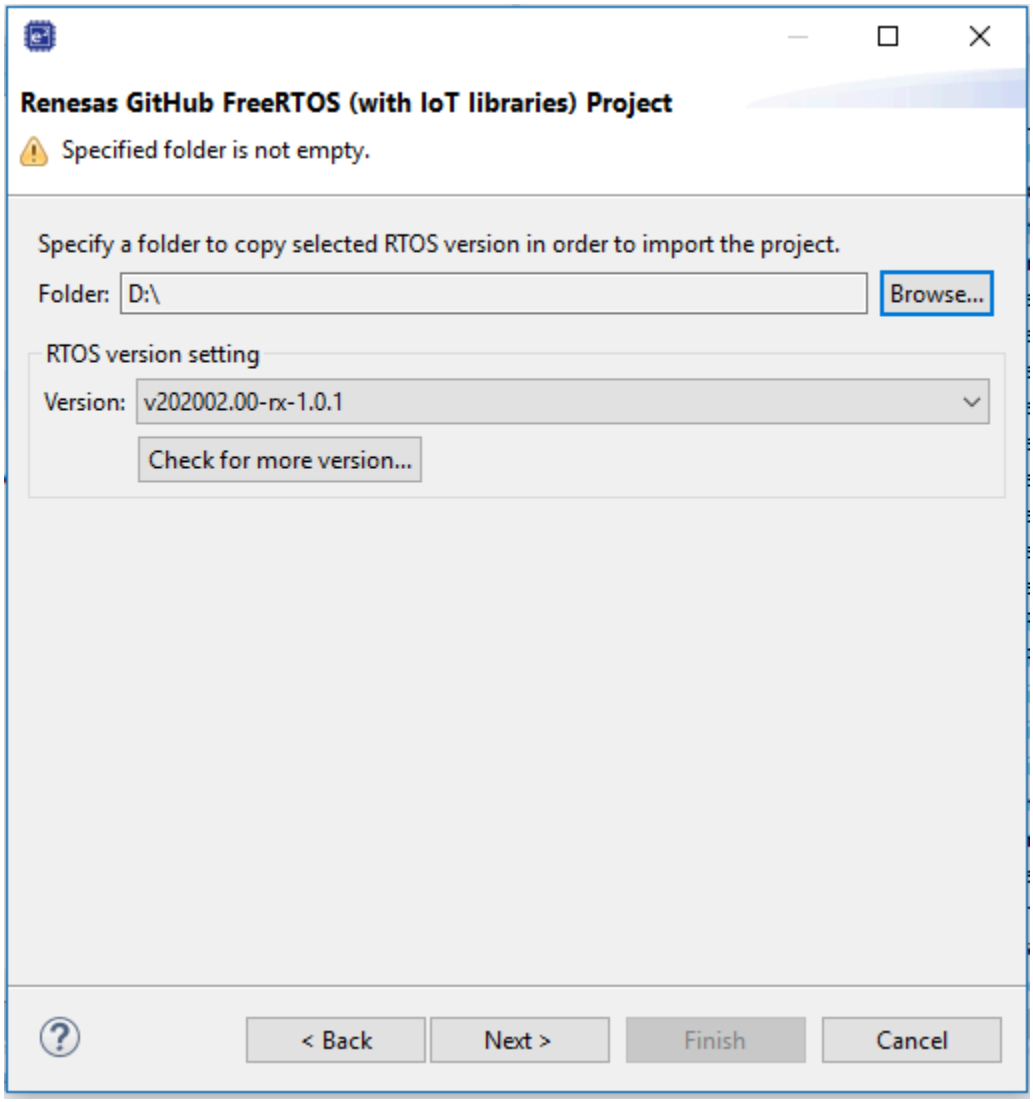
Impor, konfigurasikan file header dan buat aws_demos dan boot_loader

Untuk memulai, Anda memilih versi terbaru dari paket FreerTOS, dan ini akan diunduh GitHub dari dan diimpor secara otomatis ke proyek. Dengan cara ini Anda dapat fokus pada konfigurasi FreeRTOS dan menulis kode aplikasi.

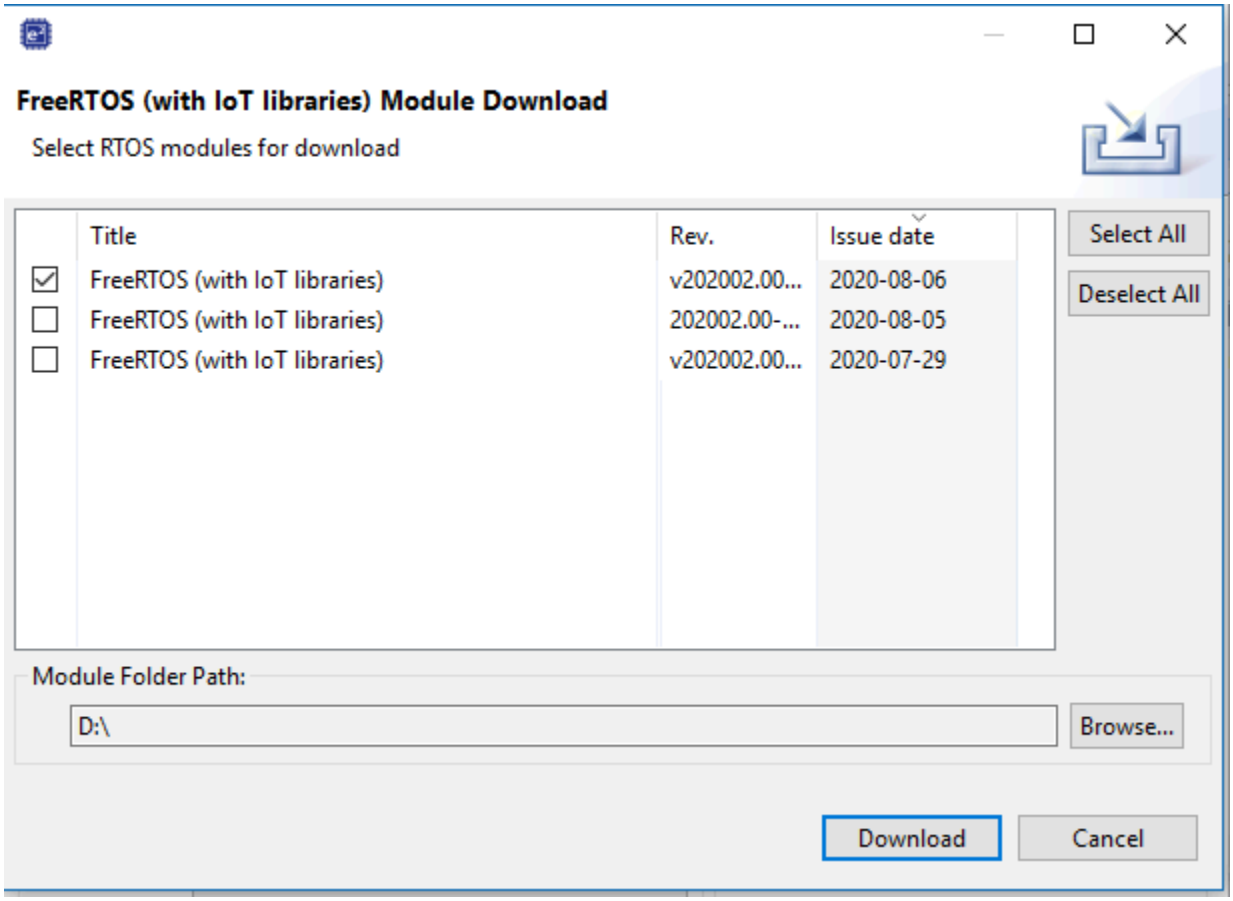
1. Luncurkan e ^{2} studio.
2. Pilih File, lalu pilih Impor...
3. Pilih Proyek Renesas FreeRTOS (dengan GitHub pustaka IoT).



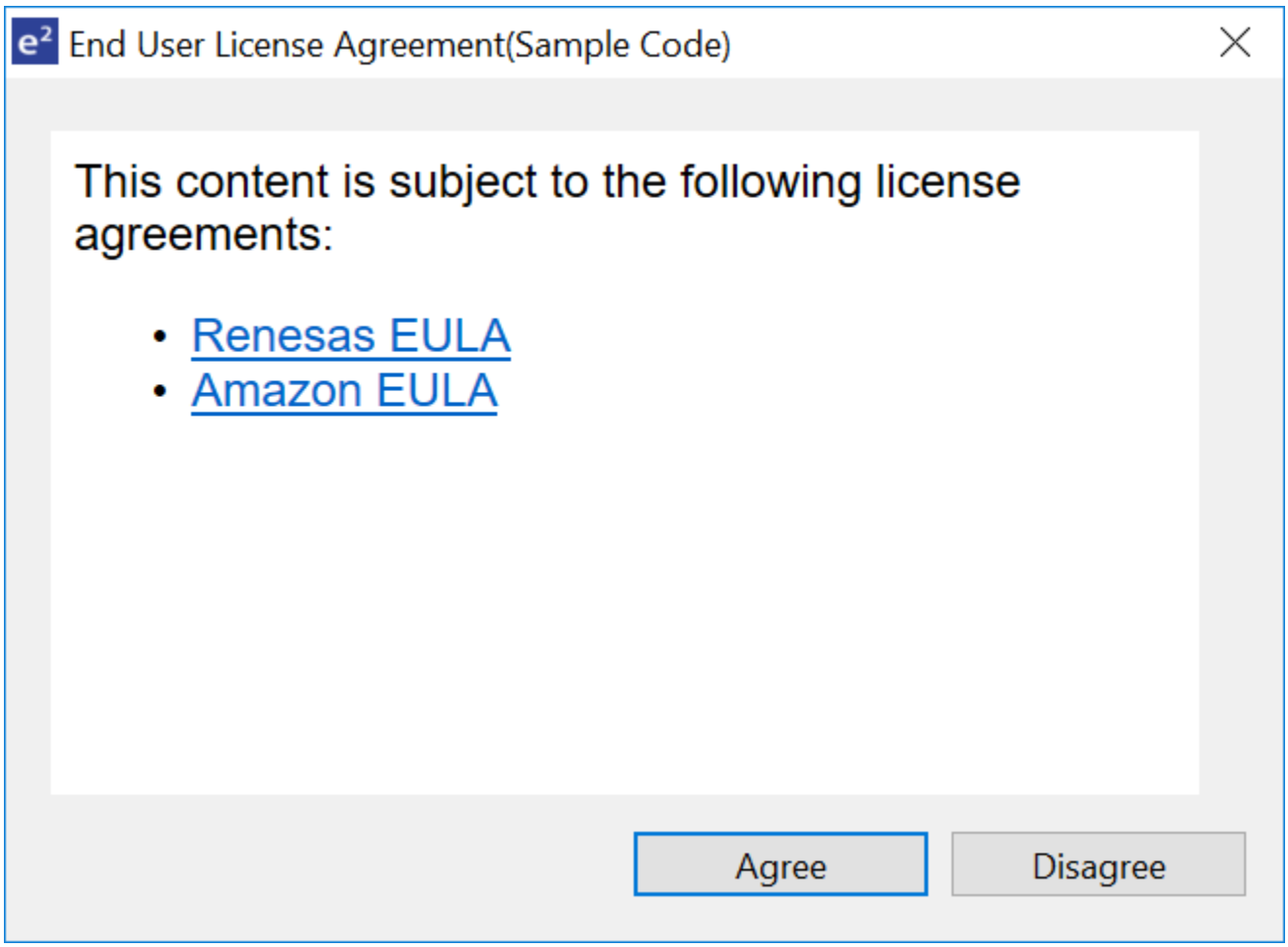
4. Pilih Periksa versi lainnya... untuk menampilkan kotak dialog unduhan.



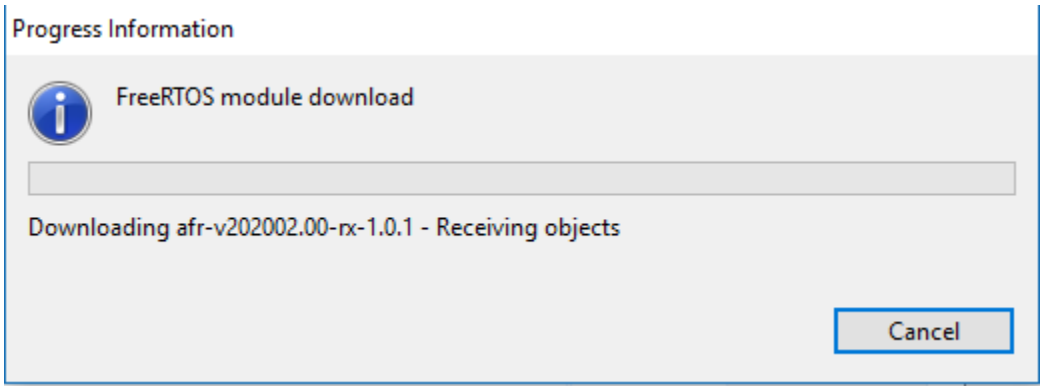
5. Pilih paket terbaru.



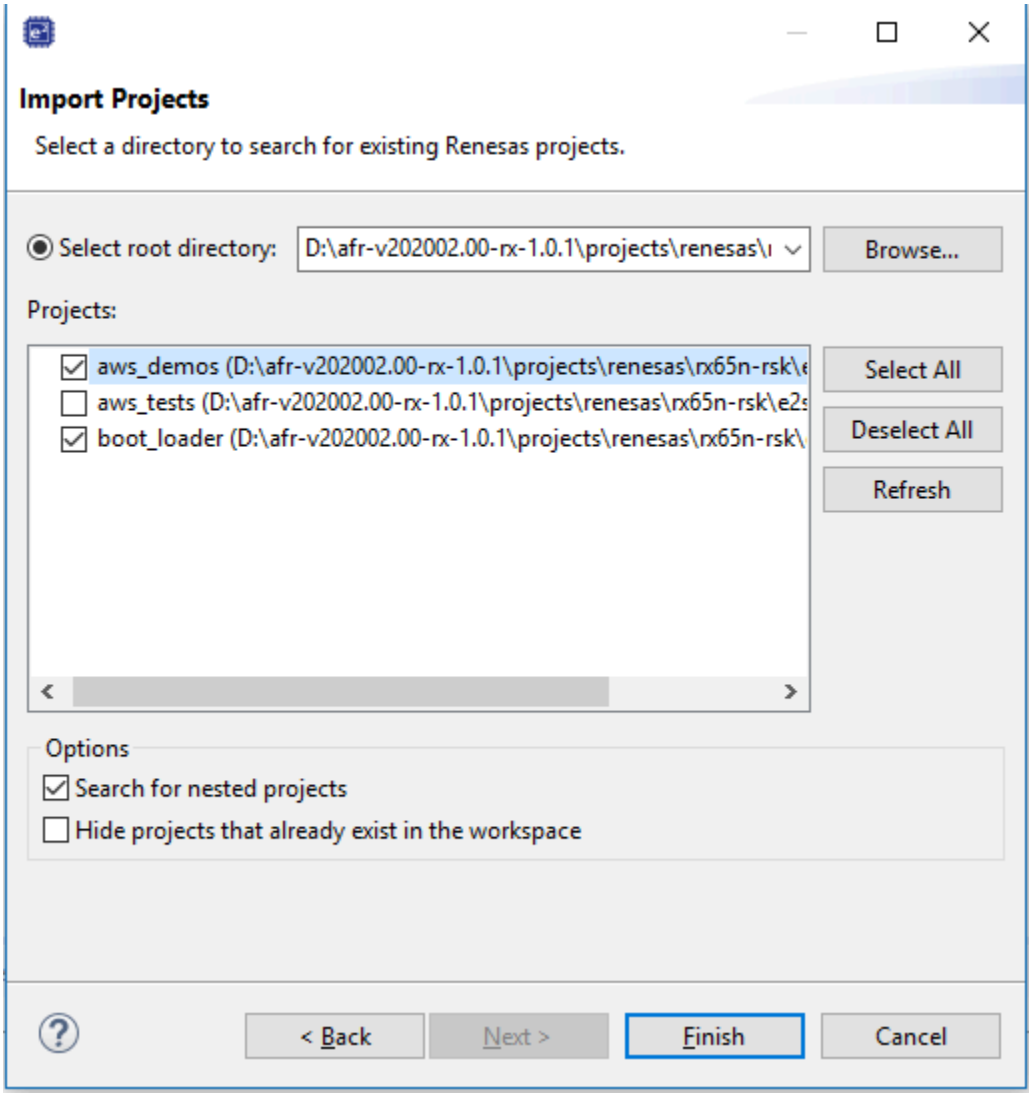
6. Pilih Setuju untuk menerima perjanjian lisensi pengguna akhir.



7. Tunggu hingga unduhan selesai.



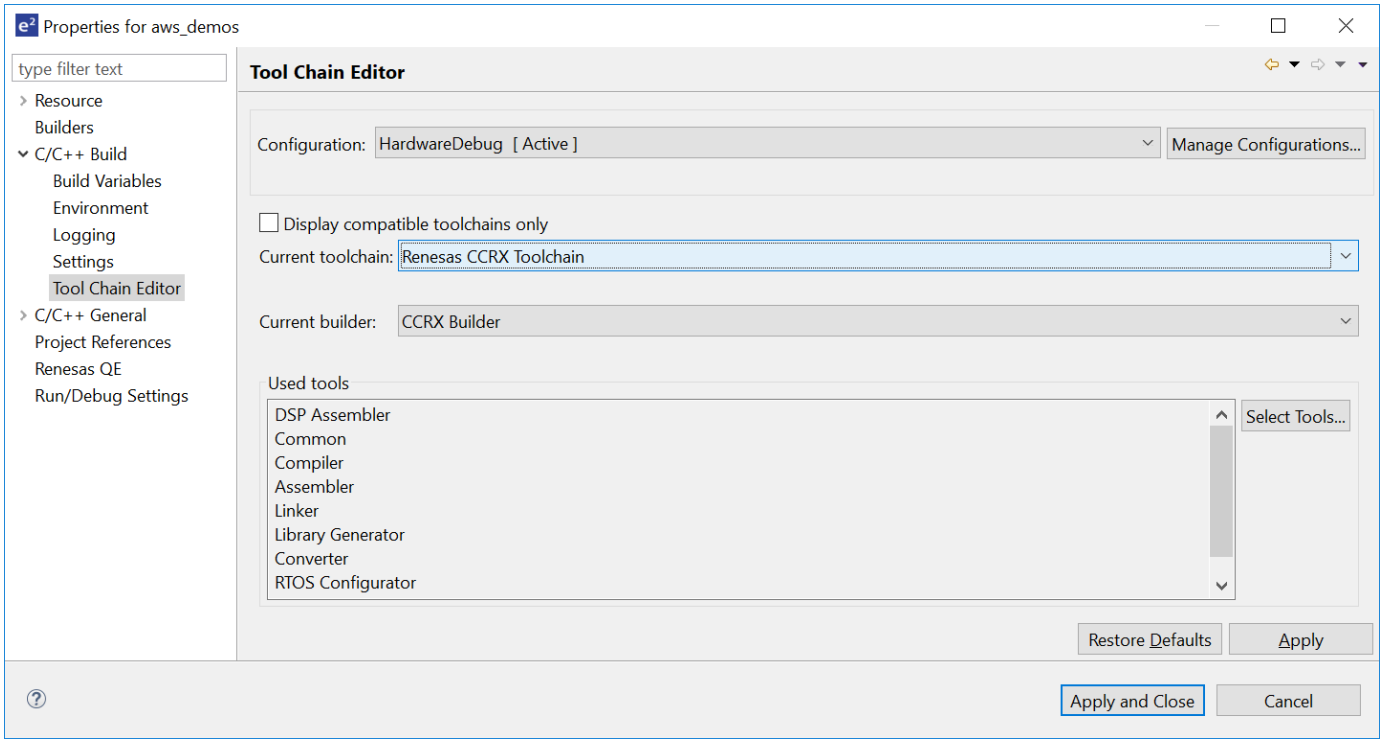
8. Pilih proyek aws_demos dan boot_loader, lalu pilih Selesai untuk mengimpornya.



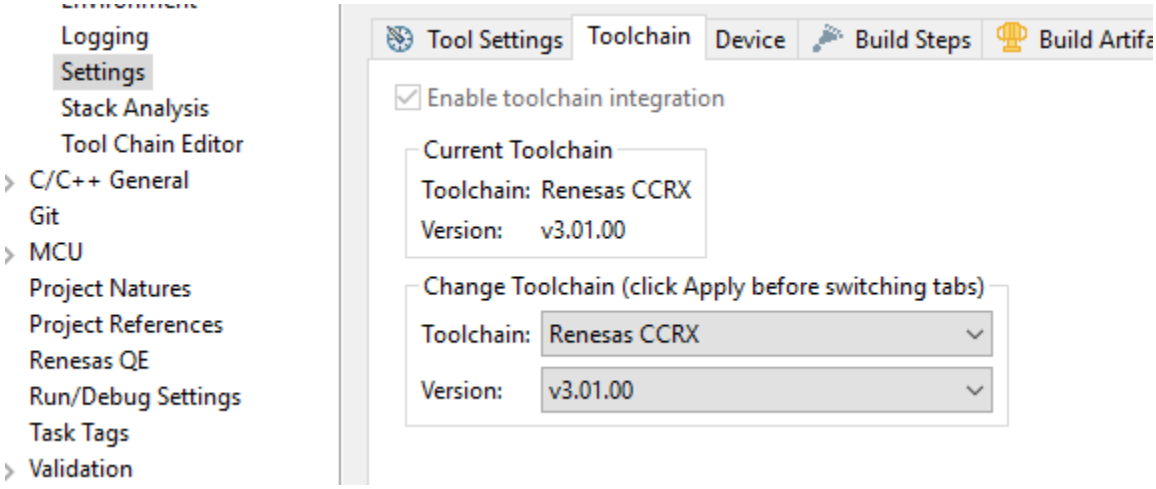
9. Untuk kedua proyek, buka properti proyek. Di panel navigasi, pilih Tool Chain Editor.

    a. Pilih toolchain saat ini.

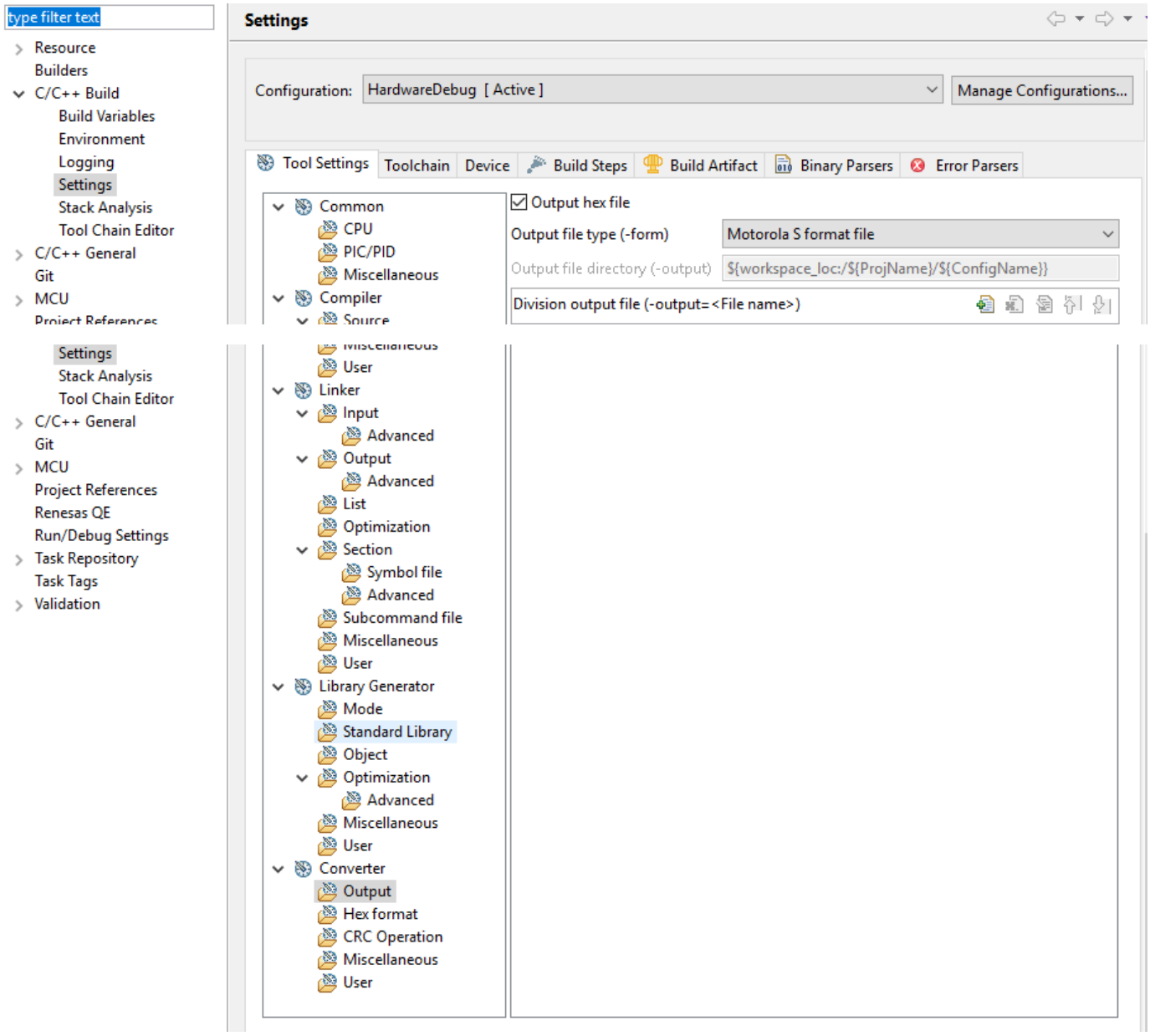
    b. Pilih pembangun saat ini.



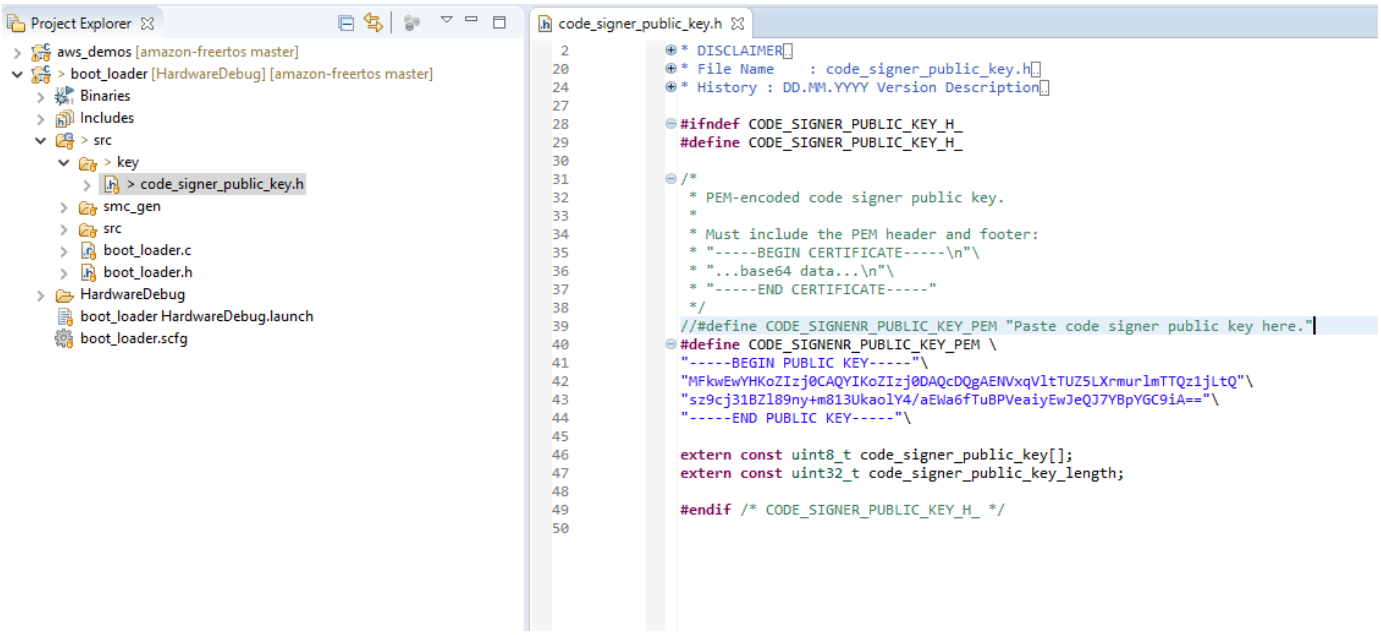
10. Pada panel navigasi, silakan pilih Pengaturan. Pilih tab Toolchain, lalu pilih Versi toolchain.



Pilih tab Pengaturan Alat, perluas Konverter dan kemudian pilih Output. Di jendela utama, pastikan Output hex file dipilih, dan kemudian pilih jenis file Output.



11. Dalam proyek bootloader, buka `projects\renesas\rx65n-rsk\e2studio\boot_loader\src\key\code_signer_public_key.h` dan masukkan kunci publik. Untuk informasi tentang cara membuat kunci publik, lihat Cara menerapkan FreeRTOS OTA dengan menggunakan Amazon Web Services RX65 di N dan bagian 7.3 “Menghasilkan ECDSA SHA256 - Pasangan Kunci dengan OpenSSL” di Kebijakan Desain Pembaruan Firmware MCU Renesas.



Kemudian bangun proyek untuk dibuat`boot_loader.mot`.

12. Buka `aws_demos` proyek.

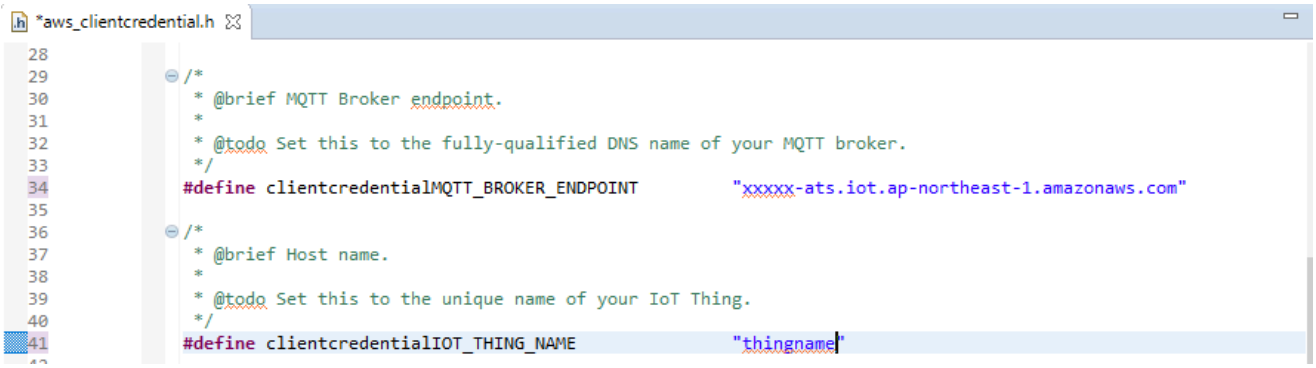
    a. Buka konsol AWS IoT.

    b. Pada panel navigasi kiri, pilih Pengaturan. Catat titik akhir kustom Anda di kotak teks titik akhir data Perangkat.

    c. Pilih Kelola, lalu pilih Things. Catat nama AWS IoT benda papan Anda.

    d. Dalam `aws_demos` proyek, buka `demos/include/aws_clientcredential.h` dan tentukan nilai-nilai berikut.

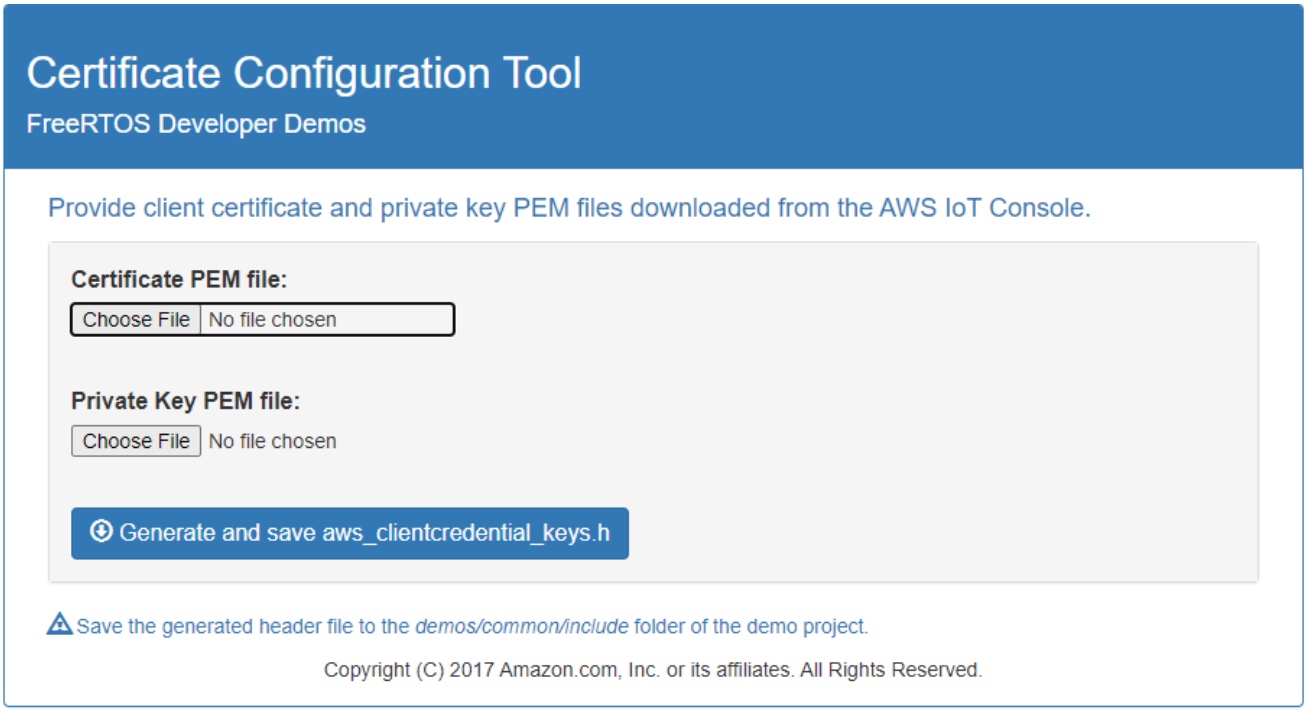
```
#define clientcredentialMQTT_BROKER_ENDPOINT[] = "Your AWS IoT endpoint";
#define clientcredentialIOT_THING_NAME "The AWS IoT thing name of your board"
```



    e. Buka file `tools/certificate_configuration/CertificateConfigurator.html`.

f. Impor file PEM sertifikat dan file PEM Kunci Pribadi yang Anda unduh sebelumnya.

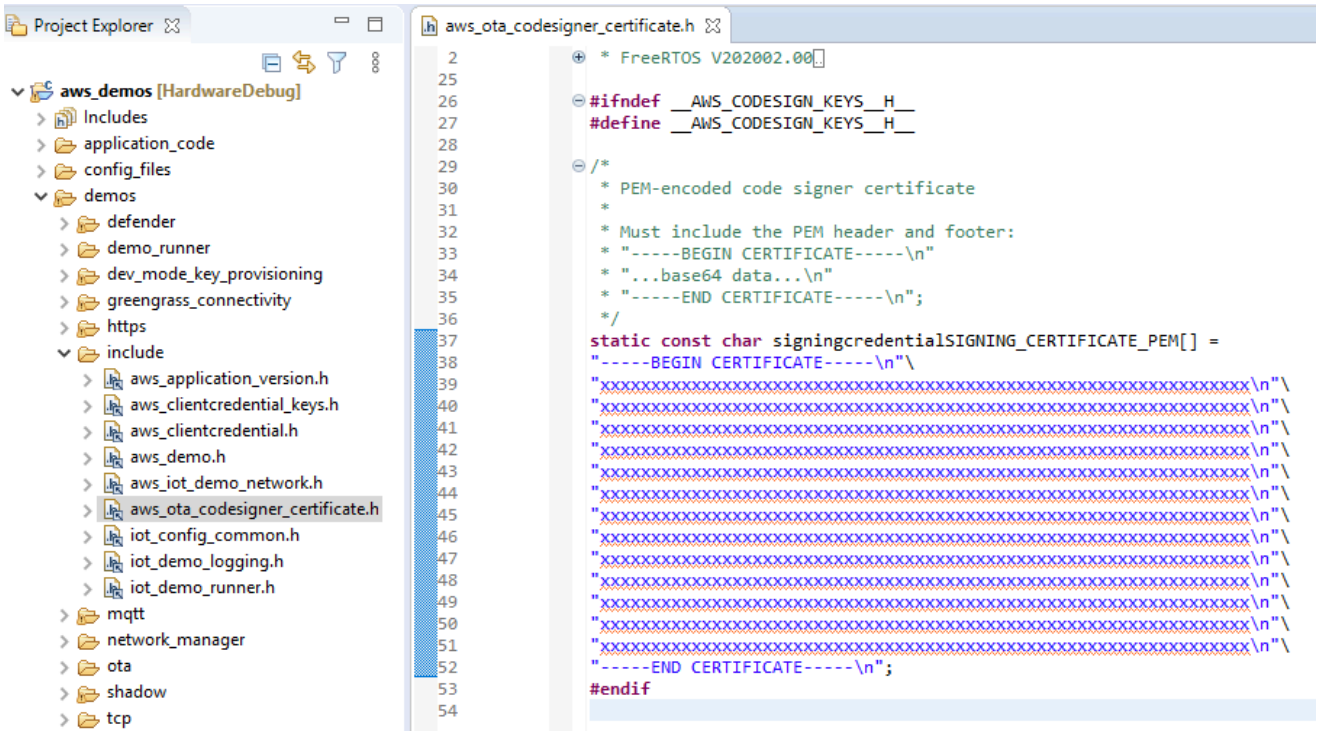
g. Pilih Generate dan save aws_clientcredential_keys.h dan ganti file ini di direktori. `demos/include/`



h. Buka `vendors/renesas/boards/rx65n-rsk/aws_demos/config_files/ota_demo_config.h` file, dan tentukan nilai-nilai ini.

```
#define otapalconfigCODE_SIGNING_CERTIFICATE [] = "your-certificate-key";
```

*your-certificate-key*Dimana nilai dari file tersebut`secp256r1.crt`. Ingatlah untuk menambahkan “\” setelah setiap baris dalam sertifikasi. Untuk informasi selengkapnya tentang membuat `secp256r1.crt` file, lihat Cara menerapkan FreerTOS OTA menggunakan Amazon Web Services RX65 di N dan bagian 7.3 “Menghasilkan ECDSA SHA256 - Pasangan Kunci dengan OpenSSL” di Kebijakan Desain Pembaruan Firmware Renesas MCU.



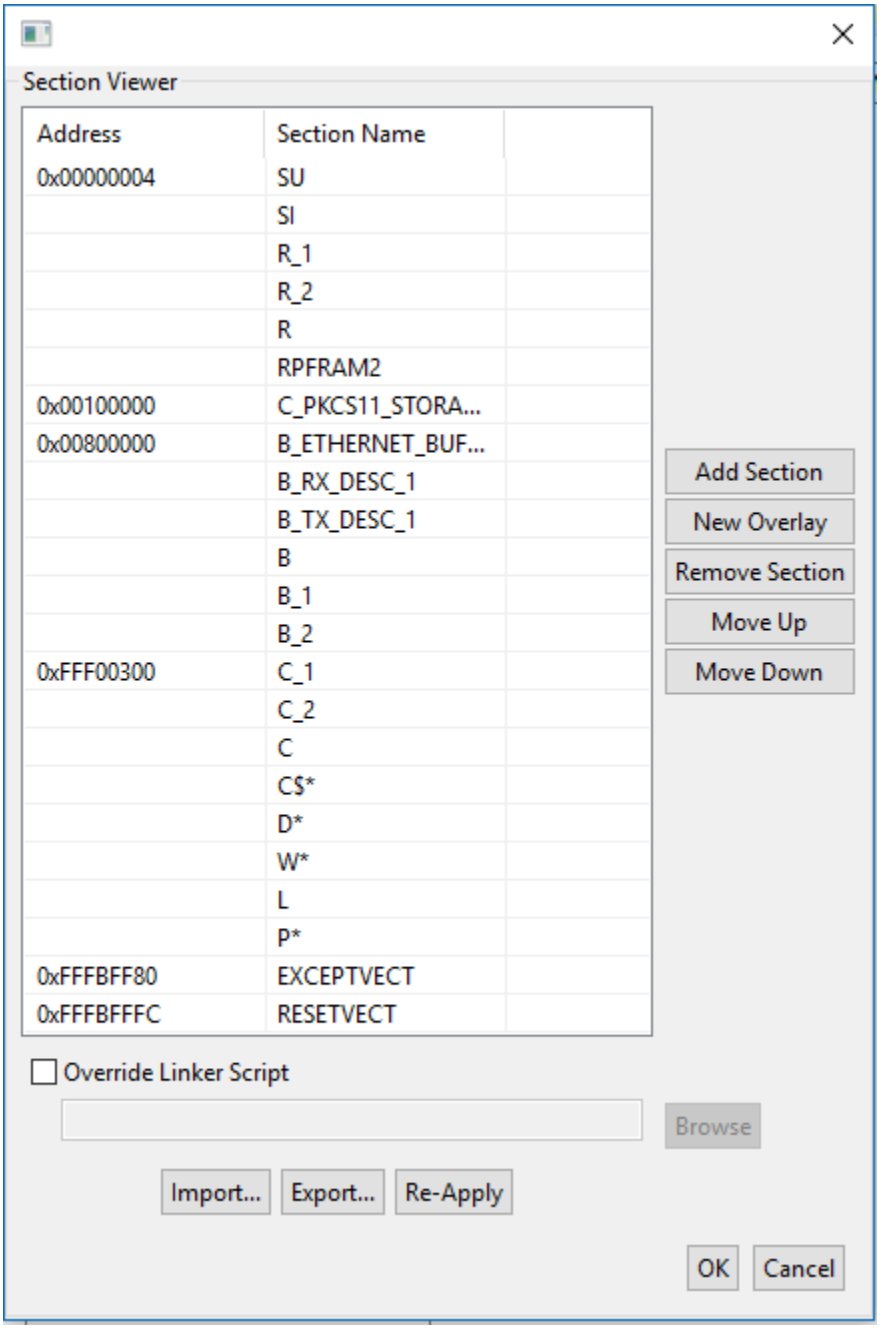
13. Tugas A: Instal versi awal firmware

    a. Buka `vendors/renesas/boards/board/aws_demos/config_files/aws_demo_config.h` file, beri komentar`#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED`, dan tentukan salah satu `CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED` atau`CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED`.

    b. Buka `demos/include/ aws_application_version.h` file, dan atur versi awal firmware ke`0.9.2`.

```
25
26  #ifndef _AWS_APPLICATION_VERSION_H_
27  #define _AWS_APPLICATION_VERSION_H_
28
29  #include "iot_appversion32.h"
30  extern const AppVersion32_t xAppFirmwareVersion;
31
32  #define APP_VERSION_MAJOR    0
33  #define APP_VERSION_MINOR    9
34  #define APP_VERSION_BUILD    2
35
36  #endif
37
```

    c. Ubah pengaturan berikut di Penampil Bagian.



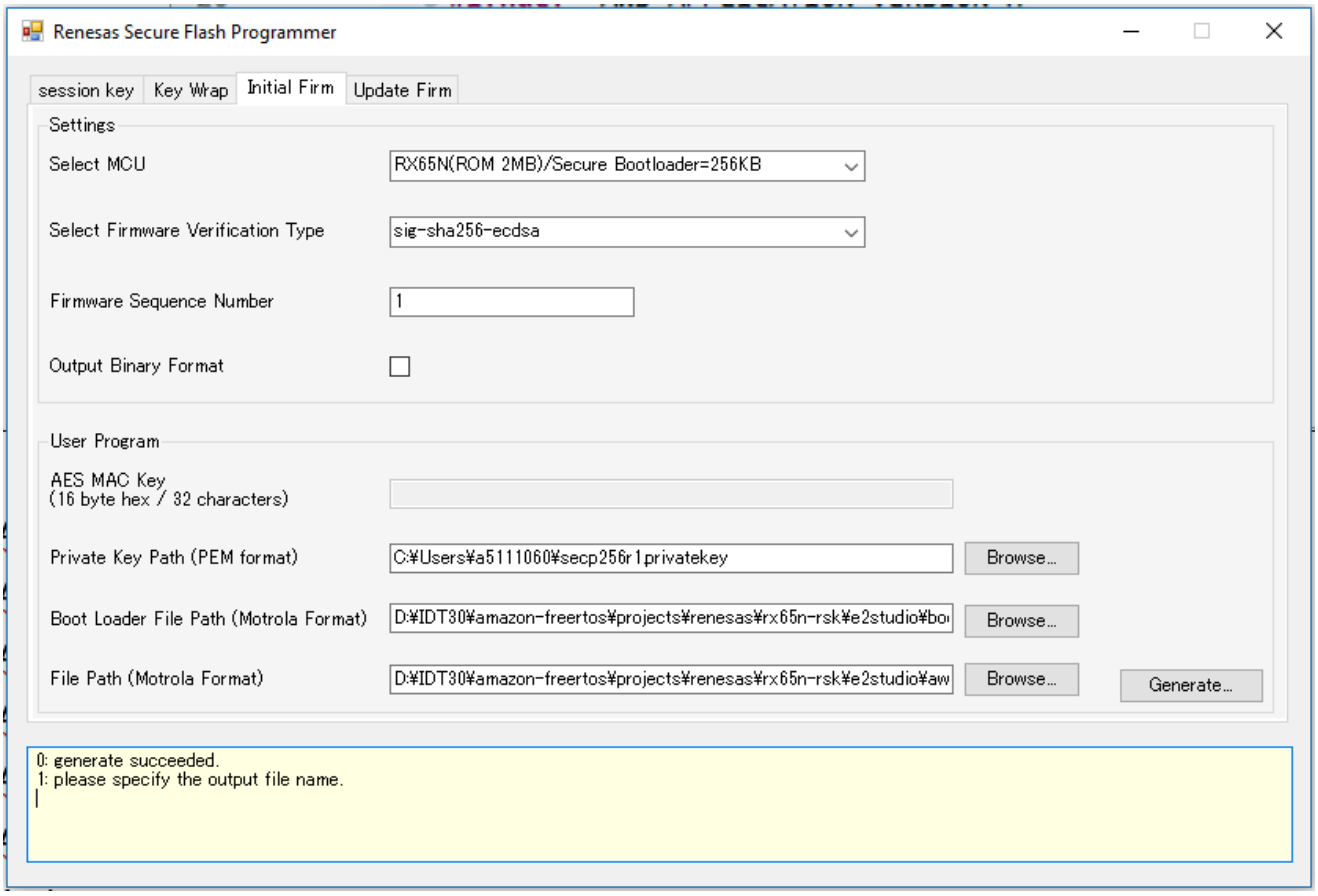
d. Pilih Build untuk membuat `aws_demos.mot` file.

14. Buat file `userprog.mot` dengan Renesas Secure Flash Programmer. `userprog.mot`adalah kombinasi dari `aws_demos.mot` dan`boot_loader.mot`. Anda dapat mem-flash file ini ke RX65 N-RSK untuk menginstal firmware awal.

    a. Unduh https://github.com/renesas/Amazon-Freertos-Tools dan buka. `Renesas Secure Flash Programmer.exe`

    b. Pilih tab Firm Awal dan kemudian atur parameter berikut:

- Private Key Path — Lokasi`secp256r1.privatekey`.
- Jalur File Boot Loader — Lokasi `boot_loader.mot` (`projects\renesas\rx65n-rsk\e2studio\boot_loader\HardwareDebug`).
- File Path — Lokasi dari `aws_demos.mot` (`projects\renesas\rx65n-rsk\e2studio\aws_demos\HardwareDebug`).



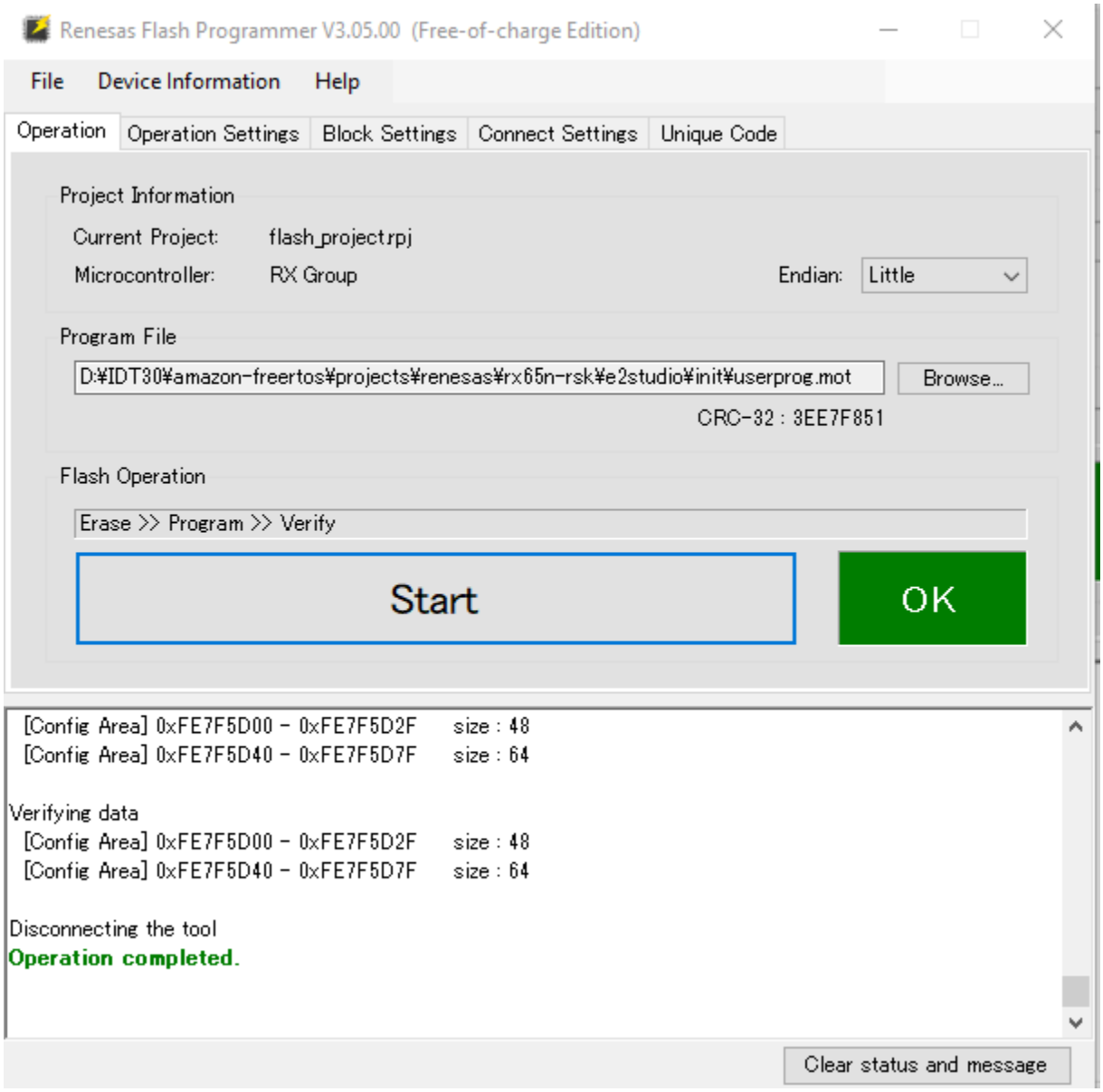
c. Buat direktori bernama`init_firmware`, Hasilkan`userprog.mot`, dan simpan ke `init_firmware` direktori. Verifikasi bahwa hasil berhasil.

15. Flash firmware awal pada RX65 N-RSK.

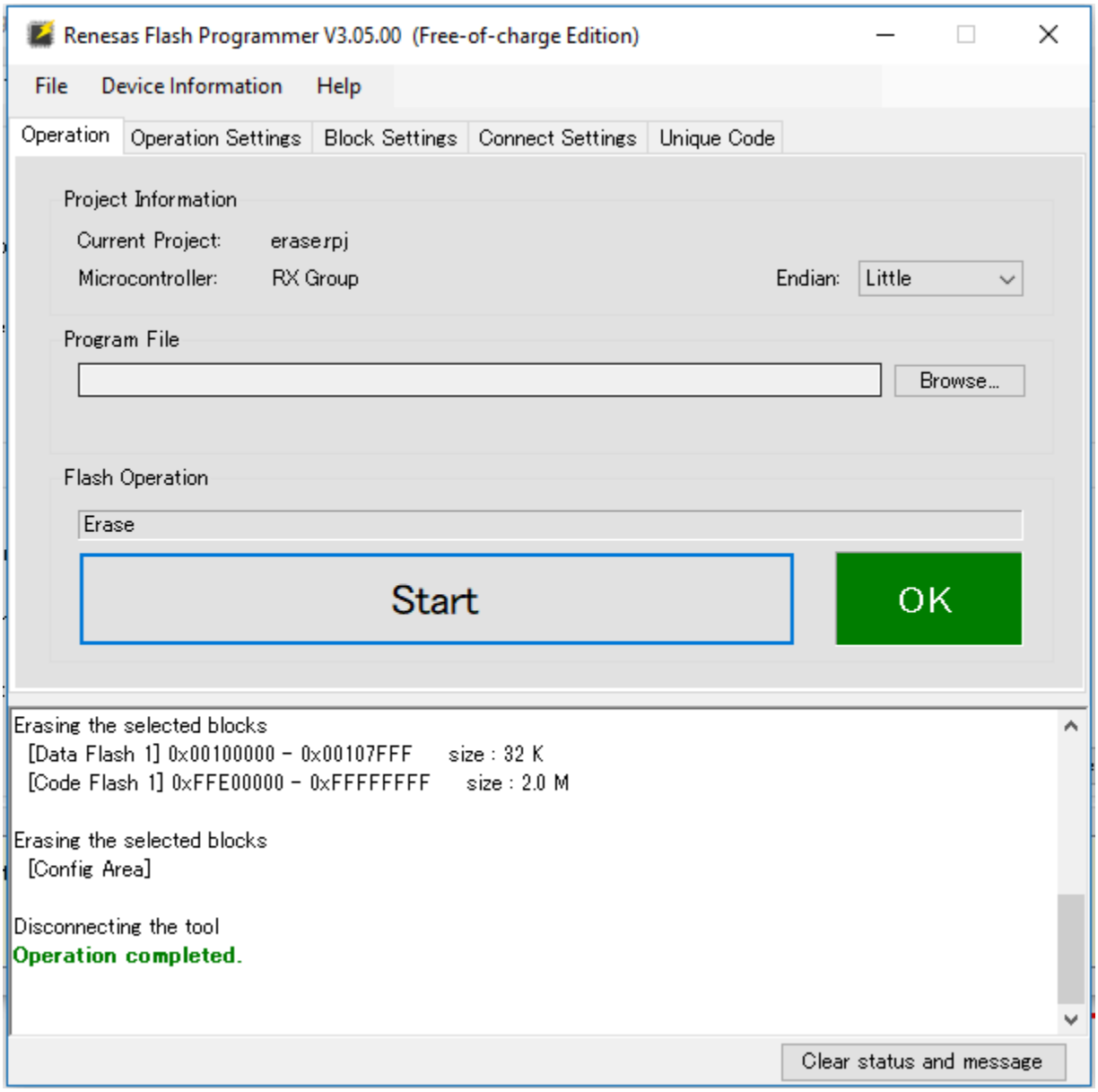
a. [Unduh versi terbaru dari Renesas Flash Programmer (Programming GUI) dari - .html. https://www.renesas.com/tw/ en/products/software-tools/tools/programmer/renesas flash-programmer-programming-gui]

b. Buka `vendors\renesas\rx_mcu_boards\boards\rx65n-rsk\aws_demos\flash_project\erase_from_bank\ erase.rpj` file untuk menghapus data di bank.

c. Pilih Mulai untuk menghapus bank.



d. Untuk mem-flash`userprog.mot`, pilih Browse... dan navigasikan ke `init_firmware` direktori, pilih `userprog.mot` file dan pilih Mulai.



16. Versi 0.9.2 (versi awal) dari firmware telah diinstal ke N-RSK Anda RX65. Papan RX65 N-RSK sekarang mendengarkan pembaruan OTA. Jika Anda telah membuka Tera Term di PC Anda, Anda melihat sesuatu seperti berikut ketika firmware awal berjalan.

```
------------------------------------------------
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. Tugas B: Perbarui versi firmware Anda

    a. Buka `demos/include/aws_application_version.h` file dan naikkan nilai `APP_VERSION_BUILD` token ke`0.9.3`.

b. Membangun kembali proyek.

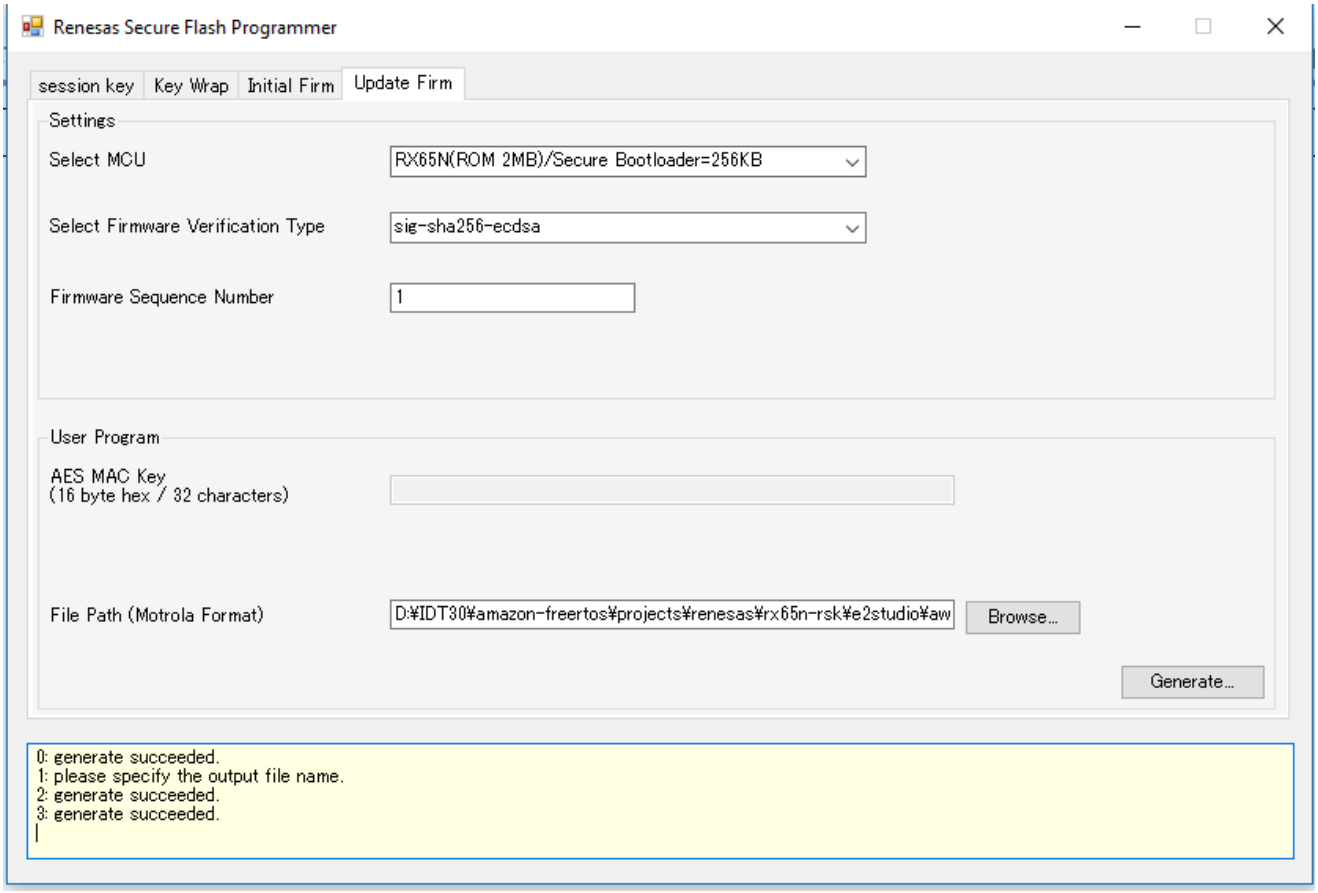
18. Buat `userprog.rsu` file dengan Renesas Secure Flash Programmer untuk memperbarui versi firmware Anda.

    a. Buka file `Amazon-FreeRTOS-Tools\Renesas Secure Flash Programmer.exe`.

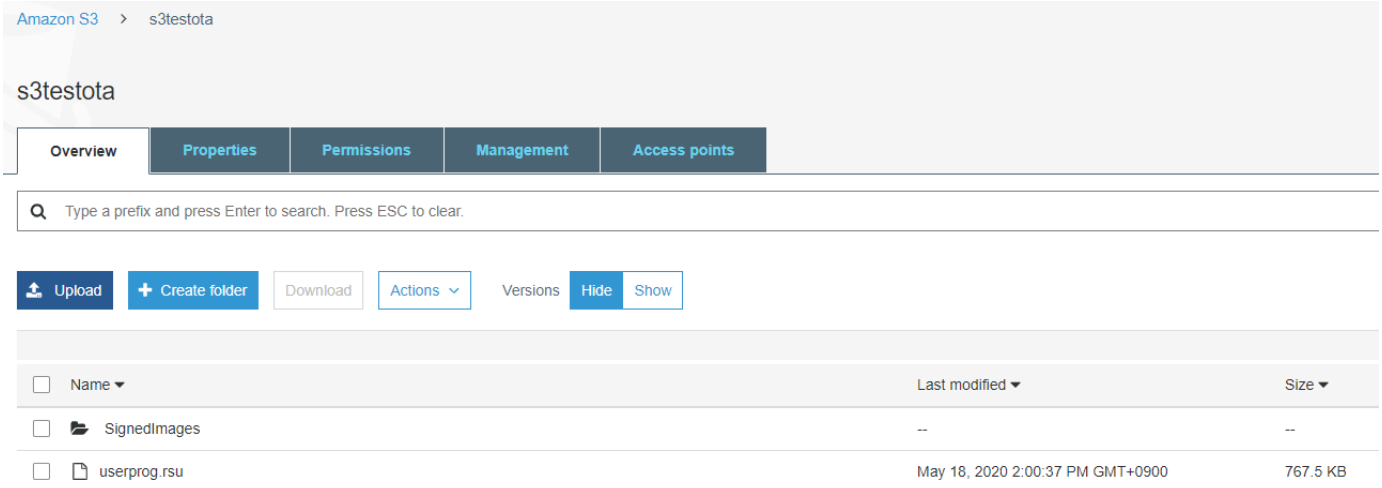
    b. Pilih tab Perbarui Perusahaan dan atur parameter berikut:

    - File Path — Lokasi `aws_demos.mot` file (`projects\renesas\rx65n-rsk\e2studio\aws_demos\HardwareDebug`).

    c. Membuat sebuah direktori bernama `update _firmware`. Hasilkan `userprog.rsu` dan simpan ke `update_firmware` direktori. Verifikasi bahwa hasil berhasil.



19. Unggah pembaruan firmware,`userproj.rsu`, ke dalam bucket Amazon S3 seperti yang dijelaskan dalam. [Buat bucket Amazon S3 untuk menyimpan pembaruan Anda]



20. Buat pekerjaan untuk memperbarui firmware pada RX65 N-RSK.

    AWS IoT Jobs adalah layanan yang memberi tahu satu atau beberapa perangkat yang terhubung tentang Job yang tertunda. Pekerjaan dapat digunakan untuk mengelola armada perangkat, memperbarui firmware dan sertifikat keamanan pada perangkat, atau melakukan tugas administratif seperti memulai ulang perangkat dan melakukan diagnostik.

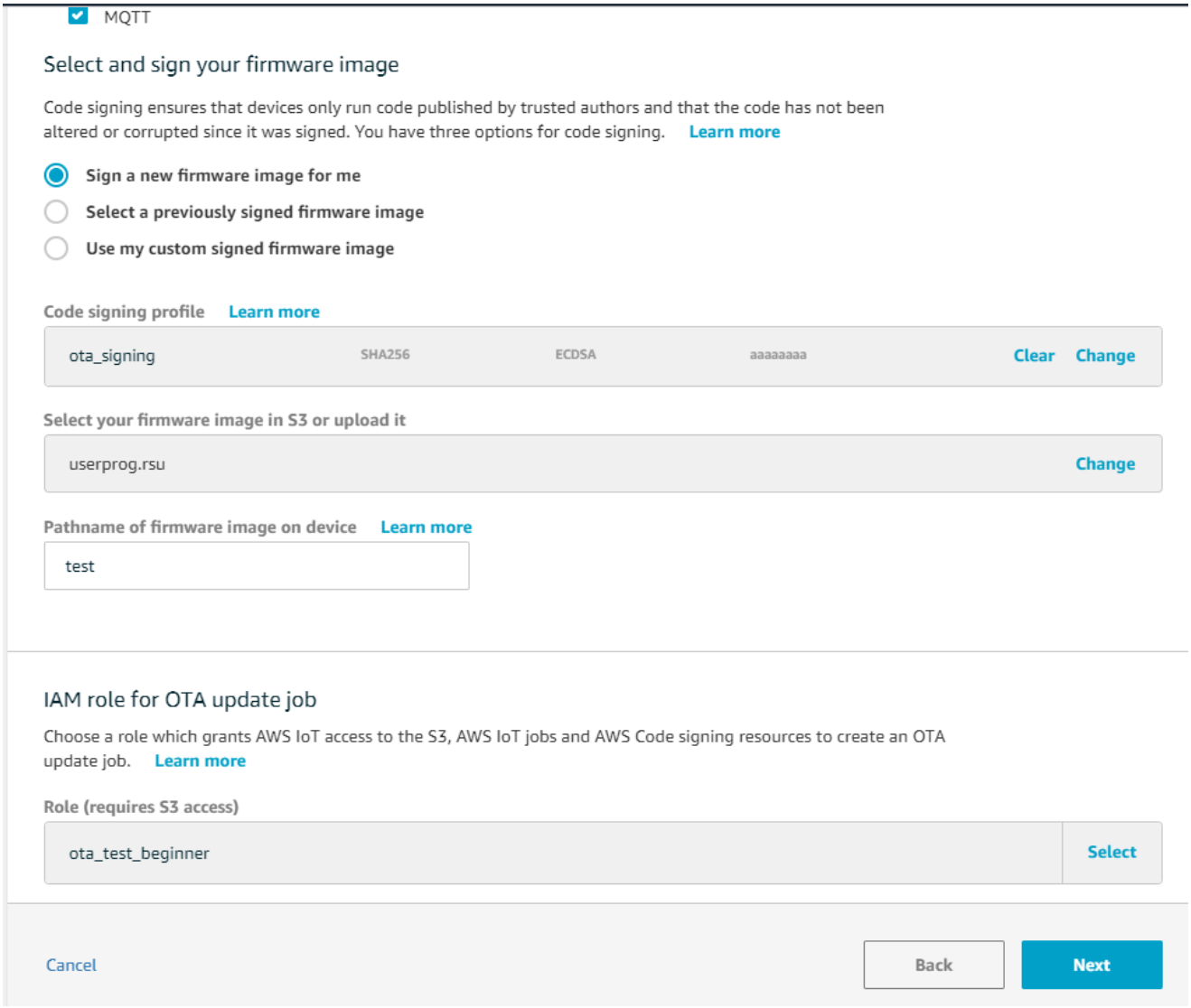
    a. Masuk ke konsol AWS IoT tersebut. Di panel navigasi, pilih Kelola, dan pilih Pekerjaan.

    b. Pilih Buat pekerjaan, lalu pilih Buat pekerjaan Pembaruan OTA. Pilih sesuatu, lalu pilih Berikutnya.

    c. Buat pekerjaan pembaruan FreeRTOS OTA sebagai berikut:

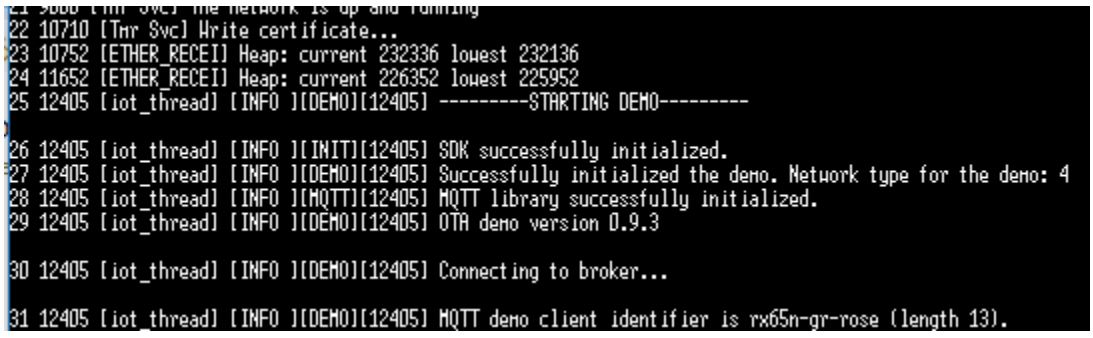
    - Pilih MQTT.
    - Pilih profil penandatanganan kode yang Anda buat di bagian sebelumnya.
    - Pilih gambar firmware yang Anda unggah ke bucket Amazon S3.
    - Untuk Pathname gambar firmware pada perangkat, masukkan**`test`**.
    - Pilih peran IAM yang Anda buat di bagian sebelumnya.

    d. Pilih Berikutnya.

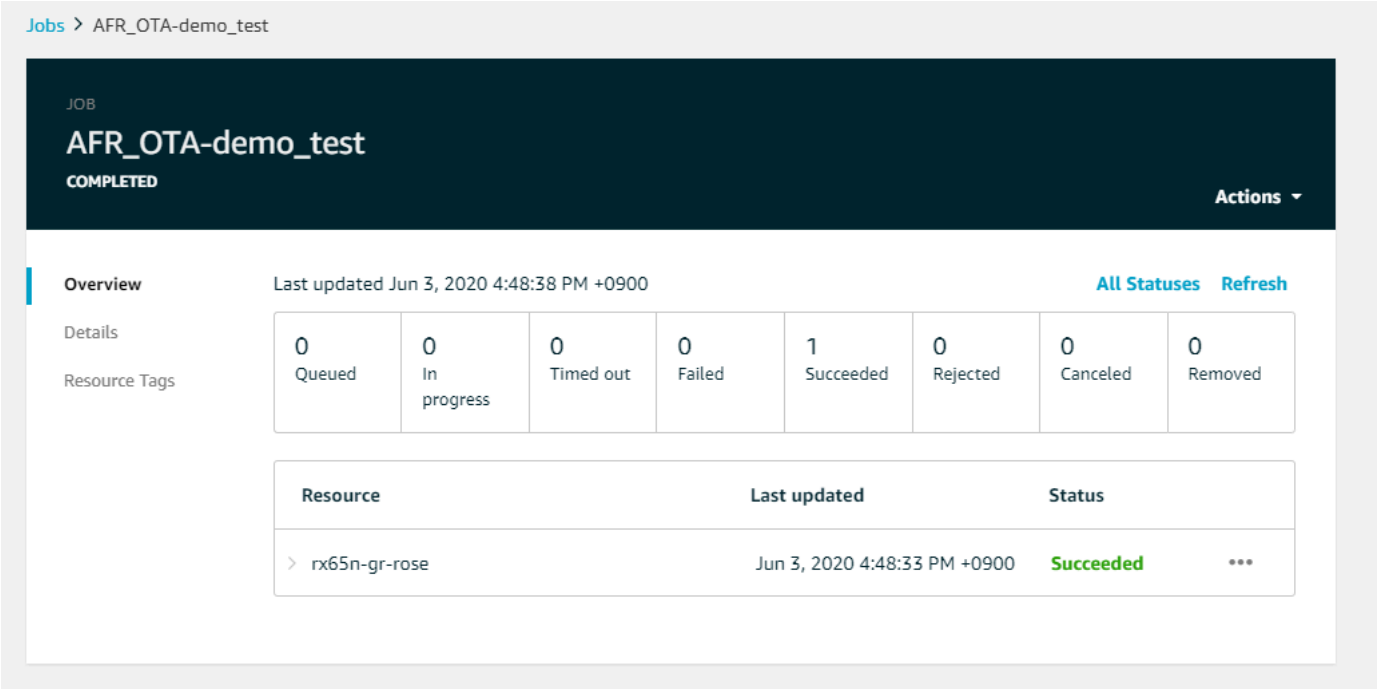


e. Masukkan ID lalu pilih Buat.

21. Buka kembali Tera Term untuk memverifikasi bahwa firmware berhasil diperbarui ke versi demo OTA 0.9.3.



22. Di AWS IoT konsol, verifikasi bahwa status pekerjaan berhasil.



Tutorial: Lakukan pembaruan OTA di Espressif menggunakan ESP32 FreeRTOS Bluetooth Low Energy

> **Important**
>
> Integrasi referensi ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-FreerTOS yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Tutorial ini menunjukkan cara memperbarui ESP32 mikrokontroler Espressif yang terhubung ke proxy Bluetooth Low Energy MQTT di perangkat Android. Ini memperbarui perangkat menggunakan pekerjaan pembaruan AWS IoT Over-the-air (OTA). Perangkat terhubung untuk AWS IoT menggunakan kredensi Amazon Cognito yang dimasukkan di aplikasi demo Android. Operator resmi memulai pembaruan OTA dari cloud. Saat perangkat terhubung melalui aplikasi demo Android, pembaruan OTA dimulai dan firmware diperbarui di perangkat.

Versi FreeRTOS 2019.06.00 Major dan yang lebih baru termasuk dukungan proxy Bluetooth Low Energy MQTT yang dapat digunakan untuk penyediaan Wi-Fi dan koneksi aman ke layanan. AWS IoT Dengan menggunakan fitur Bluetooth Low Energy, Anda dapat membangun perangkat berdaya

rendah yang dapat dipasangkan ke perangkat seluler untuk konektivitas tanpa memerlukan Wi-Fi. Perangkat dapat berkomunikasi menggunakan MQTT dengan menghubungkan melalui Android atau iOS Bluetooth Low Energy SDKs yang menggunakan profil akses generik (GAP) dan atribut generik (GATT).

Berikut adalah langkah-langkah yang akan kami ikuti untuk memungkinkan pembaruan OTA melalui Bluetooth Low Energy:

1. Konfigurasikan penyimpanan: Buat bucket dan kebijakan Amazon S3 dan konfigurasikan pengguna yang dapat melakukan pembaruan.
2. Buat sertifikat penandatanganan kode: Buat sertifikat penandatanganan dan izinkan pengguna menandatangani pembaruan firmware.
3. Konfigurasikan autentikasi Amazon Cognito: Buat penyedia kredensi, kumpulan pengguna, dan akses aplikasi ke kumpulan pengguna.
4. Konfigurasikan FreeRTOS: Siapkan Bluetooth Low Energy, kredensi klien, dan sertifikat publik penandatanganan kode.
5. Konfigurasikan aplikasi Android: Siapkan penyedia kredensi, kumpulan pengguna, dan terapkan aplikasi ke perangkat Android.
6. Jalankan skrip pembaruan OTA: Untuk memulai pembaruan OTA, gunakan skrip pembaruan OTA.

Untuk informasi selengkapnya tentang cara kerja pembaruan, lihatPembaruan FreeRTOS Over-the-Air. Untuk informasi tambahan tentang cara mengatur fungsionalitas proxy Bluetooth Low Energy MQTT, lihat posting Menggunakan Bluetooth Low Energy dengan FreeRTOS di Espressif oleh Richard Kang. ESP32

Prasyarat

Untuk melakukan langkah-langkah dalam tutorial ini, Anda memerlukan sumber daya berikut:

- Papan ESP32 pengembangan.
- Kabel microUSB ke USB A.
- AWS Akun (Tingkat Gratis sudah cukup).
- Ponsel Android dengan Android v 6.0 atau yang lebih baru dan Bluetooth versi 4.2 atau yang lebih baru.

Di komputer pengembangan Anda, Anda membutuhkan:

- Ruang disk yang cukup (~ 500 Mb) untuk toolchain Xtensa dan kode sumber FreerTOS dan contoh.
- Android Studio diinstal.
- Yang AWS CLIdiinstal.
- Python3 diinstal.
- Kit Pengembang AWS Perangkat Lunak (SDK) boto3 untuk Python.

Langkah-langkah dalam tutorial ini mengasumsikan bahwa Xtensa toolchain, ESP-IDF, dan kode FreeRTOS diinstal di direktori di direktori home Anda. `/esp` Anda harus menambahkan `~/esp/xtensa-esp32-elf/bin` ke `$PATH` variabel Anda.

Langkah 1: Konfigurasikan penyimpanan

1. Buat bucket Amazon S3 untuk menyimpan pembaruan Andadengan versi diaktifkan untuk menahan gambar firmware.
2. Buat peran layanan Pembaruan OTAdan tambahkan kebijakan terkelola berikut ke peran:
   - AWSIotPenebangan
   - AWSIotRuleActions
   - AWSIotThingsRegistration
   - AWSFreeRTOSOTAUpdate
3. Buat pengguna yang dapat melakukan pembaruan OTA. Pengguna ini dapat menandatangani dan menyebarkan pembaruan firmware ke perangkat IoT di akun, dan memiliki akses untuk melakukan pembaruan OTA di semua perangkat. Akses harus dibatasi untuk entitas tepercaya.
4. Ikuti langkah-langkahnya Buat kebijakan pengguna OTA dan lampirkan ke pengguna Anda.

Langkah 2: Buat sertifikat penandatanganan kode

1. Buat bucket Amazon S3 dengan versi yang diaktifkan untuk menyimpan gambar firmware.
2. Buat sertifikat penandatanganan kode yang dapat digunakan untuk menandatangani firmware. Perhatikan sertifikat Amazon Resource Name (ARN) saat sertifikat diimpor.

```
aws acm import-certificate --profile=ota-update-user --certificate file://
ecdsasigner.crt --private-key file://ecdsasigner.key
```

Contoh output:

```
{
"CertificateArn": "arn:aws:acm:us-east-1:<account>:certificate/<certid>"
}
```

Anda akan menggunakan ARN nanti untuk membuat profil penandatanganan. Jika mau, Anda dapat membuat profil sekarang dengan perintah berikut:

```
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
```

Contoh output:

```
{
"arn": "arn:aws:signer::<account>:/signing-profiles/esp32Profile"
}
```

Langkah 3: Konfigurasi otentikasi Amazon Cognito

Buat AWS IoT kebijakan

1. Masuk ke konsol AWS IoT tersebut.
2. Di sudut kanan atas konsol, pilih Akun Saya. Di bawah Pengaturan Akun, catat 12 digit ID akun Anda.
3. Pada panel navigasi kiri, pilih Pengaturan. Di titik akhir data Perangkat, catat nilai titik akhir. Titik akhir harus seperti`xxxxxxxxxxxxxx.iot.us-west-2.amazonaws.com`. Dalam contoh ini, AWS Region adalah “us-west-2".
4. Di panel navigasi kiri, pilih Aman, pilih Kebijakan, lalu pilih Buat. Jika Anda tidak memiliki kebijakan apa pun di akun Anda, Anda akan melihat pesan “Anda belum memiliki kebijakan apa pun” dan Anda dapat memilih Buat kebijakan.
5. Masukkan nama untuk kebijakan Anda, misalnya, “esp32_mqtt_proxy_iot_policy”.

6. Di bagian Tambahkan pernyataan, pilih Mode lanjutan. Salin dan tempel JSON berikut ke jendela editor kebijakan. Ganti `aws-account-id` dengan ID akun Anda dan `aws-region` dengan Wilayah Anda (misalnya, “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. Pilih Buat.

Buat AWS IoT sesuatu

1. Masuk ke konsol AWS IoT tersebut.
2. Di panel navigasi kiri, pilih Kelola, lalu pilih Things.
3. Di pojok kanan atas, pilih Buat. Jika Anda belum memiliki barang yang terdaftar di akun Anda, pesan “Anda belum memiliki barang” ditampilkan dan Anda dapat memilih Daftarkan sesuatu.
4. Pada halaman Creating AWS IoT things, pilih Create a single.

5. Pada halaman Tambahkan perangkat Anda ke halaman registri benda, masukkan nama untuk barang Anda (misalnya, “esp32-ble”). Hanya karakter alfanumerik, tanda hubung (-), dan garis bawah (_) yang diizinkan. Pilih Berikutnya.
6. Pada halaman Tambahkan sertifikat untuk hal Anda, di bawah Lewati sertifikat dan buat sesuatu, pilih Buat sesuatu tanpa sertifikat. Karena kami menggunakan aplikasi seluler proxy BLE yang menggunakan kredensi Amazon Cognito untuk otentikasi dan otorisasi, tidak diperlukan sertifikat perangkat.

Buat Klien Aplikasi Amazon Cognito

1. Masuk ke konsol Amazon Cognito.
2. Di spanduk navigasi kanan atas, pilih Buat kumpulan pengguna.
3. Masukkan nama pool (misalnya, “esp32_mqtt_proxy_user_pool”).
4. Pilih Tinjau default.
5. Di Klien Aplikasi, pilih Tambahkan klien aplikasi, lalu pilih Tambahkan klien aplikasi.
6. Masukkan nama klien aplikasi (misalnya “mqtt_app_client”).
7. Pastikan Hasilkan rahasia klien dipilih.
8. Pilih Buat klien aplikasi.
9. Pilih Kembali ke perincian kolam.
10. Pada halaman Ulasan kumpulan pengguna, pilih Buat kolam. Anda akan melihat pesan yang mengatakan “Kumpulan pengguna Anda berhasil dibuat.” Catat ID kolam.
11. Di panel navigasi, pilih Klien aplikasi.
12. Pilih Tampilkan Detail. Catat ID klien aplikasi dan rahasia klien aplikasi.

Buat kumpulan identitas Amazon Cognito

1. Masuk ke konsol Amazon Cognito.
2. Pilih Buat kolam identitas baru.
3. Masukkan nama untuk kumpulan identitas (misalnya, “mqtt_proxy_identity_pool”).
4. Perluas penyedia otentikasi.
5. Pilih tab Cognito.
6. Masukkan ID kumpulan pengguna dan ID klien aplikasi yang Anda catat di langkah sebelumnya.
7. Pilih Buat kolam.

8. Di halaman berikutnya, untuk membuat peran baru untuk identitas yang diautentikasi dan tidak diautentikasi, pilih Izinkan.
9. Catat ID kumpulan identitas, yang ada dalam format`us-east-1:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx`.

Lampirkan kebijakan IAM ke identitas yang diautentikasi

1. Buka Amazon Cognito konsol.
2. Pilih kumpulan identitas yang baru saja Anda buat (misalnya, “mqtt_proxy_identity_pool”).
3. Pilih Edit kolam identitas.
4. Catat Peran IAM yang ditetapkan ke peran yang diautentikasi (misalnya, “cognito_mqtt_proxy_identity_poolauth_role”).
5. Buka konsol IAM.
6. Di panel navigasi, pilih Peran.
7. Cari peran yang ditetapkan (misalnya, “Cognito_mqtt_proxy_identity_poolauth_role”), lalu pilih.
8. Pilih Tambahkan kebijakan sebaris, lalu pilih JSON.
9. Masukkan kebijakan berikut:

```
{
       "Version": "2012-10-17",
       "Statement": [
       {
          "Effect": "Allow",
          "Action": [
             "iot:AttachPolicy",
             "iot:AttachPrincipalPolicy",
             "iot:Connect",
             "iot:Publish",
             "iot:Subscribe"
          ],
          "Resource": "*"
       }]
    }
```

10. Pilih Tinjau Kebijakan.
11. Masukkan nama kebijakan (misalnya, "mqttProxyCognitoKebijakan”).
12. Pilih Buat kebijakan.

Langkah 4: Konfigurasikan Amazon FreeRTOS

1. Unduh versi terbaru kode Amazon FreeRTOS dari repo FreerTOS. GitHub
2. Untuk mengaktifkan demo pembaruan OTA, ikuti langkah-langkahnyaMemulai dengan Espressif ESP32 - DevKit C dan ESP-WROVER-KIT.
3. Buat modifikasi tambahan ini dalam file-file berikut:

   a. Buka `vendors/espressif/boards/esp32/aws_demos/config_files/aws_demo_config.h` dan tentukan`CONFIG_OTA_UPDATE_DEMO_ENABLED`.

   b. Buka `vendors/espressif/boards/esp32/aws_demos/common/config_files/aws_demo_config.h` dan ubah `democonfigNETWORK_TYPES` ke`AWSIOT_NETWORK_TYPE_BLE`.

   c. Buka `demos/include/aws_clientcredential.h` dan masukkan URL titik akhir Anda untuk`clientcredentialMQTT_BROKER_ENDPOINT`.

      Masukkan nama benda Anda untuk `clientcredentialIOT_THING_NAME` (misalnya, “esp32-ble”). Sertifikat tidak harus ditambahkan saat Anda menggunakan kredensi Amazon Cognito.

   d. Buka `vendors/espressif/boards/esp32/aws_demos/config_files/aws_iot_network_config.h` dan ubah `configSUPPORTED_NETWORKS` dan `configENABLED_NETWORKS` sertakan saja`AWSIOT_NETWORK_TYPE_BLE`.

   e. Buka `vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/ota_demo_config.h` file, dan masukkan sertifikat Anda.

      ```
      #define otapalconfigCODE_SIGNING_CERTIFICATE [] = "your-certificate-key";
      ```

   Aplikasi harus memulai dan mencetak versi 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...
   ```

Langkah 5: Konfigurasikan aplikasi Android

1. Unduh Android Bluetooth Low Energy SDK dan contoh aplikasi dari repo amazon-freertos-ble-android GitHub -sdk.
2. Buka file `app/src/main/res/raw/awsconfiguration.json` dan isi Pool Id, Region AppClientId, dan AppClientSecret menggunakan instruksi dalam contoh JSON berikut.

```
{
  "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. Buka `app/src/main/java/software/amazon/freertos/DemoConstants.java` dan masukkan nama kebijakan yang Anda buat sebelumnya (misalnya,*`esp32_mqtt_proxy_iot_policy`*) dan juga Wilayah (misalnya,*`us-east-1`*).
4. Bangun dan instal aplikasi demo.

a. Di Android Studio, pilih Build, lalu Make Module app.

b. Pilih Jalankan, lalu Jalankan aplikasi. Anda dapat membuka panel jendela logcat di Android Studio untuk memantau pesan log.

c. Di perangkat Android, buat akun dari layar login.

d. Buat pengguna. Jika pengguna sudah ada, masukkan kredensialnya.

e. Izinkan Demo Amazon FreeRTOS mengakses lokasi perangkat.

f. Pindai perangkat Bluetooth Low Energy.

g. Pindahkan slider untuk perangkat yang ditemukan ke Aktif.

h. Tekan y pada konsol debug port serial untuk file. ESP32

i. Pilih Pair & Connect.

5. Semakin banyak... link menjadi aktif setelah koneksi dibuat. Status koneksi harus berubah menjadi “BLE_CONNECTED” di logcat perangkat Android saat koneksi selesai:

```
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. Sebelum pesan dapat dikirim, perangkat Amazon FreeRTOS dan perangkat Android menegosiasikan MTU. Anda akan melihat output berikut di logcat:

```
2019-06-06 20:11:46.720 23484-23497/software.amazon.freertos.demo I/FRD:
 onMTUChanged : 512 status: Success
```

7. Perangkat terhubung ke aplikasi dan mulai mengirim pesan MQTT menggunakan proxy MQTT. Untuk mengonfirmasi bahwa perangkat dapat berkomunikasi, pastikan nilai data karakteristik MQTT_CONTROL berubah menjadi 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. Saat perangkat dipasangkan, Anda akan melihat prompt di ESP32 konsol. Untuk mengaktifkan BLE, tekan y. Demo tidak akan berfungsi sampai Anda melakukan langkah ini.

```
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.
```

Langkah 6: Jalankan skrip pembaruan OTA

1. Untuk menginstal prasyarat, jalankan perintah berikut:

   ```
   pip3 install boto3
   ```

   ```
   pip3 install pathlib
   ```

2. Tingkatkan versi aplikasi FreeRTOS di. `demos/include/aws_application_version.h`
3. Buat file.bin baru.
4. Unduh skrip python start_ota.py. Untuk melihat isi bantuan untuk skrip, jalankan perintah berikut di jendela terminal:

   ```
   python3 start_ota.py -h
   ```

   Anda akan melihat sesuatu seperti berikut:

   ```
   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. Jika Anda menggunakan AWS CloudFormation template yang disediakan untuk membuat sumber daya, jalankan perintah berikut:

```
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
```

   Anda akan melihat pembaruan dimulai di konsol ESP32 debug:

```
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. Ketika pembaruan OTA selesai, perangkat memulai ulang seperti yang dipersyaratkan oleh proses pembaruan OTA. Kemudian mencoba terhubung menggunakan firmware yang diperbarui. Jika pemutakhiran telah berhasil, firmware yang diperbarui ditandai sebagai aktif dan Anda akan melihat versi yang diperbarui di konsol:

```
13 13498 [iot_thread] OTA demo version 0.9.21
```

AWS IoT Aplikasi demo Device Shadow

> **Important**
>
> Demo ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-FreerTOS yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Pengantar

Demo ini menunjukkan cara menggunakan pustaka AWS IoT Device Shadow untuk terhubung ke layanan AWS Device Shadow. Ini menggunakan Perpustakaan CoreMQTT untuk membuat koneksi MQTT dengan TLS (Mutual Authentication) ke Broker AWS IoT MQTT dan parser perpustakaan CoreJson untuk mengurai dokumen bayangan yang diterima dari layanan Shadow. AWS Demo menunjukkan operasi bayangan dasar, seperti cara memperbarui dokumen bayangan dan cara menghapus dokumen bayangan. Demo juga menunjukkan cara mendaftarkan fungsi callback dengan pustaka CoreMQTT untuk menangani pesan seperti bayangan `/update` dan `/update/delta` pesan yang dikirim dari layanan Device Shadow. AWS IoT

Demo ini dimaksudkan sebagai latihan pembelajaran hanya karena permintaan untuk memperbarui dokumen bayangan (status) dan respons pembaruan dilakukan oleh aplikasi yang sama. Dalam skenario produksi yang realistis, aplikasi eksternal akan meminta pembaruan status perangkat dari jarak jauh, bahkan jika perangkat saat ini tidak terhubung. Perangkat akan mengakui permintaan pembaruan saat terhubung.

> **Note**
>
> Untuk mengatur dan menjalankan demo FreeRTOS, ikuti langkah-langkahnya. Memulai dengan FreeRTOS

Fungsionalitas

Demo membuat tugas aplikasi tunggal yang mengulang melalui serangkaian contoh yang mendemonstrasikan bayangan `/update` dan `/update/delta` panggilan balik untuk mensimulasikan pengalihan status perangkat jarak jauh. Ini mengirimkan pembaruan bayangan dengan `desired` status baru dan menunggu perangkat mengubah `reported` statusnya sebagai respons terhadap `desired` status baru. Selain itu, `/update` callback bayangan digunakan untuk mencetak status bayangan yang berubah. Demo ini juga menggunakan koneksi MQTT yang aman ke Broker AWS IoT MQTT, dan mengasumsikan ada status dalam bayangan perangkat. `power0n`

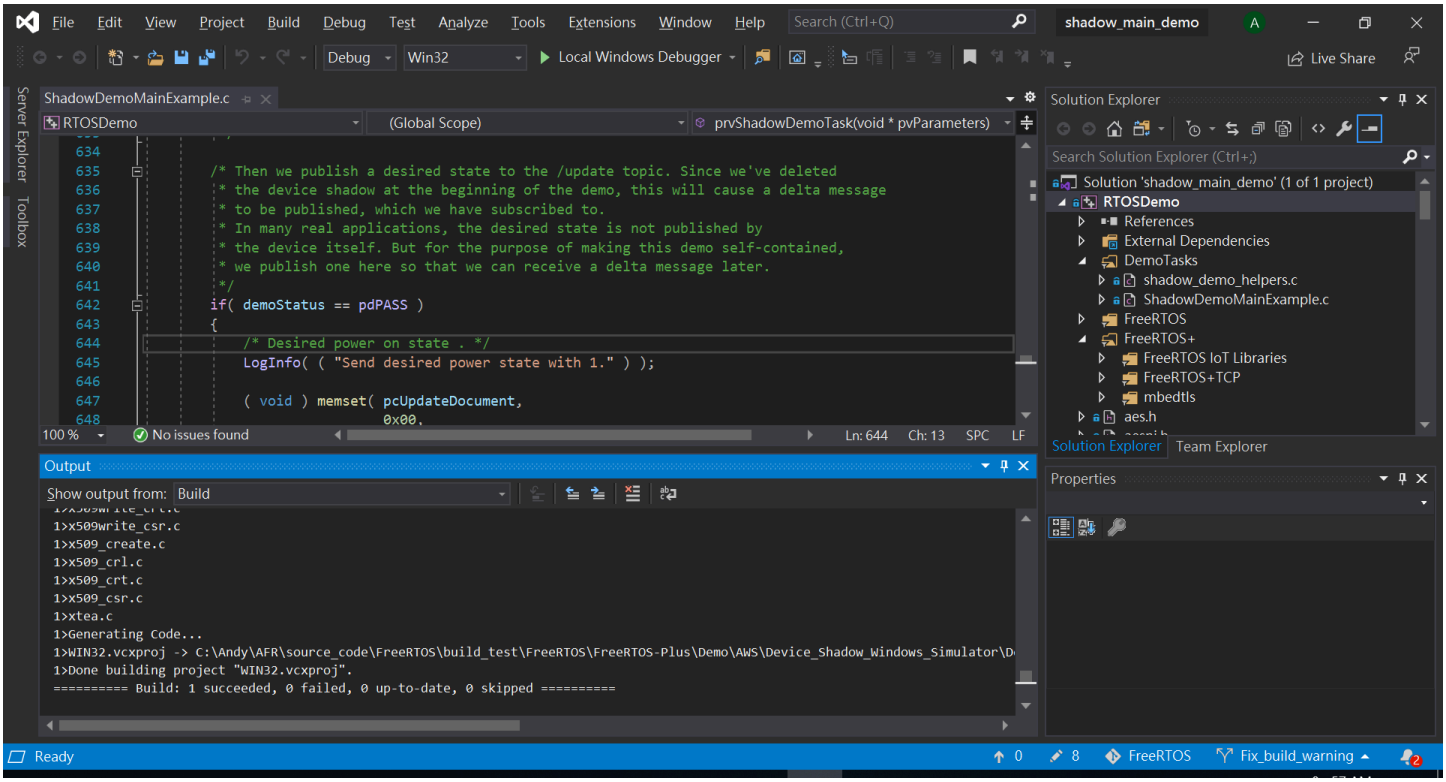
Demo melakukan operasi berikut:

1. Buat koneksi MQTT dengan menggunakan fungsi pembantu di. `shadow_demo_helpers.c`
2. Merakit string topik MQTT untuk operasi bayangan perangkat, menggunakan makro yang ditentukan oleh pustaka Device Shadow. AWS IoT
3. Publikasikan ke topik MQTT yang digunakan untuk menghapus bayangan perangkat untuk menghapus bayangan perangkat yang ada.
4. Berlangganan topik MQTT untuk`/update/delta`, `/update/accepted` dan `/update/rejected` gunakan fungsi pembantu di. `shadow_demo_helpers.c`
5. Publikasikan status yang diinginkan `power0n` menggunakan fungsi pembantu di`shadow_demo_helpers.c`. Ini akan menyebabkan `/update/delta` pesan dikirim ke perangkat.
6. Tangani pesan MQTT yang masuk`prvEventCallback`, dan tentukan apakah pesan tersebut terkait dengan bayangan perangkat menggunakan fungsi yang ditentukan oleh pustaka Device Shadow AWS IoT (). `Shadow_MatchTopic` Jika pesan adalah `/update/delta` pesan bayangan perangkat, maka fungsi demo utama akan mempublikasikan pesan kedua untuk memperbarui status yang dilaporkan kepada`power0n`. Jika `/update/accepted` pesan diterima, verifikasi bahwa pesan tersebut `clientToken` sama dengan yang dipublikasikan sebelumnya dalam pesan pembaruan. Itu akan menandai akhir dari 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]
```

Demo dapat ditemukan di file *freertos*/demos/device_shadow_for_aws/shadow_demo_main.c atau di GitHub.

Tangkapan layar berikut menunjukkan output yang diharapkan ketika demo berhasil.



Connect ke broker AWS IoT MQTT

Untuk terhubung ke broker AWS IoT MQTT, kami menggunakan metode yang sama seperti `MQTT_Connect()` di. Demo otentikasi timbal balik CoreMQTT

Hapus dokumen bayangan

Untuk menghapus dokumen bayangan, panggil `xPublishToTopic` dengan pesan kosong, menggunakan makro yang ditentukan oleh pustaka AWS IoT Device Shadow. Ini digunakan `MQTT_Publish` untuk mempublikasikan ke `/delete` topik. Bagian kode berikut menunjukkan bagaimana hal ini dilakukan dalam fungsi`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 );
```

Berlangganan topik bayangan

Berlangganan topik Device Shadow untuk menerima pemberitahuan dari AWS IoT broker tentang perubahan bayangan. Topik Device Shadow dirakit oleh makro yang ditentukan di pustaka Device Shadow. Bagian kode berikut menunjukkan bagaimana hal ini dilakukan dalam `prvShadowDemoTask` fungsi.

```
/* 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 ) );
}
```

Kirim Pembaruan Bayangan

Untuk mengirim pembaruan bayangan, demo memanggil `xPublishToTopic` dengan pesan dalam format JSON, menggunakan makro yang ditentukan oleh pustaka Device Shadow. Ini digunakan `MQTT_Publish` untuk mempublikasikan ke `/delete` topik. Bagian kode berikut menunjukkan bagaimana hal ini dilakukan dalam `prvShadowDemoTask` fungsi.

```
#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 ) );
```

Menangani pesan delta bayangan dan pesan pembaruan bayangan

Fungsi callback pengguna, yang terdaftar ke CoreMQTT Client Library menggunakan `MQTT_Init` fungsi tersebut, akan memberi tahu kami tentang peristiwa paket yang masuk. Lihat fungsi callback prvEventCallbackpada GitHub.

Fungsi callback mengonfirmasi jenis paket yang masuk`MQTT_PACKET_TYPE_PUBLISH`, dan menggunakan Device Shadow Library API `Shadow_MatchTopic` untuk mengonfirmasi bahwa pesan yang masuk adalah pesan bayangan.

Jika pesan yang masuk adalah pesan bayangan dengan tipe`ShadowMessageTypeUpdateDelta`, maka kita memanggil prvUpdateDeltaHandler untuk menangani pesan ini. Handler `prvUpdateDeltaHandler` menggunakan pustaka CoreJson untuk mengurai pesan guna mendapatkan nilai delta untuk `power0n` status dan membandingkannya dengan status perangkat saat ini yang dikelola secara lokal. Jika berbeda, status perangkat lokal diperbarui untuk mencerminkan nilai baru `power0n` status dari dokumen bayangan.

Jika pesan yang masuk adalah pesan bayangan dengan tipe`ShadowMessageTypeUpdateAccepted`, maka kita memanggil prvUpdateAcceptedHandler untuk menangani pesan ini. Handler `prvUpdateAcceptedHandler` mem-parsing pesan menggunakan pustaka CoreJson untuk mendapatkan pesan dari `clientToken` pesan. Fungsi handler ini memeriksa apakah token klien dari pesan JSON cocok dengan token klien yang digunakan oleh aplikasi. Jika tidak cocok, fungsi mencatat pesan peringatan.

Soket Aman menggemakan demo klien

> **Important**
>
> Demo ini di-host di repositori Amazon-Freertos yang tidak digunakan lagi. Kami menyarankan Anda mulai di sini ketika Anda membuat proyek baru. Jika Anda sudah memiliki proyek FreeRTOS yang sudah ada berdasarkan repositori Amazon-FreerTOS yang sekarang tidak digunakan lagi, lihat. Panduan Migrasi Repositori Github Amazon-freertos

Contoh berikut menggunakan tugas RTOS tunggal. Kode sumber untuk contoh ini dapat ditemukan di`demos/tcp/aws_tcp_echo_client_single_task.c`.

Sebelum Anda mulai, verifikasi bahwa Anda telah mengunduh FreerTOS ke mikrokontroler Anda dan membangun dan menjalankan proyek demo FreeRTOS. Anda dapat mengkloning atau mengunduh FreeRTOS dari. GitHub Lihat file README.md untuk instruksi.

Untuk menjalankan demo

> **Note**
>
> Untuk mengatur dan menjalankan demo FreeRTOS, ikuti langkah-langkahnya. Memulai dengan FreeRTOS
> Server TCP dan demo klien saat ini tidak didukung pada Kit Pengembangan Cypress CYW9439 07 AEVAL1 F dan CYW9549 07 AEVAL1 F.

1. Ikuti petunjuk dalam Menyiapkan Server Echo TLS di Panduan Porting FreeRTOS.

   Server gema TLS harus berjalan dan mendengarkan pada port 9000.

   Selama pengaturan, Anda seharusnya telah menghasilkan empat file:

   - `client.pem`(sertifikat klien)
   - `client.key`(kunci pribadi klien)
   - `server.pem`(sertifikat server)
   - `server.key`(kunci pribadi server)
2. Gunakan alat `tools/certificate_configuration/CertificateConfigurator.html` untuk menyalin sertifikat klien (`client.pem`) dan kunci pribadi klien (`client.key`) `keaws_clientcredential_keys.h`.
3. Buka file `FreeRTOSConfig.h`.
4. Atur`configECHO_SERVER_ADDR0`,`configECHO_SERVER_ADDR1`,`configECHO_SERVER_ADDR2`, dan `configECHO_SERVER_ADDR3` variabel ke empat bilangan bulat yang membentuk alamat IP tempat TLS Echo Server berjalan.
5. Atur `configTCP_ECHO_CLIENT_PORT` variabel ke`9000`, port tempat TLS Echo Server mendengarkan.
6. Atur `configTCP_ECHO_TASKS_SINGLE_TASK_TLS_ENABLED` variabel ke`1`.
7. Gunakan alat `tools/certificate_configuration/PEMfileToCString.html` untuk menyalin sertifikat server (`server.pem`) ke `cTlsECHO_SERVER_CERTIFICATE_PEM` dalam file`aws_tcp_echo_client_single_task.c`.
8. Buka*`freertos`*`/vendors/`*`vendor`*`/boards/`*`board`*`/aws_demos/config_files/aws_demo_config.h`, beri komentar`#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED`, dan tentukan

`CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED`
atau`CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED`.

Mikrokontroler dan TLS Echo Server harus berada di jaringan yang sama. Ketika demo dimulai (`main.c`), Anda akan melihat pesan log yang berbunyi`Received correct string from echo server`.

Terjemahan disediakan oleh mesin penerjemah. Jika konten terjemahan yang diberikan bertentangan dengan versi bahasa Inggris aslinya, utamakan versi bahasa Inggris.