



Panduan Pengguna

FreeRTOS



FreeRTOS: Panduan Pengguna

Copyright © 2024 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 para pelanggan, atau dengan cara apa pun yang menghina atau mendiskreditkan Amazon. Semua merek dagang lain yang tidak dimiliki oleh Amazon merupakan properti dari masing-masing pemilik, yang mungkin atau tidak berafiliasi, terkait, atau disponsori oleh Amazon.

Table of Contents

Apa itu FreeRTOS?	1
Mengunduh kode sumber FreeRTOS	1
Versi FreeRTOS	1
Dukungan Jangka Panjang FreeRTOS	2
Rencana Pemeliharaan Diperpanjang FreeRTOS	2
Arsitektur FreeRTOS	2
Platform perangkat keras berkualifikasi FreeRTOS	3
Alur kerja pengembangan	4
Sumber daya tambahan	5
Fundamental kernel FreeRTOS	6
Penjadwal kernel FreeRTOS	6
Mengelola memori	7
Alokasi memori kernel	7
Mengelola memori aplikasi	8
Koordinasi intertask	8
Antrean	8
Semaphores dan mutexes	9
Direct-to-task notifikasi	9
Buffer aliran	10
Buffer pesan	11
Dukungan multiprocessing simetris (SMP)	13
Memodifikasi aplikasi untuk menggunakan kernel FreeRTOS-SMP	13
Pengatur waktu perangkat lunak	14
Dukungan daya rendah	14
FreeRTOSConfig.h	14
AWS IoT SDK Perangkat untuk Embedded C	16
IO umum	17
Pustaka	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 FreeRTOS	21

Memulai AWS IoT dan FreeRTOS menggunakan Quick Connect	21
Eksplorasi pustaka FreeRTOS	21
Memahami cara membangun AWS IoT produk yang aman dan tangguh	22
Kembangkan produk AWS IoT aplikasi Anda	22
AWS IoT Device Tester untuk FreeRTOS	23
Suite kualifikasi FreeRTOS	23
Rangkaian pengujian khusus	24
Versi IDT yang didukung untuk FreeRTOS	25
IDT untuk FreeRTOS	25
Versi IDT sebelumnya	27
Versi IDT yang tidak didukung	34
Unduh IDT untuk FreeRTOS	67
Unduh IDT secara manual	67
Unduh IDT secara terprogram	68
Gunakan IDT dengan suite kualifikasi FreeRTOS 2.0 (FRQ 2.0)	73
Prasyarat	75
Bersiap untuk menguji papan mikrokontroler Anda untuk pertama kalinya	84
Gunakan IDT UI untuk menjalankan suite kualifikasi FreeRTOS	100
Menjalankan suite kualifikasi FreeRTOS 2.0	115
Memahami hasil dan log	119
Gunakan IDT dengan suite kualifikasi FreeRTOS 1.0 (FRQ 1.0)	123
Prasyarat	124
Bersiap untuk menguji papan mikrokontroler Anda untuk pertama kalinya	128
Gunakan UI IDT untuk menjalankan suite kualifikasi FreeRTOS	148
Menjalankan tes Bluetooth Low Energy	158
Menjalankan suite kualifikasi FreeRTOS	164
Memahami hasil dan log	169
Gunakan IDT untuk mengembangkan dan menjalankan rangkaian tes Anda sendiri	174
Unduh versi terbaru IDT untuk FreeRTOS	174
Alur kerja pembuatan rangkaian uji	175
Tutorial: Bangun dan jalankan sampel rangkaian tes IDT	176
Tutorial: Kembangkan rangkaian tes IDT sederhana	181
Versi rangkaian tes	267
Pemecahan Masalah	268
Memecahkan masalah konfigurasi perangkat	269
Memecahkan masalah kesalahan batas waktu	283

Fitur seluler dan AWS biaya	283
Kebijakan pembuatan laporan kualifikasi	284
AWS Kebijakan yang dikelola untuk AWS IoT Device Tester	284
Kebijakan terkelola	284
Pembaruan kebijakan	291
Kebijakan dukungan	294
Keamanan di AWS	296
Manajemen Identitas dan Akses	296
Audiens	297
Mengautentikasi dengan identitas	298
Mengelola akses menggunakan kebijakan	302
Bagaimana FreeRTOS bekerja dengan IAM	304
Contoh kebijakan berbasis identitas	312
Memecahkan masalah	315
Validasi kepatuhan	317
Ketahanan	318
Keamanan infrastruktur	318
Panduan Migrasi Repositori Github Amazon-freertos	320
Lampiran	320
Arsip	327
Arsip Panduan Pengguna FreeRTOS	327
Isi Panduan Pengguna FreeRTOS sebelumnya	327
Memulai dengan FreeRTOS	327
Pembaruan Over-the-Air	521
Perpustakaan FreeRTOS	607
Demo Demo demo FreeRTOS	675
.....	dccci

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).

Mengunduh kode sumber FreeRTOS

[Unduh paket FreeRTOS dan Long Term Support \(LTS\) terbaru dari halaman Unduhan di \[freertos.org\]\(https://freertos.org\).](#)

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 yyyy.mm.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, CorePKCS11, CoreJson, OTA, Jobs,, dan Device Shadow. 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.](#)

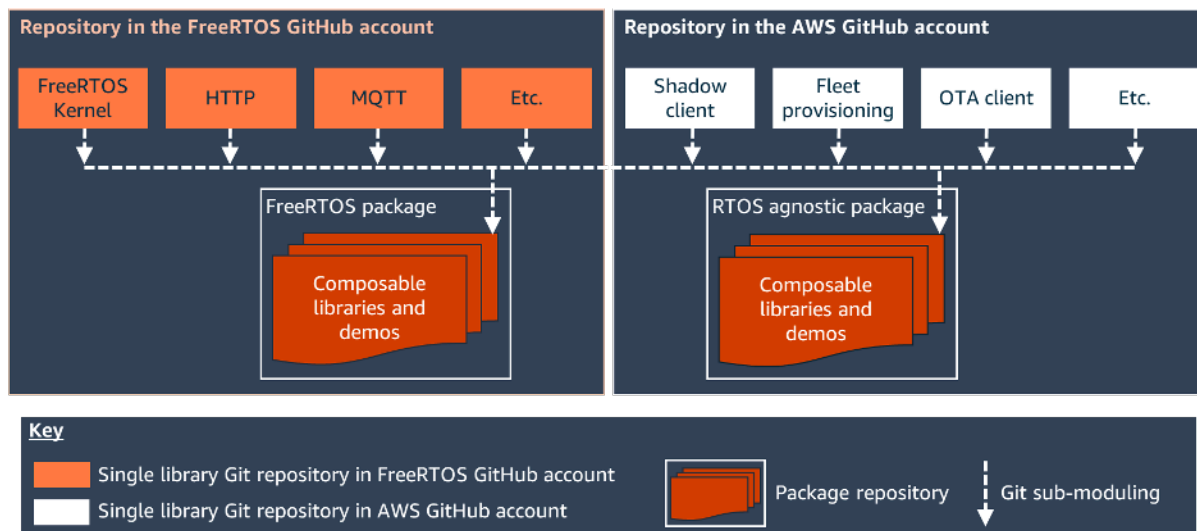
Arsitektur FreeRTOS

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 repository berisi beberapa pustaka, dan dapat berisi proyek yang telah dikonfigurasi sebelumnya yang menunjukkan penggunaan pustaka.

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

Repository git pustaka individu dibagi antara dua GitHub organisasi. Repository 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 Repository yang berisi pustaka AWS IoT tertentu (seperti klien AWS IoT over-the-air pembaruan) ada di organisasi. AWS GitHub Diagram berikut menjelaskan strukturnya.



Platform perangkat keras berkualitas Freertos

Platform perangkat keras berikut memenuhi syarat untuk FreeRTOS:

- [Kit Penyediaan Nol Sentuh ATECC608A untuk AWS IoT](#)
- [Kit Pengembangan Cypress CYW943907AEVAL1F](#)
- [Kit Pengembangan Cypress CYW954907AEVAL1F](#)
- [Cypress CY8CKIT-064S0S2-4343W Kit](#)
- [Espressif ESP32- C DevKit](#)
- [Espressif ESP-WROVER-KIT](#)

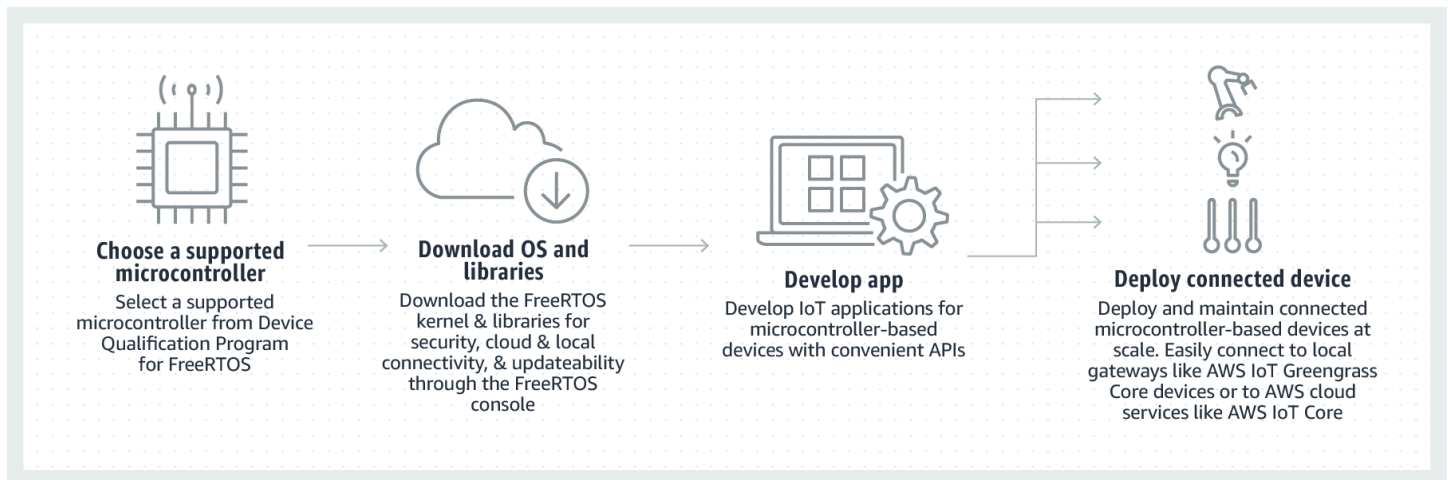
- [Espressif ESP-WROOM-32SE](#)
- [Espressif ESP32-S2-Saola-1](#)
- [Kit Konektivitas IoT Infineon XMC4800](#)
- [Kit Pemula Marvell MW320 AWS IoT](#)
- [Kit Pemula Marvell MW322 AWS IoT](#)
- [MediaTek Kit Pengembangan MT7697hx](#)
- [Microchip Curiosity PIC32MZEZ Bundel](#)
- [Nordik NRF52840-dk](#)
- [NuMaker-IoT-M487](#)
- [Modul IoT NXP LPC54018](#)
- [Solusi Keamanan OPTIGA Trust X](#)
- [Modul IoT Renesas RX65N RSK](#)
- [STMicroelectronicsSTM32L4 Kit Penemuan IoT Node](#)
- [Texas Instrumen CC3220SF-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](#).

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.



Sumber daya tambahan

Sumber daya ini mungkin bermanfaat bagi Anda.

- [Untuk Dokumentasi FreeRTOS tambahan, lihat freertos.org.](https://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 [Pusat AWS Dukungan.](#)
- Untuk pertanyaan tentang AWS penagihan, layanan akun, acara, penyalahgunaan, atau masalah lain dengan AWS, lihat halaman [Hubungi Kami.](#)

Fundamental kernel FreeRTOS

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

- Penjadwal multitasking.
- Beberapa opsi alokasi memori (termasuk kemampuan untuk membuat sistem yang dialokasikan secara statis).
- Primitif koordinasi intertask, 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 menjalankan daftar tertaut, di dalam bagian kritis atau interupsi. Kernel FreeRTOS menyertakan implementasi pengatur waktu perangkat lunak yang efisien yang tidak menggunakan waktu CPU apa pun kecuali pengatur waktu perlu diservis. Tugas yang diblokir tidak memerlukan servis berkala yang memakan waktu. Direct-to-task Pemberitahuan D memungkinkan pensinyalan tugas cepat, dengan praktis tidak ada overhead RAM. Mereka dapat digunakan di sebagian besar skenario intertask dan interrupt-to-task signaling.

Kernel FreeRTOS dirancang agar kecil, sederhana, dan mudah digunakan. Gambar biner kernel RTOS 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](#) dan semakin mendalam [Menguasai Kernel Waktu Nyata FreeRTOS](#).

Penjadwal kernel FreeRTOS

Aplikasi tertanam yang menggunakan RTOS dapat disusun sebagai serangkaian tugas independen. Setiap tugas mengeksekusi dalam konteksnya sendiri, tanpa ketergantungan pada tugas-tugas lain. Hanya satu tugas dalam aplikasi yang berlangsung pada titik waktu apa pun. Penjadwal RTOS real-time menentukan kapan setiap tugas harus dijalankan. Setiap tugas disediakan dengan tumpukannya sendiri. Ketika tugas ditukar sehingga tugas lain dapat berjalan, konteks eksekusi tugas disimpan ke

tumpukan tugas sehingga dapat dipulihkan ketika tugas yang sama kemudian ditukar kembali untuk melanjutkan pelaksanaannya.

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

Mengelola memori

Bagian ini memberikan informasi tentang alokasi memori kernel dan manajemen memori aplikasi.

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 tumpukan RTOS oleh fungsi pembuatan objek RTOS API.

Ketika objek RTOS dibuat secara dinamis, menggunakan perpustakaan C standar `malloc()` dan `free()` fungsi tidak selalu sesuai untuk 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 waktu nyata yang Anda kembangkan. Ketika kernel RTOS membutuhkan RAM, ia memanggil `pvPortMalloc()` alih-alih `malloc()` (). Saat RAM dibebaskan, kernel RTOS memanggil `pvPortFree()` alih-alih `free()` ().

Mengelola 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 menambahkan tugas tumpukan Anda sendiri.

Kernel FreeRTOS mencakup lima implementasi heap:

heap_1

Adalah implementasi yang paling sederhana. Tidak mengizinkan memori dibebaskan.

heap_2

Memungkinkan memori untuk dibebaskan, tetapi tidak menyatu blok bebas yang berdekatan.

heap_3

Membungkus `std::malloc()` dan `free()` untuk keamanan benang.

heap_4

Coalesces berdekatan blok bebas 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 intertask

Bagian ini memuat informasi tentang primitif FreeRTOS.

Antrean

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

API antrian mengizinkan waktu pemblokiran ditentukan. Ketika tugas mencoba membaca dari antrian kosong, tugas ditempatkan ke status Diblokir sampai data tersedia pada antrian atau waktu blok berlalu. Tugas dalam keadaan Diblokir tidak menghabiskan waktu CPU, memungkinkan tugas-tugas lain untuk 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 tugas memblokir antrian yang sama, tugas dengan prioritas tertinggi akan dibuka terlebih dahulu.

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

Semaphores dan mutexes

Kernel FreeRTOS menyediakan semaphores biner, menghitung semaphores, dan mutexes untuk tujuan pengecualian dan sinkronisasi bersama.

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

Mutexes adalah semaphore biner yang mencakup mekanisme pewarisan prioritas. Ini berarti bahwa jika tugas prioritas tinggi memblokir saat mencoba untuk mendapatkan mutex yang saat ini dipegang oleh tugas prioritas yang lebih rendah, prioritas tugas memegang token untuk sementara diangkat 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 notifikasi

Pemberitahuan tugas memungkinkan tugas untuk berinteraksi dengan tugas-tugas lain, dan untuk menyinkronkan dengan rutinitas layanan interupsi (ISR), 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 semaphore biner dan menghitung dan, dalam beberapa kasus, antrian. Notifikasi tugas

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

Buffer aliran

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

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

Mengirim data

`xStreamBufferSend()` digunakan untuk mengirim data ke buffer aliran dalam tugas.

`xStreamBufferSendFromISR()` digunakan untuk mengirim data ke buffer aliran dalam rutinitas layanan interupsi (ISR).

`xStreamBufferSend()` memungkinkan waktu blok yang akan ditentukan.

Jika `xStreamBufferSend()` dipanggil dengan waktu blok non-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 disebut (secara internal, oleh API FreeRTOS) 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 memberikan implementasi Anda sendiri `sbSEND_COMPLETED()` di [FreeRTOSConfig.h](#). Hal ini berguna ketika buffer aliran digunakan untuk melewati data antara core pada prosesor multicore. Dalam

skenario itu, `sbSEND_COMPLETED()` dapat diimplementasikan untuk menghasilkan interupsi di inti CPU lainnya, dan rutinitas 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 yang akan ditentukan.

Jika `xStreamBufferReceive()` dipanggil dengan waktu blok non-nol untuk membaca dari buffer aliran dan buffer kosong, tugas ditempatkan ke dalam keadaan Diblokir sampai jumlah data tertentu tersedia dalam buffer aliran, atau waktu blok berakhir.

Jumlah data yang harus dalam buffer aliran sebelum tugas diblokir disebut tingkat pemacu buffer aliran. Tugas yang diblokir dengan tingkat pemacu 10 tidak diblokir ketika setidaknya 10 byte ditulis ke buffer atau waktu pemblokiran tugas berakhir. Jika waktu pemblokiran tugas pembacaan berakhir sebelum tingkat pemacu tercapai, tugas akan menerima data apa pun yang ditulis ke buffer. Tingkat pemacu tugas harus diatur ke nilai antara 1 dan ukuran buffer aliran. Tingkat pemacu buffer aliran diatur ketika `xStreamBufferCreate()` dipanggil. Itu dapat diubah dengan menelepon `xStreamBufferSetTriggerLevel()`.

`sbRECEIVE_COMPLETED()` dan `sbRECEIVE_COMPLETED_FROM_ISR()` merupakan makro yang disebut (secara internal, oleh API FreeRTOS) 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 terpisah dengan 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 dapat ditulis ke, dan dibaca dari, buffer pesan yang sama. Pesan 10-byte hanya dapat dibaca sebagai pesan 10-byte, bukan sebagai byte individu. Buffer pesan dibangun di atas implementasi penyangga aliran. Anda dapat mengaktifkan fungsionalitas buffer pesan dengan menyertakan `filestream_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 stream yang menempatkan tugas panggilan ke status Diblokir dapat mengubah status dan nilai notifikasi tugas panggilan.

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 `tipysize_t`, yang biasanya 4 byte pada arsitektur 32-byte. Oleh karena itu, menulis pesan 10-byte ke dalam buffer pesan benar-benar mengkonsumsi 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 rutinitas layanan interupsi (ISR).

`xMessageBufferSend()` memungkinkan waktu blok yang akan ditentukan.

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

`sbSEND_COMPLETED()` dan `sbSEND_COMPLETED_FROM_ISR()` merupakan makro yang disebut (secara internal, oleh API FreeRTOS) 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 memberikan implementasi Anda sendiri `sbSEND_COMPLETED()` di [FreeRTOSConfig.h](#). Hal ini berguna ketika buffer aliran digunakan untuk melewati data antara core pada prosesor multicore. Dalam skenario itu, `sbSEND_COMPLETED()` dapat diimplementasikan untuk menghasilkan interupsi di inti CPU lainnya, dan rutinitas layanan interupsi kemudian dapat menggunakan `xStreamBufferSendCompletedFromISR()` API untuk memeriksa, dan jika perlu membuka blokir, tugas yang menunggu data.

Menerima data

`xMessageBufferReceive()` digunakan untuk membaca data dari buffer pesan dalam tugas. `xMessageBufferReceiveFromISR()` digunakan untuk membaca data dari buffer pesan dalam rutinitas layanan interupsi (ISR). `xMessageBufferReceive()` memungkinkan waktu blok yang akan ditentukan. Jika `xMessageBufferReceive()` dipanggil dengan waktu blok non-nol untuk membaca dari buffer pesan dan buffer kosong, tugas ditempatkan ke dalam keadaan Diblokir sampai salah satu data tersedia, atau waktu blok berakhir.

`sbRECEIVE_COMPLETED()` dan `sbRECEIVE_COMPLETED_FROM_ISR()` merupakan makro yang disebut (secara internal, oleh API FreeRTOS) 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 instance kernel](#) FreeRTOS untuk menjadwalkan tugas di beberapa inti prosesor yang identik. Arsitektur inti harus identik dan berbagi memori yang sama.

Memodifikasi aplikasi untuk menggunakan kernel FreeRtos-SMP

API FreeRTOS tetap sama antara versi single-core dan SMP, kecuali untuk [API tambahan ini](#). 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.

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

Satu asumsi umum lainnya adalah bahwa ISR tidak dapat berjalan secara bersamaan satu sama lain atau dengan tugas lainnya. Hal 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 ISR.

Pengatur waktu perangkat lunak

Sebuah timer perangkat lunak memungkinkan fungsi yang akan dijalankan pada waktu yang ditetapkan di future. Fungsi yang dijalankan oleh timer disebut fungsi callback timer. Waktu antara timer yang dimulai dan fungsi callback yang dijalankan disebut periode timer. Kernel FreeRTOS menyediakan implementasi pengatur waktu perangkat lunak yang efisien karena:

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

Dukungan daya rendah

Seperti kebanyakan sistem operasi tertanam, kernel FreeRTOS menggunakan timer perangkat keras untuk menghasilkan interupsi centang berkala, yang digunakan untuk mengukur waktu. Penghematan daya implementasi pengatur waktu perangkat keras reguler dibatasi oleh kebutuhan untuk keluar secara berkala dan kemudian masuk kembali ke status daya rendah untuk memproses interupsi tick. Jika frekuensi interupsi tick 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 yang paling ringan.

Untuk mengatasi batasan ini, FreeRTOS menyertakan mode pengatur waktu tickless untuk aplikasi berdaya rendah. Mode idle tickless FreeRTOS menghentikan interupsi tick 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 untuk tetap dalam keadaan hemat daya yang dalam sampai terjadi interupsi, atau sekarang saatnya kernel RTOS untuk mentransisikan tugas ke status siap.

Konfigurasi kernel

Anda dapat mengonfigurasi kernel FreeRTOS untuk papan dan aplikasi tertentu dengan file `FreeRTOSConfig.h` 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 demo FreeRTOS dan aplikasi pengujian terletak di `freertos/vendors/vendor/boards/board/aws_demos/config_files/FreeRTOSConfig.h` dan `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 SDK Perangkat 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 open source 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 AWS IoT Device Shadow, AWS IoT Jobs, AWS IoT Fleet Provisioning, dan AWS IoT Device Defender perpustakaan. SDK ini didistribusikan dalam bentuk sumber dan dapat dibangun ke 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 meng-hostingnya pada jenis prosesor apa pun (misalnya, MCU dan MPU). Namun, jika perangkat Anda memiliki memori yang cukup dan sumber daya pemrosesan, sebaiknya gunakan salah satu [SDK AWS IoT Perangkat](#) dengan urutan yang lebih tinggi.

Untuk informasi selengkapnya, lihat yang berikut:

- [AWS IoT SDK Perangkat untuk Embedded C](#)
- [AWS IoT SDK Perangkat untuk Embedded C aktif GitHub](#)
- [AWS IoT SDK Perangkat untuk Embedded C Readme](#)
- [AWS IoT SDK Perangkat untuk Sampel C Tertanam](#)

IO umum

API IO umum bertindak sebagai lapisan abstraksi perangkat keras (HAL) yang menyediakan antarmuka umum antara driver dan kode aplikasi tingkat tinggi. FreeRTOS Common IO menyediakan serangkaian API standar untuk mengakses perangkat serial umum pada papan referensi yang didukung; implementasi API ini tidak disertakan. API umum ini 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 silikon-vendor tertentu.

Note

FreeRTOS tidak memerlukan implementasi API IO Umum agar berfungsi, tetapi ia akan mencoba menggunakan API IO Umum sebagai cara untuk berinteraksi dengan periferal tertentu pada papan berbasis mikrokontroler, bukan API khusus vendor.

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

Pustaka

Saat ini, FreeRTOS menyediakan dua perpustakaan IO umum: IO umum - IO dasar dan umum - BLE.

IO umum - dasar

Gambaran Umum

[Common IO - basic](#) menyediakan API yang berhubungan dengan periferal I/O dasar dan fungsi yang mungkin Anda temukan di papan berbasis MCU. Common IO - repositori dasar tersedia pada [GitHub](#)

Periferal yang Didukung

- ADC

- GPIO
- I2C
- PWM
- SPI
- UART
- Pengawas
- Flash
- RTC
- EFUSE
- Mengatur ulang
- I2S
- Konter kinerja
- Informasi platform perangkat keras

Fitur yang didukung

- baca/tulis sinkron

Fungsi tidak kembali sampai jumlah data yang diminta ditransfer.

- baca/tulis asinkron

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

Spesifik periferai

- 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. Common IO - BLE repositori tersedia pada [GitHub](#)

Manajer 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 GAP API yang khusus untuk BLE.

Manajer Adaptor Klasik Bluetooth:

Ini menyediakan antarmuka untuk mengontrol fungsi klasik BT perangkat.

Server 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

API Common IO adalah bagian dari implementasi yang diperlukan oleh [Amazon Common Software for Devices, khususnya untuk](#) diimplementasikan dalam kit port perangkat vendor (DPK).

Apa itu ACS?

Amazon Common Software (ACS) for Devices adalah perangkat lunak yang membuatnya lebih cepat bagi Anda untuk mengintegrasikan Amazon Device SDK pada 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 [Amazon Common Software for Devices](#) memverifikasi bahwa build dari ACS DPK (Device Porting Kit) yang berjalan pada papan pengembangan berbasis mikrokontroler tertentu kompatibel dengan praktik terbaik program yang dipublikasikan dan cukup kuat untuk lulus pengujian yang dimandatkan 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 FreeRTOS

Topik:

- [Memulai AWS IoT dan FreeRTOS menggunakan Quick Connect](#)
- [Eksplorasi pustaka FreeRTOS](#)
- [Memahami cara membangun AWS IoT produk yang aman dan tangguh](#)
- [Kembangkan produk AWS IoT aplikasi Anda](#)

Memulai AWS IoT dan FreeRTOS menggunakan Quick Connect

Untuk menjelajahi 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 disertakan dengan demo Quick Connect menggunakan sistem dan alat pembuatan papan yang dipilih untuk terhubung ke AWS akun Anda. Alur data dari AWS IoT konsol di akun Anda terlihat sekarang.

Eksplorasi pustaka FreeRTOS

Setelah Anda memahami bagaimana perangkat IoT dan AWS IoT bekerja sama, Anda dapat mulai menjelajahi pustaka [FreeRTOS](#), dan pustaka [Support Jangka Panjang \(LTS\)](#).

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

- [Kernel FreeRTOS](#)
- [CoreMQTT](#)
- [AWS IoT Di udara \(OTA\)](#)

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

Memahami cara membangun AWS IoT produk yang aman dan tangguh

Lihat [AWS IoT Integrasi FreeRTOS 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 produk 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 GitHub repositori [FreeRTOS-LTS](#). 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, menyiapkan lingkungan pengembangan, dan mengintegrasikan pustaka FreeRTOS ke dalam proyek Anda. Gunakan GitHub repositori [FreeRtos-Libraries-Integration-Tests](#) untuk memvalidasi porting.

AWS IoT Device Tester untuk FreeRTOS

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

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

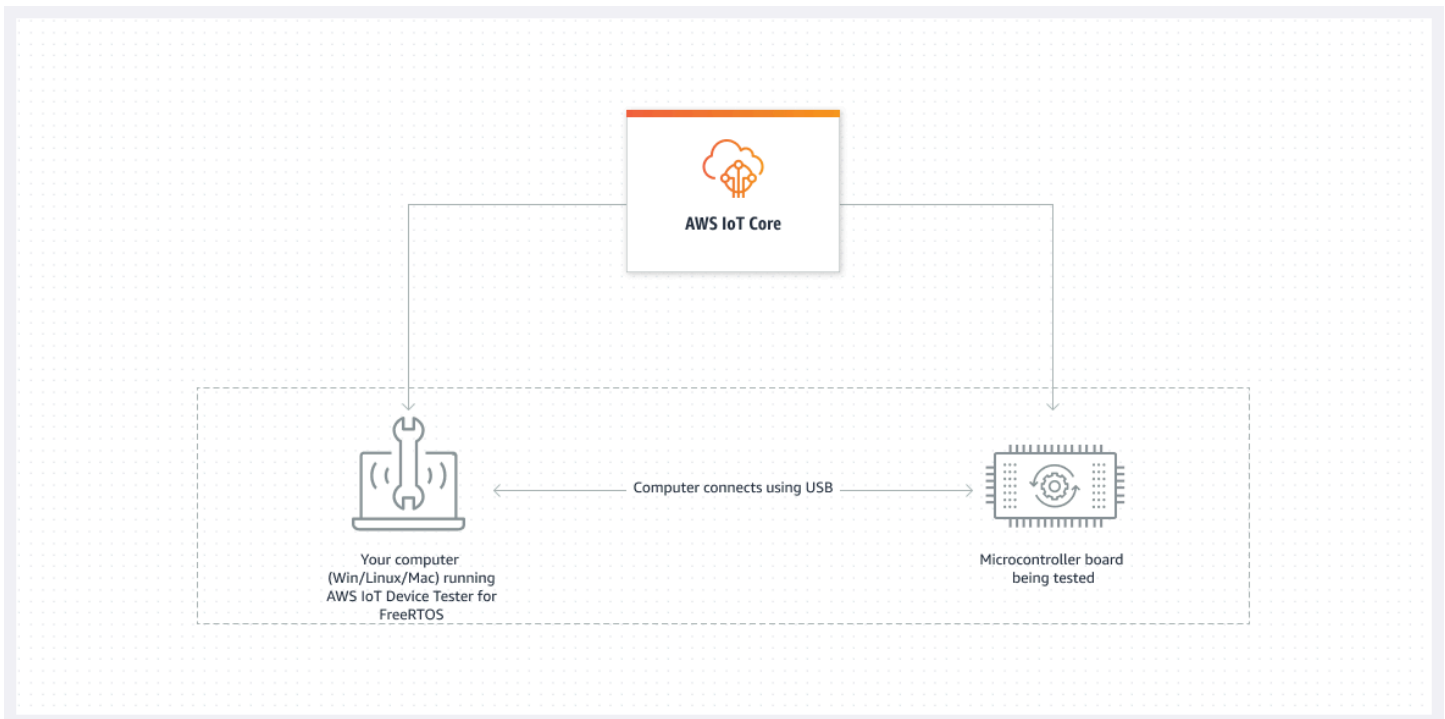
Suite kualifikasi FreeRTOS

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

[Suite FreeRTOS qualification \(FRQ\) 2.0 menggunakan kasus pengujian dari FreeRTOS-Libraries-Integration-Tests dan Device Advisor yang ditentukan dalam Panduan Kualifikasi FreeRTOS.](#)

IDT untuk FreeRTOS menghasilkan laporan pengujian yang dapat Anda kirimkan ke Jaringan AWS Mitra (APN) untuk memasukkan 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 bekerja dengan versi FreeRTOS tertentu.
- Kelompok uji adalah kumpulan kasus uji individual yang terkait dengan fitur tertentu, seperti pesan BLE dan MQTT.

Untuk informasi selengkapnya, lihat [Versi rangkaian tes](#)

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

Cara mengonfigurasi rangkaian pengujian khusus menentukan konfigurasi pengaturan yang harus Anda berikan kepada pengguna untuk menjalankan rangkaian pengujian khusus Anda. Untuk

informasi selengkapnya, lihat [Gunakan IDT untuk mengembangkan dan menjalankan rangkain tes Anda sendiri](#).

Versi yang didukung dari AWS IoT Device Tester untuk FreeRTOS

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

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

Note

Saat 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 AWS IoT Kualifikasi FreeRTOS (FRQ) 1.0 tidak menghasilkan laporan kualifikasi yang ditandatangani. Anda tidak dapat memenuhi syarat baru AWS IoT Perangkat FreeRTOS untuk dicantumkan di [AWS Katalog Perangkat Mitra](#) melalui [AWS Program Kualifikasi Perangkat](#) menggunakan versi 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 untuk menggunakan [IDT FRQ 2.0](#) untuk memenuhi syarat dan daftar perangkat FreeRTOS di [AWS Katalog Perangkat Mitra](#).

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 rangkaian tes	Versi FreeRTOS yang didukung	Tautan unduhan	Tanggal rilis	Catatan rilis
IDT v4.9.0	FRQ_2.5.0	<ul style="list-style-type: none"> • 202112.00 • 202212.00 • 202212.01 • Semua tambalan FreeRTOS 202210-LTS yang menggunakan pustaka FreeRTOS LTS. 	<ul style="list-style-type: none"> • Linux • macOS • Windows 	2023.04.04	<ul style="list-style-type: none"> • Mendukung pengujian terhadap FreeRTOS 202112, 202210, 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 <code>manifest.yml</code>. • Peningkatan waktu uji OTA E2E. • Membatasi jumlah perangkat

AWS IoT Device Tester versi	Versi rangkaian tes	Versi FreeRTOS yang didukung	Tautan unduhan	Tanggal rilis	Catatan rilis
					<p>yang tercantum dalam <code>device.json</code> untuk 1.</p> <ul style="list-style-type: none"> • Penyempurnaan dan perbaikan bug kecil.

Note

Kami tidak menyarankan untuk dijalankan oleh beberapa pengguna 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 untuk FreeRTOS

IDT untuk FreeRTOS berikut juga didukung.

Versi sebelumnya dari AWS IoT Device Tester untuk FreeRTOS

AWS IoT Device Tester versi	Versi rangkaian tes	Versi FreeRTOS yang didukung	Tautan unduhan	Tanggal rilis	Catatan rilis
IDT v4.8.1	FRQ_2.4.0	<ul style="list-style-type: none"> • 202112.00 • 202212.00 • 202212.01 	<ul style="list-style-type: none"> • Linux • macOS • Windows 	2023.01.23	<ul style="list-style-type: none"> • Lihat README.MD untuk informasi

AWS IoT Device Testerversi	Versi rangkaian tes	Versi FreeRTOS yang didukung	Tautan unduhan	Tanggal rilis	Catatan rilis
		<ul style="list-style-type: none">Semua tambalan FreeRTOS 202210-LTS yang menggunakan pustaka FreeRTOS LTS.			<p>lebih lanjut. Anda harus menyertakan versi patch untuk Freertos-LTS dimanifest.yml .</p> <ul style="list-style-type: none">Penyempurnaan dan perbaikan bug kecil.

AWS IoT Device Testerversi	Versi rangkaian tes	Versi FreeRTOS yang didukung	Tautan unduhan	Tanggal rilis	Catatan rilis
IDT v4.6.0	FRQ_2.3.0	<ul style="list-style-type: none"> • 202112.00 • 202212.00 • 202212.01 • 202210-LTS yang menggunakan pustaka FreeRTOS LTS. 	<ul style="list-style-type: none"> • Linux • macOS • Windows 	2022.11.16	<ul style="list-style-type: none"> • Lihat README.MD untuk informasi lebih lanjut. Anda harus menyertakan versi patch untuk Freertos-LTS dimanifestasikan dalam <code>ym1</code>. • Untuk informasi lebih lanjut tentang apa yang termasuk dalam FreeRTOS202210-LTSrilis, lihat ChangelOG.md berkas di GitHub. • Menambahkan kemampuan untuk mengkonfigurasi dan menjalank

AWS IoT Device Tester versi	Versi rangkaian tes	Versi FreeRTOS yang didukung	Tautan unduhan	Tanggal rilis	Catatan rilis
					<p>anAWS IoT Device Tester untuk FreeRTOS melalui antarmuka pengguna berbasis web. Lihat Gunakan antarmuka pengguna IDT for FreeRTOS untuk menjalankan suite kualifikasi FreeRTOS 2.0 (FRQ 2.0) untuk memulai.</p> <ul style="list-style-type: none"> • Menambahkan opsi untuk menyimpan salinan modifikasi dari kode sumber yang

AWS IoT Device Testerversi	Versi rangkaian tes	Versi FreeRTOS yang didukung	Tautan unduhan	Tanggal rilis	Catatan rilis
					<p>dibuat dan digunakan saat runtime untuk debugging pasca-pengujian. Lihat Mengonfigurasi pengaturan build, flash, dan pengujian untuk informasi selengkapnya.</p> <ul style="list-style-type: none"> Menambahkan dukungan SDK Klien IDT untuk Java. Untuk informasi selengkapnya tentang IDT Client SDK,

AWS IoT Device Testerversi	Versi rangkaian tes	Versi FreeRTOS yang didukung	Tautan unduhan	Tanggal rilis	Catatan rilis
					lihat Gunakan IDT untuk mengembankan dan menjalankan rangkaian tes Anda sendiri.

AWS IoT Device Testerversi	Versi rangkaian tes	Versi FreeRTOS yang didukung	Tautan unduhan	Tanggal rilis	Catatan rilis
IDT v4.5.11	FRQ_2.2.0	<ul style="list-style-type: none"> • 202112.00 • 202212.00 • 202212.01 • 202210-LTS yang menggunakan pustaka FreeRTOS LTS. 	<ul style="list-style-type: none"> • Linux • macOS • Windows 	2022.10.14	<ul style="list-style-type: none"> • Lihat README.MD untuk informasi lebih lanjut. Anda harus menyertakan versi patch untuk Freertos-LTS dimanifest.yml. • Untuk informasi lebih lanjut tentang apa yang termasuk dalam FreeRTOS202210-LTSrilis, lihat ChangelOG.md berkas di GitHub. • Penyempurnaan dan perbaikan bug kecil.

Untuk informasi selengkapnya, lihat [Kebijakan dukungan AWS IoT Device Tester untuk FreeRTOS](#).

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 [Kebijakan dukungan AWS IoT Device Tester untuk FreeRTOS](#).

Versi IDT FreeRTOS berikut ini tidak lagi didukung.

Versi FreeRTOS yang AWS IoT Device Tester tidak didukung

AWS IoT Device Testerversi	Versi rangkaian tes	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
IDT v4.10	FRQ_2.1.4	<ul style="list-style-type: none"> 202112.00 202012-LTS yang menggunakan pustaka FreeRTOS LTS. 	2022.09.02	<ul style="list-style-type: none"> Untuk informasi selengkapnya tentang apa yang disertakan dalam rilis FreeRTOS 202012-LTS, lihat file ChangeLog.md aktif. GitHub Menyelesaikan masalah yang memengaruhi kelompok OTA End to End pengujian. Dihapus FullTransportInterfacePlainText dari berjalan di kualifikasi

AWS IoT Device Testerversi	Versi rangkaian tes	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
				<p>berjalan.</p> <p>Teks biasa masih dapat dijalankan sebagai grup uji pengembangan dengan menggunakan <code>- \-group-id</code> bendera.</p> <ul style="list-style-type: none">• Meningkatkan logging dan keterbacaan konsol dan output file.• Penyempurnaan dan perbaikan bug kecil.

AWS IoT Device Testerversi	Versi rangkaian tes	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
IDT v4.5.9	FRQ_2.1.3	<ul style="list-style-type: none"> • 202112.00 • 202012.04-LTS yang menggunakan pustaka FreeRTOS LTS. 	2022.08.17	<ul style="list-style-type: none"> • Untuk informasi selengkapnya tentang apa yang disertakan dalam rilis FreeRTOS 202012.04-LTS, lihat file ChangeLog.md aktif. GitHub • Menyelesaikan masalah yang memengaruhi kelompok FreeRTOS Integrity pengujian. • Kelompok FullCloud IoT uji yang diperbarui dengan menghapus kasus uji “MQTT Connect Exponential Backoff Retries”. • Penyempurnaan dan

AWS IoT Device Testerversi	Versi rangkaian tes	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
				perbaikan bug kecil.

AWS IoT Device Testerversi	Versi rangkaian tes	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
IDT v4.6	FRQ_2.1.2	<ul style="list-style-type: none"> • 202112.00 • 202012.04-LTS yang menggunakan pustaka FreeRTOS LTS. 	2022.06.29	<ul style="list-style-type: none"> • Untuk informasi selengkapnya tentang apa yang disertakan dalam rilis FreeRTOS 202012.04-LTS, lihat file ChangeLog.md aktif. GitHub • Menambahkan kelompok uji baru FullCloud IoT yang menguji papan terhadap AWS IoT Core Device Advisor. • Menyelesaikan masalah yang memengaruhi kasus uji OTA E2E. • Penyempurnaan dan perbaikan bug kecil.

AWS IoT Device Testerversi	Versi rangkaian tes	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
IDT v4.5.5	FRQ_2.1.1	<ul style="list-style-type: none"> • 202112.00 • 202012.04-LTS yang menggunakan pustaka FreeRTOS LTS. 	2022.06.06	<ul style="list-style-type: none"> • Untuk informasi selengkapnya tentang apa yang disertakan dalam rilis FreeRTOS 202012.04-LTS, lihat file <code>ChangeLog.md</code> aktif. GitHub • Menambahkan kelompok uji baru FullCloud IoT yang menguji papan terhadap AWS IoT Core Device Advisor. • Menyelesaikan masalah yang memengaruhi kasus uji FreeRtosVersion dan FreeRtosIntegrity. • Penyempurnaan dan

AWS IoT Device Testerversi	Versi rangkaian tes	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
				perbaikan bug kecil.
IDT v4.5.5	FRQ_2.1.0	<ul style="list-style-type: none"> • 202107.00 • 202112.00 • 202012.04-LTS yang menggunakan pustaka FreeRTOS LTS. 	2022.05.31	<ul style="list-style-type: none"> • Untuk informasi selengkapnya tentang apa yang disertakan dalam rilis FreeRTOS 202012.04-LTS, lihat file <u>ChangeLog.md</u> aktif. GitHub • Menambahkan kelompok uji baru FullCloud IoT yang menguji papan terhadap AWS IoT Core Device Advisor. • Penyempurnaan dan perbaikan bug kecil.

AWS IoT Device Testerversi	Versi rangkaian tes	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
IDT v4.5.4	FRQ_2.0.0	<ul style="list-style-type: none"> • 202112.00 • 202012.04-LTS yang menggunakan pustaka FreeRTOS LTS. 	2022.05.09	<ul style="list-style-type: none"> • Untuk informasi selengkapnya tentang apa saja yang disertakan dalam rilis FreeRTOS 202012.04-LTS, lihat file ChangeLog.md aktif. GitHub • Menghapus persyaratan untuk memenuhi syarat papan hanya menggunakan versi Amazon FreeRTOS dari repositori aws/amazon-freertos GitHub • Penyempurnaan dan perbaikan bug kecil.

AWS IoT Device Testerversi	Versi rangkaian tes	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
IDT v4.5.2	FRQ_1.6.2	202107.00	2022.01.25	<ul style="list-style-type: none"> • Untuk informasi selengkapnya tentang apa saja yang disertakan dalam rilis FreeRTOS 202107.00, lihat file ChangeLog.md aktif. GitHub • Mengimplementasikan orkestrator uji IDT baru untuk mengonfigurasi rangkaian pengujian khusus. Untuk informasi selengkapnya, lihat Mengonfigurasi orkestrator pengujian IDT. • Penyempurnaan dan perbaikan bug kecil.

AWS IoT Device Testerversi	Versi rangkaian tes	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
IDT v4.0.3	FRQ_1.5.1	202012.00	2021.07.30	<ul style="list-style-type: none">• Support untuk kualifikasi perangkat dengan kredensi terkunci pada Modul Keamanan Perangkat Keras.• Penyempurnaan dan perbaikan bug kecil.

AWS IoT Device Testerversi	Versi rangkaian tes	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
IDT v4.3.0	FRQ_1.6.1	202107.00	2021.07.26	<ul style="list-style-type: none">• Untuk informasi selengkap nya tentang apa saja yang disertakan dalam rilis FreeRTOS 202107.00 , lihat file ChangeLog.md aktif. GitHub• Menambahkan kemampuan untuk mengkonfi gurasi dan menjalank an AWS IoT Device Tester FreeRTOS melalui antarmuka pengguna berbasis web. Lihat Gunakan antarmuka pengguna IDT untuk FreeRTOS untuk menjalank

AWS IoT Device Testerversi	Versi rangkaian tes	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
				an rangkaian kualifikasi FreeRTOS untuk memulai.

AWS IoT Device Testerversi	Versi rangkaian tes	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
IDT v4.1.0	FRQ_1.6.0	202107.00	2021.07.21	<ul style="list-style-type: none"> • Untuk informasi selengkap nya tentang apa saja yang disertakan dalam rilis FreeRTOS 202107.00 , lihat file ChangeLog.md aktif. GitHub • Menghapus kasus uji berikut dari kualifikasi OTA: <ul style="list-style-type: none"> • Agen OTA • Nama File yang Hilang • OTA Max Jumlah Blok yang Dikonfigurasi • Menghapus grup Both uji OTA Dataplane dari Kualifikasi OTA. Dalam

AWS IoT Device Testerversi	Versi rangkaian tes	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
				<p>device.js on file, OTADataPlaneProtocol konfigurasi sekarang hanya menerima HTTP atau MQTT sebagai nilai yang didukung.</p> <ul style="list-style-type: none"> • Mengimplementasikan perubahan berikut pada <code>freertosFileConfiguration</code> konfigurasi dalam userdata.json file untuk perubahan kode sumber FreeRTOS: <ul style="list-style-type: none"> • Mengubah nama file yang ditentukan untuk <code>otaAgentT</code>

AWS IoT Device Testerversi	Versi rangkaian tes	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
				<p>estsConfig dan otaAgentDemosConfig dari aws_ota_agent_config.h keota_config.h .</p> <ul style="list-style-type: none"> • Menambahkan konfigurasi otaDemosConfig opsional baru untuk menentukan path file ke ota_demo_config.h file baru. • Menambahkan bidang baru testStartDelays userdata.json untuk menentukan penundaan antara waktu perangkat di-flash untuk

AWS IoT Device Testerversi	Versi rangkaian tes	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
				menjalankan grup pengujian FreeRTOS dan saat mulai menjalankan pengujian . Nilai harus diberikan dalam milidetik . Penundaan ini dapat digunakan untuk memberikan IDT kesempatan untuk terhubung sehingga tidak ada output tes yang terlewatkan.

AWS IoT Device Testerversi	Versi rangkaian tes	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
IDT v4.0.1	FRQ_1.4.1	202012.00	2021.01.19	<ul style="list-style-type: none"> • Untuk informasi selengkap nya tentang apa saja yang disertakan 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

AWS IoT Device Testerversi	Versi rangkaian tes	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
				<p>untuk kualifikasi papan pengembangan FreeRTOS menggunakan konektivitas seluler.</p> <ul style="list-style-type: none"> • Memperbaiki bug dalam konfigurasi server echo. • Memungkinkan Anda untuk mengembangkan dan menjalankan rangkaian tes kustom Anda sendiri dengan menggunakan AWS IoT Device Tester untuk FreeRTOS. Untuk informasi selengkapnya, lihat Gunakan IDT untuk mengembangkan dan

AWS IoT Device Testerversi	Versi rangkaian tes	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
				<p>menjalankan rangkaian tes Anda sendiri.</p> <ul style="list-style-type: none">• Menyediakan aplikasi IDT yang ditandatangani kode, sehingga Anda tidak perlu memberikan izin saat menjalankannya di Windows atau macOS.• Sempurnakan hasil tes BLE parsing logika.

AWS IoT Device Testerversi	Versi rangkaian tes	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
IDT v3.4.0	FRQ_1.3.0	202011.01	2020.11.05	<ul style="list-style-type: none">• Untuk detail selengkapnya, lihat file ChangeLog.md di GitHub.• Fixed bug di mana 'RSA' bukan opsi konfigurasi PKCS11 valid.• Bug tetap di mana bucket Amazon S3 tidak dibersihkan dengan benar setelah pengujian OTA.• Pembaruan untuk mendukung kasus uji baru di dalam grup uji FullMQTT.

AWS IoT Device Testerversi	Versi rangkaian tes	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
IDT v3.3.0	FRQ_1.2.0	202007.00	2020.09.17	<ul style="list-style-type: none"> • Untuk detail selengkapnya, lihat file ChangeLog.md di GitHub. • Tes end-to-end baru untuk memvalidasi fitur penangguhan dan resume pembaruan Over The Air (OTA). • Fixed bug menyebabkan penggunaan pengguna di eu-central-1 Region tidak dapat lulus validasi konfigurasi untuk tes OTA. • Menambahkan --update-idt parameter ke run-suite perintah. Anda dapat menggunakan opsi ini untuk mengatur

AWS IoT Device Testerversi	Versi rangkaian tes	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
				<p>respons untuk prompt pembaruan IDT.</p> <ul style="list-style-type: none"> • Menambahkan --update-managed-policy parameter ke run-suite perintah. Anda dapat menggunakan opsi ini untuk mengatur respons untuk prompt pembaruan kebijakan terkelola. • Peningkatan internal dan perbaikan bug, termasuk: <ul style="list-style-type: none"> • Untuk pembaruan rangkaian pengujian otomatis, peningkatan peningkatan file konfigurasi.

AWS IoT Device Testerversi	Versi rangkaian tes	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
IDT v3.0.2	FRQ_1.0.1	202002.00		<ul style="list-style-type: none"> • Untuk informasi selengkapnya, lihat file ChangeLog.md di GitHub • Menambahkan update otomatis test suite dalam IDT. IDT sekarang dapat mengunduh suite pengujian terbaru yang tersedia untuk versi FreeRTOS. Dengan fitur ini, Anda dapat: <ul style="list-style-type: none"> • Unduh test suite terbaru menggunakan <code>upgrade-test-suite</code> perintah. • Unduh rangkaian

AWS IoT Device Testerversi	Versi rangkaian tes	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
				<p>pengujian terbaru dengan menetapkan bendera saat Anda memulai IDT.</p> <p>Gunakan -u <i>flag</i> opsi di mana <i>bendera</i> dapat 'y' untuk selalu mengunduh atau 'n' untuk menggunakan versi yang ada.</p> <p>Jika tersedia beberapa versi rangkaian pengujian, versi terbaru akan digunakan kecuali Anda menentukan ID rangkaian</p>

AWS IoT Device Testerversi	Versi rangkaian tes	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
				<p>pengujian saat memulai IDT.</p> <ul style="list-style-type: none"> • Gunakan <code>list-supported-versions</code> opsi baru untuk daftar FreeRTOS dan pengujian suite versi yang didukung oleh versi diinstal IDT. • Buat daftar kasus uji dalam grup dan jalankan pengujian individual. <p>Test suite berversi menggunakan <code>major minor.patchFormat</code> mulai dari 1.0.0.</p>

AWS IoT Device Testerversi	Versi rangkaian tes	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
				<ul style="list-style-type: none"> • Menambahkan <code>list-supported-products</code> perintah - Mencantumkan FreeRTOS dan pengujian suite versi yang didukung oleh versi diinstal IDT. • Menambahkan <code>list-test-cases</code> perintah - Mencantumkan kasus pengujian yang tersedia dalam grup pengujian. • Menambahkan <code>test-id</code> opsi untuk <code>run-suite</code> perintah - Gunakan opsi ini untuk menjalankan kasus pengujian

AWS IoT Device Testerversi	Versi rangkaian tes	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
				individu dalam grup pengujian .

AWS IoT Device Testerversi	Versi rangkaian tes	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
IDT	FRQ_1.0.0	202002.00		<ul style="list-style-type: none">• Untuk detail selengkapnya, lihat file ChangeLog.md di GitHub.• Mendukung metode penandatangan kode khusus untuk kasus uji end-to-end over-the-air (OTA) sehingga Anda dapat menggunakan perintah dan skrip penandatangan kode Anda sendiri untuk menandatangani muatan OTA.• Menambahkan precheck untuk port serial sebelum dimulainya tes. Pengujian akan gagal dengan cepat

AWS IoT Device Testerversi	Versi rangkaian tes	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
				<p>dengan pesan kesalahan yang ditingkatkan jika port serial salah dikonfigurasi dalam <code>device.json</code> file.</p> <ul style="list-style-type: none"> • Menambahkan Kebijakan AWS Terkelola AWSIoTDeviceTesterForFreeRTOSFullAccess dengan izin yang diperlukan untuk dijalankan AWS IoT Device Tester. Jika rilis baru memerlukan izin tambahan, menambahkannya ke kebijakan terkelola ini sehingga Anda tidak perlu update izin IAM.

AWS IoT Device Testerversi	Versi rangkaian tes	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
				<ul style="list-style-type: none"> File bernama AFQ_Report.xml dalam direktori hasil sekarangFRQ_Report.xml .
IDT	FRQ_1.0.0	202002.00		<ul style="list-style-type: none"> Mendukung pengujian opsional untuk OTA melalui HTTPS untuk memenuhi syarat papan pengembangan FreeRTOS Anda. Mendukung titik akhir AWS IoT ATS dalam pengujian. Mendukung kemampuan untuk menginformasikan pengguna pada versi IDT terbaru sebelum memulai test suite.

AWS IoT Device Testerversi	Versi rangkaian tes	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
IDT	FRQ_1.0.0	201910.00		<ul style="list-style-type: none"> • Mendukung kualifikasi perangkat FreeRTOS dengan elemen aman (kunci onboard). • Mendukung port server gema yang dapat dikonfigurasi untuk Soket Aman dan grup uji Wi-Fi. • Mendukung timeout multiplier flag untuk meningkatkan timeout, yang berguna ketika Anda memecahkan masalah untuk kesalahan terkait timeout. • Ditambahkan bug fix untuk log parsing. • Mendukung titik akhir iot

AWS IoT Device Testerversi	Versi rangkaian tes	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
				ats dalam pengujian.
IDT	FRQ_1.0.0	201908.00		<ul style="list-style-type: none"> • Menambahkan dukungan untuk pembaruan perpustakaan dan uji kasus PKCS11 baru. • Memperkenalkan kode kesalahan ditindaklanjuti. Untuk informasi selengkapnya, lihat Kode kesalahan IDT • Kebijakan IAM update digunakan untuk menjalankan IDT.

AWS IoT Device Testerversi	Versi rangkaian tes	Versi FreeRTOS yang didukung	Tanggal rilis	Catatan rilis
IDT	FRQ_1.0.0	201906.00		<ul style="list-style-type: none"> • Menambahkan dukungan untuk menguji Bluetooth Low Energy (BLE). • Peningkatan pengalaman pengguna untuk perintah antarmuka baris perintah IDT (CLI). • Kebijakan IAM update digunakan untuk menjalankan IDT.
FreeRTOS IDT v1.2	FRQ_1.0.0	<ul style="list-style-type: none"> • FreeRTOS • FreeRTOS 		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 mengirimkan 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 AWS IoT Kualifikasi FreeRTOS (FRQ) 1.0 tidak menghasilkan laporan kualifikasi yang ditandatangani. Anda tidak dapat memenuhi syarat baru AWS IoT Perangkat FreeRTOS untuk dicantumkan di [AWS Katalog Perangkat Mitra](#) melalui [AWS Program Kualifikasi Perangkat](#) menggunakan versi 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 [AWS Katalog Perangkat Mitra](#).

Topik

- [Unduh IDT secara manual](#)
- [Unduh IDT secara terprogram](#)

Dengan mengunduh perangkat lunak, Anda menyetujui AWS IoT Device Tester Perjanjian Lisensi yang terdapat 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 mengirimkan versi IDT untuk FreeRTOS. Sebagai praktik terbaik, kami menyarankan Anda menggunakan versi terbaru AWS IoT Device Tester yang mendukung versi target FreeRTOS

Anda. Rilis baru FreeRTOS mungkin mengharuskan Anda mengunduh versi baru AWS IoT Device Tester. Anda menerima pemberitahuan saat Anda memulai uji coba jika AWS IoT Device Tester tidak kompatibel dengan versi FreeRTOS yang Anda gunakan.

Lihat [Versi yang didukung dari AWS IoT Device Tester untuk FreeRTOS](#)

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 sebagai 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 Anda AWS tanda tangan, 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 tersebut mengirimkan versi IDT terbaru.

Respon API

Respon API memiliki format sebagai berikut. TheDownloadURL 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 kredensial yang Anda simpan di `AWS_ACCESS_KEY_ID` dan `AWS_SECRET_ACCESS_KEY` variabel lingkungan. Untuk mengikuti praktik keamanan terbaik, jangan simpan kredensial 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 mengirimkan `jq` untuk mengurai URL unduhan dari respons.

Warning

The `aws-sigv4` flag membutuhkan parameter kueri dari permintaan GET curl berada dalam urutan `HostOs/ProductVersion/TestSuiteType` atau `HostOs/TestSuiteType`. Pesanan yang tidak sesuai, akan mengakibatkan kesalahan mendapatkan tanda tangan yang tidak cocok untuk String Canonical dari API Gateway.

Jika parameter opsional `ProductVersion` disertakan, Anda harus menggunakan versi produk yang didukung seperti yang didokumentasikan dalam [Versi yang didukung AWS IoT Device Tester untuk FreeRTOS](#).

- Ganti `kami-barat-2` dengan Anda Wilayah AWS. Untuk daftar kode Wilayah, 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 `AWStanda tangan` yang Anda tandatangani dan hitung. Untuk informasi lebih lanjut tentang cara menandatangani dan menghitung `AWStanda tangan`, lihat [Penandatanganan AWS Permintaan API](#).

- Ganti `linux` dengan sistem operasi mesin host Anda.
- Ganti `Stempel waktu` Dengan tanggal dan waktu, seperti `20220210T004606Z`.
- Ganti `Tanggal` dengan tanggalnya, seperti `20220210`.
- Ganti `AWSRegion` dengan Anda Wilayah AWS. Untuk daftar kode Wilayah, lihat [Titik akhir regional](#).
- Ganti `AWSSignature` dengan [AWStanda 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 Python [permohonan](#) perpustakaan. Contoh ini diadaptasi dari contoh Python ke [Tanda tangan AWS Permintaan API](#) di AWS Referensi Umum.

- Ganti *kami-barat-2* dengan wilayah Anda. Untuk daftar kode Wilayah, 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)

```

Gunakan IDT dengan suite kualifikasi FreeRTOS 2.0 (FRQ 2.0)

Suite kualifikasi FreeRTOS 2.0 adalah versi terbaru dari suite kualifikasi FreeRTOS. Kami menyarankan pengembang untuk menggunakan FRQ 2.0 karena terdiri dari kasus pengujian yang

relevan untuk memenuhi syarat perangkat yang menjalankan pustaka FreeRTOS Long Term Support (LTS).

IDT untuk FreeRTOS memverifikasi port FreeRTOS pada mikro-controller Anda, dan jika itu berkomunikasi secara efektif dengan. AWS IoT Secara khusus, ini memverifikasi antarmuka lapisan porting dengan pustaka FreeRTOS, dan jika repositori pengujian FreeRTOS diimplementasikan dengan benar. Hal ini juga melakukan pengujian end-to-end dengan AWS IoT Core. Pengujian yang dijalankan oleh IDT untuk FreeRTOS didefinisikan dalam repositori [GitHubFreeRTOS](#).

IDT untuk FreeRTOS menjalankan pengujian sebagai aplikasi tertanam yang berkedip pada perangkat mikrokontroler yang diuji. Gambar biner aplikasi termasuk FreeRTOS, antarmuka FreeRTOS porting, dan driver perangkat papan. Tujuan dari pengujian ini adalah untuk memverifikasi bahwa antarmuka FreeRTOS 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 tercantum di Katalog Perangkat AWS Mitra. Untuk informasi selengkapnya, lihat [Program Kualifikasi Perangkat AWS](#).

IDT untuk FreeRTOS berjalan di komputer host (Windows, macOS, atau Linux) yang terhubung ke perangkat yang sedang diuji. IDT mengkonfigurasi dan mengatur kasus uji dan agregat hasil. IDT 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 AWS kredensi yang dikonfigurasi di `config.json` untuk membuat panggilan API atas nama Anda. Sumber daya ini disediakan pada berbagai waktu selama tes.

Ketika Anda menjalankan IDT untuk FreeRTOS pada komputer host Anda, IDT akan 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

- [Prasyarat](#)

- [Bersiap untuk menguji papan mikrokontroler Anda untuk pertama kalinya](#)
- [Gunakan antarmuka pengguna IDT for FreeRTOS untuk menjalankan suite kualifikasi FreeRTOS 2.0 \(FRQ 2.0\)](#)
- [Menjalankan suite kualifikasi FreeRTOS 2.0](#)
- [Memahami hasil dan log](#)

Prasyarat

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

Persiapkan untuk kualifikasi FreeRTOS

Note

AWS IoT Device Tester untuk FreeRTOS sangat menyarankan untuk 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 pustaka port, pengujian, dan penyiapan `manifest.yml`, lihat Memindahkan [pustaka FreeRTOS di Panduan Porting](#) FreeRTOS. FRQ 2.0 berisi proses kualifikasi yang berbeda. Lihat [Perubahan kualifikasi terbaru di panduan kualifikasi](#) FreeRTOS untuk detailnya.

[Repositori FreeRtos-Libraries-Integration-tests](#) harus ada agar IDT dapat dijalankan. Lihat [README.md](#) tentang cara mengkloning dan port repositori ini ke proyek sumber Anda. `freertos-libraries-integration-tests` harus menyertakan `manifest.yml` lokasi di root proyek Anda, agar IDT dapat dijalankan.

Note

IDT tergantung pada pelaksanaan tes repositori ini. `UNITY_OUTPUT_CHAR` Log keluaran uji dan log perangkat tidak boleh saling bertautan satu sama lain. Lihat [Mengimplementasikan bagian makro penebangan 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 memiliki izin baca dan tulis. Karena Microsoft Windows memiliki batas karakter untuk panjang jalur, ekstrak IDT untuk FreeRTOS ke direktori root seperti C:\ atau 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 sarankan Anda mengekstraksi paket IDT ke drive lokal.

Unduh Git

IDT harus memiliki Git diinstal sebagai prasyarat untuk memastikan integritas kode sumber.

Ikuti petunjuk dalam [GitHub](#) panduan 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` atau `git command not found`, instal atau instal ulang Git dan coba lagi.

Buat dan konfigurasi AWS akun

Note

Suite kualifikasi IDT lengkap hanya didukung dalam hal berikut Wilayah AWS

- US East (N. Virginia)


- US West (Oregon)
- Asia Pacific (Tokyo)
- Europe (Ireland)

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, lanjutkan ke langkah berikutnya. Lain membuat [AWSaccount](#).
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

Template 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 template kebijakan tidak memberikan hak administrator kepada pengguna, izin tersebut dapat digunakan untuk mendapatkan akses administrator ke akun AndaAWS.

- 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 China. Jika Anda berada di wilayah China, gunakan template 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

Templat 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 `IDTFreertoSiamPermissions` sebagai nama kebijakan.
 - vi. Pilih Buat kebijakan.
5. `AWSIoTDeviceTesterForFreeRTOSFullAccess` Lampirkan peran IAM Anda.
- a. Untuk melampirkan izin yang diperlukan untuk peran IAM Anda:
 - i. Pada halaman Izin, pilih Tambahkan izin.
 - ii. Pilih Pasang kebijakan.
 - iii. Cari `AWSIoTDeviceTesterForFreeRTOSFullAccess` kebijakan. Centang kotak.
 - b. Pilih Tambahkan izin.
6. Kredensi ekspor untuk IDT. Lihat [Mendapatkan kredensial peran IAM untuk akses CLI untuk detailnya](#).

Kebijakan terkelola AWS IoT Device Tester

Kebijakan yang `AWSIoTDeviceTesterForFreeRTOSFullAccess` dikelola berisi AWS IoT Device Tester izin berikut untuk pemeriksaan versi, fitur pembaruan auto, dan pengumpulan 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, rangkaian 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) Pasang AWS Command Line Interface

Anda mungkin lebih suka menggunakan AWS CLI untuk melakukan beberapa operasi. Jika Anda belum AWS CLI terpasang, ikuti petunjuk di [Instal AWS CLI](#).

Konfigurasi 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, lihat [AWS Wilayah dan Titik Akhir](#). Untuk informasi selengkapnya tentang `aws configure` lihat [Konfigurasi cepat dengan `aws configure`](#).

Bersiap untuk menguji papan mikrokontroler Anda untuk pertama kalinya

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

Tambahkan layer porting pustaka dan terapkan repositori pengujian FreeRTOS

Untuk mem-port FreeRTOS untuk perangkat Anda, lihat Panduan [Porting FreeRTOS](#). Saat mengimplementasikan repositori pengujian FreeRTOS dan mem-porting layer FreeRTOS, Anda harus menyediakan jalur `manifest.yml` dengan ke setiap pustaka, termasuk repositori pengujian. File ini akan berada di direktori root kode sumber Anda. Lihat [petunjuk file manifest](#) untuk detailnya.

Konfigurasi kredensial AWS Anda

Anda perlu mengonfigurasi AWS kredensial AWS IoT Device Tester agar dapat berkomunikasi dengan Cloud. AWS Untuk informasi selengkapnya, lihat [Menyiapkan AWS Kredensial dan Wilayah untuk Pengembangan](#). AWSKredensi 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>"
  }
}
```

```
}
```

`authAtribut config.json` file memiliki bidang metode yang mengontrol AWS otentikasi, dan dapat dinyatakan sebagai file atau lingkungan. Mengatur bidang ke lingkungan menarik AWS kredensial Anda dari variabel lingkungan mesin host Anda. Mengatur bidang untuk mengajukan impor profil tertentu dari file `.aws/credentials` konfigurasi.

Buat kumpulan perangkat di IDT untuk FreeRTOS

Perangkat yang akan diuji diatur dalam kumpulan perangkat. Setiap kolom perangkat terdiri dari satu atau beberapa perangkat yang identik. Anda dapat mengonfigurasi IDT untuk FreeRTOS untuk menguji satu perangkat, atau beberapa perangkat dalam satu kumpulan. Untuk mempercepat proses kualifikasi, IDT untuk FreeRTOS dapat menguji perangkat dengan spesifikasi yang sama secara parallel. Ini menggunakan metode round-robin untuk menjalankan grup pengujian yang berbeda pada setiap perangkat di kumpulan perangkat.

`device.jsonFile` ini memiliki array di tingkat atas. Setiap atribut array adalah kumpulan perangkat baru. Setiap kumpulan perangkat memiliki atribut array perangkat, yang memiliki beberapa perangkat yang dideklarasikan. Dalam template, ada kolom perangkat dan hanya satu perangkat di kolom 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 kolom yang sama harus memiliki spesifikasi teknis dan SKU yang sama. Untuk mengaktifkan build parallel 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 merferensikan jalur kode sumber dalam perintah build atau flash Anda menggunakan `testdata.sourcePath` variabel. IDT untuk FreeRTOS menggantikan variabel ini dengan jalur sementara kode sumber yang disalin. Untuk informasi selengkapnya, lihat [IDT untuk variabel FreeRTOS](#).

Berikut ini adalah contoh `device.json` file yang digunakan untuk membuat kolom 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 ditetapkan pengguna yang secara unik mengidentifikasi kolom perangkat. Perangkat yang termasuk dalam suatu kolom harus dari jenis yang sama. Ketika serangkaian tes berjalan, perangkat di kolom tersebut digunakan untuk memparalelkan beban kerja.

sku

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

Note

Jika Anda ingin mencantumkan forum Anda di Katalog Perangkat AWS Mitra, SKU yang Anda tentukan di sini harus sesuai dengan SKU yang Anda gunakan dalam proses pencantuman itu.

features

Rangkaian yang berisi fitur perangkat yang didukung. 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 terpercaya 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 uji PKCS11. `Onboard`, `Both` atau `No` diperlukan untuk kualifikasi.
- Gunakan `Onboard` jika papan Anda mendukung kunci pribadi on-board (misalnya, jika perangkat Anda memiliki elemen aman, atau jika Anda lebih suka membuat key pair perangkat dan sertifikat Anda sendiri). Pastikan Anda menambahkan `secureElementConfig` elemen di setiap bagian perangkat dan menempatkan path absolut ke file kunci publik di `publicKeyAsciiFilePath` bidang.
- Gunakan `Both` jika papan Anda mendukung impor kunci pribadi dan pembuatan kunci on-board untuk penyediaan kunci.
- Gunakan `No` jika papan Anda tidak mendukung penyediaan kunci. Nohanya merupakan opsi yang valid ketika perangkat Anda juga telah disediakan sebelumnya.

OTA

Menunjukkan apakah papan Anda mendukung fungsionalitas pembaruan over-the-air (OTA). `OtaDataPlaneProtocolAtribut` menunjukkan protokol dataplane OTA mana yang didukung perangkat. OTA dengan protokol dataplane HTTP atau MQTT diperlukan untuk kualifikasi. Untuk melewati menjalankan pengujian OTA saat menguji, setel fitur OTA ke `No` dan `OtaDataPlaneProtocol` atributnya `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 yang 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` adalah opsional dan tidak perlu disertakan.

Blok berikut ini jalur absolut ke file yang berisi kunci publik hex byte 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 menyalin 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 menyalin 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 diterjemahkan, bernamabase64key, kemudian output ke filepubkey . der:

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

2. Jalankan xxd perintah berikut untuk mengkonversi 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 keAWS IoT Core. Untuk informasi tentang mengunggah sertifikat AndaAWS IoT Core, lihat [sertifikat klien X.509](#) di Panduan Developer. 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 kredensi terkunci, yang tidak dapat mengimpor, membuat, atau menghancurkan objek. Jika atribut ini diatur ke Ya, Anda harus memberikan label pkcs11 yang sesuai.

devices.secureElementConfig.pkcs11JITPCodeVerifyRootCertSupport

(Opsional) Setel ke Ya jika implementasi CorePKCS11 perangkat mendukung penyimpanan untuk JITP. Ini akan memungkinkan pengujian JITP saat codeverify menguji inti PKCS 11, dan memerlukan kunci verifikasi kode, sertifikat JITP, dan sertifikat root label PKCS 11 yang akan disediakan.

identifiers

(Opsional) Susunan pasangan nama-nilai sewenang-wenang. Anda dapat menggunakan nilai-nilai ini dalam perintah build dan flash yang dijelaskan di bagian selanjutnya.

Mengonfigurasi pengaturan build, flash, dan pengujian

IDT untuk FreeRTOS membangun dan mengedipkan 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.

Mengonfigurasi 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"
    ]
  },
  "testStartDelays": 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 ini mencantumkan atribut yang digunakan dalam `userdata.json`:

sourcePath

Jalur ke akar kode sumber FreeRTOS porting.

retainModifiedSourceDirectories

(Opsional) Memeriksa apakah untuk mempertahankan direktori sumber yang dimodifikasi yang digunakan selama membangun dan berkedip untuk tujuan debugging. Jika diatur ke `true`, direktori sumber yang dimodifikasi diberi nama `RetainedSRC` dan ditemukan dalam folder log hasil di setiap grup pengujian yang dijalankan. Jika tidak disertakan, bidang default ke `false`

freeRTOSTestParamConfigPath

Jalur ke `test_param_config.h` file integrasi `Freertos-Libraries-Integration-Tests`. 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 integrasi `Freertos-Libraries-Integration-Tests`. 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 AWS IoT Device Tester FreeRTOS.

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 mereferensikan skrip flash relatif ke jalur AWS IoT Device Tester root.

Note

Integrasi baru menguji struktur dengan FRQ 2.0 tidak memerlukan variabel jalur seperti `{{enableTests}}` dan `{{buildImageName}}`. Pengujian OTA End to End dijalankan dengan template konfigurasi yang disediakan di repositori [GitHubFreertos-Libraries-Integration-Tests](#). Jika file di GitHub repositori ada di proyek sumber induk Anda, kode sumber tidak berubah di antara pengujian. Jika gambar build yang berbeda untuk OTA End to End diperlukan, Anda harus membuat gambar ini dalam skrip build dan menentukannya dalam `userdata.json` file yang ditentukan di `bawahotaConfiguration`.

testStartDelays

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

echoServerConfiguration

Konfigurasi untuk mengatur server echo untuk pengujian TLS. Bidang ini wajib diisi.

keyGenerationMethod

Server echo 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

Path ke image bin OTA yang digunakan IDT untuk pengujian OTA End to End.

otaPALCertificatePath

Jalur ke sertifikat untuk uji OTA PAL pada perangkat. IDT 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 gambar boot firmware.

codeSigningConfiguration

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 tersebut.

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

untrustedSignerCertificate

ARN atau filepath untuk sertifikat kedua yang digunakan dalam beberapa tes OTA sebagai sertifikat yang tidak dipercaya. 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 penandatanganan kode tidak disediakan atau di-flash. Itu harus dikompilasi ke dalam proyek. AWS IoT Device Tester mengambil sertifikat terpercaya dan mengkompilasi ke dalam `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 path file dari gambar yang dibangun oleh IDT yang akan ditandatangani. `{{outputSignatureFilePath}}` adalah path file dari tanda tangan yang akan dihasilkan oleh script.

pkcs11LabelConfiguration

Konfigurasi label PKCS11 memerlukan setidaknya satu set label label sertifikat perangkat, label kunci publik, dan label kunci pribadi untuk menjalankan grup pengujian PKCS11. Label PKCS11 yang diperlukan didasarkan pada konfigurasi perangkat Anda dalam file `device.json`. Jika pra-penyediaan diatur ke Ya di `device.json`, maka label yang diperlukan harus salah satu di bawah ini tergantung pada apa yang dipilih untuk fitur PKCS11.

- `PreProvisionedEC`
- `PreProvisionedRSA`

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

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

Tiga label berikut hanya diperlukan jika Anda memilih Ya untuk `pkcs11JITPCodeVerifyRootCertSupport` di `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 penyediaan kunci, label ini digunakan untuk pengujian.

Label ini mungkin berbeda dari yang ditentukan untuk kasus yang telah ditetapkan sebelumnya. Jika Anda memiliki penyediaan kunci yang disetel ke Tidak dan diatur sebelumnya ke Ya, `didevice.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 penyediaan kunci, label ini digunakan untuk pengujian. Label ini mungkin berbeda dari yang ditentukan untuk kasus yang telah ditetapkan sebelumnya. Jika Anda memiliki penyediaan kunci yang disetel ke Tidak dan diatur sebelumnya ke Ya, `didevice.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 penyediaan kunci, label ini akan digunakan untuk pengujian. Label ini mungkin berbeda dari yang ditentukan untuk kasus yang telah ditetapkan sebelumnya. Jika Anda memiliki penyediaan kunci yang disetel ke Tidak dan diatur sebelumnya ke Ya, `didevice.json`, ini akan menjadi tidak terdefinisi.

pkcs11LabelPreProvisionedECDevicePrivateKeyForTLS

(Opsional) Label ini digunakan untuk label PKCS #11 dari kunci pribadi. Untuk perangkat dengan elemen aman atau batasan perangkat keras, ini akan memiliki label berbeda untuk mempertahankan AWS IoT kredensial. Jika perangkat Anda mendukung pra-penyediaan dengan kunci EC, berikan label ini. Ketika Preprovisioned diatur ke Ya `didevice.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 batasan perangkat keras, ini akan memiliki label berbeda untuk mempertahankan AWS IoT kredensial. Jika perangkat Anda mendukung pra-penyediaan dengan kunci EC, berikan label ini. Ketika Preprovisioned diatur ke Ya `didevice.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 batasan perangkat keras, ini akan memiliki label yang berbeda untuk mempertahankan AWS IoT

kredensial. Jika perangkat Anda mendukung pra-penyediaan dengan kunci EC, berikan label ini. Ketika Preprovisioned diatur ke Ya di `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 batasan perangkat keras, ini akan memiliki label yang berbeda untuk mempertahankan AWS IoT kredensial. Jika perangkat Anda mendukung pra-penyediaan dengan kunci RSA, berikan label ini. Ketika Preprovisioned diatur ke Ya di `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 batasan perangkat keras, ini akan memiliki label yang berbeda untuk mempertahankan AWS IoT kredensial. Jika perangkat Anda mendukung pra-penyediaan dengan kunci RSA, berikan label ini. Ketika Preprovisioned diatur ke Ya di `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 batasan perangkat keras, ini akan memiliki label yang berbeda untuk mempertahankan AWS IoT kredensial. Jika perangkat Anda mendukung pra-penyediaan dengan kunci RSA, berikan label ini. Ketika Preprovisioned diatur ke Ya di `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. Saat `pkcs11JITPCodeVerifyRootCertSupport` masuk di `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. Saat `pkcs11JITPCodeVerifyRootCertSupport` masuk `device.json` diatur ke Ya, label ini harus disediakan.

IDT untuk variabel FreeRTOS

Perintah untuk membuat kode dan mem-flash perangkat mungkin memerlukan konektivitas atau informasi lain tentang perangkat Anda agar berhasil berjalan. AWS IoT Device Tester memungkinkan Anda untuk referensi informasi perangkat di flash dan membangun perintah menggunakan [JsonPath](#). Dengan menggunakan JsonPath ekspresi sederhana, Anda dapat mengambil informasi yang diperlukan yang ditentukan dalam file `Andadevice.json`.

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}}`

Perluas 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.

Gunakan antarmuka pengguna IDT for FreeRTOS untuk menjalankan 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 pengujian FreeRTOS terhadap perangkat Anda.

Untuk informasi tentang cara menggunakan baris perintah untuk menjalankan uji kualifikasi, lihat [Bersiap untuk menguji papan mikrokontroler Anda untuk pertama kalinya](#).

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

Topik

- [Prasyarat](#)
- [Konfigurasi kredensial AWS](#)
- [Buka IDT untuk FreeRTOS UI](#)
- [Buat konfigurasi baru](#)
- [Ubah konfigurasi yang ada](#)
- [Jalankan tes kualifikasi](#)

Prasyarat

Untuk menjalankan pengujian melalui AWS IoT Device Tester (IDT) untuk FreeRTOS UI, Anda harus menyelesaikan prasyarat pada [Prasyarat](#) halaman untuk IDT FreeRTOS Qualification (FRQ) 2.x.

Konfigurasi kredensial AWS

Anda harus mengonfigurasi kredensi pengguna IAM Anda untuk AWS pengguna yang Anda buat [Buat dan konfigurasi AWS akun](#). Anda dapat menentukan kredensial Anda dengan salah satu dari dua cara berikut:

- Di file kredensial
- Sebagai variabel lingkungan

Konfigurasi kredensial AWS dengan file kredensial

IDT menggunakan file kredensial yang sama sebagai AWS CLI. Untuk informasi selengkapnya, lihat [File konfigurasi dan kredensial](#).

Lokasi file kredensialnya bervariasi tergantung pada sistem operasi yang Anda gunakan:

- macOS dan Linux — `~/ .aws/credentials`

- Windows – C:\Users*UserName*\.aws\credentials

Tambahkan kredensial AWS Anda ke file `credentials` 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 [Profil bernama](#).

Konfigurasi kredensial AWS dengan variabel lingkungan

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

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

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 direktori instalasi IDT for FreeRTOS:

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

3. Jalankan perintah berikut untuk menjalankan 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 Preferensi Sistem -> Keamanan & Privasi. 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 kredensi Anda sebelum membuka UI. Jika Anda belum mengonfigurasi kredensi, tutup jendela browser IDT for FreeRTOS UI, ikuti langkah-langkah masuk [Konfigurasi kredensial AWS](#), lalu buka kembali IDT untuk FreeRTOS UI.

Buat konfigurasi baru

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

Untuk contoh `config.json`, `device.json`, dan `userdata.json` file, lihat [Bersiap untuk menguji papan mikrokontroler Anda untuk pertama kalinya](#).

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.



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 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 wizard konfigurasi untuk masuk ke pengaturan konfigurasi IDT yang digunakan untuk menjalankan tes kualifikasi. Wizard mengkonfigurasi pengaturan berikut dalam file konfigurasi JSON yang terletak di *devicetester-extract-location*/config direktori.
 - Pengaturan perangkat - Pengaturan kumpulan perangkat untuk perangkat yang akan diuji. Pengaturan ini dikonfigurasi disku kolomid dan, dan perangkat memblokir 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.	SKU Info SKU (Stock Keeping Unit) of the devices being tested.
<input type="text" value="my-device-pool"/>	<input type="text" value="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.	Serial port The serial port for device communication.
<input type="text" value="my-device"/>	<input type="text" value="/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.

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.

Pre-provisioned secure element
The device has a secure element with a pre-provisioned key that cannot be modified.

Yes
 No

PKCS #11 J1TP storage support
The device's core PKCS #11 implementation supports storage for J1TP. This enables the J1TP code verify test while testing core PKCS #11, and requires the code verification key, J1TP 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 J1TR key provisioning.

Identifiers
Arbitrary key/value pairs associated with the device.
No identifiers are associated with the device.

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 membuat AWS sumber daya selama pengujian berjalan. Pengaturan ini dikonfigurasi dalam `config.json` file.

The screenshot displays the 'AWS account settings' configuration window. On the left, a vertical sidebar lists six steps: 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), and Step 6 (Review). The main area is titled 'AWS account settings' and contains an 'Access information' section. This section includes three input fields: 'Account region' with the value 'us-west-2', 'Credentials location' with 'File' selected (radio button), and 'Profile name' with the value 'default'. Below these fields are 'Cancel', 'Previous', and 'Next' buttons. On the right side, there is an 'Access information' sidebar with explanatory text and a 'Learn more' link.

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

Device Tester for FreeRTOS > Create new configuration
FreeRTOS implementation ×

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

FreeRTOS implementation Info

Configuration for the FreeRTOS port to be tested.

Repository paths Info

Paths to elements of the FreeRTOS port, so Device Tester can hook into and use it for testing.

Repository root path
Path to the repository containing the FreeRTOS port.

FreeRTOS test parameter configuration path Info
Path to the test_param_config.h file for FreeRTOS-Libraries-Integration-Tests integration.

FreeRTOS test execution configuration path Info
Path to the test_execution_config.h file for FreeRTOS-Libraries-Integration-Tests integration.

FreeRTOS version
The FreeRTOS version of the port.

Build tool

Program to run that builds the FreeRTOS source code into an image.

Name

Version

Build commands Info
The shell commands that invoke the tool.

Command 1
 Remove

Flash tool

This tool flashes the built FreeRTOS source code onto the device.

Name

Version

Test start delay — optional
The number of milliseconds to delay tests after the flash. Set this variable if IDT misses the start of the tests.

Must be between 0 and 30000.

Flash commands Info
The shell commands that invoke the tool.

Command 1
 Remove

Cancel Previous Next

FreeRTOS implementation

Ported FreeRTOS code must be available on the local machine to begin automated testing with Device Tester. When running tests, Device Tester first makes a copy of the repository and then configures, builds, and flashes it to the device under test. This enables Device Tester to run tests end-to-end without user interaction.

This page provides information about the location of the code, how it's integrated with the testing library, what the FreeRTOS version is, and how it should be used.

- Label PKCS #11 dan server Echo - Label [PKCS #11](#) yang sesuai dengan kunci yang disediakan di perangkat keras Anda berdasarkan fungsionalitas kunci dan metode penyediaan kunci. Pengaturan konfigurasi server gema untuk tes Antarmuka Transportasi. Pengaturan ini dikonfigurasi dalam `device.json` file `userdata.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

PKCS #11 labels and Echo server

Settings for the PKCS #11 labels and Echo server creation configuration used during testing.

PKCS #11 labels [Info](#)

The labels used in PKCS #11 tests.

PKCS labels for onboard or import key provisioning devices — **Required** if the device supports onboard or import key provisioning [Info](#)

For devices with on-chip storage, this should match the non-test label.

Public key label	Private key label	Device certificate label
<input type="text" value="Enter label"/>	<input type="text" value="Enter label"/>	<input type="text" value="Enter label"/>

PKCS labels for pre-provisioned devices with EC key function — **Required** if the device is pre-provisioned with PKCS EC key function [Info](#)

For EC key function devices with secure elements or hardware limitations.

Public key label	Private key label	Device certificate label
<input type="text" value="Enter label"/>	<input type="text" value="Enter label"/>	<input type="text" value="Enter label"/>

PKCS labels for pre-provisioned devices with RSA key function — **Required** if the device is pre-provisioned with PKCS RSA key function [Info](#)

For RSA key function devices with secure elements or hardware limitations.

Public key label	Private key label	Device certificate label
<input type="text" value="Enter label"/>	<input type="text" value="Enter label"/>	<input type="text" value="Enter label"/>

PKCS Just-In-Time-Provisioning (JITP) labels — **Required** for devices with storage support JITP [Info](#)

The PKCS #11 test verifies the following labels with create/destroy objects.

Code verification key	JITP Certificate	Root Certificate
<input type="text" value="Enter label"/>	<input type="text" value="Enter label"/>	<input type="text" value="Enter label"/>

Echo server [Info](#)

Server settings.

Key generation method

The Echo server is created and configured with this key generation function.

EC

RSA

Server port number

Enter a port number where the Echo server will run.

Must be between 1024 and 49151.

Cancel Previous Next

PKCS #11 labels

Device Tester will run the Full_PKCS11 FreeRTOS-Libraries-Integration-Tests test group multiple times with different label configurations, provided if the device supports pre-provisioned credentials and other provisioning mechanisms.

For information on these labels and their configurations, refer to *Porting the corePKCS11 library* below.

Learn more [↗](#)

[Porting the corePKCS11 library](#)

- Pembaruan Over-the-air (OTA) - Pengaturan yang mengontrol tes fungsionalitas OTA. Pengaturan ini dikonfigurasi di `features_blokdevice.json` dan `userdata.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

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.

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.

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.

OTA image code signing [Info](#)

The configuration for code signing images in OTA End to End testing.

Signing method

Specifies 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.

Untrusted signer certificate ARN [Info](#)

The untrusted signer certificate uploaded to ACM.

Signer certificate file name [Info](#)

The name of the signer certificate on the device.

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

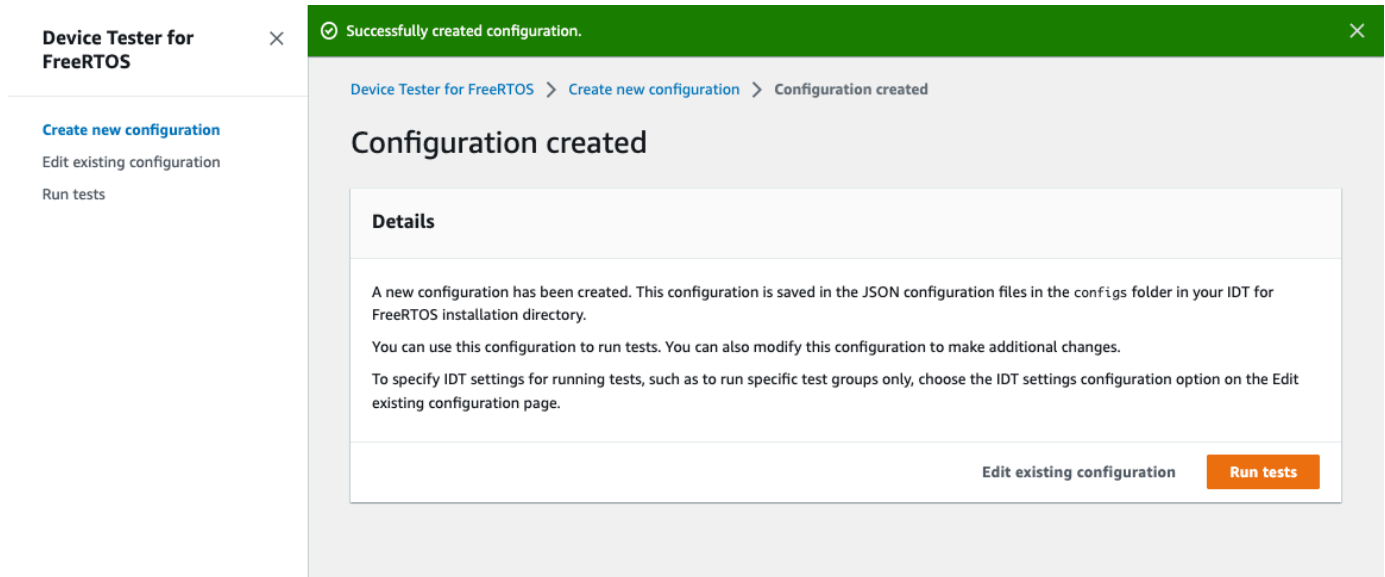
Over-the-air (OTA) updates ×

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 Ulasan, verifikasi informasi konfigurasi Anda.



Setelah selesai meninjau konfigurasi, untuk menjalankan tes kualifikasi, 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 mengubah konfigurasi yang ada. File konfigurasi yang ada harus ditempatkan 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`.

Device Tester for FreeRTOS ×

Device Tester for FreeRTOS > Edit existing configuration

Edit existing configuration Info

Edit existing configuration files that will be used for testing.

- Device settings**
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.
Status: ✔ Valid
- AWS account settings**
Settings related to the AWS account used for testing.
Status: ✔ Valid
- FreeRTOS implementation**
Configuration for the FreeRTOS port to be tested.
Status: ✔ Valid
- PKCS #11 labels and Echo server**
Settings for the PKCS #11 labels and Echo server creation configuration used during testing.
Status: ✔ Valid
- Over-the-air (OTA) updates**
The settings for over-the-air firmware update tests.
Status: ✔ Valid
- IDT test run settings**
Settings for running tests.
Status: ✔ Valid

Left sidebar: Create new configuration, Edit existing configuration, Run tests

2. Untuk mengubah pengaturan konfigurasi yang ada, selesaikan langkah-langkah berikut:
 - a. Pilih sebuah nama pengaturan konfigurasi untuk membuka halaman pengaturannya.
 - b. Ubah pengaturan, lalu pilih Simpan untuk membuat ulang file konfigurasi yang sesuai.
3. Untuk memodifikasi setelah uji coba IDT untuk FreeRTOS, pilih pengaturan uji coba IDT di tampilan edit:

Device Tester for FreeRTOS ×

Device Tester for FreeRTOS > Edit existing configuration > IDT test run settings

IDT test run settings Info

Settings for running tests.

Test selection Info
Run a subset of tests rather than the whole suite. Qualification reports won't be generated if a subset of tests are run.

- Run specific test groups:
- Run specific test cases:
Can be used only if exactly one specific group is being run.
- Skip specific test groups:
Run all test groups except for specific groups rather than the whole test suite.

Additional debugging settings

- Timeout multiplier:
Increase timeout of test suite timeouts by a specified value. Use this setting if tests are timing out.
- Stop on first failure:
Stop running tests if any fail. If selected, qualification reports won't be generated.

Buttons: Cancel, Save

Right sidebar: IDT test run settings ×
Changing settings for running tests on the run tests page.
These settings don't persist across IDT GUI executable sessions.

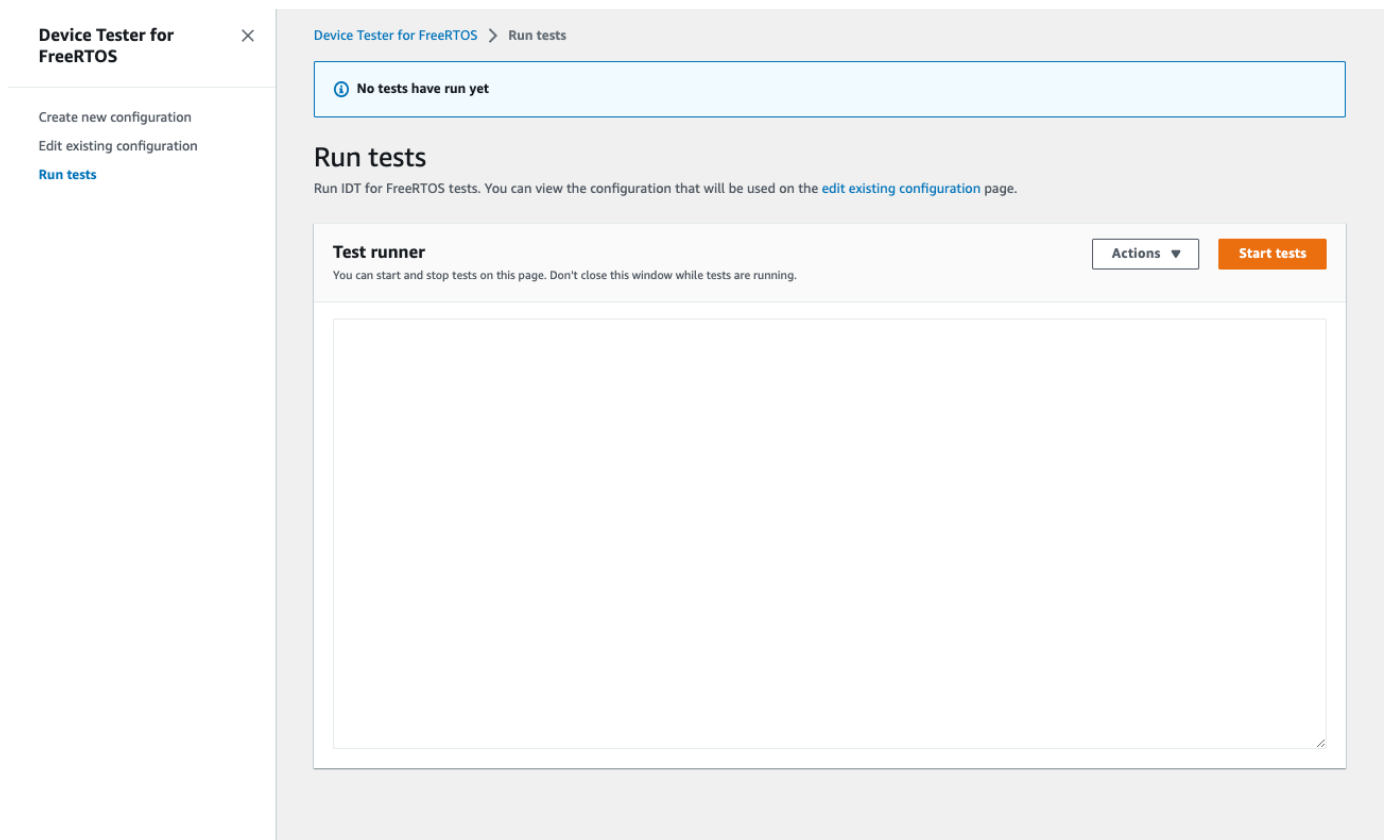
Setelah selesai memodifikasi konfigurasi, verifikasi bahwa semua pengaturan konfigurasi lulus validasi. Jika status untuk setiap pengaturan konfigurasi `Valid`, Anda dapat menjalankan pengujian kualifikasi dengan konfigurasi ini.

Jalankan tes kualifikasi

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

Untuk menjalankan tes kualifikasi

1. Di dalam menu navigasi, pilih Jalankan pengujian.
2. Pilih Mulai pengujian 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. Kemudian menampilkan ringkasan uji coba dan kesalahan apa pun di konsol Runner Test. 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 [Memahami hasil dan log](#).

The screenshot displays the AWS IoT Device Tester interface for FreeRTOS. On the left, a sidebar shows options: 'Device Tester for FreeRTOS', 'Create new configuration', 'Edit existing configuration', and 'Run tests'. The main area is titled 'Device Tester for FreeRTOS > Run tests'. A green notification box at the top states 'Tests finished running' and 'Results and logs can be found in the results folder.' Below this, the 'Run tests' section explains that the user can view the configuration used. A 'Test runner' window is open, showing a log of test execution. The log includes timestamps and status messages such as 'Upload FreeRTOS OTA test application file to S3 bucket', 'OTA update role creation completed', 'Creating OTA update job ...', 'OTA update creation completed with status CREATE_COMPLETE', 'Checking OTA update job status ...', 'OTA update job execution status SUCCEEDED', 'Device logging stopped', 'Cleaning up test resources...', and 'Cleaning up AWS resources... This may take a while...'. A 'Test Summary' section shows: 'Execution Time: 2h32m34s', 'Tests Completed: 13', 'Tests Passed: 13', 'Tests Failed: 0', and 'Tests Skipped: 0'. The 'Test Groups' section shows 'OTADataplaneMQTT: PASSED'. At the bottom, the paths for test execution logs and the aggregated JUnit report are provided.


Menjalankan 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 kolom perangkat (sekumpulan perangkat 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 ditempatkan di `devicetester_extract_location/devicetester_freertos_[win|mac|linux]/configs/` direktori.

 Note

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

Gunakan perintah berikut untuk menjalankan semua grup uji yang spesifik:

```
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 opsional jika Anda menjalankan tes suite tunggal pada kolam perangkat tunggal (yaitu, Anda hanya memiliki kolam perangkat tunggal yang ditentukan dalam `device.json` file Anda). `suite-id`

Gunakan perintah berikut untuk menjalankan uji kasus tertentu dalam grup pengujian:

```
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 uji kasus pengujian dalam grup uji.

IDT untuk FreeRTOS opsi baris perintah

group-id

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

kolam renang id

(Opsional) Kolam perangkat yang akan diuji. Ini diperlukan jika Anda menentukan beberapa kumpulan perangkat di dalamnyadevice.json. Jika Anda hanya memiliki satu kolam perangkat, Anda dapat mengabaikan opsi ini.

suite-id

(Opsional) Versi rangkaian tes yang akan dijalankan. Jika tidak ditentukan, IDT menggunakan versi terbaru dalam direktori tests pada sistem Anda.

uji-id

(Opsional) Tes untuk menjalankan, sebagai daftar yang dipisahkan koma. Jika ditentukan, group-id harus menentukan grup tunggal.

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

IDT untuk FreeRTOS perintah mendukung operasi berikut:

IDT v4.5.2 and later

help

Mencantumkan informasi tentang perintah yang ditentukan.

list-groups

Daftar grup di suite yang diberikan.

list-suites

Daftar suite yang tersedia.

list-supported-products

Daftar produk yang didukung dan versi rangkaian tes.

list-supported-versions

Mendaftar versi FreeRTOS dan rangkaian pengujian yang didukung oleh versi IDT saat ini.

list-test-cases

Daftar uji kasus dalam grup yang ditentukan.

run-suite


Menjalankan serangkaian tes pada kolam perangkat.

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

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

Example

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

 Note

Mulai IDT v3.0.0, IDT memeriksa rangkaian pengujian yang lebih baru secara online. Untuk informasi selengkapnya, lihat [Versi rangkaian tes](#).

Memahami hasil dan log

Bagian ini menjelaskan cara melihat dan menafsirkan laporan hasil IDT dan log.

Melihat Hasil

Saat berjalan, IDT menuliskan kesalahan ke konsol, file log, dan laporan tes. Setelah IDT menyelesaikan tes suite kualifikasi, ia akan 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.

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

- IDT untuk versi FreeRTOS.
- Versi FreeRTOS yang dites.
- Fitur FreeRTOS yang didukung oleh perangkat berdasarkan pengujian 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 tes.
- Perincian hasil uji kasus oleh perpustakaan yang dites berdasarkan fitur perangkat.

`FRQ_Report.xml` Ini adalah laporan dalam format [XMLJUnit](#) standar. Anda dapat mengintegrasikannya ke dalam platform CI/CD seperti [Jenkins](#), [Bamboo](#), dan sebagainya. Laporan tersebut berisi elemen berikut:

- Ringkasan agregat hasil uji kasus tes.
- Perincian hasil uji kasus oleh perpustakaan yang dites berdasarkan fitur perangkat.

Menafsirkan IDT untuk hasil FreeRTOS

Bagian laporan dalam `awsiotdevicetester_report.xml` atau `FRQ_Report.xml` daftar hasil tes yang dijalankan.

Tag XML pertama `<testsuites>` berisi ringkasan keseluruhan pelaksanaan tes. 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 uji kasus yang dilaksanakan.

failures

Jumlah uji kasus yang dijalankan, tetapi tidak lulus.

errors

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

disabled

Atribut ini tidak digunakan dan bisa diabaikan.

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

Jika kasus uji gagal atau salah, Anda dapat mengidentifikasi kasus uji yang gagal dengan meninjau tag **<testsuites>** XML-nya. Tag **<testsuite>** XML di dalam **<testsuites>** tag menunjukkan ringkasan hasil kasus uji untuk grup pengujian.

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

Format ini mirip dengan **<testsuites>** tag, tetapi dengan atribut **skipped** yang disebut yang tidak digunakan dan dapat diabaikan. Di dalam setiap tag **<testsuite>** XML-nya, ada **<testcase>** tag untuk setiap kasus uji yang dijalankan untuk grup uji. Misalnya:

```
<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. Potongan berikut menunjukkan bagaimana hal ini muncul di 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 terdapat kegagalan pengujian atau kesalahan untuk fitur yang diperlukan, perangkat Anda memenuhi persyaratan teknis untuk menjalankan FreeRTOS dan dapat bekerja sama dengan AWS IoT layanan. Jika Anda ingin mencantumkan perangkat Anda 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">
```

Format ini 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>
```

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

name

Nama uji kasus tes.

attempts

Berapa kali IDT untuk FreeRTOS mengeksekusi uji 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="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 [Pemecahan Masalah](#).

Melihat log

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, log terkait konfigurasi dan pembuatan laporan).

- *test_group_id/test_case_id/test_case_id.log*

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

Gunakan 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. [Tes 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 kredensial 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

- [Prasyarat](#)
- [Bersiap untuk menguji papan mikrokontroler Anda untuk pertama kalinya](#)
- [Gunakan antarmuka pengguna IDT untuk FreeRTOS untuk menjalankan rangkaian kualifikasi FreeRTOS](#)
- [Menjalankan tes Bluetooth Low Energy](#)
- [Menjalankan suite kualifikasi FreeRTOS](#)
- [Memahami hasil dan log](#)

Prasyarat

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

Unduh FreeRTOS

Anda dapat mengunduh rilis FreeRTOS [GitHub](#) dari 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 dari AWS IoT Device](#)

[Tester untuk FreeRTOS](#) 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` 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 dari AWS IoT Device Tester untuk FreeRTOS](#)

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 Akun AWS

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

Untuk mendaftar Akun AWS

1. Buka <https://portal.aws.amazon.com/billing/signup>.
2. Ikuti petunjuk secara online.

Anda akan diminta untuk menerima panggilan telepon dan memasukkan kode verifikasi pada keypad telepon sebagai bagian dari prosedur pendaftaran.

Saat Anda mendaftar Akun AWS, Pengguna root akun AWS akan dibuat. Pengguna root memiliki akses ke semua Layanan AWS dan sumber daya dalam akun. Sebagai praktik terbaik keamanan, [tetapkan akses administratif ke pengguna administratif](#), dan hanya gunakan pengguna root untuk melakukan [tugas yang memerlukan akses pengguna root](#).

AWS akan mengirimkan email konfirmasi kepada Anda setelah proses pendaftaran selesai. Anda dapat melihat aktivitas akun saat ini dan mengelola akun dengan mengunjungi <https://aws.amazon.com/> dan memilih Akun Saya.

Membuat pengguna administratif

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

Mengamankan Pengguna root akun AWS Anda

1. Masuk ke [AWS Management Console](#) sebagai pemilik akun dengan memilih Pengguna root dan memasukkan alamat email Akun AWS Anda. Di halaman berikutnya, masukkan kata sandi Anda.

Untuk bantuan masuk menggunakan pengguna root, lihat [Masuk sebagai pengguna root](#) dalam Panduan Pengguna AWS Sign-In.

2. Aktifkan autentikasi multi-faktor (MFA) untuk pengguna root Anda.

Untuk petunjuknya, silakan lihat [Mengaktifkan perangkat MFA virtual untuk pengguna root Akun AWS Anda \(konsol\)](#) dalam Panduan Pengguna IAM.

Membuat pengguna 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 sebuah pengguna administratif.

Untuk mendapatkan tutorial tentang menggunakan Direktori Pusat Identitas IAM sebagai sumber identitas Anda, silakan lihat [Mengonfigurasi akses pengguna dengan Direktori Pusat Identitas IAM default](#) di Panduan Pengguna AWS IAM Identity Center.

Masuk sebagai pengguna administratif

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

Untuk bantuan masuk menggunakan pengguna Pusat Identitas IAM, lihat [Masuk ke portal akses AWS](#) dalam Panduan Pengguna AWS Sign-In.

Kebijakan terkelola AWS IoT Device Tester

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](#).

Konfigurasi 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`](#).

Bersiap untuk menguji papan mikrokontroler Anda untuk pertama kalinya

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](#).

Konfigurasi kredensial AWS Anda

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

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 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 tumpukan TCP/IP 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 keYes.

Cellular

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

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 adalahECC,RSA, Both danNo. Bothmenunjukkan papan mendukung kedua ECC dan RSA algoritma.

KeyProvisioning

Menunjukkan metode penulisan sertifikat klien X.509 tepercaya ke papan Anda. Nilai yang valid adalahImport, Onboard danNo. 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). OtaDataPlaneProtocolAtribut 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 namabase64key, 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 demo/pengujian FreeRTOS.

devices.secureElementConfig.preProvisioned

(Opsional) Setel ke “Ya” jika perangkat memiliki elemen aman yang telah disediakan sebelumnya dengan kredensial 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.

Konfigurasi 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.

Konfigurasi 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"
  },
  "testStartDelays": 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. Sebagai contoh:

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

vendorPath

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

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

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

```
{ "vendorPath": "{{testData.sourcePath}}/vendors/esp8266/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`, `flashTool`. 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.

testStartDelays

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 socket 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 socket 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 `ifr-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 terpercaya yang digunakan untuk OTA.

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

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

Untuk informasi selengkapnya tentang membuat sertifikat terpercaya, 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.

Untuk informasi selengkapnya, lihat [Membuat sertifikat penandatanganan kode](#).

compileSignerCertificate

Setel ke `true` jika sertifikat verifikasi tanda tangan penandatanganan kode tidak disediakan atau di-flash, sehingga harus dikompilasi ke dalam proyek. AWS IoT Device Tester mengambil sertifikat terpercaya dan mengkompilasinya menjadi `aws_codesigner_certificate.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 `{{inputImagePath}}` dan `{{outputSignatureFilePath}}` diperlukan dalam perintah. `{{inputImagePath}}` adalah jalur file gambar yang dibangun oleh IDT untuk ditandatangani. `{{outputSignatureFilePath}}` adalah jalur file dari tanda tangan yang akan dihasilkan oleh skrip.

cmakeConfiguration

Konfigurasi CMake [Opsional]

Note

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

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 toolchain 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, PKCS11, TLS, 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 uji CMake, Anda harus memberikan nama papan, nama vendor, dan salah satu `afRToolchainPath` atau `compilerName`. Anda juga dapat memberikan `cmakeToolchainPath` jika Anda memiliki jalur khusus ke toolchain CMake.

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 `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 penandatanganan 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 FreeRTOS 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, lihat [Bersiap untuk menguji papan mikrokontroler Anda untuk pertama kalinya](#).

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

Topik

- [Prasyarat](#)
- [Memulai dengan IDT-Freertos UI](#)

Prasyarat

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 konfigurasi AWS akun](#)
- [Kebijakan terkelola AWS IoT Device Tester](#)

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
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 [GitHub](#) dari 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 dari AWS IoT Device Tester untuk FreeRTOS](#). 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 dari AWS IoT Device Tester untuk FreeRTOS](#)

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 konfigurasi AWS akun

Mendaftar Akun AWS

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

Untuk mendaftar Akun AWS

1. Buka <https://portal.aws.amazon.com/billing/signup>.
2. Ikuti petunjuk secara online.

Anda akan diminta untuk menerima panggilan telepon dan memasukkan kode verifikasi pada keypad telepon sebagai bagian dari prosedur pendaftaran.

Saat Anda mendaftar Akun AWS, Pengguna root akun AWS akan dibuat. Pengguna root memiliki akses ke semua Layanan AWS dan sumber daya dalam akun. Sebagai praktik terbaik keamanan, [tetapkan akses administratif ke pengguna administratif](#), dan hanya gunakan pengguna root untuk melakukan [tugas yang memerlukan akses pengguna root](#).

AWS akan mengirimkan email konfirmasi kepada Anda setelah proses pendaftaran selesai. Anda dapat melihat aktivitas akun saat ini dan mengelola akun dengan mengunjungi <https://aws.amazon.com/> dan memilih Akun Saya.

Membuat pengguna administratif

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

Mengamankan Pengguna root akun AWS Anda

1. Masuk ke [AWS Management Console](#) sebagai pemilik akun dengan memilih Pengguna root dan memasukkan alamat email Akun AWS Anda. Di halaman berikutnya, masukkan kata sandi Anda.

Untuk bantuan masuk menggunakan pengguna root, lihat [Masuk sebagai pengguna root](#) dalam Panduan Pengguna AWS Sign-In.

2. Aktifkan autentikasi multi-faktor (MFA) untuk pengguna root Anda.

Untuk petunjuknya, silakan lihat [Mengaktifkan perangkat MFA virtual untuk pengguna root Akun AWS Anda \(konsol\)](#) dalam Panduan Pengguna IAM.

Membuat pengguna 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 sebuah pengguna administratif.

Untuk mendapatkan tutorial tentang menggunakan Direktori Pusat Identitas IAM sebagai sumber identitas Anda, silakan lihat [Mengonfigurasi akses pengguna dengan Direktori Pusat Identitas IAM default](#) di Panduan Pengguna AWS IAM Identity Center.

Masuk sebagai pengguna administratif

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

Untuk bantuan masuk menggunakan pengguna Pusat Identitas IAM, lihat [Masuk ke portal akses AWS](#) dalam Panduan Pengguna AWS Sign-In.

Kebijakan terkelola AWS IoT Device Tester

Untuk mengaktifkan pengujian 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

- [Konfigurasi kredensial AWS](#)
- [Buka UI IDT-Freertos](#)
- [Buat konfigurasi baru](#)
- [Ubah konfigurasi yang ada](#)
- [Jalankan tes kualifikasi](#)

Konfigurasi kredensial AWS

Anda harus mengonfigurasi kredensi untuk AWS pengguna yang Anda buat. [Buat dan konfigurasi AWS akun](#) Anda dapat menentukan kredensial Anda dengan salah satu dari dua cara berikut:

- Di file kredensial
- Sebagai variabel lingkungan

Konfigurasi kredensial AWS dengan file kredensial

IDT menggunakan file kredensial 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 kredensial AWS Anda ke file `credentials` 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 [Profil bernama](#).

Konfigurasi kredensial AWS 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 kredensial 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, lihat [Gunakan 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, lihat [Bersiap untuk menguji papan mikrokontroler Anda untuk pertama kalinya](#). Untuk contoh `resource.json` file yang hanya digunakan untuk menjalankan tes Bluetooth Low Energy (BLE), lihat [Menjalankan 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. [Konfigurasi kredensial AWS](#)

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, lihat <https://docs.aws.amazon.com/freertos/latest/portingguide/afr-echo-server.html>.

- CMake - (Opsional) Pengaturan untuk menjalankan tes fungsionalitas build CMake. Konfigurasi ini hanya diperlukan jika Anda menggunakan CMake sebagai sistem build Anda. 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.

Ubah 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 [Memahami hasil dan log](#).

Menjalankan tes Bluetooth Low Energy

Bagian ini menjelaskan cara mengatur dan menjalankan tes Bluetooth 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 [Bersiap untuk menguji papan mikrokontroler Anda untuk pertama kalinya](#).
- 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 khusus 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. Konfigurasi 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 kredensi 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 kredensial 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 `OpenSSLssh-keygen`. Jika Anda sudah memiliki key pair SSH 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 keYes. 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"
    },
    ...
  ]
}
```

Menjalankan 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-langkahnya [Bersiap untuk menguji papan mikrokontroler Anda untuk pertama kalinya](#). Jika tes selain BLE gagal, maka masalahnya kemungkinan besar bukan karena konfigurasi Bluetooth.

Menjalankan 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 kolom 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

`group-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 lihat [Menjalankan 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.

Memahami hasil dan log

Bagian ini menjelaskan cara melihat dan menafsirkan laporan hasil IDT dan log.

Melihat Hasil

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 [XHTML JUnit](#) 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="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. Sebagai contoh:

```
<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. 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>
```

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. Sebagai contoh:

```
<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 [Pemecahan Masalah](#).

Melihat log

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.

Gunakan IDT untuk mengembangkan dan menjalankan rangkaian tes 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 pembuatan rangkaian uji

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

Unduhan AWS IoT Device Tester meliputi kode sumber untuk rangkaian tes 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](#)

Prasyarat

Untuk menyelesaikan tutorial ini, Anda memerlukan hal berikut ini:

- Persyaratan komputer host
 - Versi AWS IoT Device Tester terbaru
 - [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

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:

- Executable tes yang berisi logika tes
- 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](#)

Prasyarat

Untuk menyelesaikan tutorial ini, Anda memerlukan hal berikut ini:

- Persyaratan komputer host
 - Versi AWS IoT Device Tester terbaru
 - [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 dibutuhkan

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

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 dalam folder yang benar seperti yang dijelaskan dalam [langkah ini](#).
- 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.

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 [Konfigurasi group.json](#).

test.json

Berisi informasi tentang grup uji. Anda harus membuat file `test.json` untuk setiap grup uji di rangkaian tes Anda. Lihat [Konfigurasi test.json](#).

File konfigurasi opsional

test_orchestrator.yaml atau state_machine.json

Menentukan bagaimana tes akan dijalankan ketika IDT menjalankan rangkaian tes. SsE. [Konfigurasi 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, lihat [Konfigurasi 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 [Konfigurasi 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
```

Konfigurasi 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.

Konfigurasi `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`.

Konfigurasi `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 di mana `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.

Konfigurasi 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.

[Konfigurasi orkestrator uji IDT](#)

Konfigurasi `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](#).

Konfigurasi 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 berisi ID pengujian atau dependensi melingkar yang salah.

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 pengujian menggunakan ID grup uji dan satu atau lebih ID kasus uji 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
```

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 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:

- ID grup uji 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 ID grup yang 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, bukannya `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 grup uji dan satu atau beberapa ID kasus uji untuk mengidentifikasi pengujian individual yang akan dijalankan dari grup pengujian 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 grup uji dan satu atau beberapa ID kasus uji untuk mengidentifikasi pengujian individual yang akan dijalankan dari grup pengujian 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 melakukan kueri atas konteks tersebut dengan menggunakan notasi JSONPath. Sintaks untuk kueri JSONPath dalam definisi keadaan adalah `{{query}}`. Saat Anda mengakses data dari konteks orkestrator pengujian, pastikan setiap nilai mengevaluasi string, angka, atau Boolean.

Untuk informasi lebih lanjut tentang penggunaan notasi JSONPath untuk mengakses data dari konteks, lihat [Gunakan konteks IDT](#).

Konfigurasi 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 [Konfigurasi 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 dilakukan 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 keadaan. Objek *nama-keadaan* berisi definisi keadaan valid yang dipetakan ke *nama-keadaan* tersebut.

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. Serangkaian ID uji kasus dari grup yang ditentukan di 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 keadaan RunTask untuk menentukan ID grup uji tanpa menentukan ID uji kasus individu, sehingga IDT akan menjalankan semua uji kasus dalam grup uji tertentu. 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 ID uji kasus tertentu tidak valid, maka keadaan ini akan mengeluarkan kesalahan eksekusi `RunTaskError`. 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, lihat [Konteks 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. Serangkaian ID grup uji. Semua tes dalam setiap kelompok uji yang ditentukan harus lulus pada fitur yang akan didukung.

OneOfGroups

Opsional. Serangkaian ID grup uji. Semua tes dalam setidaknya satu kelompok uji yang ditentukan harus lulus pada fitur yang akan didukung.

TestCases

Opsional. Serangkaian ID grup uji. 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 telah menentukan ID uji kasus individu, atur `ResultVar` ke `group-id_test-id_passed`.
- Jika Anda tidak menentukan ID uji kasus individu, atur `ResultVar` ke `group-id_passed`.

Keadaan `AddProductFeatures` akan mengecek hasil tes dengan cara berikut:

- Jika Anda tidak menentukan ID uji kasus, maka hasil untuk setiap grup uji akan ditentukan dari nilai variabel `group-id_passed` dalam konteks state machine.
- Jika Anda tidak menentukan ID uji kasus, maka hasil untuk setiap tes akan ditentukan dari nilai variabel `group-id_test-id_passed` dalam konteks state machine.

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 bahwa Anda memberikan ID grup tes yang valid karena IDT tidak memvalidasi apakah grup tertentu 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 uji tertentu yang akan dijalankan dan bukan keseluruhan rangkaian uji, kunci ini akan ini dibuat dan berisi daftar ID grup uji tertentu.

specificTestCases

Jika test runner memilih grup uji tertentu yang akan dijalankan dan bukan keseluruhan rangkaian uji, kunci ini akan dibuat dan berisi daftar ID uji kasus tertentu.

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 melakukan kueri atas konteks tersebut dengan menggunakan notasi JSONPath. Sintaks untuk kueri JSONPath dalam definisi keadaan adalah `{{$.query}}`. Anda dapat menggunakan kueri JSONPath sebagai string placeholder dalam beberapa keadaan. IDT menggantikan string placeholder dengan nilai kueri JSONPath yang dievaluasi 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 lebih lanjut tentang penggunaan notasi JSONPath untuk mengakses data dari konteks, lihat [Gunakan 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, keadaan 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

- [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

IDT Client SDK memungkinkan Anda cara Anda menulis logika uji di executable tes Anda dengan perintah API yang dapat Anda gunakan untuk berinteraksi dengan IDT dan perangkat Anda yang sedang diuji. IDT saat ini menyediakan SDK berikut:

- IDT Client SDK for Python
- IDT Client SDK for Go
- SDK Klien IDT untuk Java

SDK ini terletak di folder `<device-tester-extract-location>/sdks`. 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 mengustomisasi 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 dijalanannya. 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 pengujianya 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 tes Anda menghasilkan hasil JUnit XML, Anda dapat menuliskan hasil tes itu ke file XML dalam folder artefak untuk langsung memberikan hasil ke IDT dan bukan mem-parsing hasilnya 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

Konfigurasi `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](#). configBidang ini juga berisi bidang tambahan berikut:

config.timeoutMultiplier

Penganda 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 kolom perangkat yang dipilih untuk uji coba. Informasi ini setara dengan elemen rangkaian devices dalam [file device.json](#) untuk perangkat yang dipilih.

devicePool

Informasi tentang kolom perangkat yang dipilih untuk uji coba. Informasi ini setara dengan elemen rangkaian kolom perangkat tingkat ata yang ditentukan di file `device.json` untuk kolom 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

Kredensial AWS yang digunakan oleh uji tersebut 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 notasi JsonPath dari file konfigurasi Anda dan dari teks yang dapat dieksekusi dengan API dan. `GetContextValue` `GetContextString` Sintaks untuk string `JSONPath` 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 lebih lanjut, lihat dokumentasi sebaris in SDK.

Tabel berikut menjelaskan operator dalam ekspresi foobar `JsonPath` khas:

Operator	Deskripsi
<code>\$</code>	Elemen root. Karena nilai konteks tingkat atas untuk IDT adalah objek, Anda biasanya akan menggunakannya <code>\$</code> . untuk memulai kueri Anda.
<code>.childName</code>	Mengakses elemen anak dengan nama <code>childName</code> 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 <code>awsRegion</code> nilai dalam <code>config</code> objek adalah <code>\$.config.awsRegion</code> .
<code>[start:end]</code>	Memfilter elemen dari array, mengambil item yang dimulai dari <code>start</code> indeks dan naik ke <code>end</code> indeks, keduanya inklusif.

Operator	Deskripsi
<code>[index1, index2, ... , indexN]</code>	Memfilter elemen dari array, mengambil item dari hanya indeks yang ditentukan.
<code>[?(<i>expr</i>)]</code>	Menyaring elemen dari array menggunakan <i>expr</i> ekspresi. Ekspresi ini harus mengevaluasi 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 JSONPath atau nilai dalam ekspresi Anda adalah rangkaian, boolean, atau nilai 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, JSONPath atau nilai di sisi kiri ekspresi Anda harus dievaluasi pada string dan sisi kanannya harus berupa nilai pola yang mengikuti [sintaks RE2](#).

Anda dapat menggunakan kueri JSONPath dalam bentuk `{{query}}` sebagai string placeholder di kolom `args` dan `environmentVariables` pada file `test.json` dan pada kolom `environmentVariables` di file `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 nyaman untuk menggunakan placeholder JSONPath 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 rangkaian tes yang ingin dijalankannya. Pengaturan ditentukan berdasarkan templat file konfigurasi yang terletak di `<device-tester-extract-location>/configs/` folder. Jika diperlukan, test runner juga harus mengatur kredensial AWS yang akan digunakan oleh 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.

Konfigurasi 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 konfigurasi 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) Konfigurasi `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) Konfigurasi `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 kolom 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 kolom 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 kolom 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) Konfigurasi `config.json`

File `config.json` berisi informasi konfigurasi untuk IDT. Biasanya, test runner tidak perlu mengubah file ini kecuali untuk memberikan kredensial pengguna AWS untuk IDT, dan secara opsional, sebuah wilayah AWS. Jika kredensial AWS dengan izin yang diperlukan disediakan, AWS IoT Device Tester akan mengumpulkan dan mengirimkan metrik penggunaan ke AWS. Ini adalah fitur opt-in dan digunakan untuk meningkatkan fungsi IDT. Untuk informasi selengkapnya, lihat [Metrik penggunaan IDT](#).

Test runner dapat mengonfigurasi kredensial AWS dengan salah satu cara berikut:

- File kredensial

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 pada Windows, gunakan `set`:

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

Untuk mengonfigurasi kredensial AWS untuk IDT, test runner akan mengedit bagian `auth` dalam file `config.json` yang terletak di `<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 `< device-tester-extract-location >`.

log.location

Jalur ke folder log di `< 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. Wilayah AWS yang akan digunakan oleh rangkaian uji. Jika tidak ditetapkan, rangkaian tes akan menggunakan wilayah default yang ditentukan dalam setiap rangkaian tes.

auth.method

Metode IDT yang digunakan untuk mengambil kredensial AWS. Nilai yang didukung adalah `file` untuk mengambil kredensial dari file kredensial, dan `environment` untuk mengambil kredensial 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 menanggapi. 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 rangkaian 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 ketika memulai rangkaian tes. 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

Skema laporan AWS IoT Device Tester

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>
</apnreport>
```

```
<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

Laporan `suite-name_Result.xml` berada dalam [format JUnitXML](#). 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` mendaftarkan tes yang dijalankan dan hasilnya.

Tag XML pertama `<testsuites>` berisi ringkasan pelaksanaan tes. Sebagai contoh:

```

<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 oleh 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. Sebagai contoh:

```
<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. Sebagai contoh:

```
<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. Sebagai contoh:

```
<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>
```

Metrik penggunaan IDT

Jika Anda memberikan kredensial AWS dengan izin yang diperlukan, AWS IoT Device Tester akan mengumpulkan dan mengirimkan 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
- Rangkaian pengujian di folder `< device-tester-extract-location >`
- 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 kredensi Anda, dan jika Anda memiliki AWS kredenal yang disimpan, jangan konfigurasi `config.json` file untuk mengaksesnya.

Mendaftar Akun AWS

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

Untuk mendaftar Akun AWS

1. Buka <https://portal.aws.amazon.com/billing/signup>.
2. Ikuti petunjuk secara online.

Anda akan diminta untuk menerima panggilan telepon dan memasukkan kode verifikasi pada keypad telepon sebagai bagian dari prosedur pendaftaran.

Saat Anda mendaftar Akun AWS, Pengguna root akun AWS akan dibuat. Pengguna root memiliki akses ke semua Layanan AWS dan sumber daya dalam akun. Sebagai praktik terbaik keamanan, [tetapkan akses administratif ke pengguna administratif](#), dan hanya gunakan pengguna root untuk melakukan [tugas yang memerlukan akses pengguna root](#).

AWS akan mengirimkan email konfirmasi kepada Anda setelah proses pendaftaran selesai. Anda dapat melihat aktivitas akun saat ini dan mengelola akun dengan mengunjungi <https://aws.amazon.com/> dan memilih Akun Saya.

Membuat pengguna administratif

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

Mengamankan Pengguna root akun AWS Anda

1. Masuk ke [AWS Management Console](#) sebagai pemilik akun dengan memilih Pengguna root dan memasukkan alamat email Akun AWS Anda. Di halaman berikutnya, masukkan kata sandi Anda.

Untuk bantuan masuk menggunakan pengguna root, lihat [Masuk sebagai pengguna root](#) dalam Panduan Pengguna AWS Sign-In.

2. Aktifkan autentikasi multi-faktor (MFA) untuk pengguna root Anda.

Untuk petunjuknya, silakan lihat [Mengaktifkan perangkat MFA virtual untuk pengguna root Akun AWS Anda \(konsol\)](#) dalam Panduan Pengguna IAM.

Membuat pengguna 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 sebuah pengguna administratif.

Untuk mendapatkan tutorial tentang menggunakan Direktori Pusat Identitas IAM sebagai sumber identitas Anda, silakan lihat [Mengonfigurasi akses pengguna dengan Direktori Pusat Identitas IAM default](#) di Panduan Pengguna AWS IAM Identity Center.

Masuk sebagai pengguna administratif

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

Untuk bantuan masuk menggunakan pengguna Pusat Identitas IAM, lihat [Masuk ke portal akses AWS](#) dalam Panduan Pengguna AWS Sign-In.

Untuk memberikan akses, tambahkan izin ke pengguna, grup, atau peran Anda:

- Pengguna dan grup di AWS IAM Identity Center:

Buat rangkaian izin. Ikuti petunjuk dalam [Buat set izin](#) dalam Panduan Pengguna AWS IAM Identity Center.

- Pengguna yang dikelola di IAM melalui penyedia identitas:

Buat peran untuk federasi identitas. Ikuti petunjuk dalam [Membuat peran untuk penyedia identitas pihak ketiga \(federasi\)](#) di Panduan Pengguna IAM.

- Pengguna IAM:

- Buat peran yang dapat diambil pengguna Anda. Ikuti petunjuk dalam [Membuat peran untuk pengguna IAM](#) dalam Panduan Pengguna IAM.

- (Tidak disarankan) Pasang kebijakan langsung ke pengguna atau tambahkan pengguna ke grup pengguna. Ikuti petunjuk di [Menambahkan izin ke pengguna \(konsol\)](#) dalam Panduan Pengguna IAM.

Berikan kredensial AWS untuk IDT

Untuk membolehkan IDT mengakses kredensial AWS dan mengirimkan metrik ke AWS, lakukan hal berikut:

1. Simpan kredensial AWS 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. Konfigurasi bagian `auth` dari file `config.json`. Lihat informasi yang lebih lengkap di [\(Opsional\) Konfigurasi config.json](#).

AWS IoT Device Tester untuk versi test suite 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 bekerja dengan versi FreeRTOS tertentu.
- Kelompok uji adalah serangkaian tes individual yang terkait dengan fitur tertentu, seperti pesan BLE dan MQTT.

Mulai di IDT v3.0.0, test suite diversi menggunakan file `major.minor.patchFormat` mulai dari 1.0.0. Ketika Anda mengunduh IDT, paket termasuk versi test suite terbaru.

Ketika Anda memulai IDT di antarmuka baris perintah, IDT memeriksa apakah versi test suite 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 [Kebijakan dukungan AWS IoT Device Tester untuk FreeRTOS](#).

Anda dapat mengunduh test suite dengan menggunakan `upgrade-test-suite` perintah. Atau, Anda dapat menggunakan parameter opsional `-upgrade-test-suite flag` ketika Anda memulai IDT di mana *bendera* 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.

Pemecahan Masalah

Setiap eksekusi test suite memiliki ID eksekusi unik yang digunakan untuk membuat folder bernama `results/execution-id` di dalam `results` direktori. Log kelompok uji individu berada di bawah `results/execution-id/logs` direktori. Gunakan IDT untuk output konsol FreeRTOS untuk menemukan id eksekusi, id kasus uji, dan id grup uji kasus uji yang gagal dan kemudian 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.
- Output eksekusi uji.
- IDT lebih verbose untuk output konsol FreeRTOS.

Kami merekomendasikan alur kerja berikut untuk pemecahan masalah:

1. Jika Anda melihat kesalahan "*pengguna/perantidak* diizinkan untuk mengakses sumber daya ini", pastikan Anda mengonfigurasi izin sebagaimana ditentukan dalam [Buat dan konfigurasi AWS akun](#).

2. Baca output konsol untuk menemukan informasi, seperti eksekusi UUID dan tugas saat ini mengeksekusi.
3. Lihat di `FRQ_Report.xml` file untuk pernyataan kesalahan dari setiap tes. Direktori ini berisi log eksekusi dari setiap grup pengujian.
4. Lihat di file log di bawah `/results/execution-id/logs`.
5. Selidiki salah satu area masalah berikut:
 - Konfigurasi perangkat, seperti file konfigurasi JSON di `/configs/` folder.
 - Antarmuka perangkat. Periksa log untuk menentukan antarmuka mana yang gagal.
 - Perangkat lunak. Pastikan bahwa rantai alat untuk membangun dan mem-flash perangkat diinstal dan dikonfigurasi dengan benar.
 - Untuk FRQ 1.x.x pastikan versi kode sumber FreeRTOS yang bersih dan kloning tersedia. Rilis FreeRTOS ditandai menurut 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

Bila Anda menggunakan IDT untuk FreeRTOS, Anda harus mendapatkan file konfigurasi yang benar sebelum Anda menjalankan biner. Jika Anda mendapatkan kesalahan parsing dan konfigurasi, langkah pertama Anda harus mencari dan menggunakan template konfigurasi yang sesuai untuk 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 UUID eksekusi, yang direferensikan sebagai `execution-id` dalam dokumentasi ini.

Selanjutnya, lihat di `FRQ_Report.xml` file di `/results/execution-id` direktori. File ini berisi semua kasus pengujian 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 <code>device.json</code> , <code>config.json</code> , atau <code>userdata.json</code> hilang atau dalam format yang salah.	Pastikan bidang yang diperlukan tidak hilang dan dalam format yang diperlukan dalam file yang terdaftar. Untuk informasi selengkapnya, lihat Bersiap untuk menguji papan mikrokontroler Anda untuk pertama kalinya .
202	ValidationError	Bidang <code>device.json</code> , <code>config.json</code> , atau <code>userdata.json</code> mengandung nilai yang tidak valid.	Periksa pesan kesalahan di sisi kanan kode kesalahan dalam laporan: <ul style="list-style-type: none"> • Tidak Valid AWS Wilayah - Tentukan valid AWS wilayah di <code>config.json</code> berkas. Untuk informasi lebih lanjut tentang AWS Sdaerah,

Kode Kesalahan	Nama Kode Kesalahan	Kemungkinan Akar Penyebab	Pemecahan Masalah
			<p>lihat Wilayah dan Titik Akhir.</p> <ul style="list-style-type: none">• Tidak ValidAWSk redensi - Set validAWSk redensyal pada mesin uji Anda (melalui variabel lingkungan atau file kredensyal). Verifikasi bahwa bidang otentikas i dikonfigurasi dengan benar. Untuk informasi selengkapnya, lihat Buat dan konfigurasi 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: <ul style="list-style-type: none">• Periksa <code>validsourcePath</code> ditentukan dalam <code>userdata.json</code> berkas.• Hapus <code>buildfolder</code> di bawah direktori kode sumber FreeRTOS, jika ada. Untuk informasi selengkapnya, lihat Konfigurasi pengaturan build, flash, dan uji.• Windows memiliki batas karakter untuk nama path file. Nama path file yang panjang akan menyebabkan kesalahan.

Kode Kesalahan	Nama Kode Kesalahan	Kemungkinan Akar Penyebab	Pemecahan Masalah
204	BuildSourceError	Tidak dapat mengompilasi kode sumber FreeRTOS.	<p>Verifikasi item berikut:</p> <ul style="list-style-type: none">• Periksa bahwa informasi di bawah <code>buildTool</code> di dalam <code>kamuuserdata.json</code> file sudah benar.• Jika Anda menggunakan <code>ancmakesebagai</code> alat build, pastikan <code>{enableTests}</code> ditentukan dalam <code>buildTool</code> perintah. Untuk informasi selengkapnya, lihat Konfigurasi pengaturan build, flash, dan uji.• Jika Anda telah mengekstrak IDT untuk FreeRTOS ke jalur file di sistem Anda yang berisi spasi, misalnya <code>C:\Users\My Name\Desktop\</code>, Anda mungkin memerlukan kutipan tambahan di dalam perintah

Kode Kesalahan	Nama Kode Kesalahan	Kemungkinan Akar Penyebab	Pemecahan Masalah
			<p>build Anda untuk memastikan jalur diurai dengan benar. Hal yang sama mungkin diperlukan untuk perintah flash Anda.</p>
205	FlashOrRunTestError	<p>IDT FreeRTOS tidak dapat mem-flash atau menjalankan FreeRTOS di DUT Anda.</p>	<p>Verifikasi informasi di bawah <code>flashTool</code> di dalam <code>kamuuserdata.json</code> file sudah benar. Untuk informasi selengkapnya, lihat Konfigurasi pengaturan build, flash, dan uji.</p>
206	StartEchoServerError	<p>IDT FreeRTOS tidak dapat memulai server gema untuk WiFi atau tes soket aman.</p>	<p>Verifikasi port yang dikonfigurasi di bawah <code>echoServerConfiguration</code> di dalam <code>kamuuserdata.json</code> file tidak digunakan atau diblokir oleh firewall atau pengaturan jaringan.</p>

Debugging kesalahan penguraian file konfigurasi

Kadang-kadang, eror ketik dalam konfigurasi JSON dapat menyebabkan penguraian eror. Sebagian besar waktu, masalahnya adalah hasil 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 menemukan kesalahan, Anda dapat melakukan validasi secara manual di IDE Anda, editor teks seperti Atom atau Sublime, atau melalui alat online seperti JSONLint.

Debugging kesalahan penguraian hasil tes

Saat menjalankan grup uji dari [Freertos-Perpustakaan-Integrasi-Tes](#), seperti FullTransportInterfaceTLS, fullpkcs11_core, fullpkcs11_onboard_ecc, fullpkcs11_onboard_rsa, fullpkcs11_PreProvisioned_ECC, FullPKCS11_PreProvisioned_RSA, atau OtaCore, IDT untuk FreeRTOS mem-parsing hasil pengujian dari perangkat uji dengan koneksi serial. Terkadang, output serial tambahan pada perangkat dapat mengganggu penguraian hasil tes.

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 keluaran serial IDT 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 LULUS.

Periksa berikut ini untuk memastikan pengujian yang optimal.

- Pastikan makro logging yang digunakan pada perangkat aman. Lihat [Menerapkan makro penebangan pustaka](#) untuk informasi lebih lanjut.
- Pastikan ada output minimal ke koneksi serial selama pengujian. Output perangkat lain dapat menjadi masalah bahkan jika makro logging Anda benar thread aman, karena hasil tes akan output dalam panggilan terpisah selama pengujian.

IDT untuk log kasus uji FreeRTOS idealnya akan menunjukkan hasil pengujian yang tidak terganggu 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
```

Debugging kegagalan pemeriksaan integritas

Jika menggunakan FreeRTOS versi FRQ 1.x.x, pemeriksaan integritas berikut berlaku.

Saat Anda menjalankan grup pengujian FreeRtosIntegrity dan Anda mengalami kegagalan, pertamanya pastikan bahwa Anda belum memodifikasi salah satu *freertos* file direktori. Jika Anda belum, dan masih melihat masalah, pastikan Anda menggunakan cabang yang benar. Jika Anda menjalankan IDT `list-supported-products` perintah, Anda dapat menemukan cabang yang ditandai dari *freertos* repo Anda harus menggunakan.

Jika Anda mengkloning cabang tag yang benar dari *freertos* repo dan masih memiliki masalah, pastikan Anda juga telah menjalankan `submodule update` perintah. 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. Untuk memenuhi syarat port FreeRTOS tanpa modifikasi apa pun pada file dan struktur folder, pastikan tidak ada file yang tercantum dalam 'exhaustive' dan 'minimal' bagian dari `checksums.json` file telah dimodifikasi. Untuk menjalankan dengan SDK yang dikonfigurasi, verifikasi bahwa tidak ada file di bawah 'minimal' bagian telah dimodifikasi.

Jika Anda menjalankan IDT dengan SDK dan telah memodifikasi beberapa file di *freertos* direktori, lalu pastikan Anda mengonfigurasi SDK dengan benar di `user_data` berkas. Jika tidak, pemeriksa Integritas akan memverifikasi semua file di *freertos* direktori.

Debugging FullWiFi kegagalan kelompok uji

Jika Anda menggunakan FRQ 1.x.x dan mengalami kegagalan di FullWiFi kelompok uji, dan "AFQP_WiFiConnectMultipleAP" tes gagal, ini bisa jadi karena kedua titik akses tidak berada dalam subnet yang sama dengan komputer host yang menjalankan IDT. Pastikan kedua titik akses berada di subnet yang sama dengan komputer host yang menjalankan IDT.

Men-debug kesalahan "parameter yang diperlukan hilang"

Karena fitur baru ditambahkan ke IDT untuk FreeRTOS, perubahan pada file konfigurasi mungkin akan diperkenalkan. Penggunaan file konfigurasi lama mungkin akan merusak konfigurasi Anda. Jika ini terjadi, `test_group_id__test_case_id.logfile` 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.

Men-debug kesalahan "test tidak dapat memulai"

Anda mungkin melihat kesalahan yang mengarah ke kegagalan selama pengujian dimulai. Karena ada beberapa kemungkinan penyebabnya, periksa bidang-bidang berikut untuk kebenaran:

- Pastikan nama pool yang Anda sertakan dalam perintah eksekusi Anda benar-benar ada. Ini direferensikan langsung dari `Anda.device.json` berkas.
- Pastikan perangkat atau perangkat di kolam Anda memiliki parameter konfigurasi yang benar.

Men-debug kesalahan "tidak dapat menemukan awal hasil pengujian"

Anda mungkin melihat kesalahan saat IDT mencoba mengurai output hasil oleh perangkat yang diuji. Ada beberapa kemungkinan penyebabnya, jadi periksa bidang-bidang 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.x.x, 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 output grup uji FreeRTOS lainnya, Anda dapat mencoba meningkatkan nilai `testStartDelayms` dalam konfigurasi user data Anda. Untuk informasi selengkapnya, lihat [Konfigurasi pengaturan build, flash, dan uji](#).

Debugging kesalahan “Uji kegagalan: diharapkan __ hasil tetapi melihat ____” kesalahan

Anda mungkin melihat kesalahan yang mengarah ke kegagalan pengujian selama pengujian. Tes mengharapkan sejumlah hasil, dan tidak melihatnya selama pengujian. Beberapa pengujian FreeRTOS dijalankan sebelum IDT melihat output dari perangkat. Jika Anda melihat kesalahan ini, Anda dapat mencoba meningkatkan nilai `testStartDelayms` di dalam kamudata pengguna konfigurasi. Untuk informasi selengkapnya, lihat [Mengonfigurasi pengaturan build, flash, dan pengujian](#).

Debugging “_____ tidak dipilih karena ConditionalTests kendala” error

Ini berarti Anda menjalankan pengujian pada kumpulan perangkat yang tidak kompatibel dengan pengujian. Ini mungkin terjadi dengan tes OTA E2E. Misalnya, saat menjalankan `OTADataplaneMQTT` kelompok uji dan di `device.jsonfile` konfigurasi, Anda telah memilih OTA sebagai `TidakatauOTADataplaneProtocol` sebagai HTTP. Kelompok pengujian yang dipilih untuk dijalankan harus sesuai dengan `device.json` pilihan kemampuan.

Debugging batas waktu IDT selama pemantauan output perangkat

IDT dapat timeout karena sejumlah alasan. Jika batas waktu terjadi selama fase pemantauan keluaran perangkat pengujian, dan Anda dapat melihat hasilnya di dalam log kasus uji IDT, itu berarti hasilnya salah diurai oleh IDT. Salah satu alasannya adalah pesan log interleaved di tengah hasil tes. Jika ini masalahnya, silakan merujuk ke [Panduan Porting FreeRTOS](#) untuk rincian lebih lanjut tentang bagaimana log UNITY harus diatur.

Alasan lain untuk batas waktu selama pemantauan output perangkat bisa menjadi reboot perangkat setelah kegagalan kasus uji TLS tunggal. Perangkat kemudian menjalankan gambar berkedip dan menyebabkan loop tak terbatas yang terlihat di log. Jika ini terjadi, pastikan perangkat Anda tidak reboot setelah kegagalan pengujian.

Men-debug kesalahan “tidak diizinkan untuk mengakses sumber daya”

Anda mungkin melihat kesalahan “*pengguna/peran* tidak berwenang untuk mengakses sumber daya ini” di output terminal atau di `test_manager.logfile` di bawah `/results/execution-id/logs`. Untuk mengatasi masalah ini, lampirkan `AWSIoTDeviceTesterForFreeRTOSFullAccess` kebijakan terkelola untuk pengguna pengetesan Anda. Untuk informasi selengkapnya, lihat [Buat dan konfigurasi AWS akun](#).

Debugging kesalahan pengujian jaringan

Untuk pengujian berbasis jaringan, IDT memulai server gema yang mengikat ke port non-reserved pada mesin host. Jika Anda mengalami kesalahan karena batas waktu atau koneksi yang tidak tersedia diWiFi atau tes soket aman, pastikan jaringan Anda dikonfigurasi untuk memungkinkan lalu lintas ke port yang dikonfigurasi dalam kisaran 1024 - 49151.

Tes soket aman menggunakan port 33333 dan 33334 secara default. YangWiFites menggunakan port 33335 secara default. Jika ketiga port ini digunakan atau diblokir oleh firewall atau jaringan, Anda dapat memilih untuk menggunakan port yang berbeda di `userdata.json` untuk pengujian. Untuk informasi selengkapnya, lihat [Konfigurasi 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 muatan versi yang sama

Jika kasus uji OTA gagal karena versi yang sama berada 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 memecahkan masalah kegagalan pembaruan OTA, mulailah dengan memastikan bahwa Anda menggunakan versi terbaru yang didukung dari sistem build Anda.

Kegagalan uji OTA aktif `PresignedUrlExpired` kasus uji

Salah satu prasyarat dari tes ini adalah bahwa waktu pembaruan OTA harus lebih dari 60 detik, jika tidak tes akan gagal. Jika ini terjadi, pesan galat berikut ditemukan di log: “Uji membutuhkan waktu kurang dari 60 detik (url waktu kedaluwarsa) untuk menyelesaikan. Tolong hubungi kami.”

Debugging 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: `/dev/tty.*` atau `/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 tertanam berbeda, yang berarti bahwa 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.
- Sebuah port baca untuk membaca output.

Anda harus mengatur port baca yang benar di `device.json` berkas. Jika tidak, membaca output dari perangkat mungkin gagal.

Dalam kasus beberapa port, pastikan untuk menggunakan port baca perangkat di `device.json` berkas. Misalnya, jika Anda mencolokkan perangkat Espressif WROver dan dua

port yang ditetapkan padanya `/dev/ttyUSB0` dan `/dev/ttyUSB1`, gunakan `/dev/ttyUSB1` di dalam `kamudevise.json` berkas.

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 `screen` perintah. Pada 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 toolchain pengembangan

Bagian ini membahas masalah yang dapat terjadi dengan toolchain Anda.

Kode Komposer Studio di Ubuntu

Versi Ubuntu yang lebih baru (17.10 dan 18.04) memiliki `versiglibc` paket yang tidak kompatibel dengan Kode Komposer Studio 7.x versi. Kami menyarankan Anda menginstal Kode Komposer 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 keluaran log yang ditampilkan di konsol selama proses build atau flash.
- Membangun perintah mencoba untuk memulai dalam mode GUI bahkan ketika dipanggil sebagai headless.

Mencatat

IDT untuk log FreeRTOS ditempatkan di satu lokasi. Dari direktori root IDT, file-file ini tersedia di `bawahresults/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` adalah 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

Kapan AWS IoT Device Tester dijalankan, kegagalan dilaporkan ke konsol dengan pesan singkat. Lihat di `results/execution-id/logs/test_group_id__test_case_id.log` untuk mempelajari lebih lanjut tentang kesalahan.

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 `bawahresults/execution-id/logs` direktori. Gunakan output konsol IDT for 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 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 uji, dan lebih banyak verbose AWS IoT Device Tester output konsol.

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 selesai, Anda mungkin mengalami masalah di mana Anda memiliki terlalu banyak bucket Amazon S3 yang telah dibuat. Ini berarti waktu berikutnya bahwa Anda menjalankan IDT tes 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 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 pengali batas waktu. Faktor ini diterapkan pada nilai batas waktu default. Nilai apa pun yang dikonfigurasi untuk bendera ini harus lebih besar dari atau sama dengan 1.0. Untuk menggunakan pengali batas waktu, gunakan bendera `--timeout-multiplier` saat menjalankan test suite.

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` di dalam `camdevice.JSON` berkas, `FullSecureSockets` akan menggunakan instans EC2 t.micro untuk menjalankan pengujian dan ini mungkin dikenakan biaya tambahan untuk Anda AWS akun. Untuk informasi lebih lanjut, lihat [Harga Amazon EC2](#).

Kebijakan pembuatan laporan kualifikasi

Laporan kualifikasi hanya dihasilkan oleh AWS IoT Device Tester Versi (IDT) yang mendukung versi FreeRTOS yang dirilis dalam dua tahun terakhir. Jika Anda memiliki pertanyaan tentang kebijakan dukungan, silakan hubungi [AWS Support](#).

AWS Kebijakan yang dikelola untuk AWS IoT Device Tester

Sebuah AWS kebijakan terkelola adalah kebijakan mandiri yang dibuat dan dikelola oleh AWS. AWS kebijakan terkelola dirancang untuk memberikan izin untuk banyak kasus penggunaan umum sehingga Anda dapat mulai menetapkan izin kepada pengguna, grup, dan peran.

Perlu diingat bahwa AWS kebijakan terkelola mungkin tidak memberikan izin hak istimewa untuk kasus penggunaan spesifik Anda karena tersedia untuk semua AWS pelanggan untuk digunakan. Kami menyarankan Anda mengurangi izin lebih lanjut dengan mendefinisikan [kebijakan yang dikelola pelanggan](#) yang khusus untuk kasus penggunaan Anda.

Anda tidak dapat mengubah izin yang ditentukan dalam AWS kebijakan yang dikelola. Jika AWS memperbarui izin yang didefinisikan dalam AWS kebijakan terkelola, pembaruan mempengaruhi semua identitas utama (pengguna, grup, dan peran) yang dilampirkan kebijakan. AWS kemungkinan besar akan memperbarui AWS kebijakan 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: AWS IoT Device Tester For FreeRTOS Full Access](#)
- [AWS IoT Device Tester memperbarui pada kebijakan terkelola AWS](#)

AWS kebijakan terkelola: AWS IoT Device Tester For FreeRTOS Full Access

Yang `AWSIoTDeviceTesterForFreeRTOSFullAccess` kebijakan terkelola berisi berikut AWS IoT Device Tester izin untuk memeriksa versi, fitur pembaruan otomatis, dan pengumpulan metrik.

Rincian izin

Kebijakan ini mencakup izin berikut:

- `iot-device-tester:SupportedVersion`

HibahAWS IoT Device Testerizin untuk mengambil daftar produk yang didukung, test suite dan versi IDT.

- `iot-device-tester:LatestIdt`

HibahAWS IoT Device Testerizin untuk mengambil versi IDT terbaru yang tersedia untuk diunduh.

- `iot-device-tester:CheckVersion`

HibahAWS IoT Device Testerizin untuk memeriksa kompatibilitas versi untuk IDT, test suite dan produk.

- `iot-device-tester:DownloadTestSuite`

HibahAWS IoT Device Testerizin untuk men-download update test suite.

- `iot-device-tester:SendMetrics`

HibahAWSizin untuk mengumpulkan metrik tentangAWS IoT Device Testerpenggunaan 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"
            ]
        }
    }
}

```

AWS IoT Device Tester memperbarui pada kebijakan terkelola AWS

Anda dapat melihat detail tentang pembaruan kebijakan terkelola AWS untuk AWS IoT Device Tester sejak saat layanan mulai melacak perubahan ini.

Versi	Perubahan	Deskripsi	Tanggal
7 (Terbaru)	Merestrukturisasiec2:CreateTags kondisi.	Menghapus penggunaanForAnyValues .	6/14/2023
6	Dihapusfreertos:ListHardwarePlatforms dari kebijakan.	Menghapus izin karena tindakan ini tidak berlaku lagi mulai 1 Maret 2023.	6/2/2023
5	Izin yang ditambahkan untuk menjalank	Ini untuk memulai dan menghentikan instans	12/15/2020

Versi	Perubahan	Deskripsi	Tanggal
	an tes server gema menggunakan EC2.	EC2 di pelanggan AWSakun.	
4	Ditambahkan <code>iot:CancelJobExecution</code>	Izin ini membatalkan pekerjaan OTA.	07/17/2020

Versi	Perubahan	Deskripsi	Tanggal
3	<p>Ditambahkan izin berikut:</p> <ul style="list-style-type: none"> • <code>iot-device-tester:DownloadTestSuite</code> , • <code>iot-device-tester:CheckVersion</code> , • <code>iot-device-tester:LatestIdt</code> , • <code>iot-device-tester:SupportedVersion</code> . 	<ul style="list-style-type: none"> • <code>iot-device-tester:DownloadTestSuite</code> — HibahAWS IoT Device Testerizin untuk mengunduh pembaruan test suite, • <code>iot-device-tester:CheckVersion</code> — HibahAWS IoT Device Testerizin untuk memeriksa kompatibilitas versi untuk IDT, test suite dan produk, • <code>iot-device-tester:LatestIdt</code> — HibahAWS IoT Device Testerizin untuk mengambil versi IDT terbaru yang tersedia untuk diunduh, • <code>iot-device-tester:SupportedVersion</code> — HibahAWS IoT Device Testerizin untuk mengambil 	03/23/2020

Versi	Perubahan	Deskripsi	Tanggal
		daftar produk yang didukung, test suite dan versi IDT.	
2	Ditambahkan <code>iot-device-tester:SendMetrics</code> izin.	Hibah AWS izin untuk mengumpulkan metrik tentang AWS IoT Device Tester penggunaan internal.	02/18/2020
1	Versi awal.		12/12/2020

Kebijakan dukungan AWS IoT Device Tester untuk FreeRTOS

Important

Pada Oktober 2022, AWS IoT Device Tester untuk AWS IoT FreeRTOS Qualification (FRQ) 1.0 tidak menghasilkan laporan kualifikasi yang ditandatangani. Anda tidak dapat memenuhi syarat perangkat AWS IoT FreeRTOS baru untuk dicantumkan di [Katalog Perangkat AWS Mitra](#) melalui [Program Kualifikasi AWS Perangkat](#) menggunakan versi 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. [Sebaiknya gunakan IDT FRQ 2.0 untuk memenuhi syarat dan mencantumkan 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 mencantulkannya di Katalog [Perangkat AWS 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 yang tersedia GitHub di FreeRTOS/FreeRTOS](#). Kami menyarankan Anda menggunakan versi terbaru dari FreeRTOS dan FreeRTOS AWS IoT Device Tester untuk memvalidasi dan memenuhi syarat perangkat Anda.

Untuk FreeRTOS-LTS, IDT mendukung validasi dan kualifikasi versi FreeRTOS 202210 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 jalur 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 membuat laporan, yang akan memungkinkan Anda mengirimkan perangkat Anda untuk kualifikasi.

Lihat [Versi yang didukung dari AWS IoT Device Tester untuk FreeRTOS](#) 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 menggunakan [Versi IDT yang tidak didukung untuk FreeRTOS](#), Anda tidak akan menerima perbaikan bug terbaru atau update.

Untuk pertanyaan tentang kebijakan dukungan, hubungi [Dukungan AWS Pelanggan](#).

Keamanan di AWS

Keamanan cloud di AWS merupakan prioritas tertinggi. Sebagai pelanggan AWS, Anda mendapatkan manfaat dari pusat data dan arsitektur jaringan yang dibangun untuk memenuhi persyaratan organisasi yang paling sensitif terhadap keamanan.

Keamanan adalah tanggung jawab bersama antara AWS dan Anda. [Model tanggung jawab bersama](#) menjelaskan hal ini sebagai keamanan dari cloud dan keamanan di cloud:

- Keamanan cloud – AWS berfungsi melindungi infrastruktur yang menjalankan layanan AWS Cloud AWS. AWS juga menyediakan 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 layanan AWS yang Anda gunakan. Anda juga bertanggung jawab atas faktor-faktor lain termasuk sensitivitas data Anda, persyaratan organisasi Anda, 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 sumber daya AWS secara aman. Administrator IAM mengontrol siapa yang dapat diautentikasi (masuk) dan diotorisasi (memiliki izin) untuk menggunakan sumber daya FreeRTOS. IAM adalah layanan Layanan AWS yang dapat Anda gunakan tanpa dikenakan 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 kredensial 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 Anda untuk mengubah izin pengguna layanan Anda. Tinjau informasi di halaman ini untuk memahami konsep Basic 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

Autentikasi adalah cara Anda untuk masuk ke AWS menggunakan kredensial identitas Anda. Anda harus terautentikasi (masuk ke AWS) sebagai Pengguna root akun AWS, sebagai pengguna IAM, atau dengan mengambil peran IAM.

Anda dapat masuk ke AWS sebagai identitas terfederasi dengan menggunakan kredensial yang disediakan melalui sumber identitas. Pengguna AWS IAM Identity Center Pengguna (Pusat Identitas IAM), autentikasi Single Sign-On perusahaan Anda, dan kredensial Google atau Facebook Anda merupakan contoh identitas terfederasi. Saat Anda masuk sebagai identitas gabungan, administrator Anda sebelumnya menyiapkan federasi identitas menggunakan peran IAM. Ketika Anda mengakses AWS dengan menggunakan federasi, Anda secara tidak langsung mengambil suatu peran.

Bergantung pada jenis pengguna Anda, Anda dapat masuk ke AWS Management Console atau portal akses AWS. Untuk informasi selengkapnya tentang cara masuk ke AWS, lihat [Cara masuk ke Akun AWS](#) dalam Panduan Pengguna AWS Sign-In.

Jika Anda mengakses AWS secara terprogram, AWS memberikan Kit Pengembangan Perangkat Lunak (SDK) dan antarmuka baris perintah (CLI) untuk menandatangani permintaan Anda secara kriptografis dengan menggunakan kredensial Anda. Jika Anda tidak menggunakan peralatan AWS, Anda harus menandatangani permintaan sendiri. Untuk informasi selengkapnya tentang cara menggunakan metode yang disarankan untuk menandatangani permintaan sendiri, lihat [Menandatangani permintaan API AWS](#) dalam Panduan Pengguna IAM.

Apa pun metode autentikasi yang digunakan, Anda mungkin diminta untuk menyediakan informasi keamanan tambahan. Sebagai contoh, AWS menyarankan Anda menggunakan autentikasi multi-faktor (MFA) untuk meningkatkan keamanan akun Anda. Untuk mempelajari lebih lanjut, lihat [Autentikasi multi-faktor](#) dalam Panduan Pengguna AWS IAM Identity Center dan [Menggunakan autentikasi multi-faktor \(MFA\) di AWS](#) dalam Panduan Pengguna IAM.

Pengguna root Akun AWS

Ketika membuat Akun AWS, Anda memulai dengan satu identitas masuk yang memiliki akses penuh ke semua Layanan AWS dan sumber daya di akun tersebut. Identitas ini disebut pengguna root Akun AWS dan diakses dengan cara masuk menggunakan alamat email dan kata sandi yang Anda gunakan untuk membuat akun. Kami sangat menyarankan agar Anda tidak menggunakan pengguna root untuk tugas sehari-hari Anda. Lindungi kredensial pengguna root Anda dan gunakan kredensial

tersebut untuk melakukan tugas yang hanya dapat dilakukan pengguna root. Untuk daftar tugas lengkap yang mengharuskan Anda masuk sebagai pengguna root, lihat [Tugas yang memerlukan kredensial pengguna root](#) dalam Panduan Pengguna IAM.

Identitas terfederasi

Praktik terbaiknya adalah mewajibkan pengguna manusia, termasuk pengguna yang memerlukan akses administrator, untuk menggunakan federasi dengan penyedia identitas untuk mengakses Layanan AWS dengan menggunakan kredensial temporer.

Identitas terfederasi adalah pengguna dari direktori pengguna perusahaan Anda, penyedia identitas web, AWS Directory Service, direktori Pusat Identitas, atau pengguna mana pun yang mengakses Layanan AWS dengan menggunakan kredensial yang disediakan melalui sumber identitas. Ketika identitas terfederasi mengakses Akun AWS, identitas tersebut mengambil peran, dan peran ini memberikan kredensial sementara.

Untuk pengelolaan akses terpusat, sebaiknya 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 Akun AWS dan aplikasi Anda. Untuk informasi tentang Pusat Identitas IAM, lihat [Apa yang dimaksud Pusat Identitas IAM?](#) dalam Panduan Pengguna AWS IAM Identity Center.

Pengguna dan grup IAM

[Pengguna IAM](#) adalah identitas dalam Akun AWS Anda yang memiliki izin khusus untuk satu orang atau aplikasi. Jika memungkinkan, sebaiknya andalkan kredensial temporer, dan bukan membuat pengguna IAM yang memiliki kredensial jangka panjang seperti kata sandi dan kunci akses. Namun, jika Anda memiliki kasus penggunaan khusus yang memerlukan kredensial jangka panjang dengan pengguna IAM, sebaiknya rotasikan 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 kumpulan pengguna IAM. Anda tidak dapat masuk sebagai grup. Anda dapat menggunakan grup untuk menentukan izin untuk beberapa pengguna sekaligus. Grup membuat izin lebih mudah dikelola untuk sekelompok besar pengguna. Misalnya, Anda dapat memiliki grup yang bernama 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 tersebut dimaksudkan untuk dapat diambil oleh siapa pun yang membutuhkannya. Pengguna

memiliki kredensial jangka panjang permanen, tetapi peran memberikan kredensial sementara. Untuk mempelajari selengkapnya, silakan lihat [Kapan harus membuat pengguna IAM \(bukan peran\)](#) dalam Panduan Pengguna IAM.

Peran IAM

[Peran IAM](#) merupakan identitas dalam Akun AWS Anda yang memiliki izin khusus. Peran ini mirip dengan pengguna IAM, tetapi tidak terkait dengan orang tertentu. Anda dapat mengambil peran IAM untuk sementara dalam AWS Management Console dengan [berganti peran](#). Anda dapat mengambil peran dengan cara memanggil operasi API AWS CLI atau AWS atau menggunakan URL kustom. Untuk informasi selengkapnya tentang metode untuk menggunakan peran, lihat [Menggunakan peran IAM](#) dalam Panduan Pengguna IAM.

Peran IAM dengan kredensial sementara berguna dalam situasi berikut:

- Akses pengguna gabungan – Untuk menetapkan izin ke sebuah identitas gabungan, Anda dapat membuat peran dan menentukan izin untuk peran tersebut. Saat identitas terfederasi diautentikasi, identitas tersebut dikaitkan dengan peran dan diberikan izin yang ditentukan oleh peran. Untuk informasi tentang peran untuk federasi, lihat [Membuat peran untuk Penyedia Identitas pihak ketiga](#) dalam Panduan Pengguna IAM. Jika Anda menggunakan Pusat Identitas IAM, Anda mengonfigurasi sekumpulan izin. Untuk mengontrol apa yang dapat diakses identitas Anda setelah identitas tersebut diautentikasi, Pusat Identitas IAM mengaitkan izin yang ditetapkan ke peran dalam IAM. Untuk informasi tentang rangkaian izin, lihat [Rangkaian 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 (pengguna utama tepercaya) dengan akun berbeda untuk mengakses sumber daya yang ada di akun Anda. Peran adalah cara utama untuk memberikan akses lintas akun. Namun, pada beberapa Layanan AWS, Anda dapat menyertakan kebijakan secara langsung ke sumber daya (bukan menggunakan peran sebagai proksi). Untuk mempelajari perbedaan antara kebijakan peran dan kebijakan berbasis sumber daya untuk akses lintas akun, lihat [Bagaimana peran IAM berbeda dari kebijakan berbasis sumber daya](#) dalam Panduan Pengguna IAM.
- Akses lintas layanan – Sebagian Layanan AWS menggunakan fitur di Layanan AWS lainnya. Contoh, ketika Anda melakukan panggilan dalam layanan, umumnya layanan tersebut menjalankan aplikasi di Amazon EC2 atau menyimpan objek di Amazon S3. Suatu layanan

mungkin melakukan hal tersebut menggunakan izin pengguna utama panggilan, menggunakan peran layanan, atau peran terkait layanan.

- Sesi akses maju (FAS) – Ketika Anda menggunakan pengguna IAM atau peran IAM untuk melakukan tindakan di AWS, Anda akan dianggap sebagai seorang pengguna utama. Saat menggunakan beberapa layanan, Anda mungkin melakukan tindakan yang kemudian dilanjutkan oleh tindakan lain pada layanan yang berbeda. FAS menggunakan izin dari pengguna utama untuk memanggil Layanan AWS, yang dikombinasikan dengan Layanan AWS yang diminta untuk membuat permintaan ke layanan hilir. Permintaan FAS hanya diajukan saat layanan menerima permintaan yang memerlukan interaksi dengan Layanan AWS lain atau sumber daya lain untuk diselesaikan. Dalam hal ini, Anda harus memiliki izin untuk melakukan kedua tindakan tersebut. Untuk detail kebijakan ketika mengajukan permintaan FAS, lihat [Meneruskan sesi akses](#).
- Peran IAM – Peran layanan adalah [peran IAM](#) yang diambil layanan untuk melakukan tindakan atas nama Anda. Administrator IAM dapat membuat, memodifikasi, dan menghapus peran layanan dari dalam IAM. Untuk informasi selengkapnya, lihat [Membuat peran untuk mendelegasikan izin ke Layanan AWS](#) dalam Panduan pengguna IAM.
- Peran terkait layanan – Peran terkait layanan adalah tipe peran layanan yang terkait dengan Layanan AWS. Layanan tersebut dapat mengambil peran untuk melakukan sebuah tindakan atas nama Anda. Peran terkait layanan akan muncul di Akun AWS Anda dan dimiliki oleh layanan tersebut. 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 kredensial sementara untuk aplikasi yang berjalan di instans EC2 dan mengajukan permintaan API AWS CLI atau AWS. Cara ini lebih dianjurkan daripada menyimpan kunci akses dalam instans EC2. Untuk menetapkan peran AWS ke instans EC2 dan menyediakannya bagi semua aplikasinya, Anda dapat membuat profil instans yang dilampirkan ke instans tersebut. Profil instans berisi peran dan memungkinkan program yang berjalan di instans EC2 mendapatkan kredensial sementara. Untuk informasi selengkapnya, lihat [Menggunakan peran IAM untuk memberikan izin ke aplikasi yang berjalan di instans Amazon EC2](#) dalam Panduan Pengguna IAM.

Untuk mempelajari apakah kita harus menggunakan peran IAM atau pengguna IAM, lihat [Kapan harus membuat peran IAM \(bukan pengguna\)](#) dalam Panduan Pengguna IAM.

Mengelola akses menggunakan kebijakan

Anda mengendalikan akses di AWS dengan membuat kebijakan dan melampirkannya ke identitas atau sumber daya AWS. Kebijakan adalah objek di AWS yang, ketika terkait dengan identitas atau sumber daya, akan menentukan izinnya. AWS mengevaluasi kebijakan-kebijakan tersebut ketika seorang pengguna utama (pengguna, pengguna root, atau sesi peran) mengajukan permintaan. Izin dalam kebijakan menentukan apakah permintaan diizinkan atau ditolak. Sebagian besar kebijakan disimpan di AWS sebagai dokumen JSON. Untuk informasi selengkapnya tentang struktur dan isi dokumen kebijakan JSON, silakan lihat [Gambaran Umum kebijakan JSON](#) dalam Panduan Pengguna IAM.

Administrator dapat menggunakan kebijakan JSON AWS untuk menentukan secara spesifik siapa yang memiliki akses terhadap apa. Artinya, pengguna utama manakah yang dapat melakukan tindakan pada sumber daya apa, dan dalam kondisi apa.

Secara default, pengguna dan peran tidak memiliki izin. Untuk memberikan izin kepada pengguna untuk melakukan tindakan pada sumber daya yang mereka perlukan, administrator IAM dapat membuat kebijakan IAM. Administrator kemudian dapat menambahkan kebijakan IAM ke peran, dan pengguna dapat menjalankan peran.

Kebijakan IAM mendefinisikan izin untuk suatu tindakan terlepas dari metode yang Anda gunakan untuk operasi. Sebagai contoh, anggap saja Anda memiliki kebijakan yang mengizinkan tindakan `iam:GetRole`. Pengguna dengan kebijakan tersebut dapat memperoleh informasi peran dari AWS Management Console, AWS CLI, atau API AWS.

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 pengguna dan peran, di sumber daya mana, dan dengan ketentuan apa. Untuk mempelajari cara membuat kebijakan berbasis identitas, lihat [Membuat kebijakan IAM](#) dalam Panduan Pengguna IAM.

Kebijakan berbasis identitas dapat dikategorikan lebih lanjut sebagai kebijakan inline atau kebijakan terkelola. 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 di Akun AWS Anda. Kebijakan terkelola meliputi kebijakan yang dikelola AWS dan kebijakan yang dikelola pelanggan. Untuk mempelajari cara memilih antara kebijakan terkelola atau kebijakan inline, lihat [Memilih antara kebijakan terkelola 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 yang dilampiri kebijakan tersebut, kebijakan ini menentukan jenis tindakan yang dapat dilakukan oleh pengguna utama tertentu di sumber daya tersebut dan apa ketentuannya. Anda harus [menentukan pengguna utama](#) dalam kebijakan berbasis sumber daya. Pengguna utama dapat mencakup akun, pengguna, peran, pengguna gabungan, atau Layanan AWS.

Kebijakan berbasis sumber daya merupakan kebijakan inline yang terletak di layanan tersebut. Anda tidak dapat menggunakan kebijakan yang dikelola AWS dari IAM dalam kebijakan berbasis sumber daya.

Daftar kontrol akses (ACL)

Daftar kontrol akses (ACL) mengendalikan pengguna utama mana (anggota akun, pengguna, atau peran) yang memiliki izin untuk mengakses sumber daya. ACL serupa dengan kebijakan berbasis sumber daya, meskipun tidak menggunakan format dokumen kebijakan JSON.

Amazon S3, AWS WAF, dan Amazon VPC adalah contoh layanan yang mendukung ACL. Untuk mempelajari ACL selengkapnya, silakan lihat [Gambaran umum daftar kontrol akses \(ACL\)](#) di Panduan Developer Layanan Penyimpanan Ringkas Amazon.

Tipe kebijakan lain

AWS mendukung jenis kebijakan tambahan yang kurang umum. Tipe-tipe kebijakan ini dapat mengatur izin maksimum yang diberikan kepada Anda berdasarkan tipe kebijakan yang lebih umum.

- Batasan izin – Batasan izin adalah fitur lanjutan di mana Anda menetapkan izin maksimum yang dapat diberikan oleh kebijakan berbasis identitas kepada entitas IAM (pengguna 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 secara eksplisit terhadap salah satu kebijakan ini akan mengesampingkan izin tersebut. Untuk informasi selengkapnya tentang batasan izin, lihat [Batasan izin untuk entitas IAM](#) dalam Panduan Pengguna IAM.

- Kebijakan kontrol layanan (SCP) – SCP adalah kebijakan JSON yang menentukan izin maksimum untuk sebuah organisasi atau unit organisasi (OU) di AWS Organizations. AWS Organizations adalah layanan untuk mengelompokkan dan mengelola beberapa akun AWS yang dimiliki bisnis Anda secara terpusat. Jika Anda mengaktifkan semua fitur di organisasi, Anda dapat menerapkan kebijakan kontrol layanan (SCP) ke salah satu atau semua akun Anda. SCP membatasi izin untuk entitas dalam akun anggota, termasuk setiap Pengguna root akun AWS. Untuk informasi selengkapnya tentang Organisasi dan SCP, lihat [Cara kerja SCP](#) dalam Panduan Pengguna AWS Organizations.
- Kebijakan sesi – Kebijakan sesi adalah kebijakan lanjutan yang Anda teruskan sebagai parameter saat Anda membuat sesi sementara secara terprogram untuk peran atau pengguna gabungan. 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 di salah satu kebijakan ini akan membatalkan izin tersebut. Untuk informasi selengkapnya, lihat [Kebijakan sesi](#) dalam Panduan Pengguna IAM.

Berbagai jenis kebijakan

Jika beberapa jenis kebijakan diberlakukan untuk satu permintaan, izin yang dihasilkan lebih rumit untuk dipahami. Untuk mempelajari cara AWS menentukan apakah akan mengizinkan permintaan ketika ada beberapa jenis kebijakan, lihat [Logika evaluasi kebijakan](#) dalam 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

Fitur IAM	Dukungan FreeRTOS
kunci-kunci persyaratan kebijakan (spesifik layanan)	Ya
ACL	Tidak
ABAC (tanda dalam kebijakan)	Parsial
Kredensial sementara	Ya
Izin pengguna utama	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, [AWSlihat 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 pengguna dan peran, di sumber daya mana, dan dengan ketentuan apa. Untuk mempelajari cara membuat kebijakan berbasis identitas, lihat [Membuat kebijakan IAM](#) dalam Panduan Pengguna IAM.

Dengan kebijakan berbasis identitas IAM, Anda dapat menentukan tindakan dan sumber daya yang diizinkan atau ditolak, serta ketentuan terkait jenis tindakan yang diizinkan atau ditolak. Anda tidak dapat menentukan pengguna utama dalam kebijakan berbasis identitas karena kebijakan ini berlaku untuk pengguna atau peran yang dilampiri kebijakan. Untuk mempelajari semua elemen yang dapat digunakan 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 yang dilampiri kebijakan tersebut, kebijakan ini menentukan jenis tindakan yang dapat dilakukan oleh pengguna utama tertentu di sumber daya tersebut dan apa ketentuannya. Anda harus [menentukan pengguna utama](#) dalam kebijakan berbasis sumber daya. Pengguna utama dapat mencakup akun, pengguna, peran, pengguna gabungan, atau Layanan AWS.

Untuk mengaktifkan akses lintas akun, Anda dapat menentukan seluruh akun atau entitas IAM di akun lain sebagai pengguna utama dalam kebijakan berbasis sumber daya. Menambahkan pengguna utama lintas akun ke kebijakan berbasis sumber daya bagian dari membangun hubungan kepercayaan. Ketika pengguna utama dan sumber daya berada di Akun AWS yang berbeda, administrator IAM di akun tepercaya juga harus memberikan izin kepada entitas pengguna utama (pengguna atau peran) untuk mengakses sumber daya. Izin diberikan dengan melampirkan kebijakan berbasis identitas ke entitas tersebut. Namun, jika kebijakan berbasis sumber daya memberikan akses kepada pengguna utama dalam akun yang sama, kebijakan berbasis identitas lainnya tidak diperlukan. Untuk informasi selengkapnya, lihat [Perbedaan peran IAM dengan kebijakan berbasis sumber daya](#) di Panduan Pengguna IAM.

Tindakan kebijakan untuk FreeRTOS

Mendukung tindakan kebijakan Ya

Administrator dapat menggunakan kebijakan JSON AWS untuk menentukan siapa yang memiliki akses ke apa. Yaitu, pengguna utama mana yang dapat melakukan tindakan pada sumber daya apa, dan dalam 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 seperti operasi API AWS 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.

Menyertakan tindakan dalam suatu kebijakan untuk memberikan izin 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:

```
aws
```

Untuk menetapkan secara spesifik beberapa tindakan dalam satu pernyataan, pisahkan tindakan-tindakan tersebut dengan koma.

```
"Action": [  
  "aws:action1",  
  "aws: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 JSON AWS untuk menentukan siapa yang memiliki akses ke apa. Yaitu, pengguna utama mana yang dapat melakukan tindakan pada sumber daya apa, dan dalam kondisi apa.

Elemen kebijakan JSON `Resource` menentukan objek atau beberapa 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 mengindikasikan bahwa pernyataan tersebut berlaku untuk semua sumber daya.

```
"Resource": "*" 
```

Untuk melihat daftar jenis sumber daya FreeRTOS dan ARNnya, 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 spesifik layanan	Ya
--	----

Administrator dapat menggunakan kebijakan JSON AWS untuk menentukan siapa yang memiliki akses ke apa. Yaitu, pengguna utama mana yang dapat melakukan tindakan pada sumber daya apa, dan dalam kondisi apa.

Elemen Condition (atau blok Condition) memungkinkan Anda menentukan kondisi di mana suatu pernyataan akan diterapkan. Elemen Condition bersifat opsional. Anda dapat membuat ekspresi kondisional 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 satu pernyataan, atau beberapa kunci dalam satu elemen Condition, AWS akan mengevaluasinya dengan menggunakan operasi AND logis. Jika Anda menentukan beberapa nilai untuk satu kunci persyaratan, AWS akan mengevaluasi syarat tersebut menggunakan operasi OR yang 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, silakan lihat [Elemen kebijakan IAM: variabel dan tanda](#) di Panduan Pengguna IAM.

AWS mendukung kunci kondisi global dan kunci kondisi spesifik layanan. Untuk melihat semua kunci kondisi global AWS, lihat [kunci konteks kondisi global AWS](#) dalam 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](#)

ACL di FreeRTOS

Mendukung ACL

Tidak

Daftar kontrol akses (ACL) mengontrol pengguna utama (anggota akun, pengguna, atau peran) yang memiliki izin untuk mengakses sumber daya. ACL sama dengan kebijakan berbasis sumber daya, meskipun tidak menggunakan format dokumen kebijakan JSON.

ABAC dengan FreeRTOS

Mendukung ABAC (tanda dalam kebijakan)

Parsial

Kontrol akses berbasis atribut (ABAC) adalah strategi otorisasi yang menentukan izin berdasarkan atribut. Di AWS, atribut ini disebut tag. Anda dapat melampirkan tanda ke entitas IAM (pengguna atau peran) dan ke banyak sumber daya AWS. Pemberian tanda ke entitas dan sumber daya adalah langkah pertama dari ABAC. Kemudian rancanglah kebijakan ABAC untuk mengizinkan operasi-operasi ketika tanda milik pengguna utama cocok dengan tanda yang ada di sumber daya yang ingin diakses.

ABAC sangat berguna di lingkungan yang berkembang dengan cepat dan berguna dalam situasi di mana pengelolaan kebijakan menjadi rumit.

Untuk mengendalikan akses berdasarkan tag, berikan informasi tentang tanda di [elemen syarat](#) dari sebuah kebijakan dengan menggunakan kunci-kunci persyaratan `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 hanya untuk beberapa jenis sumber daya, nilainya adalah Parsial.

Untuk informasi selengkapnya tentang ABAC, lihat [Apa itu ABAC?](#) di Panduan Pengguna IAM. Untuk melihat tutorial terkait langkah-langkah penyiapan ABAC, lihat [Menggunakan kontrol akses berbasis atribut \(ABAC\)](#) di Panduan Pengguna IAM.

Menggunakan kredensial sementara dengan FreeRTOS

Mendukung kredensial sementara	Ya
--------------------------------	----

Beberapa Layanan AWS tidak berfungsi saat Anda masuk menggunakan kredensial sementara. Sebagai informasi tambahan, termasuk tentang Layanan AWS mana saja yang berfungsi dengan kredensial sementara, lihat [Layanan AWS yang berfungsi dengan IAM](#) di Panduan Pengguna IAM.

Anda menggunakan kredensial sementara jika Anda masuk ke AWS Management Console menggunakan metode apa pun kecuali nama pengguna dan kata sandi. Misalnya, ketika Anda mengakses AWS dengan menggunakan tautan masuk tunggal (SSO) milik perusahaan Anda, proses itu secara otomatis akan membuat kredensial temporer. Anda juga akan membuat kredensial sementara secara otomatis saat masuk ke konsol sebagai pengguna dan kemudian beralih peran. Untuk informasi selengkapnya tentang cara beralih peran, lihat [Beralih peran \(konsol\)](#) di Panduan Pengguna IAM.

Anda dapat membuat kredensial sementara secara manual menggunakan AWS CLI atau AWS API. Anda kemudian dapat menggunakan kredensial sementara untuk mengakses AWS. AWS menyarankan Anda membuat kredensial sementara secara dinamis, 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
---------------------------------	----

Jika menggunakan pengguna IAM atau peran IAM untuk melakukan tindakan di AWS, Anda akan dianggap sebagai pengguna utama. Jika menggunakan beberapa layanan, Anda mungkin melakukan tindakan yang kemudian dilanjutkan oleh tindakan lain di layanan yang berbeda. FAS menggunakan izin dari pengguna utama untuk memanggil Layanan AWS, yang dikombinasikan dengan Layanan AWS yang diminta untuk membuat permintaan ke layanan hilir. Permintaan FAS hanya diajukan saat layanan menerima permintaan yang memerlukan interaksi dengan Layanan AWS lain atau sumber daya lain untuk diselesaikan. Dalam hal ini, Anda harus memiliki izin untuk melakukan kedua tindakan tersebut. Untuk detail kebijakan ketika mengajukan permintaan FAS, lihat [Meneruskan sesi akses](#).

Peran layanan untuk FreeRTOS

Mendukung peran layanan	Ya
-------------------------	----

Peran layanan adalah [peran IAM](#) yang diambil oleh layanan untuk melakukan tindakan atas nama Anda. Administrator IAM dapat membuat, mengubah, dan menghapus peran layanan dari dalam IAM. Untuk informasi selengkapnya, lihat [Membuat 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 tertaut layanan adalah jenis peran layanan yang terkait dengan Layanan AWS. Layanan ini dapat menjalankan peran untuk melakukan tindakan atas nama Anda. Peran terkait layanan akan muncul di Akun AWS Anda dan dimiliki oleh layanan tersebut. Administrator IAM dapat melihat, tetapi tidak dapat mengedit izin untuk peran terkait layanan.

Untuk detail tentang pembuatan atau pengelolaan peran terkait layanan, lihat [Layanan AWS yang berfungsi dengan IAM](#). Temukan sebuah 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. Pengguna dan peran tersebut juga tidak dapat melakukan tugas dengan menggunakan AWS Management Console, AWS Command Line Interface (AWS CLI), atau API AWS. Untuk memberikan izin kepada pengguna untuk melakukan tindakan pada sumber daya yang mereka perlukan, administrator IAM dapat membuat kebijakan IAM. Administrator kemudian dapat menambahkan kebijakan IAM ke peran, dan pengguna dapat menjalankan peran.

Untuk mempelajari cara membuat kebijakan berbasis identitas IAM menggunakan contoh dokumen kebijakan JSON ini, lihat [Membuat kebijakan IAM](#) dalam Panduan Pengguna IAM.

Untuk detail tentang tindakan dan jenis sumber daya yang ditentukan oleh FreeRTOS, termasuk format ARN untuk setiap jenis sumber daya, [lihat Tindakan, sumber daya, dan kunci kondisi untuk FreeRTOS di Referensi](#) Otorisasi Layanan.

Topik

- [Praktik terbaik kebijakan](#)
- [Menggunakan konsol FreeRTOS](#)
- [Izinkan 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 dikenai biaya untuk Akun AWS Anda. Ketika Anda membuat atau mengedit kebijakan berbasis identitas, ikuti panduan dan rekomendasi ini:

- Mulai menggunakan kebijakan yang dikelola AWS dan beralih ke izin dengan hak akses paling rendah – Untuk mulai memberikan izin kepada pengguna dan beban kerja Anda, gunakan kebijakan yang dikelola AWS yang memberikan izin untuk banyak kasus penggunaan umum. Kebijakan ini ada di Akun AWS Anda. Sebaiknya Anda mengurangi izin lebih lanjut dengan menentukan kebijakan yang dikelola pelanggan AWS yang khusus untuk kasus penggunaan Anda. Untuk informasi selengkapnya, lihat [kebijakan yang dikelola AWS](#) atau [kebijakan yang dikelola AWS untuk fungsi pekerjaan](#) di 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 melakukan ini dengan menentukan tindakan yang dapat diambil pada sumber daya tertentu dalam kondisi tertentu, juga dikenal sebagai izin hak akses paling rendah. Untuk informasi selengkapnya tentang cara menggunakan IAM untuk menerapkan izin, lihat [Kebijakan dan izin di IAM](#) di Panduan Pengguna IAM.
- Gunakan kondisi dalam kebijakan IAM untuk membatasi akses lebih lanjut – Anda dapat menambahkan kondisi ke kebijakan Anda untuk membatasi akses ke tindakan dan sumber daya. Misalnya, Anda dapat menulis syarat kebijakan untuk menentukan bahwa semua pengajuan harus dikirim menggunakan SSL. Anda juga dapat menggunakan kondisi untuk memberi akses ke tindakan layanan jika digunakan melalui Layanan AWS yang spesifik, seperti AWS CloudFormation. Untuk informasi selengkapnya, lihat [Elemen kebijakan JSON IAM: Syarat](#) di Panduan Pengguna IAM.
- Menggunakan IAM Access Analyzer untuk memvalidasi kebijakan IAM Anda guna memastikan izin yang aman dan berfungsi – 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 Analizer Akses IAM](#) di Panduan Pengguna IAM.
- Wajibkan autentikasi multi-faktor (MFA) – Jika Anda memiliki skenario yang mengharuskan pengguna IAM atau pengguna root di Akun AWS Anda, aktifkan MFA untuk keamanan tambahan. Untuk mewajibkan MFA saat operasi API dipanggil, tambahkan kondisi MFA pada kebijakan Anda. Untuk informasi selengkapnya, lihat [Mengonfigurasi akses API yang dilindungi MFA](#) di Panduan Pengguna IAM.

Untuk informasi selengkapnya tentang praktik terbaik dalam IAM, lihat [Praktik terbaik keamanan di IAM](#) di 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 memberikan izin konsol minimum untuk pengguna yang melakukan panggilan hanya ke AWS CLI atau API AWS. Sebaliknya, izinkan akses hanya ke tindakan yang cocok dengan operasi API yang coba dilakukan.

Untuk memastikan bahwa pengguna dan peran masih dapat menggunakan konsol FreeRTOS, lampirkan juga FreeRTOS atau kebijakan terkelola ke *ConsoleAccess* entitas *ReadOnlyAWS*. Untuk informasi selengkapnya, lihat [Menambahkan izin ke pengguna](#) di Panduan Pengguna IAM.

Izinkan pengguna melihat izin mereka sendiri

Contoh ini menunjukkan cara membuat kebijakan yang mengizinkan para pengguna IAM melihat kebijakan inline dan terkelola yang dilampirkan ke identitas pengguna mereka. Kebijakan ini mencakup izin untuk menyelesaikan tindakan pada konsol atau menggunakan AWS CLI atau AWS API secara terprogram.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ViewOwnUserInfo",
      "Effect": "Allow",
      "Action": [
        "iam:GetUserPolicy",
        "iam:ListGroupForUser",
        "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](#)
- [Saya ingin mengizinkan 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 `aws:GetWidget` rekaan.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:  
aws: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 `aws:GetWidget`.

Jika Anda membutuhkan bantuan, hubungi administrator AWS 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.

Sebagian Layanan AWS mengizinkan Anda untuk memberikan peran yang sudah ada ke layanan tersebut alih-alih membuat peran layanan baru atau peran terkait-layanan. Untuk melakukan tindakan tersebut, Anda harus memiliki izin untuk memberikan peran pada layanan tersebut.

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 membutuhkan bantuan, hubungi administrator AWS Anda. Administrator Anda adalah orang yang memberi Anda kredensial masuk.

Saya ingin mengizinkan orang di luar saya Akun AWS untuk mengakses sumber daya FreeRTOS saya

Anda dapat membuat peran yang dapat digunakan pengguna di akun lain atau pengguna 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 (ACL), Anda dapat menggunakan kebijakan tersebut untuk memberi pengguna akses ke sumber daya Anda.

Untuk mempelajari selengkapnya, periksa hal berikut:

- Untuk mengetahui apakah FreeRTOS mendukung fitur-fitur ini, lihat [Bagaimana FreeRTOS bekerja dengan IAM](#)
- Untuk mempelajari cara memberikan akses ke sumber daya di seluruh Akun AWS yang Anda miliki, lihat [Menyediakan akses ke pengguna IAM di Akun AWS lainnya yang Anda miliki](#) dalam Panduan Pengguna IAM.

- Untuk mempelajari cara memberikan akses ke sumber daya Anda ke pihak ketiga Akun AWS, lihat [Menyediakan akses ke akun Akun AWS yang dimiliki oleh pihak ketiga](#) dalam Panduan Pengguna IAM.
- Untuk mempelajari cara memberikan akses melalui federasi identitas, lihat [Menyediakan akses ke pengguna terautentikasi eksternal \(gabungan identitas\)](#) dalam Panduan Pengguna IAM.
- Untuk mempelajari perbedaan antara penggunaan peran dan kebijakan berbasis sumber daya untuk akses lintas akun, lihat [Perbedaan antara peran IAM dan kebijakan berbasis sumber daya](#) di Panduan Pengguna IAM.

Validasi kepatuhan

Untuk mempelajari apakah an Layanan AWS berada dalam lingkup program kepatuhan khusus, lihat [Layanan AWS di Scope oleh Program](#) Program Kepatuhan yang Anda minati. Untuk informasi umum, lihat [Program Kepatuhan 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 berdasarkan sensitivitas data Anda, tujuan kepatuhan perusahaan Anda, serta hukum dan peraturan yang berlaku. AWS menyediakan sumber daya berikut untuk membantu kepatuhan:

- [Panduan Mulai Cepat Keamanan dan Kepatuhan](#) – Panduan deployment ini membahas pertimbangan arsitektur dan menyediakan langkah-langkah untuk melakukan deployment lingkungan dasar di AWS yang menjadi fokus keamanan dan kepatuhan.
- [Merancang Keamanan dan Kepatuhan HIPAA di Amazon Web Services](#) – Laporan resmi ini menjelaskan cara perusahaan dapat menggunakan AWS untuk membuat aplikasi yang memenuhi syarat HIPAA.

Note

Tidak semua Layanan AWS memenuhi syarat HIPAA. Untuk informasi selengkapnya, lihat [Referensi Layanan yang Memenuhi Syarat HIPAA](#).

- [Sumber Daya Kepatuhan AWS](#) – Kumpulan buku kerja dan panduan ini mungkin berlaku untuk industri dan lokasi Anda.

- [Panduan Kepatuhan Pelanggan AWS](#) – Pahami model tanggung jawab bersama melalui lensa kepatuhan. Panduan ini merangkum praktik terbaik untuk mengamankan Layanan AWS dan memetakan panduan kontrol keamanan di banyak kerangka kerja (termasuk National Institute of Standards and Technology (NIST), Payment Card Industry Security Standards Council (PCI), dan International Organization for Standardization (ISO)).
- [Mengevaluasi Sumber Daya dengan Aturan](#) dalam Panduan Developer AWS Config – Layanan AWS Config menilai seberapa baik konfigurasi sumber daya Anda dalam mematuhi praktik-praktik internal, pedoman industri, dan regulasi internal.
- [AWS Security Hub](#) – Layanan AWS ini memberikan pandangan komprehensif tentang status keamanan Anda di dalam AWS. Security Hub menggunakan kontrol keamanan untuk mengevaluasi sumber daya AWS Anda dan memeriksa kepatuhan Anda terhadap standar industri keamanan dan praktik terbaik. Untuk daftar layanan dan kontrol yang didukung, lihat [Referensi kontrol Security Hub](#).
- [AWS Audit Manager](#) – Layanan AWS ini akan membantu Anda untuk terus-menerus mengaudit penggunaan AWS untuk menyederhanakan cara Anda mengelola risiko dan kepatuhan terhadap regulasi dan standar industri.

Ketahanan di AWS

Infrastruktur global AWS dibangun di sekitar Wilayah dan Availability Zone AWS. AWS Kawasan menyediakan beberapa Zona Ketersediaan yang terpisah dan terisolasi secara fisik, yang tersambung dengan jejaring jaringan latensi rendah, throughput tinggi, dan sangat redundan. Dengan Zona Ketersediaan, Anda dapat merancang dan mengoperasikan aplikasi dan basis data yang melakukan secara otomatis pinda saat gagal/failover di antara zona-zona tanpa terputus. Zona Ketersediaan lebih sangat tersedia, lebih toleran kesalahan, dan lebih dapat diskalakan daripada infrastruktur pusat data tunggal atau multi tradisional.

Untuk informasi selengkapnya tentang Wilayah AWS dan Zona Ketersediaan, lihat [Infrastruktur Global AWS](#).

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 dengan menggunakan ID kunci akses dan kunci akses rahasia yang terkait dengan pengguna utama 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 diperbarui dengan perbaikan keamanan terbaru yang tersedia untuk umum. Periksa [Pustaka FreeRTOS LTS](#) halaman untuk pembaruan, atau berlangganan [Freertos-LTS](#) GitHub repositori untuk menerima patch LTS terbaru dengan perbaikan bug kritis dan keamanan. Anda dapat mengunduh atau mengkloning patch FreeRTOS LTS terbaru yang diperlukan langsung dari individu GitHub repositori.
2. Pertimbangkan refactoring implementasi antarmuka transportasi jaringan untuk mengoptimalkan platform perangkat keras Anda. API abstrak seperti [soket aman](#) dan [API Wifi](#) tidak diwajibkan oleh yang terbaru [CoreMQTT](#) perpustakaan. Lihat [Antarmuka Transportasi](#) untuk rincian lebih lanjut.

Lampiran

Tabel berikut memberikan rekomendasi untuk semua proyek demo, pustaka lama, dan API abstrak dalam repositori Amazon-Freertos.

Pustaka dan demo yang dimigrasi

Nama	Tipe	Rekomendasi
CoreHTTP	demo dan perpustakaan	Kloning atau unduh pustaka CoreHTTP langsung dari CoreHTTP repositori (sub-modul jika menggunakan git) di Organisasi FreeRTOS Github . Demo CoreHTTP ada di distribusi FreeRTOS primer . Untuk detail selengkapnya, lihat Halaman CoreHttp .
CoreMQTT	demo dan perpustakaan	Kloning atau unduh pustaka CoreMQTT langsung dari CoreMQTT repositori (sub-modul jika menggunakan git)

Nama	Tipe	Rekomendasi
		<p>di Organisasi FreeRTOS Github. Demo CoreMQTT ada didistribusi FreeRTOS primer. Untuk detail selengkapnya, lihat Halaman CoreMQTT.</p>
Agen CoreMQTT	demo dan perpustakaan	<p>Kloning atau unduh pustaka CoreMQTT-Agent langsung dari Agen CoreMQTT repositori (sub-modul jika menggunakan git) di Organisasi FreeRTOS Github. Demo agen CoreMQTT ada di Demo Agen CoreMQTT repositori. Untuk detail selengkapnya, lihat Halaman agen CoreMQTT.</p>
device_defender_for_aws	demo dan perpustakaan	<p>The AWS IoT Pustaka Device Defender ada di repositori di AWS GitHub organisasi. Kloning atau unduh (sub-modul jika menggunakan git) langsung dari AWS IoT Pembela Perangkat repositori. The AWS IoT Demo Device Defender ada didistribusi FreeRTOS primer. Untuk lebih jelasnya, lihat AWS IoT Halaman Device Defender.</p>

Nama	Tipe	Rekomendasi	
device_shadow_for_aws	demo dan perpustakaan	<p>TheAWS IoT Pustaka Device Shadow ada di repositori di AWS GitHub organisasi. Kloning atau unduh (sub-modul jika menggunakan git) langsung dari AWS IoT Device Shadow repositori. TheAWS IoT Demo Device Shadow ada di distribusi FreeRTOS primer. Untuk detail selengkapnya, lihat AWS IoT Halaman Device Shadow.</p>	
jobs_for_aws	demo dan perpustakaan	<p>TheAWS IoT Pustaka pekerjaan ada di repositori di AWS GitHub organisasi. Kloning atau unduh (sub-modul jika menggunakan git) langsung dari AWS IoT Lowongan repositori. TheAWS IoT Demo pekerjaan ada di distribusi FreeRTOS primer. Untuk detail selengkapnya, lihat AWS IoT Halaman Lowongan.</p>	
OTA	demo dan perpustakaan	<p>TheAWS IoT Pustaka Pembaruan Over-The-Air (OTA) ada di repositori di AWS GitHub organisasi. Kloning atau unduh (sub-modul jika menggunakan git) langsung dari AWS IoT OTA repositori. TheAWS IoT Demo OTA ada di distribusi FreeRTOS primer. Untuk lebih jelasnya, lihat AWS IoT Halaman OTA.</p>	

Nama	Tipe	Rekomendasi
CLI dan Freertos_Plus_CLI	demo dan perpustakaan	Ada contoh CLI yang berjalan di WinSim. Merujuk ke Antarmuka Baris Perintah FreeRTOS Plus halaman untuk lebih jelasnya. Integrasi referensi IoT FreeRTOS Unggulan pada NXP i.MX RT1060 dan STM32U5 platform, juga memberikan contoh CLI pada perangkat keras yang sebenarnya.
pencatatan log	makro	Ada implementasi makro logging untuk platform perangkat keras tertentu yang digunakan oleh beberapa pustaka FreeRTOS. Merujuk ke halaman logging untuk cara mengimplementasikan makro logging. Merujuk ke salah satu referensi IoT unggulan FreeRTOS untuk contoh yang berjalan pada perangkat keras yang sebenarnya.
greengrass_s_connectivity	demo	[Migrasi sedang berlangsung] Proyek demo ini mengasumsikan bahwa konektivitas cloud tersedia sebelum menghubungkan ke AWS IoT Perangkat 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 yang tidak digunakan lagi

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 opsi pihak ketiga seperti gesit untuk mengoptimalkan proyek Anda dengan sebaik-baiknya.
dev_mode_key_provisioning	demo	Integrasi referensi IoT FreeRTOS Unggulan pada NXP i.MX RT1060 , STM32U5 , atau ESP32-C3 platform memberikan contoh penyediaan penting menggunakan CLI.
posix	abstraksi dan demo	Tidak direkomendasikan untuk digunakan.
wifi_provisioning	contoh	Contoh ini menunjukkan cara penyediaan WiFi kredensial pada perangkat menggunakan perpustakaan Amazon-Freertos BLE. Lihat referensi IoT Unggulan FreeRTOS di Platform ESP32C3 Untuk contoh WiFi penyediaan melalui BLE.
API abstrak lama	code	Ini adalah API yang dibuat untuk menyediakan antarmuka abstrak untuk berbagai tumpukan perangkat lunak pihak ketiga, modul konektivitas

Nama	Tipe	Rekomendasi	
		<p>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/<code>libraries/abstractions/</code>. API ini tidak diperlukan saat menggunakan Pustaka FreeRTOS LTS.</p>	

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 mensubmodulnya langsung dari repositori pihak ketiga mereka.

- cMock: kloningnya (submodule jika Anda menggunakan git) langsung dari [Cmock](#) repositori.
- jsmn: tidak direkomendasikan dan tidak lagi didukung.
- lwip: kloningnya (submodule jika Anda menggunakan git) langsung dari [lwip-tcpip](#) repositori.
- lwip_osal: lihat Integrasi Referensi Unggulan FreeRTOS di [ii.mx RT1060](#) atau [STM32U5](#) untuk cara mengimplementasikan lwip_osal pada platform/papan perangkat keras Anda.
- mbedtls: kloningnya (submodule jika Anda menggunakan git) langsung dari [MBED-TLS](#) repositori. Konfigurasi dan utilitas mbedtls dapat digunakan kembali; buat salinan lokal dalam kasus ini.
- pkcs11: kloningnya (submodule jika Anda menggunakan git) langsung dari salah satu [CorePKCS11](#) Perpustakaan atau [OASIS PKCS 11](#) repositori.
- tinycbor: kloningnya (submodule jika Anda menggunakan git) langsung dari [tinycbor](#) repositori.
- 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 [tinycrypt](#) repositori.
- tracealyzer_recorder: kloningnya (submodule jika Anda menggunakan git) langsung dari Percepio [perekam jejak](#) repositori.
- kesatuan: kloningnya (submodule jika Anda menggunakan git) langsung dari [ThrowTheSwitch/Persatuan](#) repositori.

- win_pcap: win_pcap tidak lagi dipertahankan. Kami menyarankan Anda menggunakan libslirp, libpcap (posix), atau npcap sebagai gantinya.

Tes porting dan tes integrasi

Semua tes di bawah/testsfolder yang diperlukan untuk memvalidasi integrasi pustaka FreeRTOS dimigrasikan ke[Freertos-Perpustakaan-Integrasi-Tes](#) repositori. Ini dapat digunakan untuk menguji implementasi PAL dan integrasi perpustakaan. Tes yang sama digunakan oleh AWS IoT Device Tester (IDT) untuk[AWS Program Kualifikasi Perangkat untuk FreeRTOS](#).

Dokumentasi yang diarsipkan FreeRTOS

Arsip Panduan Pengguna FreeRTOS

Versi sebelumnya dari Panduan Pengguna FreeRTOS ini 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

Isi Panduan Pengguna FreeRTOS sebelumnya

Konten ini sudah usang tetapi disediakan di sini untuk referensi.

Lihat [Memulai 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 ada berdasarkan repositori Amazon-freertos yang sekarang tidak digunakan lagi, lihat [Panduan Migrasi Repositori Github Amazon-freertos](#).

Tutorial Memulai dengan FreeRTOS ini menunjukkan cara mengunduh dan mengkonfigurasi FreeRTOS pada mesin host, dan kemudian mengkompilasi dan menjalankan aplikasi demo sederhana di [papan mikrokontroler yang memenuhi syarat](#).

Sepanjang tutorial ini, kami berasumsi bahwa Anda sudah familiar dengan AWS IoT dan AWS IoT konsol. Jika tidak, sebaiknya Anda menyelesaikan tutorial [AWS IoT Memulai](#) sebelum melanjutkan.

Topik:

- [Aplikasi demo demo FreeRTOS aplikasi demo aplikasi demo](#)

- [Langkah pertama](#)
- [Memulai masalah saat memulai](#)
- [Menggunkan CMake dengan FreeRTOS](#)
- [Penyediaan kunci mode pengembang](#)
- [Panduan memulai khusus papan](#)
- [Langkah selanjutnya dengan FreeRTOS](#)

Aplikasi demo demo FreeRTOS aplikasi demo aplikasi demo

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 ada berdasarkan repositori Amazon-freertos yang sekarang tidak digunakan lagi, lihat [Panduan Migrasi Repositori Github Amazon-freertos](#).

Aplikasi demo dalam tutorial ini adalah demo CoreMQTT Agen didefinisikan dalam *freertos/demos/coreMQTT_Agent/mqtt_agent_task.c* file. Ini menggunakan [Perpustakaan CoreMQTT](#) untuk terhubung ke AWS Cloud dan kemudian mempublikasikan pesan secara berkala ke topik MQTT yang diselenggarakan oleh [brokerAWS IoT MQTT](#).

Hanya satu aplikasi demo FreeRTOS yang dapat berjalan dalam satu waktu. Saat Anda membangun proyek demo FreeRTOS, demo pertama yang diaktifkan di file *freertos/vendors/vendor/boards/board/aws_demos/config_files/aws_demo_config.h* header adalah aplikasi yang berjalan. Untuk tutorial ini, Anda tidak perlu mengaktifkan atau menonaktifkan demo apa pun. Demo agen CoreMQTT Agen Agen diaktifkan secara default.

Untuk informasi selengkapnya tentang aplikasi demo yang disertakan dengan FreeRTOS, lihat [Demo Demo 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, ESP-WROVER-KIT, atau ESP32-WROOM-32SE, lewati langkah-langkah ini dan pergi ke. [Memulai dengan Espressif ESP32- DevKit C dan ESP-WROVER-KIT](#)
- Jika Anda menggunakan Nordic NRF52840-dk, lewati langkah-langkah ini dan pergi ke. [Memulai dengan Nordic NRF52840-DK](#)

1. [Menyiapkan AWS akun dan izin Anda](#)
2. [Mendaftarkan papan MCU Anda dengan AWS IoT](#)
3. [Mengunduh FreeRTOS](#)
4. [Mengkonfigurasi demo FreeRTOS](#)

Menyiapkan AWS akun dan izin Anda

Mendaftar Akun AWS

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

Untuk mendaftar Akun AWS

1. Buka <https://portal.aws.amazon.com/billing/signup>.
2. Ikuti petunjuk secara online.

Anda akan diminta untuk menerima panggilan telepon dan memasukkan kode verifikasi pada keypad telepon sebagai bagian dari prosedur pendaftaran.

Saat Anda mendaftarkan Akun AWS, Pengguna root akun AWS akan dibuat. Pengguna root memiliki akses ke semua Layanan AWS dan sumber daya dalam akun. Sebagai praktik terbaik keamanan, [tetapkan akses administratif ke pengguna administratif](#), dan hanya gunakan pengguna root untuk melakukan [tugas yang memerlukan akses pengguna root](#).

AWS akan mengirimkan email konfirmasi kepada Anda setelah proses pendaftaran selesai. Anda dapat melihat aktivitas akun saat ini dan mengelola akun dengan mengunjungi <https://aws.amazon.com/> dan memilih Akun Saya.

Membuat pengguna administratif

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

Mengamankan Pengguna root akun AWS Anda

1. Masuk ke [AWS Management Console](#) sebagai pemilik akun dengan memilih Pengguna root dan memasukkan alamat email Akun AWS Anda. Di halaman berikutnya, masukkan kata sandi Anda.

Untuk bantuan masuk menggunakan pengguna root, lihat [Masuk sebagai pengguna root](#) dalam Panduan Pengguna AWS Sign-In.

2. Aktifkan autentikasi multi-faktor (MFA) untuk pengguna root Anda.

Untuk petunjuknya, silakan lihat [Mengaktifkan perangkat MFA virtual untuk pengguna root Akun AWS Anda \(konsol\)](#) dalam Panduan Pengguna IAM.

Membuat pengguna 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 sebuah pengguna administratif.

Untuk mendapatkan tutorial tentang menggunakan Direktori Pusat Identitas IAM sebagai sumber identitas Anda, silakan lihat [Mengonfigurasi akses pengguna dengan Direktori Pusat Identitas IAM default](#) di Panduan Pengguna AWS IAM Identity Center.

Masuk sebagai pengguna administratif

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

Untuk bantuan masuk menggunakan pengguna Pusat Identitas IAM, lihat [Masuk ke portal akses AWS](#) dalam Panduan Pengguna AWS Sign-In.

Untuk memberikan akses, tambahkan izin ke pengguna, grup, atau peran Anda:

- Pengguna dan grup di AWS IAM Identity Center:

Buat rangkaian izin. Ikuti petunjuk dalam [Buat set izin](#) dalam Panduan Pengguna AWS IAM Identity Center.

- Pengguna yang dikelola di IAM melalui penyedia identitas:

Buat peran untuk federasi identitas. Ikuti petunjuk dalam [Membuat peran untuk penyedia identitas pihak ketiga \(federasi\)](#) di Panduan Pengguna IAM.

- Pengguna IAM:

- Buat peran yang dapat diambil pengguna Anda. Ikuti petunjuk dalam [Membuat peran untuk pengguna IAM](#) dalam Panduan Pengguna IAM.
- (Tidak disarankan) Pasang kebijakan langsung ke pengguna atau tambahkan pengguna ke grup pengguna. Ikuti petunjuk di [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": "xxxxxxxxxxxxx-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 Wilayah dan ID akun AWS 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 IoTkonsol](#).
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 melanjutkan [Mengunduh 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

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 IoTkonsol](#).
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 kredensial 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` *Jenis keamanan jaringan Wi-Fi Anda*

Jenis keamanan yang valid adalah:

- `eWiFiSecurityOpen` (Terbuka, tidak ada keamanan)
- `eWiFiSecurityWEP` (Keamanan WEP)
- `eWiFiSecurityWPA` (Keamanan WPA)
- `eWiFiSecurityWPA2` (Keamanan WPA2)

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 kredensial 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, bukakan `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 papan](#) Aplikasi demo yang digunakan dalam tutorial Memulai adalah demo CoreMQTT Mutual Authentication, yang terletak di. `demos/coreMQTT/mqtt_demo_mutual_auth.c`

Memulai masalah saat 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 ada berdasarkan repositori Amazon-freertos yang sekarang tidak digunakan lagi, lihat [Panduan Migrasi Repositori Github Amazon-freertos](#).

Berikut ini dapat membantu Anda memecahkan masalah yang terjadi saat memulai dengan FreeRTOS:

Topik

- [Umum memulai tips pemecahan masalah](#)
- [Memasang emulator terminal](#)

Untuk pemecahan masalah khusus papan, lihat [Memulai dengan FreeRTOS](#) panduan untuk papan Anda.

Umum memulai tips pemecahan masalah

Tidak ada pesan yang muncul diAWS IoT konsol setelah saya menjalankan proyek demo Hello World. Apa yang harus saya lakukan?

Coba cara berikut ini:

1. Buka sebuah jendela terminal untuk melihat output logging dari 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 dipotong. Bagaimana saya bisa meningkatkan panjangnya?

Tingkatkan nilai `configLOGGING_MAX_MESSAGE_LENGTH` ke 255 dalam `FreeRTOSconfig.h` file untuk demo yang Anda jalankan:

```
#define configLOGGING_MAX_MESSAGE_LENGTH    255
```

Memasang 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, dan macOS, dan dan.

Anda harus menghubungkan papan ke komputer sebelum mencoba membuat koneksi serial ke papan Anda dengan emulator terminal.

Gunakan pengaturan berikut untuk mengonfigurasi emulator terminal:

Pengaturan Terminal	Nilai
laju BAUD	115200
Data	8 bit
Paritas	tidak ada
Berhenti	1 bit

Pengaturan Terminal	Nilai
Kontrol aliran	tidak ada

Menemukan port serial papan Anda

Jika Anda tidak tahu port serial papan 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 menggunakan CMake untuk membangun dan menjalankan aplikasi.

Papan berikut mendukung CMake:

- Espressif ESP32-DevKit C
- Espressif ESP-WROVER KIT
- Kit Konektivitas IoT Infineon XMC4800
- KitAWS IoT Pemula Marvell MW320
- KitAWS IoT Pemula Marvell MW322
- Bundel Microchip Keingintahuan PIC32MZEF
- Kit Pengembangan Nordic NRF52840 DK
- Kit Penemuan STMicroelectronicsSTM32L4 IoT Node
- Texas Instruments CC3220SF-LAUNCHXL
- Microsoft Windows Simulator Simulator Windows Simulator

Lihat topik di bawah ini untuk informasi selengkapnya tentang cara menggunakan 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:

- Toolchain kompilasi perangkat Anda harus mendukung sistem operasi alat berat. CMake mendukung semua versi Windows, dan Linux

Subsistem Windows untuk Linux (WSL) tidak didukung. Gunakan CMake asli pada mesin Windows.


- Anda harus menginstal CMake versi 3.13 atau lebih tinggi.

Anda dapat mengunduh distribusi biner CMake dari [CMake.org](https://cmake.org).

 Note

Jika Anda mengunduh distribusi biner CMake, pastikan Anda menambahkan CMake yang dapat dieksekusi ke variabel lingkungan PATH sebelum Anda menggunakan CMake dari baris perintah.


Anda juga dapat mengunduh dan menginstal CMake menggunakan pengelola paket, seperti [homebrew](https://brew.sh) di macOS, dan [scoop](https://scoop.sh) atau [chocolatey](https://chocolatey.org) di Windows.

 Note

Versi paket CMake yang disediakan dalam 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](https://linuxbrew.sh) atau [nix](https://nixos.org).

- Anda harus memiliki sistem build native yang kompatibel.

CMake dapat menargetkan banyak sistem build asli, termasuk [GNU Make](https://www.gnu.org/software/make/) atau [Ninja](https://ninja-build.org/). Make dan Ninja dapat diinstal dengan manajer paket di Linux, macOS dan Windows. Jika Anda menggunakan Make on Windows, Anda dapat menginstal versi mandiri dari [Equation](https://equation-engine.com/), atau Anda dapat menginstal [MinGW](https://mingw-w64.org/), 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 baris perintah CMake untuk membangun proyek Anda dari dalam VS Code. Untuk informasi selengkapnya tentang cara menggunakan CMake untuk membangun aplikasi FreeRTOS, lihat [Membangun FreeRTOS dengan CMake](#).

Untuk debugging, Anda dapat memberikan Kode VS 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 compiler yang ingin Anda gunakan. Di FreeRTOS, kami menyediakan file toolchain default di `freertos/tools/cmake/toolchains`. Cara untuk menyediakan file ini ke CMake tergantung pada apakah Anda menggunakan antarmuka baris perintah CMake atau GUI. Untuk lebih jelasnya, ikuti [Menghasilkan file build \(alat baris perintah CMake\)](#) petunjuk di bawah ini. Untuk informasi lebih lanjut tentang cross-compiling di CMake, lihat [CrossCompiling](#) di wiki CMake resmi.

Untuk membangun proyek berbasis CMake

1. Jalankan CMake untuk menghasilkan file build untuk sistem build asli, seperti Make atau Ninja.

Anda dapat menggunakan [alat baris perintah CMake](#) atau [CMake GUI](#) untuk menghasilkan file build untuk sistem build asli Anda.

Untuk informasi tentang membuat file build FreeRTOS, lihat [Menghasilkan file build \(alat baris perintah CMake\)](#) dan [Menghasilkan file build \(CMake GUI\)](#).

2. Memanggil 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 (alat baris perintah CMake)

Anda dapat menggunakan alat baris perintah CMake (cmake) untuk menghasilkan file build untuk FreeRTOS. Untuk menghasilkan file build, Anda perlu menentukan papan target, kompilator, 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

Compiler harus dalam PATH variabel sistem, atau Anda harus menentukan lokasi kompilator.

Misalnya, jika vendornya adalah Texas Instruments, dan papannya adalah Launchpad CC3220, dan kompilernya adalah GCC untuk ARM, Anda dapat mengeluarkan perintah berikut untuk membuat 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. Misalnya:

```
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, sehingga Anda tidak perlu menggunakan nama lengkap vendor dan papan untuk `VENDOR` dan `BOARD` pilihan. Nama sebagian bekerja, asalkan ada satu pertandingan. 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 default `cmake/toolchains`. Misalnya:

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

Jika file toolchain tidak menggunakan path absolut untuk compiler Anda, dan Anda tidak menambahkan compiler 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 tersebut. Opsi ini mencari jalur direktori toolchain yang ditentukan dan subfolder toolchain di bawah `bin`. Misalnya:

```
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` ke `debug`. Dengan opsi ini diaktifkan, CMake menambahkan tanda 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
```

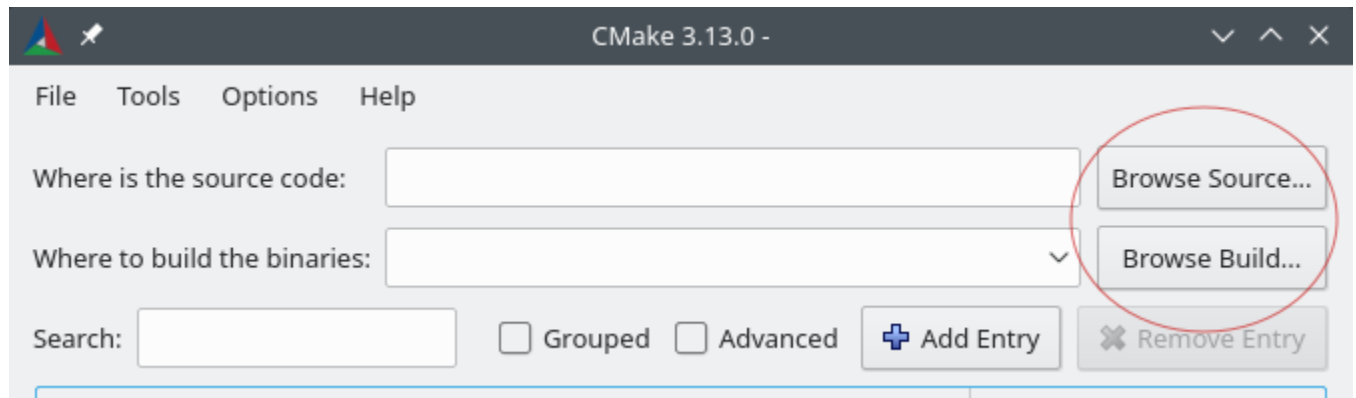
Anda juga dapat mengatur `CMAKE_BUILD_TYPE` to `release` untuk menambahkan bendera optimasi ke opsi kompilasi.

Menghasilkan file build (CMake GUI)

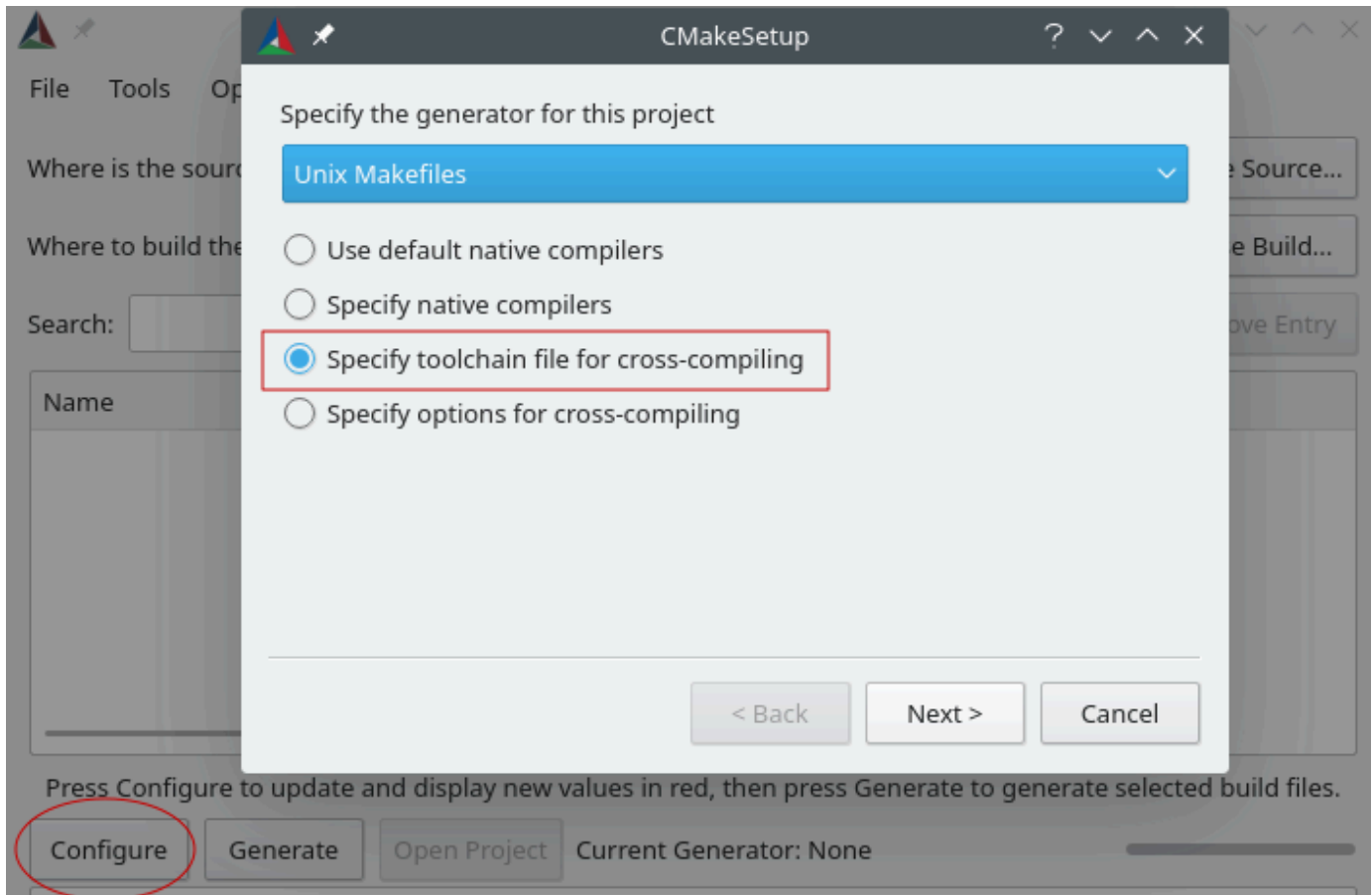
Anda dapat menggunakan CMake GUI untuk menghasilkan file build FreeRTOS.

Untuk menghasilkan file build dengan CMake GUI

1. Dari baris perintah, masalah `cmake-gui` untuk memulai GUI.
2. Pilih Jelajahi Sumber dan tentukan input sumber, lalu pilih Jelajahi Bangun dan tentukan keluaran build.



3. Pilih Konfigurasi, dan di bawah Tentukan generator build untuk proyek ini, temukan dan pilih sistem build yang ingin Anda gunakan untuk membuat file build yang dihasilkan. jika Anda tidak melihat jendela pop up, Anda mungkin menggunakan kembali direktori build yang ada. Dalam hal ini, hapus cache CMake 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 template, yang tidak menyediakan target lapisan portabel apa pun. Akibatnya, sebuah jendela muncul dengan pesan `Error in configuration process`.

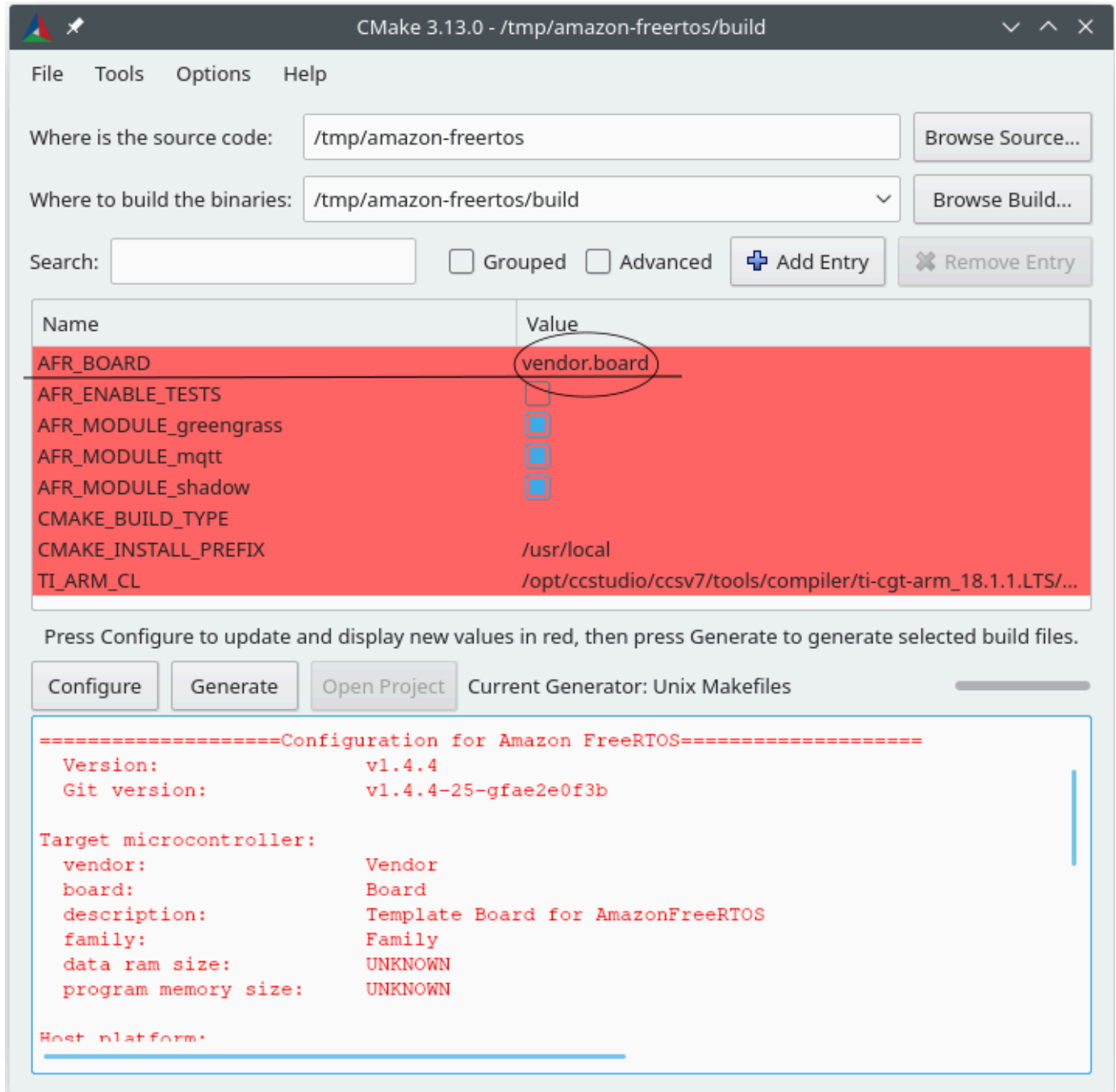
Note

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 compiler tidak dalam variabel `PATH` lingkungan Anda. Anda dapat mengatur `AFR_TOOLCHAIN_PATH` variabel dalam GUI untuk memberitahu CMake di mana Anda menginstal compiler Anda. Jika Anda tidak melihat `AFR_TOOLCHAIN_PATH` variabelnya, pilih Add Entry. Di jendela pop up, di bawah Nama, ketik `AFR_TOOLCHAIN_PATH`. Di bawah Compiler Path ketik path ke kompiler Anda. misalnya, `C:/toolchains/arm-none-eabi-gcc`.

6. GUI seharusnya terlihat seperti ini:



Pilih AFR_BOARD, pilih papan Anda, lalu pilih Konfigurasi lagi.

7. Pilih Hasilkan. CMake menghasilkan file sistem build (misalnya, makefiles atau file ninja), dan file-file ini muncul di direktori build yang Anda tentukan pada langkah pertama. Ikuti petunjuk di bagian berikutnya 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 as your native build system, jalankan perintah berikut:

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

Membangun dengan CMake

Anda juga dapat menggunakan alat baris perintah CMake untuk membangun FreeRTOS. CMake menyediakan lapisan abstraksi untuk memanggil sistem build asli. Misalnya:

```
cmake --build build_dir
```

Berikut adalah beberapa kegunaan umum lainnya dari mode build alat baris perintah CMake:

```
# Take advantage of CPU cores.
cmake --build build_dir --parallel 8
```

```
# Build specific targets.
cmake --build build_dir --target afr_kernel
```

```
# Clean first, then build.
cmake --build build_dir --clean-first
```

Untuk informasi selengkapnya tentang mode build CMake, lihat [dokumentasi CMake](#).

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 krypto khusus. Bagian ini menyediakan logika untuk bekerja dalam kemampuan kriptografi perangkat Anda.

Opsi #1: kunci pribadi impor dari AWS IoT

Untuk tujuan pengujian lab, jika perangkat Anda mengizinkan impor kunci pribadi, ikuti petunjuk di dalamnya [Mengkonfigurasi demo FreeRTOS](#).

Opsi #2: pembuatan kunci pribadi onboard

Jika perangkat Anda memiliki elemen aman, atau jika Anda lebih suka membuat key pair perangkat dan sertifikat Anda sendiri, ikuti petunjuk di sini.

Konfigurasi awal

Pertama, lakukan langkah-langkah di [Mengkonfigurasi demo FreeRTOS](#), tetapi lewati langkah terakhir (yaitu, jangan lakukan Untuk memformat AWS IoT kredensial Anda). Hasil bersih harus `bahwademos/include/aws_clientcredential.h` file telah diperbarui dengan pengaturan Anda, tetapi `idemos/include/aws_clientcredential_keys.h` file belum.

Konfigurasi Proyek Demo

Buka demo CoreMQTT Mutual Authentication seperti yang dijelaskan dalam panduan untuk papan Anda di [Panduan memulai khusus papan](#).

Dalam proyek, buka `fileaws_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 dikecualikan dari proyek demo FreeRTOS untuk mengurangi ukuran ROM. Sebagai gantinya, untuk tujuan pengujian lab, buat kunci privat pada 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 Anda 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 penandatanganan CSR di perangkat keras terpisah, dan mengonfigurasi otoritas sertifikat sehingga hanya mengeluarkan sertifikat untuk permintaan yang telah ditandatangani oleh kunci tertentu tersebut. Kunci itu juga harus tetap berada di bawah kendali administrator yang ditunjuk.

Sertifikat Impor

Dengan sertifikat yang dikeluarkan, langkah selanjutnya adalah mengimpornya ke perangkat Anda. Anda juga perlu mengimpor sertifikat Otoritas Sertifikat (CA) Anda, karena diperlukan agar otentikasi pertama kali AWS IoT berhasil saat menggunakan JITP. Dalam `aws_clientcredential_keys.h` file dalam proyek Anda, aturkey `CLIENT_CERTIFICATE_PEM` makro menjadi konten `DeviceCert.pem` dan aturkey `JITR_DEVICE_CERTIFICATE_AUTHORITY_PEM` makro menjadi isinyarootCA.pem.

Otorisasi Perangkat

Impor `deviceCert.pem` ke AWS IoT registri seperti yang dijelaskan di [Gunakan Sertifikat Anda Sendiri](#). Anda harus membuat AWS IoT hal baru, melampirkan sertifikat PENDING dan kebijakan untuk hal Anda, kemudian menandai 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 papan

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 menyelesaikan [Langkah pertama](#), lihat panduan papan Anda untuk petunjuk khusus papan tentang memulai dengan FreeRTOS:

- [Memulai dengan Cypress CYW943907AEVAL1F Development Kit](#)
- [Memulai dengan Cypress CYW954907AEVAL1F Development Kit](#)
- [Memulai dengan kit Cypress CY8CKIT-064S0S2-4343W](#)
- [Memulai dengan Kit Konektivitas IoT Infineon XMC4800](#)
- [Memulai dengan MW32xAWS IoT Starter Kit](#)
- [Memulai dengan kit pengembangan MediaTek MT7697hx](#)
- [Memulai dengan Microchip Curiosity PIC32MZ EF](#)
- [Memulai dengan Nuvoton NuMaker -IoT-M487](#)
- [Memulai dengan Modul IoT NXP LPC54018](#)
- [Memulai dengan Renesas Starter Kit+untuk RX65N-2MB](#)
- [Memulai dengan STMicroelectronics STM32L4 Discovery Kit IoT Node](#)

- [Memulai dengan Texas Instruments CC3220SF-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 ATECC608A Secure Element dengan simulator Windows](#)
- [Memulai dengan Espressif ESP32- DevKit C dan ESP-WROVER-KIT](#)
- [Memulai dengan Espressif ESP32-WROOM-32SE](#)
- [Memulai dengan Espressif ESP32-S2](#)
- [Memulai dengan Infineon OPTIGA Trust X dan XMC4800 IoT Connectivity Kit](#)
- [Memulai dengan Nordic NRF52840-DK](#)

Memulai dengan Cypress CYW943907AEVAL1F Development 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 Cypress CYW943907AEVAL1F Development Kit. Jika Anda tidak memiliki Kit Pengembangan Cypress CYW943907AEVAL1F, kunjungi Katalog Perangkat AWS Mitra untuk membelinya dari [mitra](#) kami.

Note

Tutorial ini memandu Anda melakukan penyiapan dan menjalankan demo CoreMQTT Mutual Authentication. Port FreeRTOS untuk papan ini saat ini tidak mendukung server TCP dan demo klien.

Sebelum memulai, Anda harus mengonfigurasi AWS IoT dan mengunduh FreeRTOS untuk menghubungkan perangkat Anda ke AWS Cloud. 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 mengandung karakter spasi.
- Panjang maksimal jalur file di Microsoft Windows adalah 260 karakter. Jalur direktori unduhan FreeRTOS yang panjang dapat menyebabkan kegagalan build.
- Karena kode sumber mungkin berisi tautan simbolik, 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:

- Atur `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`, `git submodule update --init --recursive`).
- Seperti disebutkan dalam [Mengunduh 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. Cross kompilasi 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 melalui koneksi serial, untuk tujuan pemantauan dan debugging.

Menyiapkan lingkungan pengembangan Anda

Unduh dan instal SDK WICED Studio

Dalam panduan Memulai ini, Anda menggunakan Cypress WICED Studio SDK untuk memprogram papan Anda dengan demo FreeRTOS. Kunjungi situs web [Perangkat Lunak WICED](#) untuk mengunduh WICED Studio SDK dari Cypress. Anda harus mendaftar akun Cypress gratis untuk mengunduh perangkat lunak. WICED Studio SDK kompatibel dengan sistem operasi Windows, macOS, dan Linux.

Note

Beberapa sistem operasi memerlukan langkah instalasi tambahan. Pastikan bahwa Anda membaca dan mengikuti semua petunjuk instalasi untuk sistem operasi dan versi WICED Studio yang 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 bernama `WICED-Studio-m.n` pada mesin Anda di manam dann 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. Ketika 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, dan kemudian 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 mesinmu, dan tambahkan berikut ini ke baris terakhir file:

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

2. Mulai ulang mesinmu.
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 dan papan Anda

1. Connect papan ke komputer host Anda dengan kabel USB Standard-A ke Micro-B.
2. Identifikasi nomor port serial USB untuk koneksi ke papan pada 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 lebih lanjut tentang menginstal terminal dan menyiapkan koneksi serial, lihat [Memasang emulator terminal](#).

Memantau pesan MQTT di cloud

Sebelum Anda menjalankan proyek demo FreeRTOS, Anda dapat mengatur klien MQTT di AWS IoT konsol untuk memantau pesan yang dikirim perangkat Anda ke AWS Cloud.

Untuk berlangganan topik MQTT dengan klien AWS IoT MQTT

1. Masuk ke [konsol AWS IoT](#) tersebut.
2. Di panel navigasi, pilih Uji, lalu pilih klien uji MQTT untuk membuka klien MQTT.
3. Di Topik berlangganan *your-thing-name/example/topic*, masukkan, lalu pilih Berlangganan topik.

Bangun dan jalankan proyek demo FreeRTOS

Setelah Anda mengatur koneksi serial ke papan Anda, Anda dapat membangun proyek demo FreeRTOS, flash demo ke papan Anda, dan kemudian menjalankan demo.

Untuk membangun dan menjalankan proyek demo FreeRTOS di WICED Studio

1. Meluncurkan WICED Studio.
2. Dari menu File, pilih Impor. Perluas General folder, pilih Proyek yang Ada ke Ruang Kerja, lalu pilih Berikutnya.
3. Di Pilih direktori root, pilih Browse... , arahkan ke jalur *freertos/projects/cypress/CYW943907AEVAL1F/wicedstudio*, dan kemudian pilih OK.
4. Di bawah Proyek, centang kotak untuk hanya proyek aws_demo. Pilih Finish untuk mengimpor proyek. Proyek target aws_demo akan muncul di jendela Make Target.
5. Perluas menu Platform WICED dan pilih Filter WICED.

6. Di jendela Make Target, luaskan `aws_demo`, klik `kanandemo.aws_demo` file, lalu pilih Build Target untuk membangun dan mengunduh demo ke papan Anda. Demo harus berjalan secara otomatis setelah dibangun dan diunduh ke papan Anda.

Pemecahan Masalah

- Jika Anda menggunakan Windows, Anda mungkin menerima galat berikut ketika Anda membangun dan menjalankan proyek demo:

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

Untuk memecahkan masalah eror 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 eror berikut saat membuat dan menjalankan project demo:

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

Untuk memecahkan masalah eror ini, gunakan perintah berikut untuk memperbarui paket `libusb-dev`:

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

Untuk informasi pemecahan masalah umum tentang Memulai FreeRTOS, lihat [Memulai masalah saat memulai](#).

Memulai dengan Cypress CYW954907AEVAL1F Development 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 petunjuk untuk memulai dengan Cypress CYW954907AEVAL1F Development Kit. Jika Anda tidak memiliki Kit Pengembangan Cypress CYW954907AEVAL1F, kunjungi Katalog Perangkat AWS Mitra untuk membelinya dari [mitra](#) kami.

Note

Tutorial ini memandu Anda mengatur dan menjalankan demo CoremQtt Mutual Authentication. Port FreeRTOS untuk papan ini saat ini tidak mendukung server TCP dan demo klien.

Sebelum memulai, Anda harus mengonfigurasi AWS IoT dan mengunduh FreeRTOS Anda untuk menghubungkan perangkat Anda ke AWS Cloud. Lihat [Langkah pertama](#) untuk instruksi. Dalam tutorial ini, path ke direktori download 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 mengandung karakter spasi.
- Panjang maksimal jalur file di Microsoft Windows adalah 260 karakter. Jalur direktori unduhan FreeRTOS yang panjang dapat menyebabkan kegagalan build.
- Karena kode sumber mungkin berisi tautan simbolik, 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 in Windows 10!](#).

Jika Anda menggunakan Git di bawah Windows, Anda harus mengaktifkan Mode Pengembang atau Anda harus:

- Atur `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`, `git submodule update --init --recursive`).
- Seperti disebutkan dalam [Mengunduh 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. Cross kompilasi 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 melalui koneksi serial, untuk tujuan pemantauan dan debugging.

Menyiapkan lingkungan pengembangan Anda

Unduh dan instal SDK WICED Studio

Dalam panduan Memulai ini, Anda menggunakan Cypress WICED Studio SDK untuk memprogram papan Anda dengan demo FreeRTOS. Kunjungi situs web [Perangkat Lunak WICED](#) untuk mengunduh WICED Studio SDK dari Cypress. Anda harus mendaftar akun Cypress gratis untuk mengunduh perangkat lunak. WICED Studio SDK kompatibel dengan sistem operasi Windows, macOS, dan Linux.

Note

Beberapa sistem operasi memerlukan langkah instalasi tambahan. Pastikan bahwa Anda membaca dan mengikuti semua petunjuk instalasi untuk sistem operasi dan versi WICED Studio yang 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 bernama `WICED-Studio-m.n` pada mesin Anda di manam dann 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. Ketika 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, dan kemudian 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 mesinmu, dan tambahkan berikut ini ke baris terakhir file:

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

2. Mulai ulang mesinnya.
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 dan papan Anda

1. Connect papan ke komputer host Anda dengan kabel USB Standard-A ke Micro-B.
2. Identifikasi nomor port serial USB untuk koneksi ke papan pada 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 lebih lanjut tentang menginstal terminal dan menyiapkan koneksi serial, lihat [Memasang emulator terminal](#).

Memantau pesan MQTT di cloud

Sebelum Anda menjalankan proyek demo FreeRTOS, Anda dapat mengatur klien MQTT di AWS IoT konsol untuk memantau pesan yang dikirim perangkat Anda ke AWS Cloud.

Untuk berlangganan topik MQTT dengan klien AWS IoT MQTT

1. Masuk ke [konsol AWS IoT](#) tersebut.
2. Di panel navigasi, pilih Uji, lalu pilih klien uji MQTT untuk membuka klien MQTT.

3. Di Topik berlanggananyour-thing-name/example/topic, masukkan, lalu pilih Berlangganan topik.

Bangun dan jalankan proyek demo FreeRtos

Setelah Anda mengatur koneksi serial ke papan Anda, Anda dapat membangun proyek demo FreeRTOS, flash demo ke papan Anda, dan kemudian menjalankan demo.

Untuk membangun dan menjalankan proyek demo FreeRTOS di WICED Studio

1. Meluncurkan WICED Studio.
2. Dari menu File, pilih Impor. PerluasGeneral folder, pilih Proyek yang Ada ke Ruang Kerja, lalu pilih Berikutnya.
3. Di Pilih direktori root, pilih Browse... , arahkan ke jalurfreertos/projects/cypress/CYW954907AEVAL1F/wicedstudio, dan kemudian pilih OK.
4. Di bawah Proyek, centang kotak untuk hanya proyek aws_demo. Pilih Finish untuk mengimpor proyek. Proyek target aws_demo akan muncul di jendela Make Target.
5. Perluas menu Platform WICED dan pilih Filter WICED.
6. Di jendela Make Target, luaskan aws_demo, klik kanandemo . aws_demo file, lalu pilih Build Target untuk membangun dan mengunduh demo ke papan Anda. Demo harus berjalan secara otomatis setelah dibangun dan diunduh ke papan Anda.

Pemecahan Masalah

- Jika Anda menggunakan Windows, Anda mungkin menerima galat berikut ketika Anda membangun dan menjalankan proyek demo:

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

Untuk memecahkan masalah eror ini, lakukan hal berikut:

1. JelajahiWICED-Studio-SDK-PATH\WICED-Studio-6.2\43xxx_Wi-Fi\tools\OpenOCD \Win32 dan klik dua kaliopenocd-all-brcm-libftdi.exe.
2. JelajahiWICED-Studio-SDK-PATH\WICED-Studio-6.2\43xxx_Wi-Fi\tools\drivers \CYW9WCD1EVAL1 dan klik dua kaliInstallDriver.exe.

- Jika Anda menggunakan Linux atau macOS, Anda mungkin menerima error berikut saat membuat dan menjalankan project demo:

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

Untuk memecahkan masalah error ini, gunakan perintah berikut untuk memperbarui paket libusb-dev:

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

Untuk informasi pemecahan masalah umum tentang Memulai FreeRTOS, lihat [Memulai masalah saat memulai](#).

Memulai dengan kit Cypress CY8CKIT-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 instruksi untuk memulai dengan [CY8CKIT-064S0S2-4343W](#) kit. Jika Anda belum memilikinya, Anda dapat menggunakan tautan itu untuk membeli kit. Anda juga dapat menggunakan tautan itu untuk mengakses panduan pengguna kit.

Mulai

Sebelum Anda mulai, Anda harus mengkonfigurasi AWS IoT dan FreeRTOS untuk menghubungkan perangkat Anda ke AWS. Untuk petunjuk, lihat [Langkah pertama](#). Setelah Anda menyelesaikan prasyarat, Anda akan memiliki paket FreeRTOS AWS IoT Core kredensi.

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 CMake atau Make. Anda dapat menggunakan ModusToolbox untuk 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 Windows, macOS, dan Linux.

Sebelum Anda memulai, unduh dan instal hal terbaru [ModusToolbox perangkat lunak](#). Untuk informasi lebih lanjut, lihat [ModusToolbox Panduan Instalasi](#).

Memperbarui alat untuk ModusToolbox 2.1 atau lebih

Jika Anda menggunakan ModusToolbox 2.1 Eclipse IDE untuk memprogram kit ini, Anda harus memperbarui alat OpenOCD dan Firmware-loader.

Di 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 OpenOCD

1. Pergi ke [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 di `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. Pergi ke [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 di `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 aliran CMake, unduh dan instal Cypress Programmer dari [Solusi Pemrograman](#) halaman web. Untuk informasi selengkapnya, lihat [Menggunakan CMake dengan FreeRTOS](#).

Menyiapkan perangkat Anda

Ikuti langkah-langkah berikut.

1. Menyediakan kit Anda

Ikuti [Panduan Penyediaan untuk Kit CY8CKIT-064S0S2-4343W](#) instruksi untuk menyediakan kit Anda dengan aman AWS IoT.

Kit ini membutuhkan CySecureTools 3.1.0 atau lebih baru.

2. Siapkan koneksi serial
 - a. Sambungkan 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 Pengalih aplikasi di bawah Pelabuhan (COM & LPT).
 - c. Mulai terminal serial dan buka koneksi dengan pengaturan berikut:
 - Tingkat baud: 115200
 - Data: 8 bit
 - Paritas: Tidak ada
 - Hentikan bit: 1

- Pengalih aplikasi

Bangun dan jalankan proyek Demo FreeRTOS

Di bagian ini Anda membangun dan menjalankan demo.

1. Pastikan untuk mengikuti langkah-langkah di [Panduan Penyediaan untuk Kit CY8CKIT-064S0S2-4343W](#).

2. Bangun Demo FreeRTOS.

- a. Buka Pengalih aplikasi untuk ModusToolbox dan pilih, atau buat, ruang kerja.
- b. Dari Berkas menu, pilih Impor.

Perluas Umum, pilih Proyek yang Ada Ke Ruang Kerja, dan kemudian pilih Berikutnya.

- c. Di Direktori Root, 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 Panel cepat, pilih Bangun Aplikasi `aws_demos`.
 - Pilih Proyek dan pilih Bangun semua.

Pastikan proyek dikompilasi tanpa kesalahan.

3. Memantau Pesan MQTT di Cloud

Sebelum Anda menjalankan demo, Anda dapat mengatur klien MQTT di AWS IoT konsol untuk memantau pesan yang dikirim perangkat Anda ke AWS Awan. Untuk berlangganan topik dengan AWS IoT Klien MQTT, ikuti langkah-langkah ini.

- a. Masuk ke [konsol AWS IoT](#) tersebut.
- b. Di panel navigasi, pilih Uji, lalu pilih Pengalih aplikasi untuk membuka klien MQTT.
- c. Untuk Topik berlangganan, masukkan `your-thing-name/example/topic`, dan kemudian pilih Berlangganan topik.

4. Jalankan proyek demo FreeRTOS

- a. Pilih proyek `aws_demos` di ruang kerja.

- b. Dari Panel cepat, pilih Program aws_demos (KitProg3). Ini memprogram papan dan aplikasi demo mulai berjalan setelah pemrograman selesai.
- c. Anda dapat melihat status aplikasi yang menjalankan dalam terminal serial. Gambar berikut menunjukkan bagian dari output terminal.

```

COMS - Tera Term VT
File Edit Setup Control Window Help
WLAN MAC Address : CC:08:79:24:DB:8B
WLAN Firmware   : v10: Jul 30 2019 01:54:48 version 7.45.98.89 (x718486 CY) FWID 01-81376c4b
WLAN CWM       : API: 12.2 Data: 9.10.39 Compiler: 1.29.4 ClmImport: 1.36.3 Creation: 2019-07-30 01:43:02
WHD VERSION    : v1.30.0-rc3-dirty : v1.30.0-rc3 : GCC 7.2 : 2019-08-27 16:29:32 +0800
1 3518 [Tr Svc] Wi-Fi Connected to AP. Creating tasks which use network...
2 3518 [Tr Svc] IP Address acquired 192.168.43.207
3 5083 [Tr Svc] Write certificate...
4 5623 [Tr Svc] Device credential provisioning succeeded.
5 5627 [Iot_thread] [INFO] [INIT][lu] SDK successfully initialized.
6 8504 [Iot_thread] [INFO] [IDEMO][lu] Successfully initialized the demo. Network type for the demo: 1
7 8504 [Iot_thread] [INFO] [MQTT][lu] MQTT library successfully initialized.
8 8504 [Iot_thread] [INFO] [IDEMO][lu] MQTT demo client identifier is cy8cproto-kit (length 13).
9 13409 [Iot_thread] [INFO] [MQTT][lu] Establishing new MQTT connection.
10 13411 [Iot_thread] [INFO] [MQTT][lu] Anonymous metrics (SDK language, SDK version) will be provided to AWS IoT. Recompile with AWS
    [IOT] MQTT_ENABLE_METRICS set to 0 to disable.
11 13412 [Iot_thread] [INFO] [MQTT][lu] <MQTT connection 0x800b4c8, CONNECT operation 0x800b2d0> Waiting for operation completion.
12 13753 [Iot_thread] [INFO] [MQTT][lu] <MQTT connection 0x800b4c8, CONNECT operation 0x800b2d0> Wait complete with result SUCCESS.
13 13754 [Iot_thread] [INFO] [MQTT][lu] New MQTT connection 0x800c864 established.
14 13755 [Iot_thread] [INFO] [MQTT][lu] <MQTT connection 0x800b4c8> SUBSCRIBE operation scheduled.
15 13755 [Iot_thread] [INFO] [MQTT][lu] <MQTT connection 0x800b4c8, SUBSCRIBE operation 0x800b5e0> Waiting for operation completion.
16 14065 [Iot_thread] [INFO] [MQTT][lu] <MQTT connection 0x800b4c8, SUBSCRIBE operation 0x800b5e0> Wait complete with result SUCCESS.
17 14065 [Iot_thread] [INFO] [IDEMO][lu] All demo topic filter subscriptions accepted.
18 14065 [Iot_thread] [INFO] [IDEMO][lu] Publishing messages 0 to 1.
19 14067 [Iot_thread] [INFO] [MQTT][lu] <MQTT connection 0x800b4c8> MQTT PUBLISH operation queued.
20 14069 [Iot_thread] [INFO] [MQTT][lu] <MQTT connection 0x800b4c8> MQTT PUBLISH operation queued.
21 14069 [Iot_thread] [INFO] [IDEMO][lu] Waiting for 2 publishes to be received.
22 14378 [Iot_thread] [INFO] [IDEMO][lu] MQTT PUBLISH 0 successfully sent.
23 14424 [Iot_thread] [INFO] [IDEMO][lu] Incoming PUBLISH received:
Subscription topic filter: iotdemo/topic/1
Publish topic name: iotdemo/topic/1
Publish retain flag: 0
Publish QoS: 1
Publish pay24 14424 [Iot_thread] [INFO] [MQTT][lu] <MQTT connection 0x800b4c8> MQTT PUBLISH operation queued.
25 14425 [Iot_thread] [INFO] [IDEMO][lu] Acknowledgment message for PUBLISH 0 will be sent.
26 14680 [Iot_thread] [INFO] [IDEMO][lu] MQTT PUBLISH 1 successfully sent.
27 14708 [Iot_thread] [INFO] [IDEMO][lu] Incoming PUBLISH received:
Subscription topic filter: iotdemo/topic/2
Publish topic name: iotdemo/topic/2
Publish retain flag: 0
Publish QoS: 1
Publish pay28 14708 [Iot_thread] [INFO] [MQTT][lu] <MQTT connection 0x800b4c8> MQTT PUBLISH operation queued.
29 14708 [Iot_thread] [INFO] [IDEMO][lu] Acknowledgment message for PUBLISH 1 will be sent.
30 14710 [Iot_thread] [INFO] [IDEMO][lu] 2 publishes received.
31 14710 [Iot_thread] [INFO] [IDEMO][lu] Publishing messages 2 to 3.

```

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. Saat 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 FreeTos

Anda juga dapat menggunakan CMake untuk membangun dan menjalankan aplikasi demo. Untuk menyiapkan CMake dan sistem build asli, lihat [Prasyarat](#).

- a. Gunakan perintah berikut untuk menghasilkan file. Tentukan papan target dengan `-DBOARD` pilihan.

```
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` pilihan 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 `arm-none-eabi-gcc` tidak ada di jalur shell Anda, Anda juga perlu mengatur `AFR_TOOLCHAIN_PATH` variabel CMake.

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, lihat [Demo Demo demo FreeRTOS](#).

- Demo Bluetooth Energi Rendah
- Demo Pengalih aplikasi
- Demo Klien Echo Soket Aman
- AWS IoT Demo Pengalih aplikasi

Debugging

The KitProg3 pada kit mendukung debugging melalui protokol SWD.

- Untuk men-debug aplikasi FreeRTOS, pilih proyek `aws_demos` di ruang kerja dan kemudian pilih `aws_demo Debug (KitProg3)` dari Panel cepat.

Pembaruan OTA

PSoC 64 MCU telah lulus semua tes kualifikasi FreeRTOS yang diperlukan. Namun, opsional over-the-air (OTA) fitur diimplementasikan dalam PSoC 64 Standard Secure AWS perustakaan firmware masih menunggu evaluasi. Fitur OTA sebagaimana diterapkan saat ini lulus semua tes kualifikasi OTA kecuali [aws_ota_test_case_rollback_if_unable_to_connect_after_update.py](#).

Saat gambar OTA yang berhasil divalidasi diterapkan ke perangkat menggunakan PSoC64 Standard Secure — AWS MCU dan perangkat tidak dapat berkomunikasi AWS IoT Core, perangkat tidak dapat secara otomatis mengembalikan ke gambar bagus asli yang diketahui. Hal ini dapat menyebabkan perangkat tidak dapat menjalankan AWS IoT Core untuk pembaruan lebih lanjut. Fungsionalitas ini masih dalam pengembangan oleh tim Cypress.

Untuk informasi selengkapnya, lihat [Pengalih aplikasi dengan AWS dan Kit CY8CKIT-064S0S2-4343W](#). Jika Anda memiliki pertanyaan lebih lanjut atau memerlukan dukungan teknis, hubungi [Komunitas Pengalih aplikasi](#).

Memulai dengan Microchip ATECC608A 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 ATECC608A Secure Element dengan Windows Simulator.

Anda memerlukan perangkat keras berikut:

- [Microchip ATECC608A elemen aman clickboard](#)
- [SAM21 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.*

Ikhtisar

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 ATECC608A

Sebelum Anda dapat berinteraksi dengan perangkat Microchip ATECC608A Anda, Anda harus terlebih dahulu memprogram SAM21.

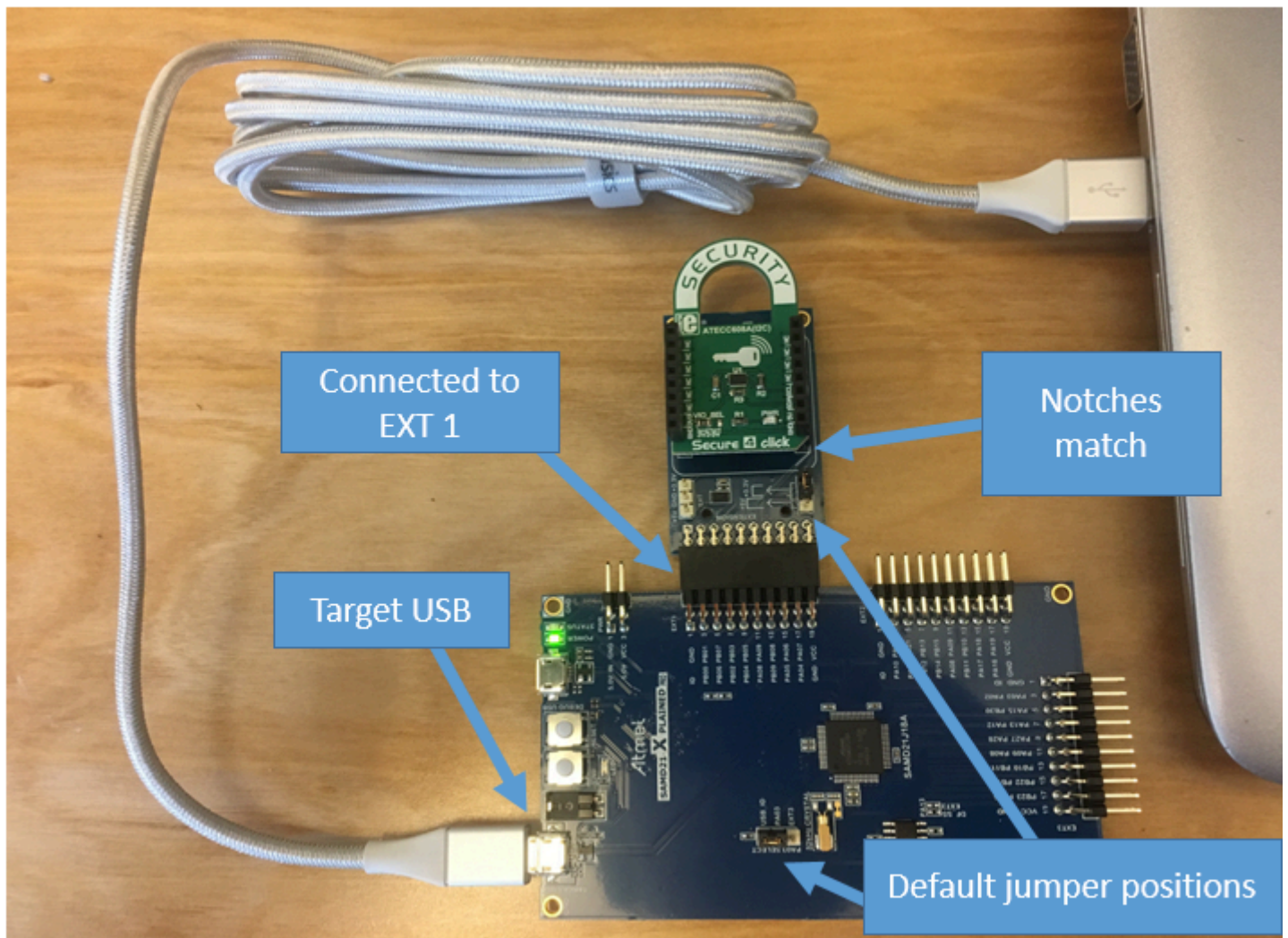
Untuk mengatur papan SAMD21 XPlained Pro

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 papan SAMD21 di lokasi EXT1.
3. Colokkan papan ATECC608A Secure 4 Click ke adaptor MikroBusX XPlained 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 Akun AWS

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

Untuk mendaftar Akun AWS

1. Buka <https://portal.aws.amazon.com/billing/signup>.
2. Ikuti petunjuk secara online.

Anda akan diminta untuk menerima panggilan telepon dan memasukkan kode verifikasi pada keypad telepon sebagai bagian dari prosedur pendaftaran.

Saat Anda mendaftar Akun AWS, Pengguna root akun AWS akan dibuat. Pengguna root memiliki akses ke semua Layanan AWS dan sumber daya dalam akun. Sebagai praktik terbaik

keamanan, [tetapkan akses administratif ke pengguna administratif](#), dan hanya gunakan pengguna root untuk melakukan [tugas yang memerlukan akses pengguna root](#).

AWS akan mengirimkan email konfirmasi kepada Anda setelah proses pendaftaran selesai. Anda dapat melihat aktivitas akun saat ini dan mengelola akun dengan mengunjungi <https://aws.amazon.com/> dan memilih Akun Saya.

Membuat pengguna administratif

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

Mengamankan Pengguna root akun AWS Anda

1. Masuk ke [AWS Management Console](#) sebagai pemilik akun dengan memilih Pengguna root dan memasukkan alamat email Akun AWS Anda. Di halaman berikutnya, masukkan kata sandi Anda.

Untuk bantuan masuk menggunakan pengguna root, lihat [Masuk sebagai pengguna root](#) dalam Panduan Pengguna AWS Sign-In.

2. Aktifkan autentikasi multi-faktor (MFA) untuk pengguna root Anda.

Untuk petunjuknya, silakan lihat [Mengaktifkan perangkat MFA virtual untuk pengguna root Akun AWS Anda \(konsol\)](#) dalam Panduan Pengguna IAM.

Membuat pengguna 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 sebuah pengguna administratif.

Untuk mendapatkan tutorial tentang menggunakan Direktori Pusat Identitas IAM sebagai sumber identitas Anda, silakan lihat [Mengonfigurasi akses pengguna dengan Direktori Pusat Identitas IAM default](#) di Panduan Pengguna AWS IAM Identity Center.

Masuk sebagai pengguna administratif

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

Untuk bantuan masuk menggunakan pengguna Pusat Identitas IAM, lihat [Masuk ke portal akses AWS](#) dalam Panduan Pengguna AWS Sign-In.

Untuk memberikan akses, tambahkan izin ke pengguna, grup, atau peran Anda:

- Pengguna dan grup di AWS IAM Identity Center:

Buat rangkaian izin. Ikuti petunjuk dalam [Buat set izin](#) dalam Panduan Pengguna AWS IAM Identity Center.

- Pengguna yang dikelola di IAM melalui penyedia identitas:

Buat peran untuk federasi identitas. Ikuti petunjuk dalam [Membuat peran untuk penyedia identitas pihak ketiga \(federasi\)](#) di Panduan Pengguna IAM.

- Pengguna IAM:

- Buat peran yang dapat diambil pengguna Anda. Ikuti petunjuk dalam [Membuat peran untuk pengguna IAM](#) dalam Panduan Pengguna IAM.
- (Tidak disarankan) Pasang kebijakan langsung ke pengguna atau tambahkan pengguna ke grup pengguna. Ikuti petunjuk di [Menambahkan izin ke pengguna \(konsol\)](#) dalam Panduan Pengguna IAM.

Menyiapkan

1. [Unduh repo FreeRTOS dari repositori FreeRTOS. GitHub](#)

Untuk mengunduh FreeRTOS dari: GitHub

1. Jelajahi repositori [FreeRTOS GitHub](#).
2. Pilih Klon 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`, `git submodule update -\-init -\-recursive`).

4. Dari *freertos* direktori, periksa cabang yang akan digunakan.
2. Siapkan lingkungan pengembangan Anda.
 - a. Instal [WinPcap](#) versi terbaru.
 - 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 aktif.

Bangun dan jalankan proyek demo FreeRTOS

Important

Perangkat Microchip ATECC608A 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 Debug untuk menjalankan proyek. Pada proses pertama, Anda perlu mengkonfigurasi antarmuka perangkat Anda dan mengkompilasi ulang. Untuk informasi selengkapnya, lihat [Konfigurasi antarmuka jaringan Anda](#).

4. Menyediakan Microchip ATECC608A.

Microchip telah menyediakan beberapa alat skrip untuk membantu pengaturan bagian ATECC608A. 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 ATECC608A dan ekspor kunci publik dan nomor seri perangkat.
3. Buat 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!

Memecahkan masalah

Untuk informasi pemecahan masalah umum, lihat. [Memulai masalah saat 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 ESP32-C3.](#)

Ikuti tutorial ini untuk memulai dengan Espressif ESP32- DevKit C dilengkapi dengan modul ESP32-WROOM-32, ESP32-SOLO-1, atau ESP-WROVER dan ESP-WROVER-KIT-VB. Untuk membeli satu dari mitra kami di katalog Perangkat AWS Mitra, gunakan tautan berikut:

- [ESP32-WROOM-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-KIT v4.1 di situs web Espressif](#).

Note

Saat ini, port FreeRTOS untuk ESP32-WROVER-KIT dan DevKit ESP C tidak mendukung fitur Symmetric multiprocessing (SMP).

Ikhtisar

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 Akun AWS

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

Untuk mendaftar Akun AWS

1. Buka <https://portal.aws.amazon.com/billing/signup>.
2. Ikuti petunjuk secara online.

Anda akan diminta untuk menerima panggilan telepon dan memasukkan kode verifikasi pada keypad telepon sebagai bagian dari prosedur pendaftaran.

Saat Anda mendaftar Akun AWS, Pengguna root akun AWS akan dibuat. Pengguna root memiliki akses ke semua Layanan AWS dan sumber daya dalam akun. Sebagai praktik terbaik keamanan, [tetapkan akses administratif ke pengguna administratif](#), dan hanya gunakan pengguna root untuk melakukan [tugas yang memerlukan akses pengguna root](#).

AWS akan mengirimkan email konfirmasi kepada Anda setelah proses pendaftaran selesai. Anda dapat melihat aktivitas akun saat ini dan mengelola akun dengan mengunjungi <https://aws.amazon.com/> dan memilih Akun Saya.

Membuat pengguna administratif

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

Mengamankan Pengguna root akun AWS Anda

1. Masuk ke [AWS Management Console](#) sebagai pemilik akun dengan memilih Pengguna root dan memasukkan alamat email Akun AWS Anda. Di halaman berikutnya, masukkan kata sandi Anda.

Untuk bantuan masuk menggunakan pengguna root, lihat [Masuk sebagai pengguna root](#) dalam Panduan Pengguna AWS Sign-In.

2. Aktifkan autentikasi multi-faktor (MFA) untuk pengguna root Anda.

Untuk petunjuknya, silakan lihat [Mengaktifkan perangkat MFA virtual untuk pengguna root Akun AWS Anda \(konsol\)](#) dalam Panduan Pengguna IAM.

Membuat pengguna 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 sebuah pengguna administratif.

Untuk mendapatkan tutorial tentang menggunakan Direktori Pusat Identitas IAM sebagai sumber identitas Anda, silakan lihat [Mengonfigurasi akses pengguna dengan Direktori Pusat Identitas IAM default](#) di Panduan Pengguna AWS IAM Identity Center.

Masuk sebagai pengguna administratif

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

Untuk bantuan masuk menggunakan pengguna Pusat Identitas IAM, lihat [Masuk ke portal akses AWS](#) dalam Panduan Pengguna AWS Sign-In.

Untuk memberikan akses, tambahkan izin ke pengguna, grup, atau peran Anda:

- Pengguna dan grup di AWS IAM Identity Center:

Buat rangkaian izin. Ikuti petunjuk dalam [Buat set izin](#) dalam Panduan Pengguna AWS IAM Identity Center.

- Pengguna yang dikelola di IAM melalui penyedia identitas:

Buat peran untuk federasi identitas. Ikuti petunjuk dalam [Membuat peran untuk penyedia identitas pihak ketiga \(federasi\)](#) di Panduan Pengguna IAM.

- Pengguna IAM:

- Buat peran yang dapat diambil pengguna Anda. Ikuti petunjuk dalam [Membuat peran untuk pengguna IAM](#) dalam Panduan Pengguna IAM.

- (Tidak disarankan) Pasang kebijakan langsung ke pengguna atau tambahkan pengguna ke grup pengguna. Ikuti petunjuk di [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 pengembangan ESP-WROVER-KIT, 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 [GitHub](#)FreeRTOS 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, Anda harus menginstal driver CP210x 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 Membuat [Koneksi Serial dengan ESP32](#).
- c. Setelah Anda membuat koneksi serial, catat port serial untuk koneksi papan Anda. Anda membutuhkannya untuk mem-flash demo.

Konfigurasi 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 defaultAWS. 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`(Keamanan WPA2)

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 kredensial 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. Konfigurasi 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
```

- Gunakan `idf.py` skrip untuk mem-flash biner aplikasi ke papan Anda.

```
idf.py flash
```

- 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
```

Bangun 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

- Di jendela baris perintah, arahkan ke root direktori unduhan FreeRTOS Anda.
- 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 konektivitas [Perpustakaan Bluetooth Energi Rendah](#).

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 [SDK Seluler untuk perangkat Bluetooth FreeRTOS](#) untuk mengunduh dan menginstal SDK untuk platform seluler Anda di komputer host Anda.
2. Ikuti petunjuk [Aplikasi demo SDK Seluler FreeRTOS Bluetooth Rendah Energi](#) 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 CMake Anda sendiri untuk ESP32

Jika Anda ingin menggunakan FreeRTOS dalam proyek CMake Anda 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 perintah CMake berikut. Pastikan kompiler ESP32 ada di 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 menjalankan CMake, Anda dapat menemukan semua komponen yang tersedia di 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_
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:
dev_mode_key_provisioning,
                     backoff_algorithm, common, core_http, core_mqtt,
                     core_mqtt_agent, crypto, demo_base,
ota,
                     freertos, http_demo_helpers, kernel, mqtt, mqtt_agent_
                     interface, mqtt_demo_helpers, mqtt_subscription_manager,
                     ota_demo_version, pkcs11_mbedtls, serializer, tls,
                     transport_interface_secure_sockets, utils
3rdparty dependencies:
Available demos:     jsmn, mbedtls, pkcs11, tinycbor
demo_core_mqtt_     demo_cli_uart, demo_core_http, 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 FreeTos. Secara default, CMake akan mencari `freertos/demos/include/` direktori `freertos/vendors/espessif/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 sendiri `sdkconfig.default`, Anda dapat mengatur variabel `CMAKE_IDF_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 pengantiannya](#) di halaman tersebut.

Ringkasan

Jika Anda memiliki proyek dengan komponen yang disebut `example_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)
```

Memecahkan masalah

- Jika Anda menjalankan macOS dan sistem operasi tidak mengenali ESP-WROVER-KIT Anda, 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 untuk mengaktifkan GDbStub untuk komunikasi dengan gdb tanpa memerlukan perangkat keras JTAG khusus. Untuk informasi lebih lanjut, 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 CMake Anda sendiri untuk 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 fungsi `target_link_libraries` CMake standar.
- 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 [Memulai masalah saat 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 ESP32 DevKitc.

C232HM-DDHSL-0 Wire Color	ESP32 GPIO Pin	JTAG Signal Name
Brown (pin 5)	I014	TMS
Yellow (pin 3)	I012	TDI
Black (pin 10)	GND	GND
Orange (pin 2)	I013	TCK
Green (pin 4)	I015	TDO

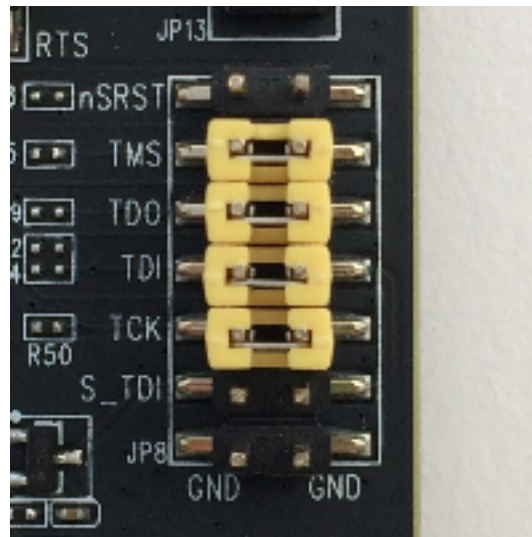
Pengaturan ESP-WROVER-KIT JTAG

Untuk kabel FTDI C232HM-DDHSL-0, ini adalah koneksi ke ESP32-WROVER-KIT.

C232HM-DDHSL-0 Wire Color	ESP32 GPIO Pin	JTAG Signal Name
Brown (pin 5)	I014	TMS
Yellow (pin 3)	I012	TDI
Orange (pin 2)	I013	TCK
Green (pin 4)	I015	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, letakkan 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 ID Perangkat Keras. 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 ID ini cocok dengan ID di `projects/esp8266/esp32/make/aws_demos/esp32_devkitj_v1.cfg`. ID 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/espessif/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 ID agar sesuai dengan ID 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

```

- 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 port serial ESP32 (UART) dan digunakan untuk mengunggah kode ke memori flash ESP32.

- Di jendela terminal, arahkan ke root direktori unduhan FreeRTOS Anda, dan gunakan perintah berikut untuk menjalankan OpenOCD.

```
idf.py openocd
```

- Buka terminal lain, arahkan ke root direktori unduhan FreeRTOS Anda, dan jalankan perintah berikut.

```
idf.py flash monitor
```

- Buka terminal lain, navigasikan root direktori unduhan FreeRTOS Anda, dan jalankan perintah berikut:

```
idf.py gdb
```

Program ini harus berhenti `main()`.

Memulai dengan Espressif ESP32-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 ESP32-C3.](#)
- Saat ini, port FreeRTOS untuk ESP32-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](#).

Ikhtisar

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 Akun AWS

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

Untuk mendaftar Akun AWS

1. Buka <https://portal.aws.amazon.com/billing/signup>.
2. Ikuti petunjuk secara online.

Anda akan diminta untuk menerima panggilan telepon dan memasukkan kode verifikasi pada keypad telepon sebagai bagian dari prosedur pendaftaran.

Saat Anda mendaftar Akun AWS, Pengguna root akun AWS akan dibuat. Pengguna root memiliki akses ke semua Layanan AWS dan sumber daya dalam akun. Sebagai praktik terbaik keamanan, [tetapkan akses administratif ke pengguna administratif](#), dan hanya gunakan pengguna root untuk melakukan [tugas yang memerlukan akses pengguna root](#).

AWS akan mengirimkan email konfirmasi kepada Anda setelah proses pendaftaran selesai. Anda dapat melihat aktivitas akun saat ini dan mengelola akun dengan mengunjungi <https://aws.amazon.com/> dan memilih Akun Saya.

Membuat pengguna administratif

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

Mengamankan Pengguna root akun AWS Anda

1. Masuk ke [AWS Management Console](#) sebagai pemilik akun dengan memilih Pengguna root dan memasukkan alamat email Akun AWS Anda. Di halaman berikutnya, masukkan kata sandi Anda.

Untuk bantuan masuk menggunakan pengguna root, lihat [Masuk sebagai pengguna root](#) dalam Panduan Pengguna AWS Sign-In.

2. Aktifkan autentikasi multi-faktor (MFA) untuk pengguna root Anda.

Untuk petunjuknya, silakan lihat [Mengaktifkan perangkat MFA virtual untuk pengguna root Akun AWS Anda \(konsol\)](#) dalam Panduan Pengguna IAM.

Membuat pengguna 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 sebuah pengguna administratif.

Untuk mendapatkan tutorial tentang menggunakan Direktori Pusat Identitas IAM sebagai sumber identitas Anda, silakan lihat [Mengonfigurasi akses pengguna dengan Direktori Pusat Identitas IAM default](#) di Panduan Pengguna AWS IAM Identity Center.

Masuk sebagai pengguna administratif

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

Untuk bantuan masuk menggunakan pengguna Pusat Identitas IAM, lihat [Masuk ke portal akses AWS](#) dalam Panduan Pengguna AWS Sign-In.

Untuk memberikan akses, tambahkan izin ke pengguna, grup, atau peran Anda:

- Pengguna dan grup di AWS IAM Identity Center:

Buat rangkaian izin. Ikuti petunjuk dalam [Buat set izin](#) dalam Panduan Pengguna AWS IAM Identity Center.

- Pengguna yang dikelola di IAM melalui penyedia identitas:

Buat peran untuk federasi identitas. Ikuti petunjuk dalam [Membuat peran untuk penyedia identitas pihak ketiga \(federasi\)](#) di Panduan Pengguna IAM.

- Pengguna IAM:

- Buat peran yang dapat diambil pengguna Anda. Ikuti petunjuk dalam [Membuat peran untuk pengguna IAM](#) dalam Panduan Pengguna IAM.

- (Tidak disarankan) Pasang kebijakan langsung ke pengguna atau tambahkan pengguna ke grup pengguna. Ikuti petunjuk di [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](#) - C V4. DevKit

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 [GitHub](#)FreeRTOS 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 CP210x 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.

Konfigurasi 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 ATECC608A 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. Konfigurasi Proyek Demo FreeRTOS dengan mengikuti langkah-langkah di [Mengkonfigurasi demo FreeRTOS](#) Saat Anda sampai ke langkah terakhir Untuk memformat AWS IoT kredensial Anda, hentikan, dan lakukan langkah-langkah berikut.
2. Microchip telah menyediakan beberapa alat skrip untuk membantu pengaturan bagian ATECC608A. 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 di ATECC608A 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 untuk [Penyediaan 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. Konfigurasi 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
```

Bangun 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 papan Espressif ESP32, lihat topik berikut:

- [Menggunakan FreeRTOS dalam proyek CMake Anda sendiri untuk ESP32](#)
- [Memecahkan masalah](#)
- [Debugging](#)

Memulai dengan Espressif ESP32-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 ESP32-C3.](#)

[Tutorial ini menunjukkan kepada Anda bagaimana memulai dengan papan pengembangan Espressif ESP32-S2 SoC dan ESP32-S2-Saola-1.](#)

Ikhtisar

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 Akun AWS

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

Untuk mendaftar Akun AWS

1. Buka <https://portal.aws.amazon.com/billing/signup>.
2. Ikuti petunjuk secara online.

Anda akan diminta untuk menerima panggilan telepon dan memasukkan kode verifikasi pada keypad telepon sebagai bagian dari prosedur pendaftaran.

Saat Anda mendaftar Akun AWS, Pengguna root akun AWS akan dibuat. Pengguna root memiliki akses ke semua Layanan AWS dan sumber daya dalam akun. Sebagai praktik terbaik keamanan, [tetapkan akses administratif ke pengguna administratif](#), dan hanya gunakan pengguna root untuk melakukan [tugas yang memerlukan akses pengguna root](#).

AWS akan mengirimkan email konfirmasi kepada Anda setelah proses pendaftaran selesai. Anda dapat melihat aktivitas akun saat ini dan mengelola akun dengan mengunjungi <https://aws.amazon.com/> dan memilih Akun Saya.

Membuat pengguna administratif

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

Mengamankan Pengguna root akun AWS Anda

1. Masuk ke [AWS Management Console](#) sebagai pemilik akun dengan memilih Pengguna root dan memasukkan alamat email Akun AWS Anda. Di halaman berikutnya, masukkan kata sandi Anda.

Untuk bantuan masuk menggunakan pengguna root, lihat [Masuk sebagai pengguna root](#) dalam Panduan Pengguna AWS Sign-In.

2. Aktifkan autentikasi multi-faktor (MFA) untuk pengguna root Anda.

Untuk petunjuknya, silakan lihat [Mengaktifkan perangkat MFA virtual untuk pengguna root Akun AWS Anda \(konsol\)](#) dalam Panduan Pengguna IAM.

Membuat pengguna 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 sebuah pengguna administratif.

Untuk mendapatkan tutorial tentang menggunakan Direktori Pusat Identitas IAM sebagai sumber identitas Anda, silakan lihat [Mengonfigurasi akses pengguna dengan Direktori Pusat Identitas IAM default](#) di Panduan Pengguna AWS IAM Identity Center.

Masuk sebagai pengguna administratif

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

Untuk bantuan masuk menggunakan pengguna Pusat Identitas IAM, lihat [Masuk ke portal akses AWS](#) dalam Panduan Pengguna AWS Sign-In.

Untuk memberikan akses, tambahkan izin ke pengguna, grup, atau peran Anda:

- Pengguna dan grup di AWS IAM Identity Center:

Buat rangkaian izin. Ikuti petunjuk dalam [Buat set izin](#) dalam Panduan Pengguna AWS IAM Identity Center.

- Pengguna yang dikelola di IAM melalui penyedia identitas:

Buat peran untuk federasi identitas. Ikuti petunjuk dalam [Membuat peran untuk penyedia identitas pihak ketiga \(federasi\)](#) di Panduan Pengguna IAM.

- Pengguna IAM:
 - Buat peran yang dapat diambil pengguna Anda. Ikuti petunjuk dalam [Membuat peran untuk pengguna IAM](#) dalam Panduan Pengguna IAM.
 - (Tidak disarankan) Pasang kebijakan langsung ke pengguna atau tambahkan pengguna ke grup pengguna. Ikuti petunjuk di [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 [GitHub](#)FreeRTOS 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 CP210x 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.

Konfigurasi 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 defaultAWS. 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`(Keamanan WPA2)
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:

- Membuat 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 kredensial 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 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. Konfigurasi 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
```

Bangun 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 papan Espressif ESP32, lihat topik berikut:

- [Menggunakan FreeRTOS dalam proyek CMake Anda sendiri untuk ESP32](#)
- [Memecahkan masalah](#)
- [Debugging](#)

Memulai dengan Kit Konektivitas IoT Infineon XMC4800

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 XMC4800 IoT Connectivity Kit. [Jika Anda tidak memiliki Kit Konektivitas IoT Infineon XMC4800, kunjungi Katalog Perangkat Mitra untuk membelinya AWS dari mitra kami.](#)

Jika Anda ingin membuka koneksi serial dengan papan untuk melihat informasi logging dan debugging, Anda memerlukan konverter USB/serial 3.3V, selain Kit Konektivitas IoT XMC4800. [CP2104 adalah konverter USB/serial umum yang tersedia secara luas di papan seperti Teman CP2104 Adafruit.](#)

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*

Ikhtisar

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 XMC4800. 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 [unduh 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 XMC4800 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 XMC4800 menggunakan koneksi serial UART pada pin P0.0 dan P0.1, yang diberi label pada silkscreen papan XMC4800 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 XMC4800 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 XMC4800.

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 IoT XMC4800 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 `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 menggunakan CMake 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 di Windows dengan MinGW sebagai sistem build asli. Untuk informasi selengkapnya tentang penggunaan CMake dengan sistem operasi dan opsi lain, lihat [Menggunakan 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 [unduh 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 petunjuk, lihat [Prasyarat CMake](#).

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
```

- 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.

- Flash dan jalankan gambar dengan [JLINK](#).
 - 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
```

- Flash gambar menggunakan executable JLINK.

```
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.

Memecahkan 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. [Memulai masalah saat memulai](#)

Memulai dengan Infineon OPTIGA Trust X dan XMC4800 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 ada berdasarkan repositori Amazon-freertos yang sekarang tidak digunakan lagi, lihat [Panduan Migrasi Repositori Github Amazon-freertos](#).

Tutorial ini memberikan petunjuk untuk memulai dengan Infineon OPTIGA Trust X Secure Element dan XMC4800 IoT Connectivity Kit. Dibandingkan dengan [Memulai dengan Kit Konektivitas IoT Infineon XMC4800](#) tutorial, panduan ini menunjukkan kepada Anda bagaimana memberikan kredensi aman menggunakan Elemen Aman Infineon OPTIGA Trust X.

Anda memerlukan perangkat keras berikut:

1. Host MCU - Infineon XMC4800 IoT Connectivity Kit, kunjungi Katalog Perangkat AWS Mitra untuk membelinya dari [mitra](#) kami.
2. Paket Ekstensi Keamanan:
 - Elemen Aman - Infineon OPTIGA Trust X.

Kunjungi Katalog Perangkat AWS Mitra untuk membelinya dari [mitra](#) kami.
 - Dewan Personalisasi - Dewan Personalisasi Infineon OPTIGA.
 - Papan Adaptor - Adaptor Infineon Mylo T.

Untuk mengikuti langkah-langkah di sini, Anda harus membuka koneksi serial dengan papan untuk melihat informasi logging dan debugging. (Salah satu langkahnya mengharuskan Anda 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 IoT XMC4800. Konverter USB/Serial [JBTek EL-PN-47310126](#) diketahui berfungsi untuk demo ini. Anda juga memerlukan tiga [kabel male-to-male jumper](#) (untuk menerima (RX), mengirimkan (TX), dan tanah (GND)) untuk menghubungkan kabel serial ke papan Adaptor Infineon Mylo T.

Sebelum memulai, Anda harus mengonfigurasi AWS IoT dan mengunduh FreeRTOS Anda untuk menghubungkan perangkat Anda ke AWS Cloud. Untuk petunjuk, lihat [Opsi #2: pembuatan kunci pribadi onboard](#). Dalam tutorial ini, path ke direktori download 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. Cross-kompilasi aplikasi demo FreeRTOS ke gambar biner.
3. Muat gambar biner aplikasi ke papan Anda, dan kemudian jalankan aplikasi.
4. Untuk tujuan pemantauan dan debugging, berinteraksi dengan aplikasi yang berjalan di papan Anda melalui koneksi serial.

Siapkan lingkungan pengembangan Anda

FreeRTOS menggunakan lingkungan pengembangan DAVE Infineon untuk memprogram XMC4800. Sebelum memulai, unduh dan instal DAVE dan beberapa driver J-Link untuk berkomunikasi dengan debugger on-board.

Instal DAVE

1. Buka halaman [unduh perangkat lunak DAVE](#) Infineon.
2. Pilih paket DAVE untuk sistem operasi Anda dan kirimkan informasi pendaftaran Anda. Setelah mendaftar, Anda akan menerima email konfirmasi dengan tautan untuk mengunduh file.zip.
3. Unduh file paket.zip DAVE (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 unzip file. Kami menyarankan Anda menggunakan program pihak ketiga seperti 7-Zip.

4. Untuk meluncurkan DAVE, jalankan file executable yang ditemukan di DAVE_*version_os_date*.zip folder unzip.

Untuk informasi selengkapnya, lihat [DAVE Panduan Mulai Cepat](#).

Instal driver Segger J-Link

Untuk berkomunikasi dengan probe debugging on-board kit XMC4800 IoT Connectivity, Anda memerlukan driver yang disertakan dalam paket Perangkat Lunak dan Dokumentasi J-Link. Anda dapat men-download software J-Link dan paket Dokumentasi dari halaman [download software J-Link](#) Segger ini.

Buat koneksi serial

Connect kabel konverter USB/serial ke Adaptor Infineon Shield2Go. Hal ini memungkinkan papan Anda untuk mengirim informasi logging dan debugging dalam bentuk yang dapat Anda lihat pada mesin pengembangan Anda. Untuk mengatur koneksi serial:

1. Connect pin RX ke pin TX USB/serial converter Anda.
2. Connect pin TX ke pin RX USB/serial converter Anda.
3. Connect pin ground konverter serial Anda ke salah satu pin GND di papan Anda. Perangkat harus berbagi kesamaan.

Daya dipasok dari port debugging USB, jadi jangan sambungkan pin tegangan positif adaptor serial Anda ke papan.

Note

Beberapa kabel serial menggunakan level pensinyalan 5V. Papan XMC4800 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 [GNU Screen](#). Baud rate diatur ke 115200 secara default dengan 8 bit data, tanpa paritas, dan 1 stop bit.

Memantau pesan MQTT di cloud

Sebelum Anda menjalankan proyek demo FreeRTOS, Anda dapat mengatur klien MQTT diAWS IoT konsol untuk memantau pesan yang dikirim perangkat Anda keAWS Cloud.

Untuk berlangganan topik MQTT dengan klienAWS IoT MQTT

1. Masuk ke [konsol AWS IoT](#) tersebut.

2. Di panel navigasi, pilih Uji, lalu pilih klien uji MQTT untuk membuka klien MQTT.
3. Di Topik berlanggananyour-thing-name/example/topic, masukkan, lalu pilih Berlangganan topik.

Ketika proyek demo berhasil berjalan pada 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 Proyek DAVE, lalu pilih Berikutnya.
3. Di jendela Impor Proyek DAVE, pilih Pilih Direktori Root, pilih Browse, dan kemudian pilih proyek demo XMC4800.

Di direktori tempat Anda membuka ritsleting unduhan FreeRTOS, proyek demo berada di `diprojects/infineon/xmc4800_plus_optiga_trust_x/dave4/aws_demos/dave4`.

Pastikan bahwa Copy Projects Into Workspace dihapus.

4. Pilih Selesai.

aws_demosProyek harus diimpor ke ruang kerja Anda dan diaktifkan.

5. Dari menu Project, pilih Build Active Project.

Pastikan bahwa proyek dibangun tanpa kesalahan.

Jalankan proyek demo FreeRTOS

1. Dari menu Project, pilih Rebuild Active Project untuk membangun kembaliaws_demos dan mengonfirmasi bahwa perubahan konfigurasi Anda diambil.
2. Dari Project Explorer, klik kananaws_demos, pilih Debug As, lalu 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 breakpointmain(), dari menu Run, pilih Lanjutkan.

Pada titik ini, lanjutkan dengan langkah ekstraksi kunci publik [Opsi #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----
```

Bangun demo FreeRTOS dengan CMake

Bagian ini mencakup penggunaan CMake di Windows dengan MingW sebagai sistem build asli. Untuk informasi selengkapnya tentang menggunakan CMake dengan sistem operasi dan opsi lain, lihat [Menggunakan 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 menggunakan CMake 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 versi Windows dari toolchain 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 pemasang toolchain yang diunduh, dan ikuti instruksi di penuntun.
 - 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 petunjuknya, lihat [Prasyarat CMake](#).

3. Buat folder berisi file build yang dihasilkan (*build-folder*).
4. Ubah direktori ke direktori unduhan FreeRTOS Anda (*freertos*), dan gunakan perintah berikut untuk membuat 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 output bineraws_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 JLINK executable.

```
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 publik [Opsis #2: pembuatan kunci pribadi onboard](#). Setelah semua langkah selesai, pergi ke 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 [Memulai masalah saat memulai](#).

Memulai dengan MW32xAWS 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 88MW320/88MW322, Mikrokontroler Cortex M4 terintegrasi terbaru dari NXP, yang mengintegrasikan Wi-Fi 802.11b/g/n pada chip

mikrokontroler tunggal. Kit pengembangan bersertifikat FCC. Untuk informasi selengkapnya, lihat [Katalog Perangkat AWS Mitra](#) untuk membelinya dari mitra kami. Modul 88MW320/88MW322 juga bersertifikat FCC dan tersedia untuk penyesuaian dan penjualan volume.

Panduan memulai ini menunjukkan kepada Anda cara mengompilasi aplikasi bersama SDK di komputer host, lalu memuat file biner yang dihasilkan ke papan menggunakan alat yang disediakan bersama SDK. Saat 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
- ARM toolchain versi 4_9_2015q3
- Eclipse 4.9.0 IDE

Toolchain ARM diperlukan untuk mengompilasi aplikasi Anda dan SDK secara silang. SDK memanfaatkan versi terbaru toolchain untuk mengoptimalkan footprint gambar dan memasukkan lebih banyak fungsionalitas ke dalam ruang yang lebih sedikit. Panduan ini mengasumsikan bahwa Anda menggunakan versi 4_9_2015q3 dari toolchain. Menggunakan versi toolchain yang lebih lama tidak disarankan. Kit pengembangan sudah di-flash dengan firmware proyek Demo Mikrokontroler Nirkabel.

Topik

- [Mengatur perangkat keras Anda](#)
- [Menyiapkan lingkungan pengembangan](#)
- [Bangun dan jalankan proyek demo FreeRTOS](#)
- [Debugging](#)
- [Pemecahan Masalah](#)

Mengatur perangkat keras Anda

Connect papan MW32x ke laptop Anda dengan menggunakan kabel mini-USB ke USB. Connect kabel mini-USB ke satu-satunya konektor mini-USB yang ada di papan tulis. Anda tidak perlu perubahan jumper.

Jika papan terhubung ke laptop atau komputer desktop, Anda tidak memerlukan catu daya eksternal.

Koneksi USB ini menyediakan yang berikut:

- Akses konsol ke papan. Port tty/com virtual terdaftar dengan host pengembangan yang dapat digunakan untuk mengakses konsol.
- Akses JTAG ke papan tulis. Ini dapat digunakan untuk memuat atau membongkar gambar firmware ke RAM atau flash papan, atau untuk tujuan debugging.

Menyiapkan lingkungan pengembangan

Untuk tujuan pengembangan, persyaratan minimum adalah toolchain ARM dan alat yang dibundel dengan SDK. Bagian berikut memberikan perincian 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 menganggap bahwa GNU compiler toolchain binari tersedia di PATH pengguna dan dapat dipanggil dari Makefiles. Makefiles juga berasumsi bahwa nama file biner 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 `gcc-arm-embedded` toolchain.

Prosedur Pengaturan Toolchain Linux

Ikuti langkah berikut untuk menyiapkan toolchain GCC di Linux.

1. Unduh tarball toolchain yang tersedia di [GNU Arm Embedded Toolchain 4.9-2015-q3-update](#). Berkasnya 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 membuka file.

```
tar -vxf filename
```

4. Tambahkan jalur toolchain 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-eabi-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 rekomendasikan Anda memperbarui ke rilis terbaru. Langkah-langkah berikut telah diverifikasi untuk bekerja di Ubuntu 16.04 dan berasumsi bahwa Anda menggunakan versi itu.

Menginstal Paket

SDK menyertakan skrip untuk mengaktifkan penyiapan cepat lingkungan pengembangan Anda pada mesin Linux yang baru disiapkan. Skrip mencoba mendeteksi jenis mesin secara auto dan menginstal perangkat lunak yang sesuai, termasuk perpustakaan C, perpustakaan USB, perpustakaan FTDI, ncurses, python, dan latexs. Pada 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 melakukannya, masukkan 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 mulus.

Menyiapkan Konsol Serial

Masukkan kabel USB ke dalam 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. Dalam contoh di atas, ini bernama "TTYusb1".

Dalam panduan ini, kita menggunakan minicom untuk melihat output konsol serial. Anda mungkin juga menggunakan program serial lainnya seperti `ty`. Jalankan perintah berikut untuk mengeksekusi minicom dalam mode pengaturan.

```
minicom -s
```

Di minicom, arahkan ke Serial Port Setup dan menangkap 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 digunakan di future. 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 mengekspos dua antarmuka USB untuk host. Antarmuka pertama dikaitkan dengan fungsionalitas JTAG MCU dan antarmuka kedua dikaitkan dengan port UARTX 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 fungsi Eclipse. Jika versi sebelumnya, seperti versi 0.7, diinstal pada host Linux Anda, hapus repositori itu dengan perintah yang sesuai untuk distribusi Linux yang sedang 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 memasang 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. Mengatur openocd dengan perintah berikut.

```
./configure --enable-ftdi --enable-jlink
```

5. Jalankan utilitas make untuk mengkompilasi opencd.

```
make install
```

Mengatur Eclipse

Note

Bagian ini mengasumsikan bahwa Anda telah menyelesaikan langkah-langkah di [Menghindari sudo](#)

Eclipse adalah IDE yang disukai untuk pengembangan aplikasi dan debugging. Ini menyediakan IDE yang kaya dan ramah pengguna dengan dukungan debugging terintegrasi, termasuk debugging sadar benang. Bagian ini menjelaskan penyiapan Eclipse umum untuk semua host pengembangan yang didukung.

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 Unduhan](#) di situs web Oracle.

2. Unduh dan instal “Eclipse IDE untuk 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 Eclipse khusus platform yang dapat dieksekusi untuk memulai aplikasi.

Bangun dan jalankan proyek demo FreeRTOS

Ada dua cara untuk menjalankan proyek demo FreeRTOS:

- Gunakan baris perintah.
- Menggunakan 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`

Misalnya:

```
#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 tertutup 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.

```
marvell@pe-1t586:amzsdk$
marvell@pe-1t586:amzsdk$ cmake -DVENDOR=marvell -DBOARD=mw300_rd -DCOMPILER=arm-gcc -S . -
Bbuild -DAFR_ENABLE_TESTS=0

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

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

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

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

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

-----
-- Configuring done
-- Generating done
-- Build files have been written to: /home/marvell/gerrit/amzsdk/build
marvell@pe-1t586:amzsdk$
```

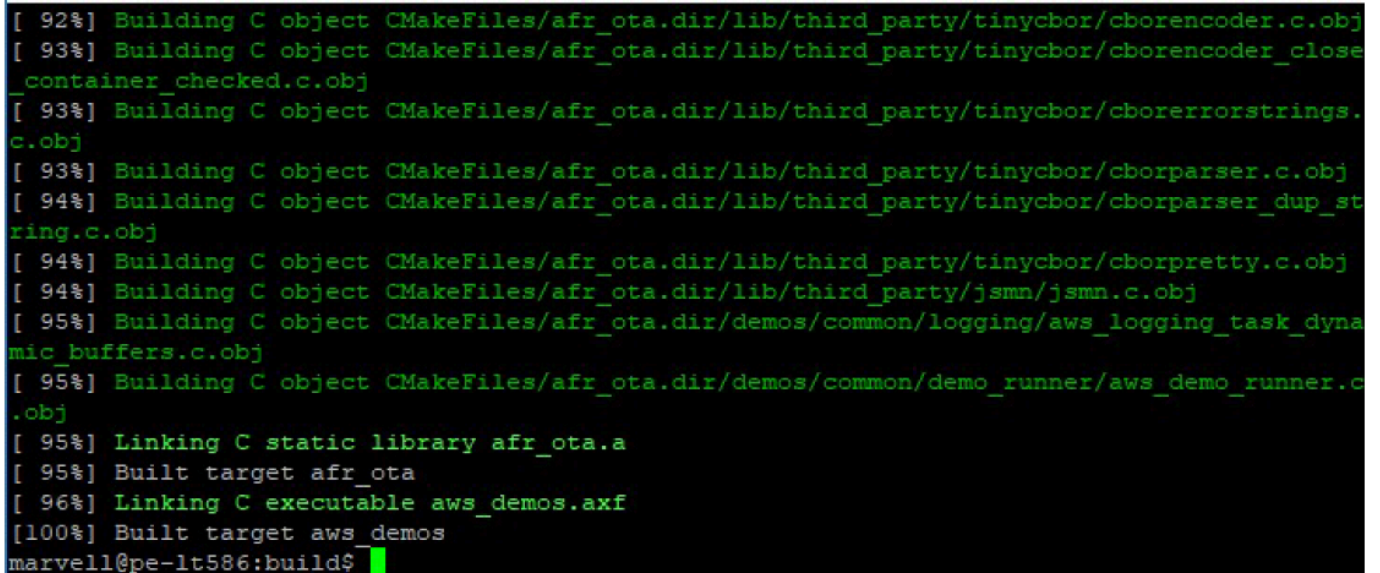
2. Buka direktori build.

```
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_string.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_dynamic_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 membuat SDK sebelum 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
```

flashprogPerintah memulai hal berikut:

- Tata Letak - 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. flashprogPerintah ini juga menulis bootloader ke flash. Ini adalah tugas bootloader untuk memuat gambar firmware mikrokontroler setelah itu 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
```

- Firmware menggunakan chipset Wi-Fi untuk fungsinya, dan chipset Wi-Fi memiliki firmware sendiri yang juga harus ada di 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 perintah serupa dengan gambar di bawah.

```
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. Atur ulang papan. Anda akan melihat log untuk aplikasi demo.

e. Untuk menjalankan aplikasi pengujian, flashaws_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 serupa dengan yang ditunjukkan pada gambar di bawah.


```
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.

```
Network connection successful.
Wi-Fi Connected to AP. Creating tasks which use network...
2 6293 [Startup Hook] Write certificate...
3 6296 [Startup Hook] Write device private key...
4 6362 [Startup Hook] Creating MQTT Echo Task...
6 11668 [MQTTEcho] MQTT echo connected to connect to a2wtm15blvjjs8-ats.iot.us-east-2.amazonaws.com
7 11668 [MQTTEcho] MQTT echo test echoing task created.
8 11961 [MQTTEcho] MQTT Echo demo subscribed to freertos/demos/echo
9 12248 [MQTTEcho] Echo successfully published 'Hello World 0'
10 12591 [Echoing] Message returned with ACK: 'Hello World 0 ACK'
11 17633 [MQTTEcho] Echo successfully published 'Hello World 1'
12 17927 [Echoing] Message returned with ACK: 'Hello World 1 ACK'
13 22953 [MQTTEcho] Echo successfully published 'Hello World 2'
14 23276 [Echoing] Message returned with ACK: 'Hello World 2 ACK'
15 28245 [MQTTEcho] Echo successfully published 'Hello World 3'
16 28575 [Echoing] Message returned with ACK: 'Hello World 3 ACK'
17 33542 [MQTTEcho] Echo successfully published 'Hello World 4'
18 33980 [Echoing] Message returned with ACK: 'Hello World 4 ACK'
19 38823 [MQTTEcho] Echo successfully published 'Hello World 5'
20 39279 [Echoing] Message returned with ACK: 'Hello World 5 ACK'
21 44139 [MQTTEcho] Echo successfully published 'Hello World 6'
22 44501 [Echoing] Message returned with ACK: 'Hello World 6 ACK'
23 49516 [MQTTEcho] Echo successfully published 'Hello World 7'
24 50270 [Echoing] Message returned with ACK: 'Hello World 7 ACK'
25 54796 [MQTTEcho] Echo successfully published 'Hello World 8'
26 55129 [Echoing] Message returned with ACK: 'Hello World 8 ACK'
27 60080 [MQTTEcho] Echo successfully published 'Hello World 9'
28 60389 [Echoing] Message returned with ACK: 'Hello World 9 ACK'
29 65378 [MQTTEcho] Echo successfully published 'Hello World 10'
30 65998 [Echoing] Message returned with ACK: 'Hello World 10 ACK'
31 70658 [MQTTEcho] Echo successfully published 'Hello World 11'
32 70964 [Echoing] Message returned with ACK: 'Hello World 11 ACK'
33 75958 [MQTTEcho] MQTT echo demo finished.
34 75958 [MQTTEcho] ---Demo finished---
```

7. (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 meluncurkan file eksekusi segera. 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.

```
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)
target state: halted
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x00007f14 msp: 0x20001000
75072 bytes written at address 0x00100000
8 bytes written at address 0x00112540
468 bytes written at address 0x20000040
downloaded 75548 bytes in 0.636127s (115.979 KiB/s)
verified 75548 bytes in 0.959023s (76.930 KiB/s)
shutdown command invoked
```

Ketika eksekusi perintah selesai, Anda akan melihat log aplikasi demo.

Bangun dan jalankan demo FreeRTOS menggunakan Eclipse IDE

1. Sebelum Anda mengatur ruang kerja Eclipse, Anda harus menjalankan `cmake` perintah.

Jalankan perintah berikut untuk bekerja dengan `projekaws_demos` Eclipse.

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

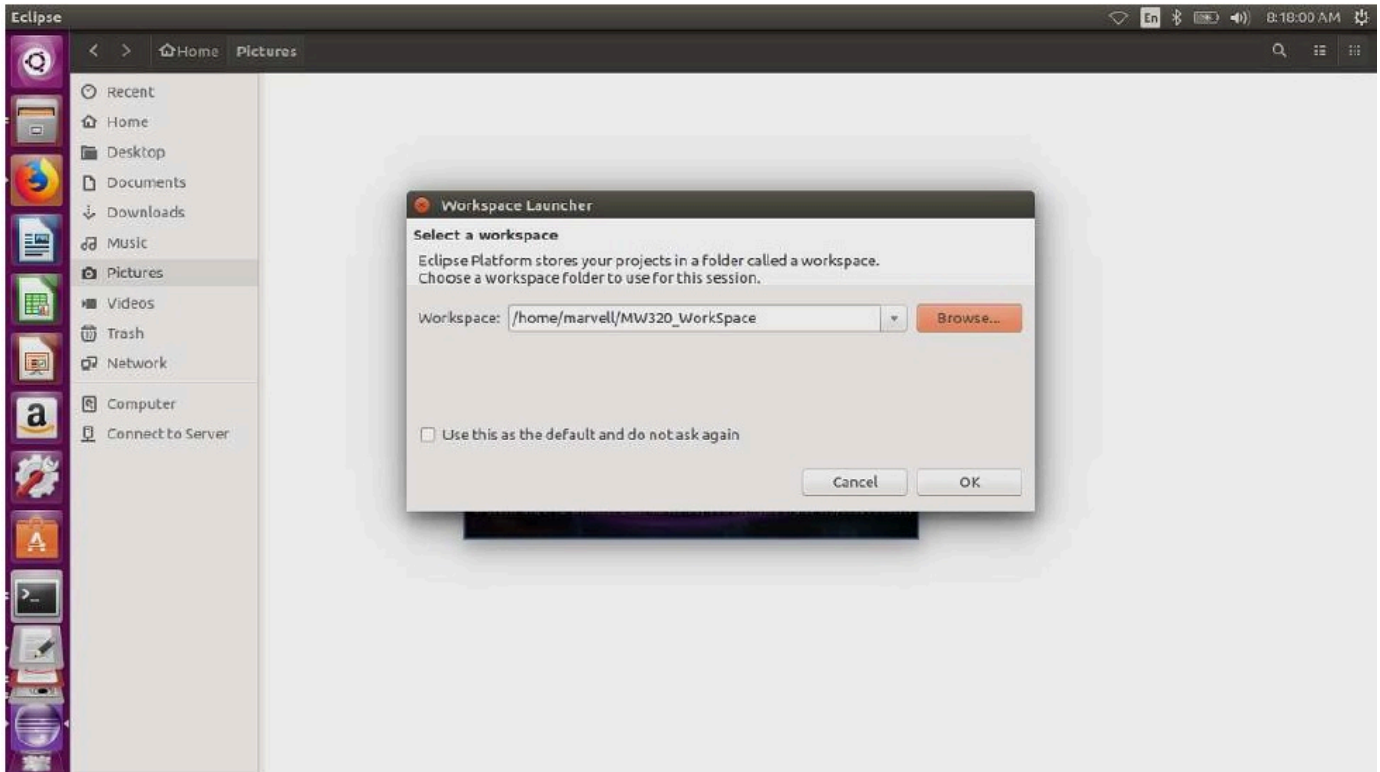
Jalankan perintah berikut untuk bekerja dengan `projekaws_tests` Eclipse.

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

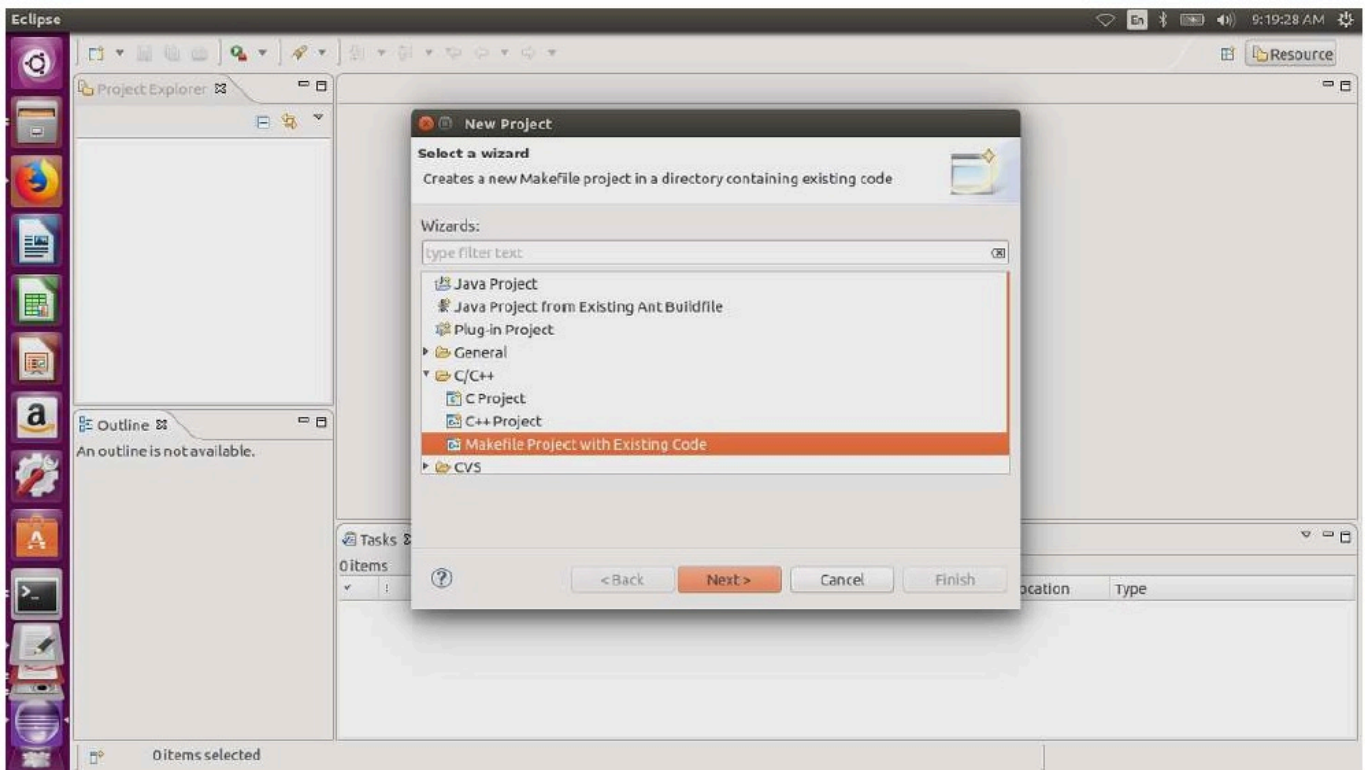
Tip

Jalankan `make` perintah setiap kali Anda beralih antara `aws_demos` proyek dan `aws_tests` proyek.

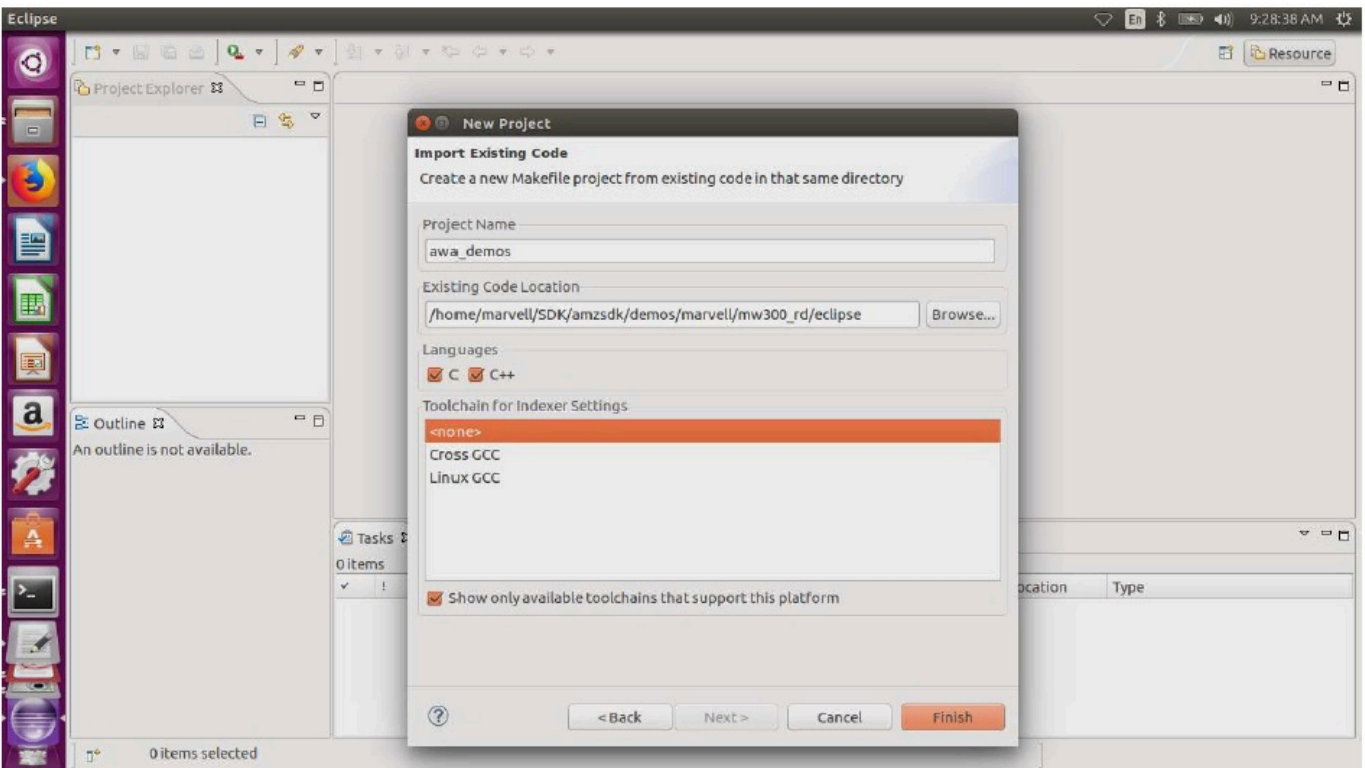
2. Buka Eclipse dan, ketika diminta, pilih ruang kerja Eclipse Anda seperti yang ditunjukkan pada gambar di bawah ini.



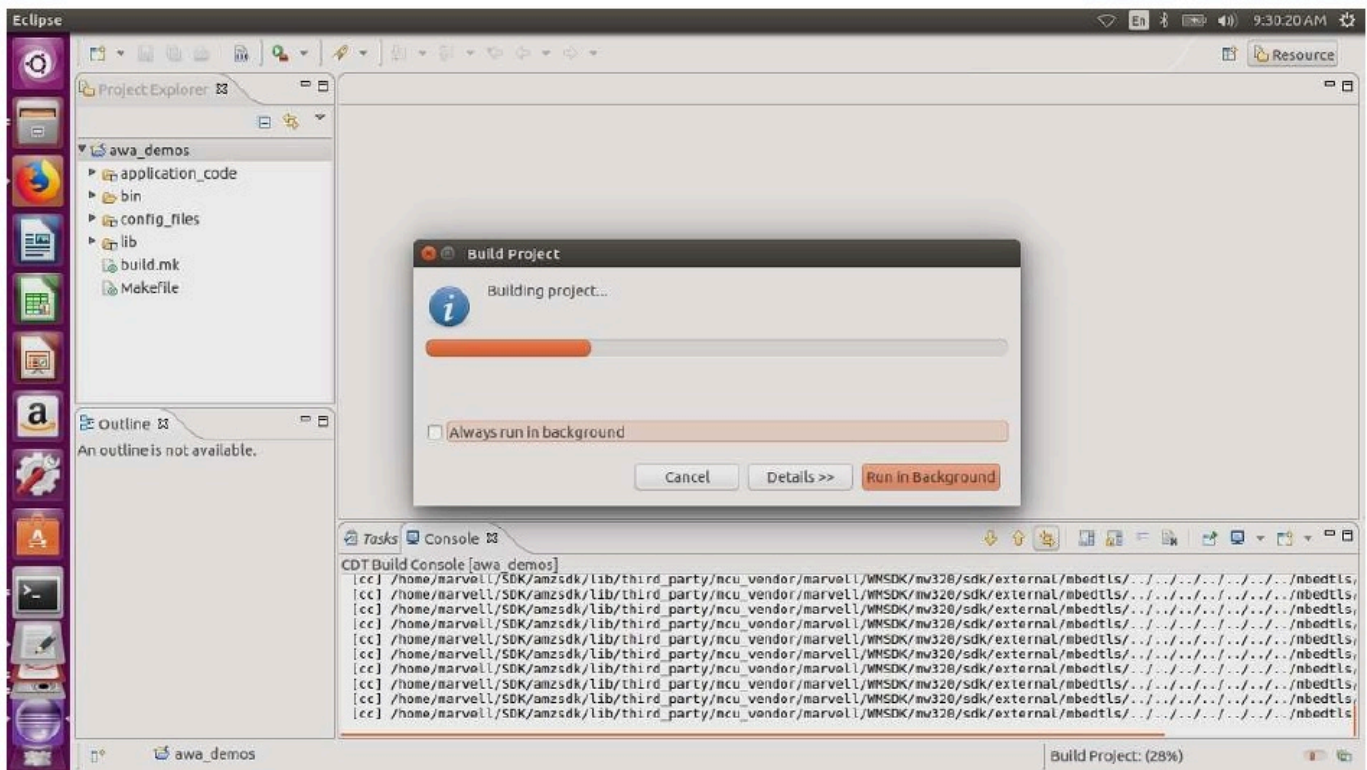
3. Pilih opsi untuk membuat Proyek Makefile: dengan Kode yang Ada seperti yang ditunjukkan pada gambar di bawah ini.



4. Pilih Jelajahi, tentukan direktori kode yang ada, lalu pilih Selesai.



5. Di panel navigasi, pilih aws_demos di explorer proyek. Klik kanan aws_demos 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
```

flashprogPerintah memulai hal berikut:

- Tata Letak - 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 di flash. Anda menggunakan `flashprog.py` utilitas untuk mem-flash firmware Wi-Fi dengan cara yang sama seperti yang Anda lakukan untuk mem-flash boot2 boot-loader 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 perintah serupa dengan gambar di bawah.

```
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. Atur ulang papan. Anda akan melihat log untuk aplikasi demo.
- e. Untuk menjalankan aplikasi pengujian, flashaws_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 serupa dengan yang ditunjukkan pada gambar di bawah.

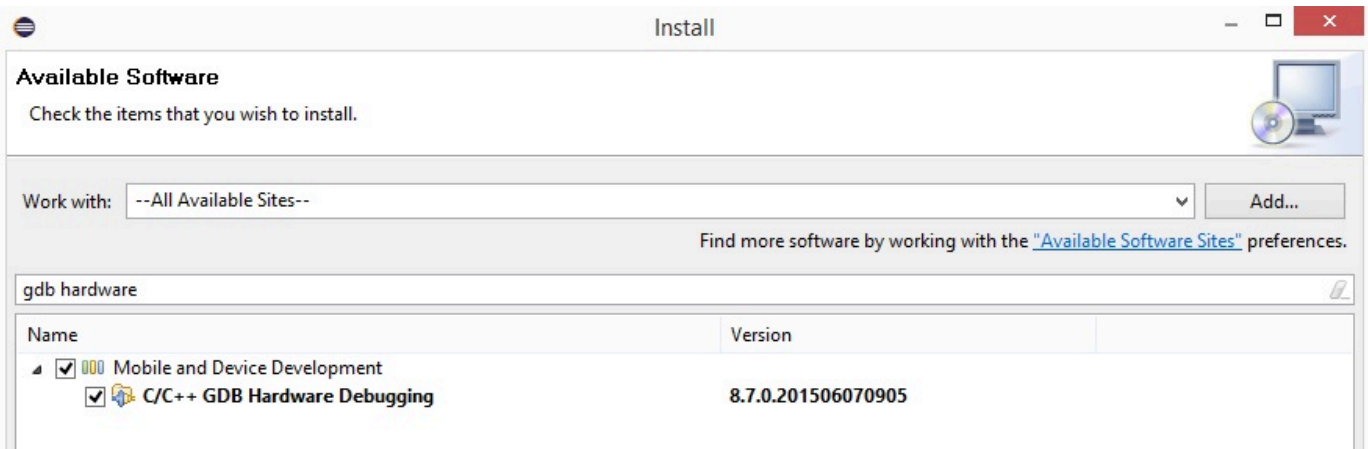

```
target state: halted
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x00007f14 msp: 0x20001000
29088 bytes written at address 0x00100000
downloaded 29088 bytes in 0.245499s (115.708 KiB/s)
verified 29088 bytes in 0.350231s (81.107 KiB/s)
semihosting is enabled

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_nrst_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...
```

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 Debugging Perangkat Keras C/C ++ GDB dan instal plugin.



Pemecahan Masalah

Masalah jaringan

Periksa kredensi jaringan Anda. Lihat “Provisioning” di [Bangun dan jalankan proyek demo FreeRTOS](#).

Mengaktifkan log tambahan

- Aktifkan log khusus papan.

Aktifkan panggilan `kewmstdio_init(UART0_ID, 0)prvMiscInitialization` dalam fungsi 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 MT7697hx

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 Kit Pengembangan MediaTek MT7697hx. [Jika Anda tidak memiliki Kit Pengembangan MediaTek MT7697hx, 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*

Ikhtisar

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 Kit Pengembangan MediaTek MT7697hx.

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 μ Vision IDE dan μ Vision 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 (Seri MT76x7).

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 μ Vision

1. Dari menu Start, buka Keil μ Vision 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 Kit Pengembangan MediaTek MT7697hx 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.

μ Vision 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 Kit Pengembangan MediaTek MT7697hx Anda.

Memecahkan masalah

Mendebug proyek FreeRTOS di Keil μ Vision

Saat ini, Anda harus mengedit MediaTek paket yang disertakan dengan Keil μ Vision sebelum Anda dapat men-debug proyek demo FreeRTOS dengan Keil μ Vision. 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 bahwa IRAM1 dan IRAM2 keduanya dipilih.
3. Pilih tab Debug, lalu pilih CMSIS-DAP Debugger.
4. Bukavendors/mediatek/boards/mt7697hx-dev-kit/aws_demos/application_code/main.c, dan atur makro MTK_DEBUGGER ke 1.
5. Membangun kembali proyek demo di μ Vision.
6. Atur Kit Pengembangan MediaTek MT7697hx 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.

μ Vision memberi tahu Anda saat unduhan selesai.

8. Tekan tombol RESET pada Kit Pengembangan MediaTek MT7697hx Anda.
9. Dari menu μ Vision, 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 ) ; }
```

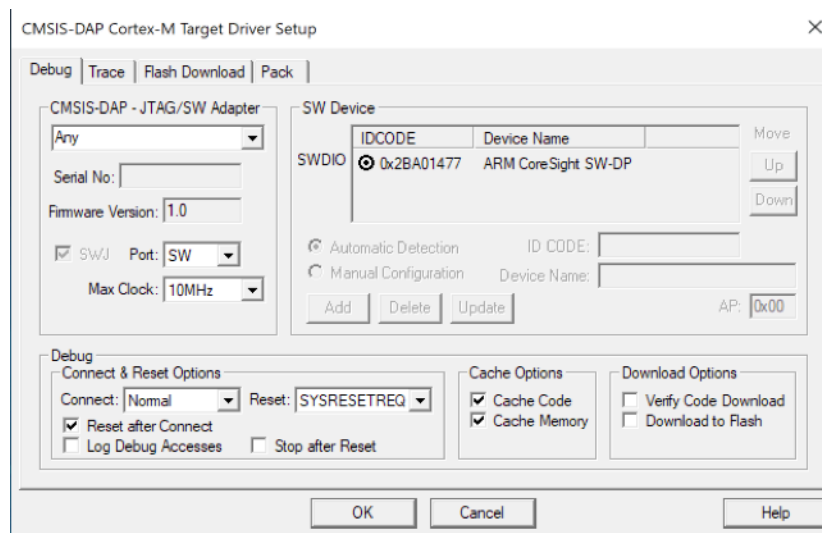
11. Di jendela Call Stack + Locals, ubah nilainya `wait_ice` menjadi 0.
12. Tetapkan breakpoint dalam kode sumber proyek Anda, dan jalankan kode.

Memecahkan masalah pengaturan debugger IDE

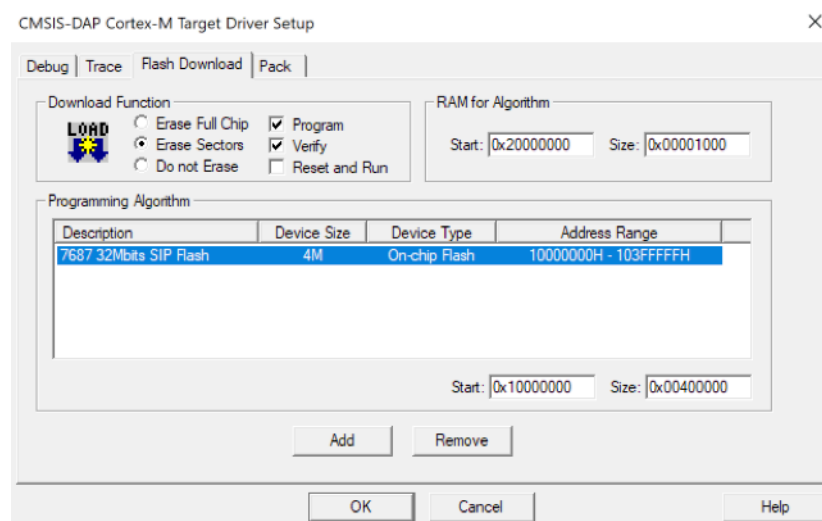
Jika Anda mengalami kesulitan men-debug aplikasi, pengaturan debugger Anda mungkin salah.

Untuk memverifikasi bahwa pengaturan debugger Anda sudah benar

1. Buka Keil μ Vision.
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. [Memulai masalah saat memulai](#)

Memulai dengan Microchip Curiosity PIC32MZ 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 PIC32MZEF (DM320104) dari cabang utama repositori FreeRTOS Reference Integration dan tidak akan lagi membawanya dalam rilis baru. Microchip telah mengeluarkan [pemberitahuan resmi](#) bahwa PIC32MZEF (DM320104) tidak lagi direkomendasikan untuk desain baru. Proyek PIC32MZEF dan kode sumber masih dapat diakses melalui tag rilis sebelumnya. Microchip merekomendasikan agar pelanggan menggunakan Curiosity [PIC32MZ-EF-2.0 Development board \(DM320209\)](#) untuk desain baru. Platform Pic32Mzv1 masih dapat ditemukan di [v202012.00](#) dari repositori Integrasi Referensi FreeRTOS. Namun, platform ini tidak lagi didukung oleh [v202107.00](#) dari Referensi FreeRTOS.

Tutorial ini memberikan instruksi untuk memulai dengan Microchip Curiosity PIC32MZ EF. Jika Anda tidak memiliki bundel Microchip Curiosity PIC32MZ EF, kunjungi Katalog Perangkat AWS Mitra untuk membelinya dari [mitra](#) kami.

Bundel tersebut mencakup item berikut:

- [Curiosity PIC32MZ EF Development Board](#)
- [MikroElektronika Papan Klik UART USB](#)
- [MikroElektronika WiFi 7 klik Papan](#)
- [PIC32 LAN8720 PHY papan putri](#)

Anda juga memerlukan item berikut untuk debugging:

- [Debugger Sirkuit Dalam Snap MPLAB](#)

- (Opsional) [Kit Kabel Pemrograman PicKit 3](#)

Sebelum memulai, Anda harus mengonfigurasi AWS IoT dan mengunduh FreeRTOS Anda untuk menghubungkan perangkat Anda ke AWS Cloud. Lihat [Langkah pertama](#) untuk instruksi.

Important

- Dalam topik ini, jalur ke direktori unduhan FreeRTOS disebut sebagai *freertos*.
- Karakter spasi di *freertos* jalur dapat menyebabkan kegagalan build. Saat Anda mengkloning atau menyalin repositori, pastikan jalur yang Anda buat tidak mengandung karakter spasi.
- Panjang maksimal jalur file pada Microsoft Windows adalah 260 karakter. Jalur direktori unduhan FreeRTOS yang panjang dapat menyebabkan kegagalan build.
- Karena kode sumber mungkin berisi tautan simbolik, 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 selengkapnya, lihat entri blog [Symlink di Windows 10!](#).

Jika Anda menggunakan Git di bawah Windows, Anda harus mengaktifkan Mode Pengembang atau Anda harus:

- Atur `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`, dan `git 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. Cross kompilasi 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 melalui koneksi serial, untuk tujuan pemantauan dan debugging.

Mengatur perangkat keras Microchip Curiosity PIC32MZ EF

1. Connect MikroElektronika USB UART klik Board ke konektor MicroBus 1 pada Microchip Curiosity PIC32MZ EF.
2. Connect papan putri PIC32 LAN8720 PHY ke header J18 pada Microchip Curiosity PIC32MZ EF.
3. Connect Papan klik UART MikroElektronika USB ke komputer Anda menggunakan kabel USB A ke USB mini-B.
4. Untuk terhubung ke internet, gunakan salah satu opsi berikut:
 - Untuk menggunakan Wi-Fi, sambungkan Papan klik MikroElektronika Wi-Fi 7 ke konektor MicroBus 2 pada Microchip Curiosity PIC32MZ EF. Lihat [Mengkonfigurasi demo FreeRTOS](#).
 - Untuk menggunakan Ethernet untuk menghubungkan Microchip Curiosity PIC32MZ EF Board ke internet, hubungkan papan putri PIC32 LAN8720 PHY ke header J18 pada Microchip Curiosity PIC32MZ EF. Connect salah satu ujung kabel Ethernet ke papan putri LAN8720 PHY. Connect ujung lainnya ke router atau port internet lainnya. Anda juga harus menentukan makro `preprocessorPIC32_USE_ETHERNET`.
5. Jika belum selesai, solder konektor sudut ke header ICSP pada Microchip Curiosity PIC32MZ EF.
6. Connect salah satu ujung kabel ICSP dari Kit Kabel Pemrograman PicKit 3 ke Microchip Curiosity PIC32MZ EF.

Jika Anda tidak memiliki Kit Kabel Pemrograman PicKit 3, Anda dapat menggunakan jumper kawat MF Dupont untuk menghubungkan koneksi sebagai gantinya. Perhatikan bahwa lingkaran putih menandakan posisi Pin 1.

7. Connect ujung lain dari kabel ICSP (atau jumper) ke MPLAB Snap Debugger. Pin 1 dari 8-pin SIL Programming Connector ditandai dengan segitiga hitam di kanan bawah papan.

Pastikan kabel apa pun ke Pin 1 pada Microchip Curiosity PIC32MZ EF, ditandai dengan lingkaran putih, sejajar dengan Pin 1 pada MPLAB Snap Debugger.

Untuk informasi selengkapnya tentang MPLAB Snap Debugger, lihat [Lembar Informasi Debugger Snap In-Circuit MPLAB](#).

Mengatur perangkat keras Microchip Curiosity PIC32MZ EF menggunakan PicKit On Board (PKOB)

Kami menyarankan agar Anda mengikuti prosedur pengaturan di bagian sebelumnya. Namun, Anda dapat mengevaluasi dan menjalankan demo FreeRTOS dengan debugging dasar menggunakan programmer/debugger PicKit On Board (PKOB) terintegrasi dengan mengikuti langkah-langkah ini.

1. Connect MikroElektronika USB UART klik Board ke konektor MicroBus 1 pada Microchip Curiosity PIC32MZ EF.
2. Untuk terhubung ke internet, lakukan salah satu hal berikut:
 - Untuk menggunakan Wi-Fi, sambungkan Papan klik MikroElektronika Wi-Fi 7 ke konektor MicroBus 2 pada Microchip Curiosity PIC32MZ EF. (Ikuti langkah-langkah “Untuk mengkonfigurasi Wi-Fi Anda” di [Mengkonfigurasi demo FreeRTOS](#)).
 - Untuk menggunakan Ethernet untuk menghubungkan Microchip Curiosity PIC32MZ EF Board ke internet, hubungkan papan putri PIC32 LAN8720 PHY ke header J18 pada Microchip Curiosity PIC32MZ EF. Connect salah satu ujung kabel Ethernet ke papan putri LAN8720 PHY. Connect ujung lainnya ke router atau port internet lainnya. Anda juga harus menentukan makro preprocessor `PIC32_USE_ETHERNET`.
3. Connect port USB Micro-B bernama “USB DEBUG” pada Microchip Curiosity PIC32MZ EF Board ke komputer Anda menggunakan kabel USB tipe A ke USB Micro-B.
4. Connect Papan klik UART MikroElektronika 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 dari Kompiler MPLAB XC32 dan versi 2.X.X dari MPLAB Harmony Configurator (MHC).

1. Instal [Python versi 3.x](#) atau yang lebih baru.
2. Instal IDE MPLAB X:

 Note

IntegrasiAWS Referensi FreeRTOS v202007.00 saat ini didukung hanya di MPLabv5.35. Versi IntegrasiAWS Referensi FreeRTOS sebelumnya didukung di MPLaBv5.40.

MPLaBv5.35 unduhan

- [Lingkungan Pengembangan Terpadu MPLAB X untuk Windows](#)
- [MPLAB X Integrated Development Environment untuk macOS](#)
- [Lingkungan Pengembangan Terpadu MPLAB X untuk Linux](#)

Unduhan MPLab terbaru (MPLaBv5.40)

- [Lingkungan Pengembangan Terpadu MPLAB X untuk Windows](#)
- [MPLAB X Integrated Development Environment untuk macOS](#)
- [Lingkungan Pengembangan Terpadu MPLAB X untuk Linux](#)

3. Instal Kompiler MPLAB XC32:
 - [MPLAB XC32/32++ Compiler untuk Windows](#)
 - [MPLAB XC32/32++ Compiler untuk macOS](#)
 - [MPLAB XC32/32++ Compiler 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 Anda menjalankan proyek demo FreeRTOS, Anda dapat mengatur klien MQTT di AWS IoT konsol untuk memantau pesan yang dikirim perangkat Anda ke AWS Cloud.

Untuk berlangganan topik MQTT dengan klien AWS IoT MQTT

1. Masuk ke [konsol AWS IoT](#) tersebut.
2. Di panel navigasi, pilih Uji, lalu pilih klien uji MQTT untuk membuka klien MQTT.
3. Di Topik berlangganan ***your-thing-name/example/topic***, masukkan, lalu pilih Berlangganan topik.

Ketika proyek demo berhasil berjalan pada perangkat Anda, Anda melihat “Hello World!” dikirim beberapa kali ke topik yang Anda berlangganan.

Bangun dan jalankan proyek demo FreeRTOS

Buka demo FreeRTOS di IDE MPLAB

1. Buka MPLAB IDE. Jika Anda memiliki lebih dari satu versi compiler diinstal, Anda perlu memilih compiler yang ingin Anda gunakan dari dalam IDE.
2. Dari menu File, pilih Open Project.
3. Jelajahi dan buka `projects/microchip/curiosity_pic32mzef/mplab/aws_demos`.
4. Pilih Buka proyek.

Note

Ketika Anda membuka proyek untuk pertama kalinya, Anda mungkin mendapatkan pesan kesalahan tentang compiler. Di IDE, navigasikan ke Tools, Options, Embedded, dan kemudian pilih compiler yang Anda gunakan untuk proyek Anda.

Untuk menggunakan Ethernet untuk terhubung, Anda harus menentukan makro preprocessor `PIC32_USE_ETHERNET`.

Menggunakan Ethernet untuk menyambung menggunakan IDE MPLAB

1. Di IDE MPLAB, klik kanan proyek dan pilih Properties.
2. Di kotak dialog Project Properties, pilih **compiler-name** (Global Options) untuk mengembangkannya, dan kemudian 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. Bangun kembali proyek Anda.
2. Pada tab Proyek, klik kanan folder `aws_demos` tingkat atas, dan kemudian pilih Debug.
3. Ketika debugger berhenti di `breakpointmain()`, dari menu Run, pilih Lanjutkan.

Bangun demo FreeRTOS dengan CMake

Jika Anda memilih untuk tidak menggunakan IDE untuk pengembangan FreeRTOS, Anda dapat menggunakan CMake 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 biner Hexmate dan toolchain, seperti `C:\Program Files\Microchip\xc32\v2.40\bin` jalur `C:\Program Files (x86)\Microchip\MPLABX\v5.35\mplab_platform\bin` dan.

3. Ubah direktori ke direktori build (**build-directory**), lalu jalankan `make` dari direktori tersebut.

Untuk informasi selengkapnya, lihat [Menggunakan CMake dengan FreeRTOS](#).

Untuk menggunakan Ethernet untuk terhubung, Anda harus menentukan makro `preprocessorPIC32_USE_ETHERNET`.

Pemecahan Masalah

Untuk informasi pemecahan masalah, lihat [Memulai masalah saat memulai](#).

Memulai dengan 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 tidak digunakan lagi, lihat [Panduan Migrasi Repositori Github Amazon-freertos](#).

Tutorial ini memberikan petunjuk untuk memulai dengan Nordic NRF52840-dk. Jika Anda tidak memiliki Nordic NRF52840-dk, kunjungi Katalog Perangkat AWS Mitra untuk membelinya dari [mitra](#) kami.

Sebelum memulai, Anda harus melakukannya [Mengatur 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. Cross kompilasi 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 melalui koneksi serial, untuk tujuan pemantauan dan debugging.

Siapkan perangkat keras Nordik

Connect komputer host Anda ke port USB berlabel J2, terletak tepat di atas dudukan baterai sel koin di papan Nordic NRF52840 Anda.

Untuk informasi selengkapnya tentang pengaturan Nordic NRF52840-dk, lihat [Panduan Pengguna Kit Pengembangan NRF52840](#).

Siapkan lingkungan pengembangan Anda

Unduh dan instal Segger Embedded Studio

FreeRTOS mendukung Segger Embedded Studio sebagai lingkungan pengembangan untuk Nordic NRF52840-DK.

Untuk pengaturan lingkungan Anda, Anda perlu mengunduh dan menginstal Segger Embedded Studio di komputer host Anda.

Untuk mengunduh dan menginstal Segger Embedded Studio

1. Buka halaman [Segger Embedded Studio Downloads](#), dan pilih opsi Embedded Studio for ARM untuk sistem operasi Anda.
2. Jalankan penginstal dan ikuti petunjuk penyelesaian.

Mengatur 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.

Mengatur aplikasi Demo SDK Mobile FreeRTOS Bluetooth Low Energy

1. Ikuti petunjuk [SDK Seluler untuk perangkat Bluetooth FreeRTOS](#) untuk mengunduh dan menginstal SDK untuk platform seluler Anda di komputer host Anda.

- Ikuti petunjuk [Aplikasi demo SDK Seluler FreeRTOS Bluetooth Rendah Energi](#) untuk mengatur aplikasi mobile demo pada perangkat mobile Anda.

Buat koneksi serial

Segger Embedded Studio menyertakan emulator terminal yang dapat Anda gunakan untuk menerima pesan log melalui koneksi serial ke papan Anda.

Untuk membuat koneksi serial dengan Segger Embedded Studio

- Buka Segger Embedded Studio.
- Dari menu atas, pilih Target, Connect J-Link.
- Dari menu atas, pilih Tools, Terminal Emulator, Properties, dan atur properti seperti yang diinstruksikan [Memasang emulator terminal](#).
- 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 GNU Screen. Konfigurasi terminal untuk terhubung ke papan Anda dengan koneksi serial seperti yang diinstruksikan [Memasang emulator terminal](#).

Unduh dan mengonfigurasi FreeRTOS

Setelah menyiapkan perangkat keras dan lingkungan, Anda dapat mengunduh FreeRTOS.

Unduh FreeRTOS

Untuk mengunduh FreeRTOS untuk Nordic NRF52840-DK, buka [GitHub halaman FreeRTOS](#) dan kloning repositori. Lihat file [README.md](#) untuk instruksi.

Important

- Dalam topik ini, jalur ke direktori unduhan FreeRTOS disebut sebagai *freertos*.
- Karakter spasi di *freertos* jalur dapat menyebabkan kegagalan build. Saat mengkloning atau menyalin repositori, pastikan jalur yang Anda buat tidak mengandung karakter spasi.

- Panjang maksimal jalur file pada Microsoft Windows adalah 260 karakter. Jalur direktori unduhan FreeRTOS yang panjang dapat menyebabkan kegagalan build.
- Karena kode sumber mungkin berisi tautan simbolik, 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 selengkapnya, lihat entri blog [Symlink di Windows 10!](#) .

Jika Anda menggunakan Git di bawah Windows, Anda harus mengaktifkan Mode Pengembang atau Anda harus:

- Atur `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`, `git submodule update --init --recursive`).

Konfigurasi 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 Anda [Mengatur AWS IoT dan Amazon Cognito untuk FreeRTOS Bluetooth Low Energy](#).

Untuk mengonfigurasi AWS IoT titik akhir

1. Masuk ke [konsol AWS IoT](#) tersebut.
2. Di panel navigasi, 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`. Ingatlah titik akhir ini.

3. Dalam panel navigasi, pilih Kelola, lalu pilih Hal. Ingatlah namaAWS IoT benda untuk perangkat Anda.
4. DenganAWS IoT endpoint dan namaAWS IoT benda Anda di tangan, buka`freertos/demos/include/aws_clientcredential.h` di IDE Anda, dan tentukan nilai untuk`#define` konstanta berikut:
 - `clientcredentialMQTT_BROKER_ENDPOINT`*AWS IoTEndpoint Anda*
 - `clientcredentialIOT_THING_NAME`*NamaAWS IoT hal papan Anda*

Untuk mengaktifkan demo

1. Periksa apakah Demo GATT Bluetooth Energi Rendah diaktifkan. Pergi ke `vendors/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. Buk `vendors/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 UUID khusus vendor. Tingkatkan jumlah ini sebesar 1 saat Anda menambahkan UUID khusus vendor baru.

- NRF_SDH_BLE_GATTS_ATTR_TAB_SIZE

Ukuran tabel atribut dalam byte. Ukurannya harus kelipatan dari 4. Nilai ini menunjukkan jumlah memori yang ditetapkan khusus 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 tersebut `freertos/vendors/nordic/boards/nrf52840-dk/aws_tests/config_files/sdk_config.h`.)

Bangun dan jalankan proyek demo FreeRTOS

Setelah Anda men-download 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 Segger Embedded Studio. Dari menu atas, pilih File, pilih Open Solution, dan kemudian arahkan 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 untuk output dari koneksi serial Anda.

3. Klik kanan proyekaws_demo di Project Explorer, dan pilih Build.

Note

Jika ini adalah pertama kalinya Anda menggunakan Segger Embedded Studio, Anda mungkin melihat Anda peringatan “Tidak ada lisensi untuk penggunaan komersial”. Segger Embedded Studio dapat digunakan secara gratis untuk perangkat Nordic Semiconductor. [Minta lisensi gratis](#), selama pengaturan pilih Aktifkan Lisensi Gratis Anda, dan ikuti instruksinya.

4. Pilih Debug, lalu pilih Pergi.

Setelah demo dimulai, ia menunggu untuk dipasangkan dengan perangkat seluler di Bluetooth Low Energy.

5. Ikuti petunjuk untuk [MQTT melalui Aplikasi Demo Bluetooth Energi Rendah 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 [Memulai masalah saat memulai](#).

Memulai dengan Nuvoton NuMaker -IoT-M487

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 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-M487. Seri mikrokontroler, dan termasuk built-in RJ45 Ethernet dan Wi-Fi modul. Jika Anda tidak memiliki Nuvoton NuMaker -IOT-M487, kunjungi [Katalog Perangkat AWS Mitra](#) untuk membelinya dari mitra kami.

Sebelum memulai, Anda harus mengonfigurasi AWS IoT dan perangkat lunak FreeRTOS Anda untuk menghubungkan papan pengembangan Anda ke AWS Cloud. Untuk petunjuk, lihat [Langkah pertama](#). Dalam tutorial ini, path ke direktori download 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. Cross-kompilasi aplikasi demo FreeRTOS ke gambar biner.
3. Muat gambar biner aplikasi ke papan Anda, dan kemudian 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 discount harga untuk MCU seri Nuvoton Cortex-M4. Keil MDK hanya didukung pada Windows.

Untuk menginstal alat pengembangan untuk NuMaker -IoT-M487

1. Unduh [Edisi Keil MDK Nuvoton](#) dari situs Keil MDK.
2. Instal Keil MDK pada mesin host Anda menggunakan lisensi Anda. Keil MDK menyertakan IDE Keil μ Vision, toolchain kompilasi C/C++, dan debugger μ Vision.

Jika Anda mengalami masalah selama instalasi, hubungi [Nuvoton](#) untuk mendapatkan 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 IDE Keil μ Vision.
2. Pada menu File, pilih Buka. Di kotak dialog Open file, pastikan pemilih jenis file diatur ke Project Files.
3. Pilih proyek demo Wi-Fi atau Ethernet yang akan dibuat.
 - Untuk membuka proyek demo Wi-Fi, pilih proyek `targetaws_demos.uvproj` di `freertos\projects\nuvoton\numaker_iot_m487_wifi\uvision\aws_demos` direktori.
 - Untuk membuka proyek demo Ethernet, pilih proyek `targetaws_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, dan kemudian pilih Opsi. (Lihat [Pemecahan Masalah](#) untuk lebih detail.)
5. Pada tab Utilitas, verifikasi bahwa Gunakan Driver Target untuk Pemrograman Flash dipilih, dan Nuvoton Nu-Link Debugger ditetapkan sebagai driver target.
6. Pada tab Debug, di samping Nuvoton Nu-Link Debugger, pilih Pengaturan.
7. Verifikasi bahwa Jenis Chip diatur ke M480.
8. Di panel navigasi Proyek Keil μ Vision IDE, pilih `aws_demos` proyek. Pada menu Project, pilih Build Target.

Anda dapat menggunakan klien MQTT di AWS IoT konsol untuk memantau pesan yang dikirim perangkat Anda ke AWS Cloud.

Untuk berlangganan topik MQTT dengan klien AWS IoT MQTT

1. Masuk ke [konsol AWS IoT](#) tersebut.
2. Di panel navigasi, pilih Uji, lalu pilih klien uji MQTT untuk membuka klien MQTT.
3. Di Topik berlangganan `your-thing-name/example/topic`, masukkan, lalu pilih Berlangganan topik.

Untuk menjalankan proyek demo FreeRTOS

1. Connect papan Numaker-IoT-M487 Anda ke mesin host Anda (komputer).
2. Membangun kembali proyek.
3. Di Keil μ Vision 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, dan kemudian pilih Run (F5).

Anda akan melihat pesan MQTT yang dikirim oleh perangkat Anda di klien MQTT diAWS IoT konsol.

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 build CMake. Ikuti petunjuk di [Menggunakan CMake dengan FreeRTOS](#), dan kemudian ikuti langkah-langkah di bagian ini.

Note

Pastikan path ke lokasi compiler (Keil) ada di variabel sistem Path Anda, misalnya, `C:\Keil_v5\ARM\ARMCC\bin`.

Anda juga dapat menggunakan klien MQTT diAWS IoT konsol untuk memantau pesan yang dikirim perangkat Anda keAWS Cloud.

Untuk berlangganan topik MQTT dengan klienAWS IoT MQTT

1. Masuk ke [konsol AWS IoT](#) tersebut.
2. Di panel navigasi, pilih Uji, lalu pilih klien uji MQTT untuk membuka klien MQTT.
3. Di Topik berlangganan `your-thing-name/example/topic`, masukkan, lalu pilih Berlangganan topik.

Untuk menghasilkan file build dari file sumber dan menjalankan proyek demo

1. Pada mesin host Anda, buka prompt perintah dan arahkan ke folder *freertos*.
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:

Buka direktori yang berisi berkas sumber yang berisi berkas sumber untuk proyek demo FreeRTOS. Kemudian, buat berkas build dengan menjalankan perintah berikut.

```
cmake -DVENDOR=nuvoton -DBOARD=numaker_iot_m487_wifi -DCOMPILER=arm-keil -S . -B BUILD_FOLDER -G Ninja
```

- Untuk Ethernet:

Buka direktori yang berisi berkas sumber yang berisi berkas sumber untuk proyek demo FreeRTOS. Kemudian, buat berkas 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 mem-flash ke M487 dengan menjalankan perintah berikut.

```
cmake --build BUILD_FOLDER
```

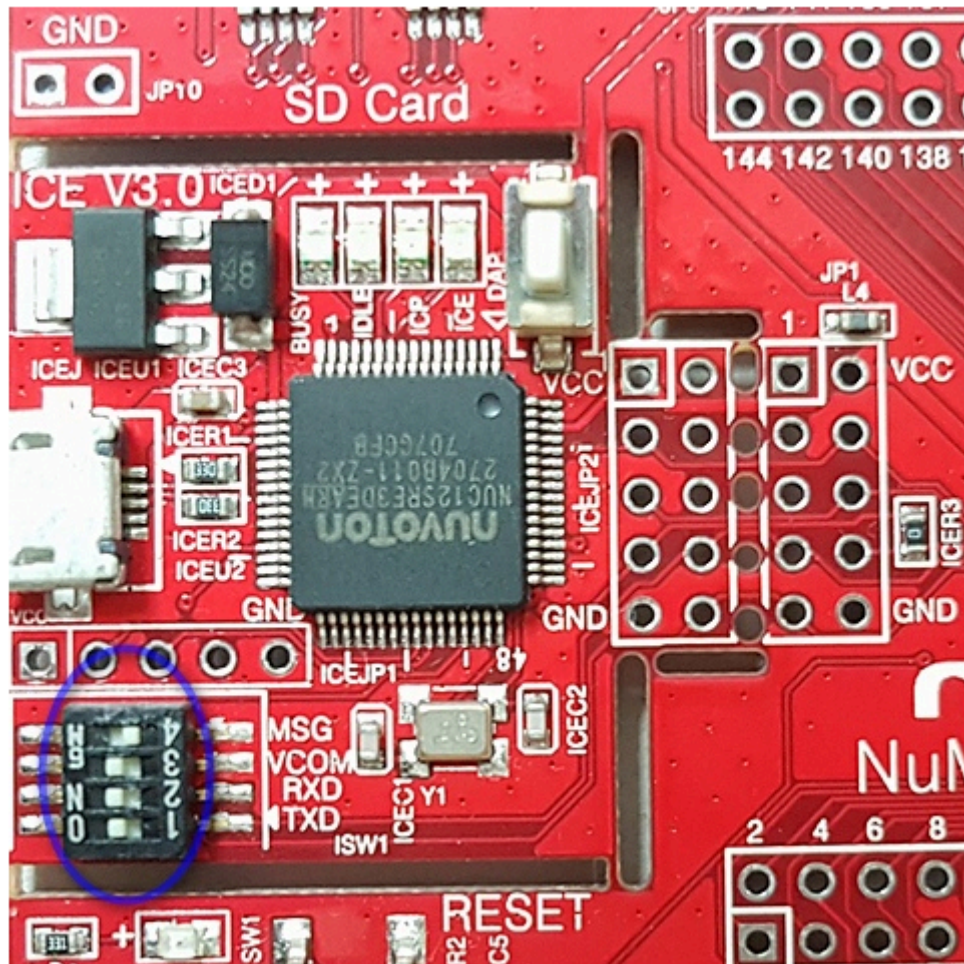
Pada titik ini, file bineraws_demos.bin harus ada di *BUILD_FOLDER*/vendors/Nuvoton/boards/numaker_iot_m487_wifi folder.

5. Untuk mengkonfigurasi papan untuk mode berkedip, pastikan sakelar MSG (No.4 dari ISW1 di 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 [Memasang emulator terminal](#).
7. Jalankan proyek demo dengan menyalin biner yang dihasilkan ke perangkat.

Jika Anda berlangganan topik MQTT dengan klien AWS IoT MQTT, Anda akan melihat pesan MQTT yang dikirim oleh perangkat Anda di AWS IoT konsol.

Pemecahan Masalah

- Jika jendela Anda tidak dapat mengenali perangkat VCOM, instal driver port serial NuMaker windows dari tautan [Driver USB Nu-Link v1.6](#).
- Jika Anda menghubungkan perangkat Anda ke Keil MDK (IDE) melalui Nu-Link, pastikan sakelar MSG (No.4 dari ISW1 di ICE) MATI, seperti yang ditunjukkan.



Jika Anda mengalami masalah saat menyiapkan lingkungan pengembangan atau terhubung ke papan Anda, hubungi [Nuvoton](#).

Debugging proyek FreeRTOS di Keil μ Vision

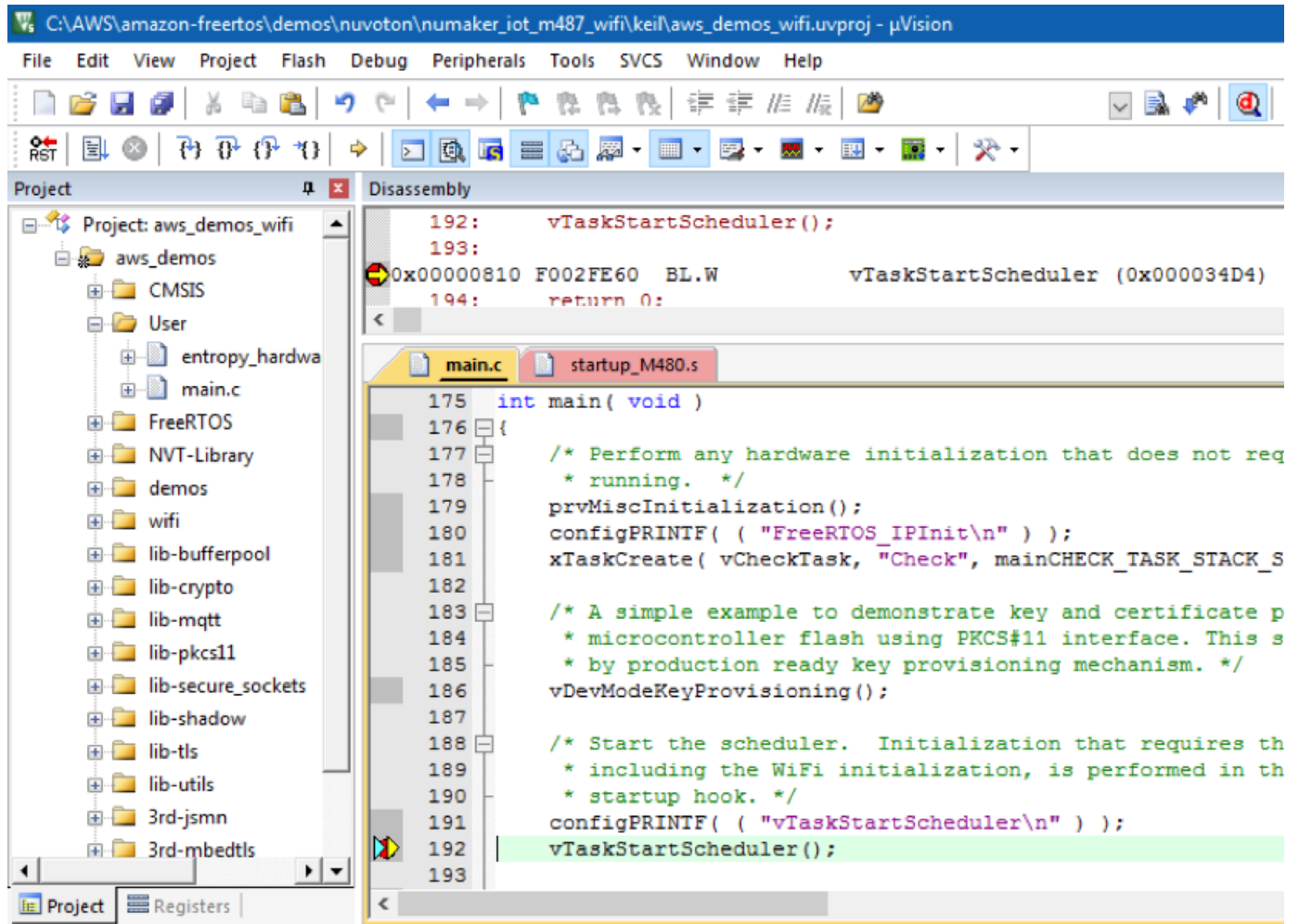
Untuk memulai sesi debug di Keil μ Vision

1. Buka Keil μ Vision.
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. μ Vision mem-flash demo ke papan, menjalankan demo, dan berhenti di awal `main()` fungsi.

4. Tetapkan breakpoint dalam kode sumber proyek Anda, dan kemudian jalankan kode. Proyek akan terlihat seperti berikut ini.



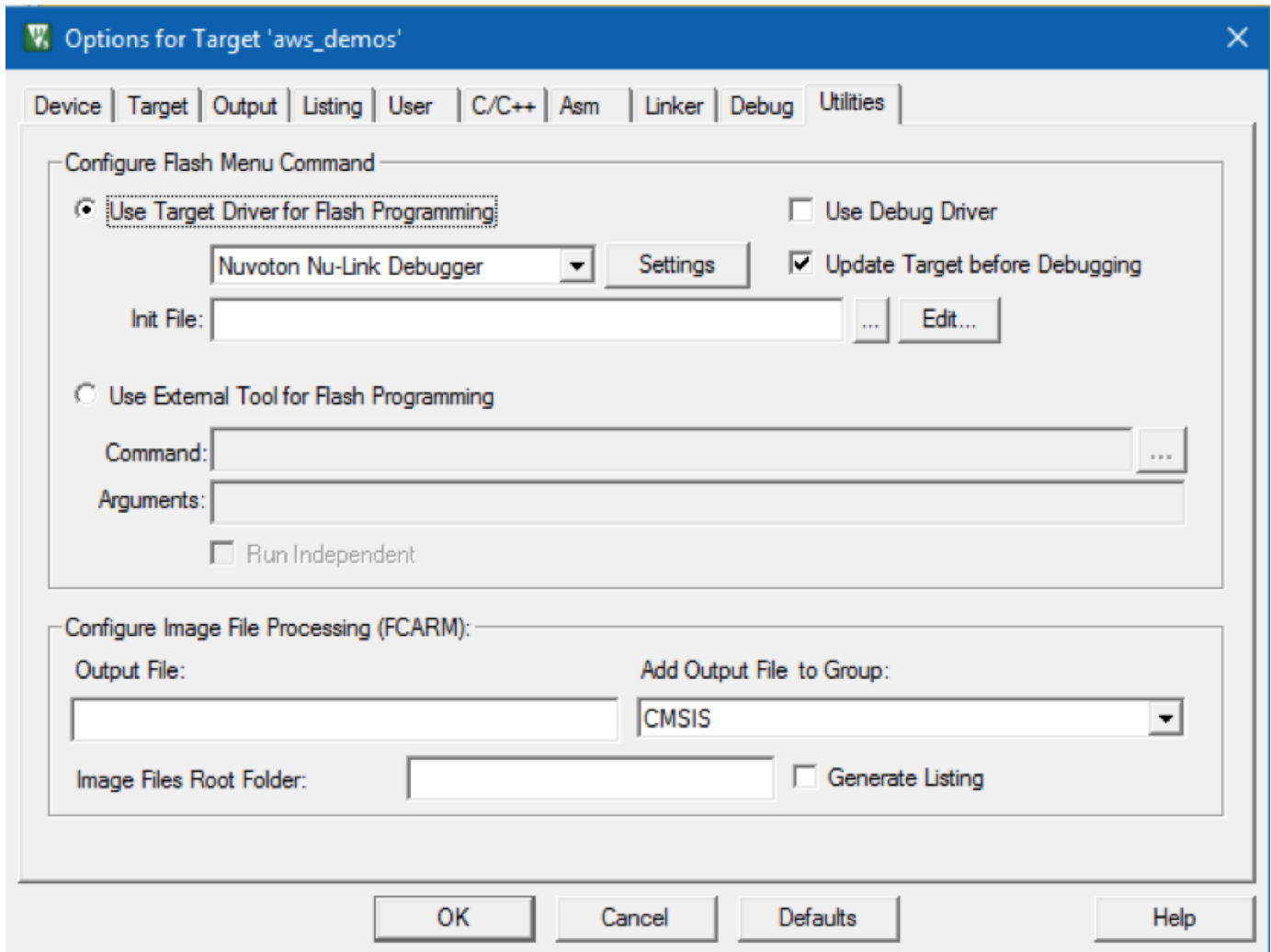
Memecahkan masalah pengaturan debug μ Vision

Jika Anda mengalami masalah saat men-debug aplikasi, periksa apakah pengaturan debug Anda diatur dengan benar di Keil μ Vision.

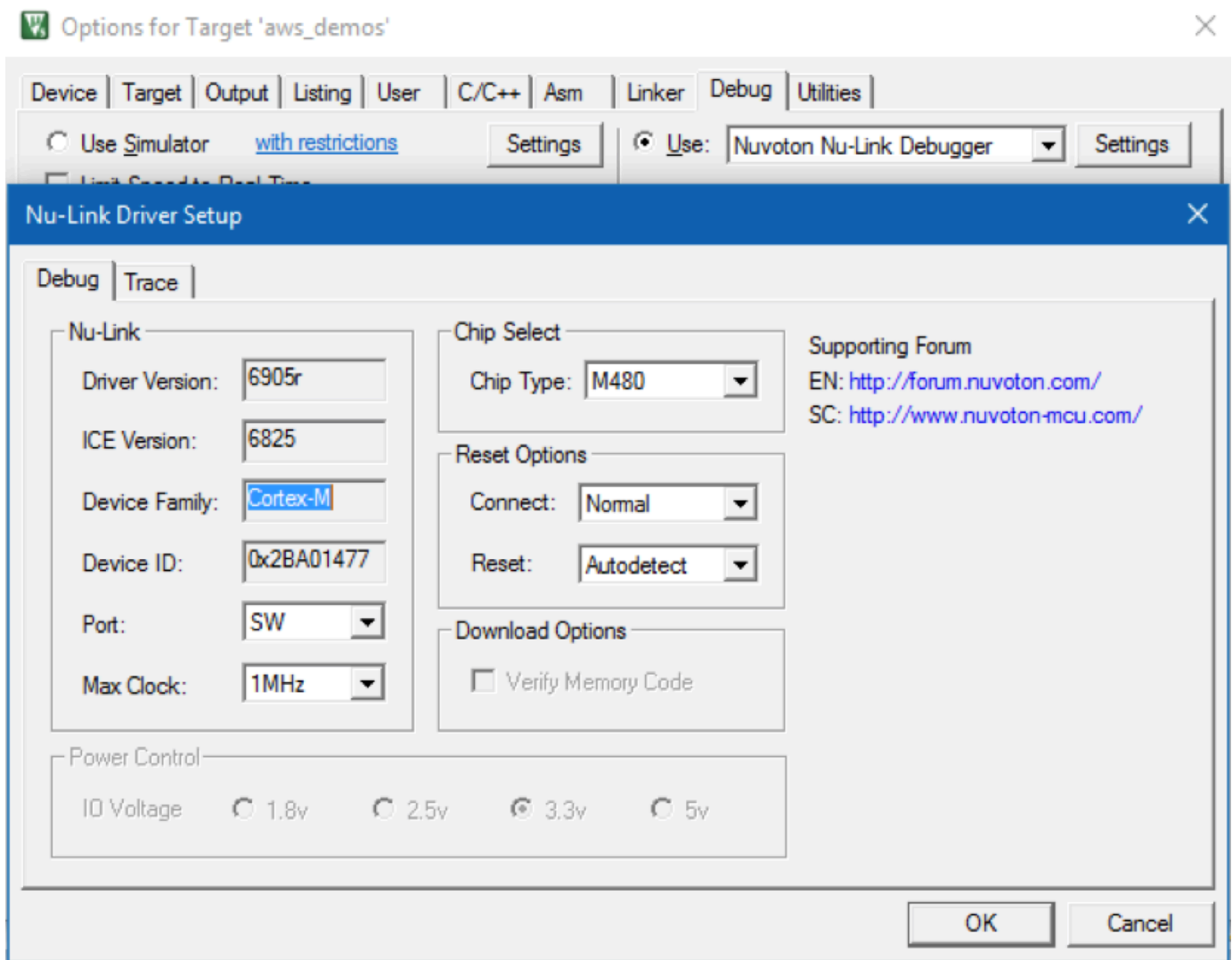
Untuk memverifikasi bahwa pengaturan debug μ Vision sudah benar

1. Buka Keil μ Vision.
2. Klik kanan `aws_demo` proyek di IDE, dan kemudian pilih Options.

3. Pada tab Utilitas, verifikasi bahwa Gunakan Driver Target untuk Pemrograman Flash dipilih, dan Nuvoton Nu-Link Debugger ditetapkan sebagai driver target.



4. Pada tab Debug, di samping Nuvoton Nu-Link Debugger, pilih Pengaturan.



5. Verifikasi bahwa Jenis Chip diatur ke M480.

Memulai dengan Modul IoT NXP LPC54018

⚠ 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 IoT NXP LPC54018. [Jika Anda tidak memiliki Modul IoT NXP LPC54018, kunjungi Katalog Perangkat Mitra AWS untuk membelinya dari mitra kami.](#) Gunakan kabel USB untuk menghubungkan Modul IoT NXP LPC54018 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*

Ikhtisar

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 LPC54018

- Hubungkan komputer Anda ke port USB pada NXP LPC54018.

Untuk mengatur debugger JTAG

Anda memerlukan debugger JTAG untuk meluncurkan dan men-debug kode Anda yang berjalan di papan NXP LPC54018. FreeRTOS diuji menggunakan Modul IoT OM40006. [Untuk informasi selengkapnya tentang debugger yang didukung, lihat Panduan Pengguna untuk Modul IoT NXP LPC54018 yang tersedia dari halaman produk Modul IoT OM40007 LPC54018.](#)

1. Jika Anda menggunakan debugger Modul IoT OM40006, gunakan kabel konverter untuk menghubungkan konektor 20-pin dari debugger ke konektor 10-pin pada modul NXP IoT.
2. Hubungkan NXP LPC54018 dan Debugger Modul IoT OM40006 ke port USB di komputer Anda menggunakan kabel mini-USB ke USB.


Siapkan lingkungan pengembangan Anda

FreeRTOS mendukung dua IDE untuk Modul IoT NXP LPC54018: IAR Embedded Workbench dan MCUxPresso.

Sebelum Anda mulai, instal salah satu IDE ini.

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 penginstal MCUxPresso 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 Modul LPC54018-IoT.
6. Verifikasi detail perangkat keras, lalu pilih Build MCUxPresso 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 tab SDK Terinstal.
9. Seret dan lepas file arsip SDK yang diunduh ke jendela SDK 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 Open Projects From File System.

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 J-Link Device Selection 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 MCUxPresso IDE

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 debugger MCUxPresso untuk modul NXP54018-IoT).

4. Ketika debugger berhenti di breakpoint `main()`, dari menu Debug, pilih Go.

Memecahkan masalah

Untuk informasi pemecahan masalah umum tentang Memulai FreeRTOS, lihat. [Memulai masalah saat memulai](#)

Memulai dengan Renesas Starter Kit+untuk RX65N-2MB

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 Renesas Starter Kit+untuk RX65N-2MB. Jika Anda tidak memiliki Renesas RSK+ untuk RX65N-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 AWS Cloud. Lihat [Langkah pertama](#) untuk instruksi. Dalam tutorial ini, path ke direktori download 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. Cross kompilasi 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 RX65N-2MB

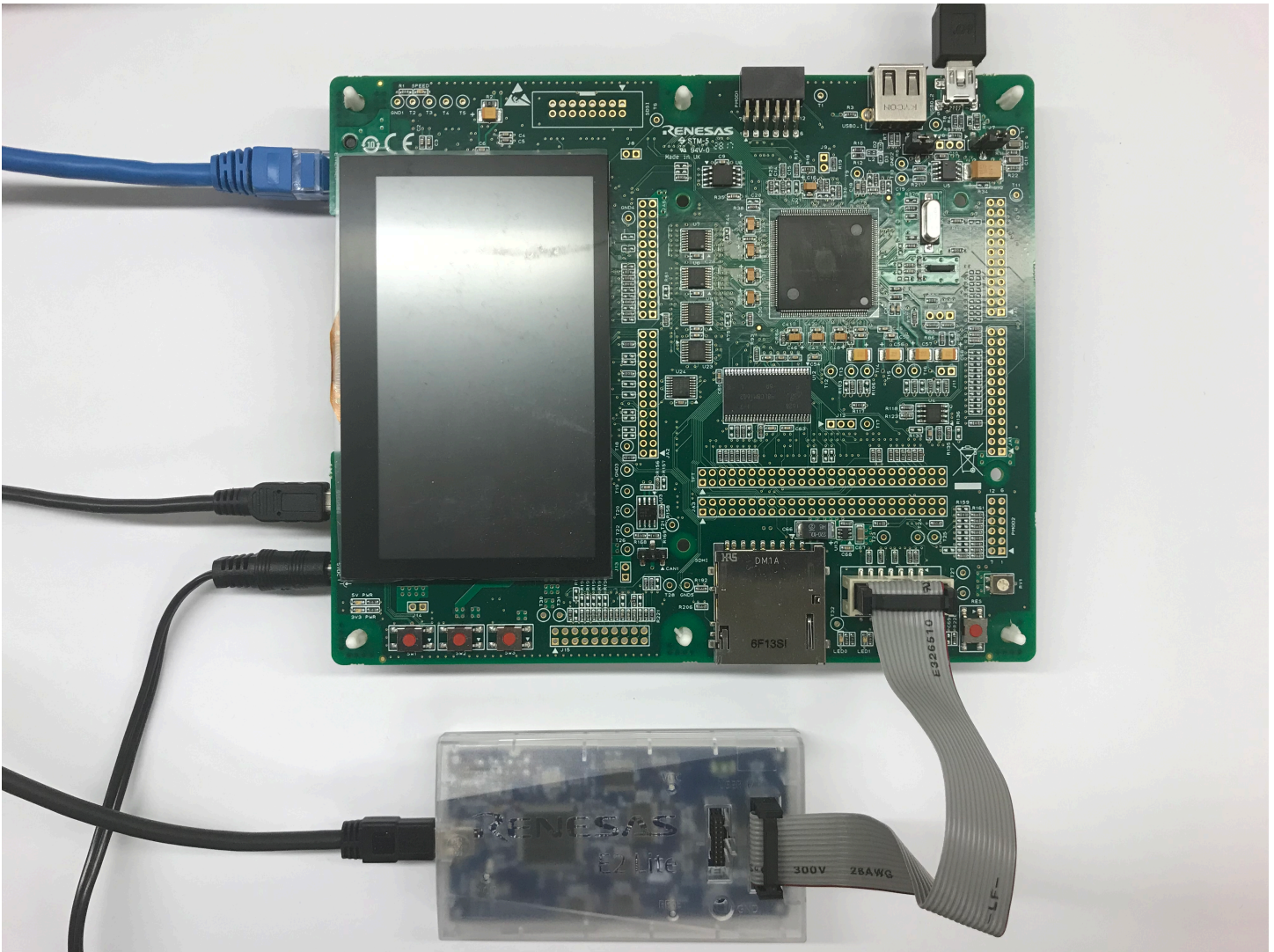
1. Connect adaptor daya +5V positif ke konektor PWR pada RSK+untuk RX65N-2MB.

2. Connect komputer Anda ke port USB2.0 FS pada RSK+untuk RX65N-2MB.
3. Connect komputer Anda ke port USB-ke-Serial pada RSK+untuk RX65N-2MB.
4. Connect router atau port Ethernet yang terhubung ke internet ke port Ethernet pada RSK+untuk RX65N-2MB.

Untuk mengatur modul Debugger E2 Lite

1. Gunakan kabel pita 14-pin untuk menghubungkan modul E2 Lite Debugger ke port 'E1/E2 Lite' pada RSK+untuk RX65N-2MB.
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 dan RSK+untuk RX65N-2MB, driver debugger E2 Lite mulai menginstal.

Perhatikan bahwa hak istimewa administrator diperlukan untuk menginstal driver.



Siapkan lingkungan pengembangan Anda

Untuk mengatur konfigurasi FreeRTOS untuk RSK+ untuk RX65N-2MB, gunakan Renesas e² studio IDE dan CC-RX compiler.

Note

Renesas e² studio IDE dan CC-RX compiler hanya didukung pada sistem operasi Windows 7, 8, dan 10.

Untuk mengunduh dan menginstal e² studio

1. Pergi ke [Renesas e² studio installer](#) download halaman, dan men-download installer offline.

2. Anda diarahkan ke halaman Login Renesas.

Jika Anda memiliki akun dengan Renesas, masukkan kredensial masuk Anda, lalu pilih Login.

Jika Anda tidak memiliki akun, pilih Daftar sekarang, dan ikuti langkah pendaftaran pertama. Anda akan menerima email dengan tautan atur ulang akun Anda. Ikuti tautan ini untuk menyelesaikan pendaftaran Anda dengan Renesas, lalu masuk ke Renesas.

3. Setelah Anda masuk, unduh installer studio e² ke komputer Anda.
4. Buka installer dan ikuti langkah-langkah untuk menyelesaikan.

Untuk informasi lebih lanjut, lihat [e² studio](#) di situs Renesas.

Untuk mengunduh dan menginstal Package Compiler RX Family C/C ++

1. Buka halaman unduhan [Package Kompiler RX Family C/C ++](#), dan unduh paket V3.00.00.
2. Buka executable dan instal compiler.

Untuk informasi selengkapnya, lihat [Package Kompiler C/C++ untuk Keluarga RX](#) di situs web Renesas.

Note

Compiler 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 Anda telah mengkonfigurasi proyek demo, Anda siap untuk membangun dan menjalankan proyek di papan Anda.

Membangun Demo FreeRTOS di e² studio

Untuk mengimpor dan membangun demo di e² studio

1. Luncurkan e² 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 e² studio, jendela Toolchain Registry terbuka. Pilih Renesas Toolchains, dan konfirmasi **CC-RX v3.00.00** bahwa telah dipilih. Pilih Daftar, lalu pilih OK.
4. Jika Anda membuka e² studio untuk pertama kalinya, jendela Registrasi Generator Kode muncul. Pilih OKE.
5. Kode Generator COM komponen mendaftar jendela muncul. Di bawah Silakan restart e² studio untuk menggunakan Code Generator, pilih OK.
6. Restart e² jendela studio muncul. Pilih OKE.
7. e² restart studio. Pada jendela Pilih direktori sebagai ruang kerja, pilih Luncurkan.
8. Di layar selamat datang studio e², pilih ikon panah meja kerja studio e².
9. Klik kanan jendela Project Explorer, dan pilih Impor.
10. Di wizard impor, pilih Umum, Proyek yang Ada ke Ruang Kerja, lalu pilih Berikutnya.
11. Pilih Jelajahi, 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 compiler CC-RX. Untuk menginstal License Manager, lihat halaman unduhan [License Manager](#).

Memantau pesan MQTT di cloud

Sebelum Anda menjalankan proyek demo FreeRTOS, Anda dapat mengatur klien MQTT di AWS IoT konsol untuk memantau pesan yang dikirim perangkat Anda ke AWS Cloud.

Untuk berlangganan topik MQTT dengan klien AWS IoT MQTT

1. Masuk ke [konsol AWS IoT](#) tersebut.
2. Di panel navigasi, pilih Uji, lalu pilih klien uji MQTT untuk membuka klien MQTT.
3. Di Topik berlangganan ***your-thing-name/example/topic***, masukkan, lalu pilih Berlangganan topik.

Ketika proyek demo berhasil berjalan pada perangkat Anda, Anda melihat “Hello World!” dikirim beberapa kali ke topik yang Anda berlangganan.

Jalankan proyek FreeRTOS

Untuk menjalankan proyek di e² studio

1. Konfirmasikan bahwa Anda telah menghubungkan modul E2 Lite Debugger ke RSK+Anda untuk RX65N-2MB
2. Dari menu atas, pilih Run, Debug Configuration.
3. Perluas Renesas GDB Hardware Debugging, dan pilih aws_demos HardwareDebug.
4. Pilih tab Debugger, dan kemudian 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. Itu e² Studio mungkin meminta untuk mengubah ke Renesas Debug Perspektif. Pilih Ya.

LED 'ACT' hijau pada Debugger E2 Lite menyala.

7. Setelah kode diunduh ke papan, 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` garpuamazon-freertos repositori aktif [GitHub](#).

Pemecahan Masalah

Untuk informasi pemecahan masalah umum tentang Memulai FreeRTOS, lihat [Memulai masalah saat memulai](#).

Memulai dengan STMicroelectronics STM32L4 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 STM32L4 Discovery Kit IoT Node. [Jika Anda belum memiliki STMicroelectronics STM32L4 Discovery Kit IoT Node, kunjungi Katalog Perangkat Mitra untuk membelinya AWS 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, SubGHz](#), Wi-Fi. 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*

Ikhtisar

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 [OpenSTM32.org](#).
2. Daftar di halaman web OpenSTM32. Anda harus masuk untuk mengunduh System Workbench.
3. Jelajahi [Meja Kerja Sistem untuk penginstal STM32](#) untuk mengunduh dan menginstal Meja Kerja Sistem.

Jika Anda mengalami masalah selama instalasi, lihat FAQ di situs web [System Workbench](#).

Bangun dan jalankan proyek demo FreeRTOS

Impor demo FreeRTOS ke Meja Kerja Sistem STM32

1. Buka Meja Kerja Sistem STM32 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/stm321475_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 STM32L4 Discovery Kit IoT Node ke komputer Anda. (Periksa dokumentasi pabrikan yang disertakan dengan papan Anda untuk port USB yang benar untuk digunakan.)
2. Dari Project Explorer, klik kanan, pilih Debug As `aws_demos`, 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 Debugging `Ac6 STM32`.)
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 `main()`, dari menu Run, pilih Lanjutkan.

Menggunakan CMake dengan FreerTos

Jika Anda memilih untuk tidak menggunakan IDE untuk pengembangan Freertos, Anda dapat menggunakan CMake untuk membangun dan menjalankan aplikasi demo atau aplikasi yang telah Anda kembangkan menggunakan editor kode pihak ketiga dan alat debugging.

Pertama buat folder yang berisi file build yang dihasilkan (*build-folder*).

Gunakan perintah berikut untuk menghasilkan file build:

```
cmake -DVENDOR=st -DBOARD=stm321475_discovery -DCOMPILER=arm-gcc -S freertos -B build-  
folder
```

Jika tidak `arm-none-eabi-gcc` ada di jalur shell Anda, Anda juga perlu mengatur variabel `AFR_TOOLCHAIN_PATH` CMake. Sebagai contoh:

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

Untuk informasi selengkapnya tentang menggunakan CMake dengan FreerTos, lihat. [Menggunakan CMake dengan FreeRTOS](#)

Memecahkan 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: xxxxxxxxxxxxxx  
[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, SubGHz](#), Wi-Fi. 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. [Memulai masalah saat memulai](#)

Memulai dengan Texas Instruments CC3220SF-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 CC3220SF-LAUNCHXL. [Jika Anda tidak memiliki Kit Pengembangan Texas Instruments \(TI\) CC3220SF-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*

Ikhtisar

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 dua IDE untuk Kit Pengembangan TI CC3220SF-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 MCU.
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 FAQ, 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 SDK SimpleLink CC3220

Instal SDK [SimpleLink CC3220](#). SimpleLink Wi-Fi CC3220 SDK berisi driver untuk MCU CC3220SF 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 MCU. Uniflash memiliki GUI, baris perintah, dan antarmuka scripting.

Instal paket layanan terbaru

1. Pada TI CC3220SF-LAUNCHXL Anda, letakkan jumper SOP di set tengah pin (posisi = 1) dan setel ulang papan.
2. Mulai Uniflash. Jika LaunchPad papan CC3220SF 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 SimpleLink CC3220SF. 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.

Konfigurasi penyedia Wi-Fi

Untuk mengonfigurasi pengaturan Wi-Fi untuk papan Anda, lakukan salah satu hal berikut:

- Konfigurasi aplikasi demo FreeRTOS yang dijelaskan dalam [Mengkonfigurasi demo FreeRTOS](#)
- Gunakan [SmartConfig](#) dari 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) di Texas Instruments CC3220SF-LAUNCHXL Anda berada di posisi 0. Untuk informasi selengkapnya, lihat Panduan Pengguna [SimpleLink Prosesor Jaringan Wi-Fi CC3x20, CC3x3x](#).
2. Gunakan kabel USB untuk menyambungkan Texas Instruments CC3220SF-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) di Texas Instruments CC3220SF-LAUNCHXL Anda berada di posisi 0. Untuk informasi selengkapnya, lihat Panduan Pengguna [SimpleLink Prosesor Jaringan Wi-Fi CC3x20, CC3x3x](#).
2. Gunakan kabel USB untuk menyambungkan Texas Instruments CC3220SF-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 menggunakan CMake 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 kompilator TI CGT berada (misalnya). `C:\ti\ccs910\ccs\tools\compiler\ti-cgt-arm_18.12.2.LTS\bin`

Jika Anda menggunakan kompilator 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](#).

Memecahkan 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. [Memulai masalah saat memulai](#)

Memulai dengan Simulator Perangkat Windows

Tutorial ini memberikan petunjuk untuk memulai dengan FreeRTOS Windows Device Simulator.

Sebelum memulai, Anda harus mengonfigurasi AWS IoT dan mengunduh FreeRTOS Anda untuk menghubungkan perangkat Anda ke AWS Cloud. Lihat [Langkah pertama](#) untuk instruksi. Dalam tutorial ini, path ke direktori download 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 porting untuk dijalankan di Windows. Kumpulan file ini disebut sebagai simulator FreeRTOS untuk Windows.

Note

Tutorial ini tidak dapat berhasil dijalankan pada instans Windows Amazon EC2.

Siapkan lingkungan pengembangan Anda

1. Instal versi terbaru [Npcap](#). 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 Windows 10 SDK terbaru. Anda dapat memilih ini di bawah bagian Opsional pengembangan Desktop dengan komponen C ++.

3. Pastikan Anda memiliki koneksi Ethernet aktif.
4. (Opsional) Jika Anda ingin menggunakan sistem build berbasis CMake untuk membangun proyek FreeRTOS, instal versi terbaru [CMake](#). FreeRTOS memerlukan CMake versi 3.13 atau yang lebih baru.

Memantau pesan MQTT di cloud

Sebelum Anda menjalankan proyek demo FreeRTOS, Anda dapat mengatur klien MQTT diAWS IoT konsol untuk memantau pesan yang dikirim perangkat Anda keAWS Cloud.

Untuk berlangganan topik MQTT dengan klienAWS IoT MQTT

1. Masuk ke [konsol AWS IoT](#) tersebut.
2. Di panel navigasi, pilih Uji, lalu pilih klien uji MQTT untuk membuka klien MQTT.
3. Di Topik berlangganan ***your-thing-name/example/topic***, masukkan, lalu pilih Berlangganan topik.

Ketika proyek demo berhasil berjalan pada 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 membuat project FreeRTOS.

Membangun dan menjalankan proyek demo FreeRTOS dengan Visual Studio IDE

1. Memuat proyek ke Visual Studio.

Di Visual Studio, dari menu File, pilih Buka. Pilih File/Solution, navigasikan ke `projects/pc/windows/visual_studio/aws_demos/aws_demos.sln` file, lalu pilih Buka.

2. Menargetkan ulang proyek demo.

Proyek demo yang disediakan bergantung pada Windows SDK, tetapi tidak memiliki versi Windows SDK yang ditentukan. Secara default, IDE mungkin mencoba membuat demo dengan versi SDK yang tidak ada di komputer Anda. Untuk mengatur versi Windows SDK, klik `kananaws_demos` dan kemudian pilih `Retarget Proyek`. Ini akan membuka jendela `Review Solution Actions`. Pilih versi Windows SDK yang ada di komputer Anda (nilai awal dalam dropdown baik-baik saja), dan kemudian pilih `OK`.

3. Bangun dan jalankan proyek.

Dari menu `Build`, pilih `Build Solution`, dan pastikan solusi dibuat tanpa kesalahan atau peringatan. Pilih `Debug`, `Mulai Debugging` untuk menjalankan proyek. Pada saat pertama, Anda harus [memilih antarmuka jaringan](#).

Membangun dan menjalankan proyek demo FreeRTOS dengan CMake

Kami menyarankan Anda menggunakan CMake GUI bukan alat baris perintah CMake 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 Dimana kode sumbernya.

Ditetapkan *freertos/build* untuk Dimana untuk membangun biner.

2. Konfigurasi Proyek CMake.

Di CMake GUI, pilih `Add Entry`, dan pada jendela `Add Cache Entry`, atur nilai berikut:

Nama

`AFR_BOARD`

Tipe

`STRING`

Nilai

`pc.jendela`

Deskripsi

(Opsional)

3. Pilih Configure (Konfigurasi). Jika CMake meminta Anda untuk membuat direktori build, pilih Ya, lalu pilih generator di bawah Tentukan generator untuk proyek ini. Sebaiknya gunakan Visual Studio sebagai generator, tetapi Ninja juga didukung. (Perhatikan bahwa saat menggunakan Visual Studio 2019, platform harus diatur ke Win32 alih-alih pengaturan defaultnya.) Jaga agar opsi generator lainnya tidak berubah dan pilih Selesai.
4. Hasilkan dan Buka Proyek CMake.

Setelah Anda mengkonfigurasi proyek, CMake GUI menampilkan semua opsi yang tersedia untuk proyek yang dihasilkan. Untuk tujuan tutorial ini, Anda dapat meninggalkan opsi pada nilai default mereka.

Pilih Generate untuk membuat solusi Visual Studio, dan kemudian pilih Open Project untuk membuka proyek di Visual Studio.

Di Visual Studio, klik kanan `aws_demos` proyek dan pilih Set as StartUp Project. Hal ini memungkinkan Anda untuk membangun dan menjalankan proyek. Pada saat pertama, Anda harus [memilih antarmuka jaringan](#).

Untuk informasi selengkapnya tentang menggunakan CMake dengan FreeRTOS, lihat [Menggunakan CMake dengan FreeRTOS](#).

Konfigurasi antarmuka jaringan Anda

Pada proyek demo pertama, Anda harus memilih antarmuka jaringan yang akan digunakan. Program ini menghitung antarmuka jaringan Anda. Temukan nomor untuk antarmuka Ethernet terprogram Anda. Output-nya 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 adalah 1.

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 galat 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 [Memulai masalah saat 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 petunjuk untuk memulai dengan Xilinx Avnet MicroZed Industrial IoT Kit. Jika Anda tidak memiliki Xilinx Avnet MicroZed Industrial IoT 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 AWS Cloud. Lihat [Langkah pertama](#) untuk instruksi. Dalam tutorial ini, path ke direktori download FreeRTOS disebut sebagai *freertos*.

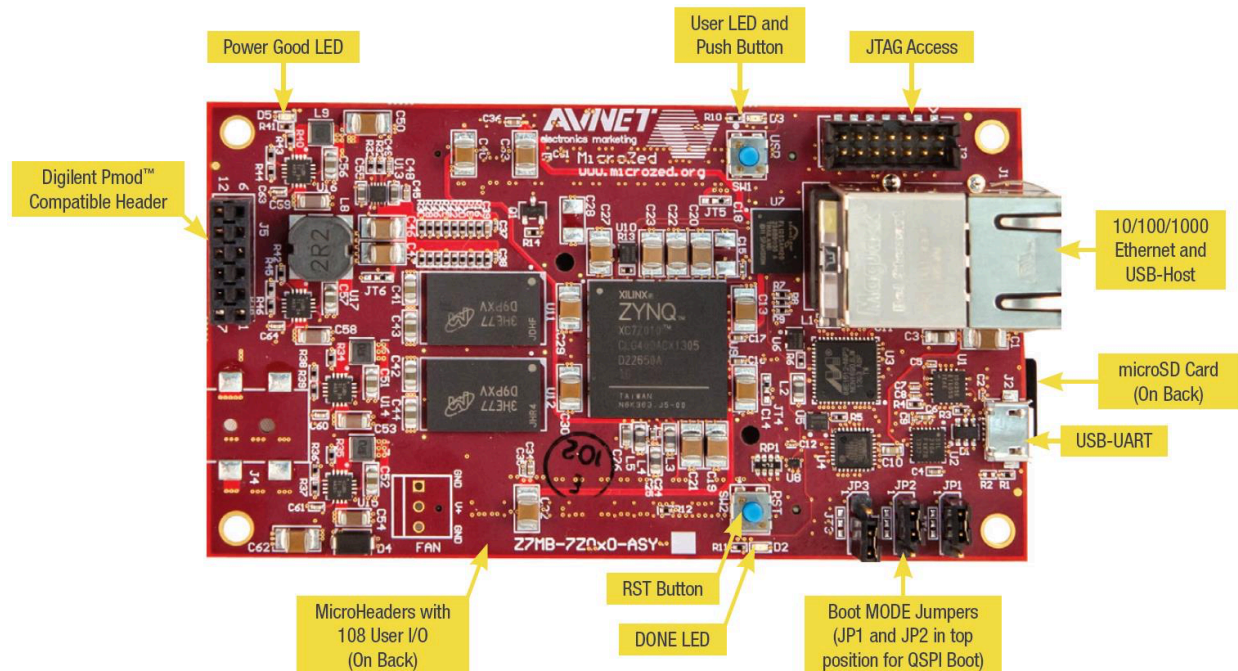
Gambaran Umum

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. Cross kompilasi aplikasi demo FreeRTOS ke gambar biner.
4. Memuat gambar biner aplikasi ke papan Anda, dan kemudian menjalankan aplikasi.

Mengatur perangkat MicroZed keras

Diagram berikut mungkin berguna saat Anda menyiapkan perangkat MicroZed keras:



Untuk mengatur MicroZed papan

1. Connect komputer Anda ke port USB-UART di MicroZed papan Anda.
2. Connect komputer Anda ke port JTAG Access di MicroZed papan Anda.
3. Connect router atau port Ethernet yang terhubung ke internet ke port Ethernet dan USB-Host di MicroZed papan Anda.

Siapkan lingkungan pengembangan Anda

Untuk mengatur konfigurasi FreeRTOS untuk MicroZed kit, Anda harus menggunakan Xilinx Software Development Kit (XSDK). XSDK didukung di Windows dan Linux.

Unduh dan instal XSDK

Untuk menginstal perangkat lunak Xilinx, Anda memerlukan akun Xilinx gratis.

Untuk mengunduh XSDK

1. Pergi ke halaman download [Software Development Kit Standalone WebInstall Client](#).
2. Pilih opsi yang sesuai untuk sistem operasi Anda.
3. Anda akan diarahkan ke halaman masuk Xilinx.

Jika Anda memiliki akun dengan Xilinx, masukkan kredensial masuk, lalu pilih Masuk.

Jika Anda tidak memiliki akun, pilih Buat akun Anda. Setelah Anda 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 memulai.
5. Simpan file `Xilinx_SDK_version_os`.

Untuk menginstal XSDK

1. Buka file `Xilinx_SDK_version_os`.
2. Di Pilih Edisi untuk Menginstal, pilih Xilinx Software Development Kit (XSDK) dan kemudian pilih Berikutnya.
3. Pada halaman berikut dari wizard instalasi, di bawah Opsi Instalasi, pilih Instal Driver Kabel dan kemudian pilih Berikutnya.

Jika komputer Anda tidak mendeteksi koneksi USB-UART, instal driver VCP CP210x USB-to-UART Bridge secara manual. MicroZed Untuk petunjuknya, lihat [Panduan Instalasi USB-to-UART Silicon Labs CP210x](#).

Untuk informasi selengkapnya tentang XSDK, lihat [Memulai dengan Xilinx SDK](#) di situs web Xilinx.

Memantau pesan MQTT di cloud

Sebelum Anda menjalankan proyek demo FreeRTOS, Anda dapat mengatur klien MQTT diAWS IoT konsol untuk memantau pesan yang dikirim perangkat Anda keAWS Cloud.

Untuk berlangganan topik MQTT dengan klienAWS IoT MQTT


1. Masuk ke [konsol AWS IoT](#) tersebut.
2. Di panel navigasi, pilih Uji, lalu pilih klien uji MQTT untuk membuka klien MQTT.
3. Di Topik berlangganannya ***your-thing-name/example/topic***, masukkan, lalu pilih Berlangganan topik.

Bangun dan jalankan proyek demo FreeRTOS

Buka demo FreeRTOS di XSDK IDE

1. Luncurkan XSDK IDE dengan direktori workspace diatur 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, luaskan Umum, pilih Proyek yang Ada ke Ruang Kerja, lalu pilih Berikutnya.
5. Pada halaman Proyek Impor, pilih Pilih direktori root, dan kemudian masukkan direktori root proyek demo Anda: ***freertos/projects/xilinx/microzed/xsdk/aws_demos***. Untuk menelusuri direktori, pilih Jelajahi.

Setelah Anda menentukan direktori root, proyek di direktori itu muncul di halaman Proyek Impor. Semua proyek yang tersedia dipilih secara default.

 Note

Jika Anda melihat peringatan di bagian atas halaman Proyek Impor (“Beberapa proyek tidak dapat diimpor karena sudah ada di ruang kerja.”) Anda dapat mengabaikannya.

6. Dengan semua proyek yang dipilih, pilih Finish.
7. Jika Anda tidak melihat `aws_bsp`, `fsbl`, dan `MicroZed_hw_platform_0` proyek di panel proyek, ulangi langkah-langkah sebelumnya mulai dari #3 tetapi dengan direktori root diatur ke `freertos/vendors/xilinx`, dan impor `aws_bsp`, `fsbl`, dan `MicroZed_hw_platform_0`.
8. Dari menu, pilih Jendela, lalu pilih Preferensi.
9. Di panel navigasi, perluas Jalankan/Debug, pilih Substitusi String, lalu pilih Baru.
10. Dalam Variabel Substitusi String Baru, untuk Nama, masukkan **AFR_ROOT**. Untuk Nilai, masukkan jalur root dari `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 Bersihkan, biarkan opsi pada nilai defaultnya, lalu pilih OK. XSDK membersihkan dan membangun semua proyek, dan kemudian menghasilkan `.elf` file.

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 gambar 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 samping Output jalur file BIF, pilih Browse, dan kemudian pilih `aws_demos.bif` terletak di `<freertos>/vendors/xilinx/microzed/aws_demos/aws_demos.bif`.
4. Pilih Tambahkan.
5. Pada Tambahkan partisi gambar boot baru, di samping File path, pilih Browse, dan kemudian 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 `B00T.bin` file di `projects/xilinx/microzed/xsdk/aws_demos/B00T.bin`.

Debug JTAG

1. Atur jumper mode boot MicroZed papan Anda ke mode boot JTAG.

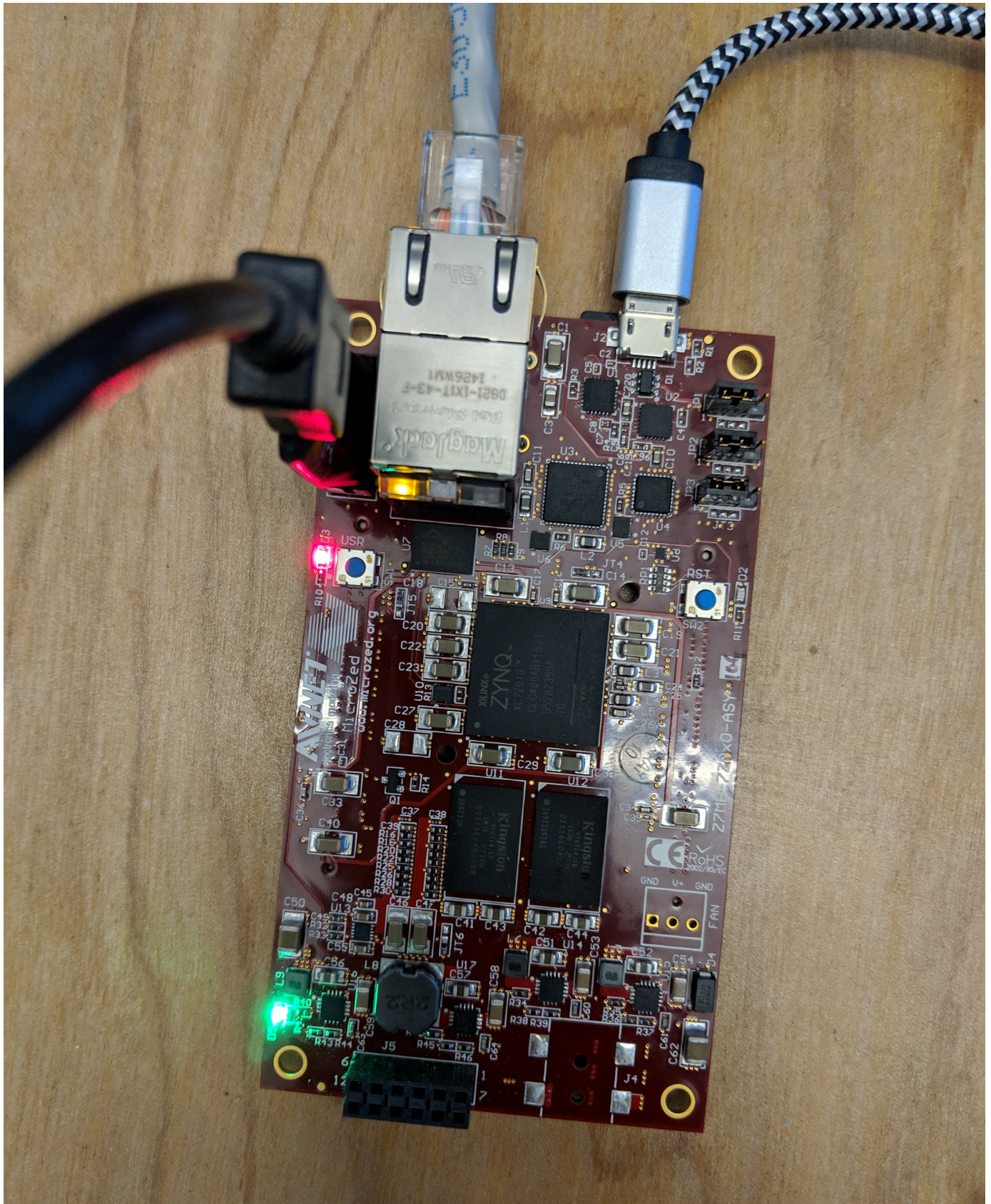


2. Masukkan kartu microSD Anda ke dalam slot kartu microSD yang terletak tepat di bawah port USB-UART.

Note

Sebelum Anda men-debug, pastikan untuk mencadangkan konten apa pun yang Anda miliki di kartu microSD.

Dewan Anda akan terlihat serupa dengan yang berikut ini:



3. Di XSDK IDE, klik kanan `aws_demos`, pilih `Debug As`, dan kemudian pilih `1 Launch on System Hardware (System Debugger)`.
4. Ketika debugger berhenti di `breakpointmain()`, dari menu, pilih `Run`, dan kemudian pilih `Lanjutkan`.

Note

Pertama kali Anda menjalankan aplikasi, key pair sertifikat baru diimpor ke memori non-volatile. Untuk menjalankan berikutnya, Anda dapat berkomentar `vDevModeKeyProvisioning()` di `main.c` file sebelum Anda membangun kembali gambar dan `B00T.bin` file. Ini mencegah penyalinan sertifikat dan kunci penyimpanan pada setiap proses.

Anda dapat memilih untuk mem-boot MicroZed papan Anda dari kartu microSD atau dari QSPI flash untuk menjalankan proyek demo FreeRTOS. Untuk instruksi, lihat [Hasilkan gambar 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 menyiapkan MicroZed papan untuk menjalankan proyek demo FreeRTOS, lihat diagram di [Mengatur perangkat MicroZed keras](#). Pastikan bahwa Anda telah terkoneksi MicroZed ke komputer Anda.

Boot proyek FreeRTOS dari kartu microSD

Format kartu microSD yang disediakan dengan Kit IoT MicroZed Industri Xilinx.

1. Salin `B00T.bin` file ke kartu microSD.
2. Masukkan kartu ke dalam 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 kabelnya.

Boot proyek demo FreeRTOS dari flash QSPI

1. Atur jumper mode boot MicroZed papan Anda ke mode boot JTAG.



2. Verifikasi bahwa komputer Anda terkoneksi ke port USB-UART dan JTAG Access. Lampu LED Power Good hijau harus diterangi.
3. Di XSDK IDE, dari menu, pilih Xilinx, lalu 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, dan kemudian tentukan server Anda.

5. Di File Gambar, masukkan jalur direktori ke fileB00T.bin gambar Anda. Pilih Browse untuk menelusuri file sebagai gantinya.
6. Di Offset, masukkan **0x0**.
7. Di File FSBL, masukkan path direktori ke fsbl.e1f file Anda. 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 dalam 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 kabelnya.


Pemecahan Masalah

Jika Anda menemukan kesalahan build yang terkait dengan jalur yang salah, cobalah untuk membersihkan dan membangun kembali proyek, seperti yang dijelaskan dalam [Bangun proyek demo FreeRTOS](#).

Jika Anda menggunakan Windows, pastikan bahwa Anda menggunakan garis miring maju ketika Anda mengatur variabel substitusi string di Windows XSDK IDE.

Untuk informasi pemecahan masalah umum tentang Memulai FreeRTOS, lihat [Memulai masalah saat 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 ada berdasarkan repositori Amazon-freertos yang sekarang tidak digunakan lagi, lihat [Panduan Migrasi Repositori Github Amazon-freertos](#).

Setelah membuat, 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 cara melakukan tugas-tugas penting, berinteraksi dengan AWS IoT layanan, dan kemampuan khusus papan program (seperti modem seluler). Untuk informasi selengkapnya, lihat halaman [Kategori Perpustakaan Perpustakaan FreeRTOS Library Categories](#).

Situs web FreeRtos.org juga memiliki informasi mendalam tentang [FreeRTOS Kernel](#) serta konsep sistem operasi real-time yang mendasar. 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 melakukan pembaruan Over-the-air (OTA).

Pembaruan Over-the-air (OTA) memungkinkan Anda untuk menyebarkan pembaruan firmware ke satu atau lebih perangkat di armada Anda. Meskipun pembaruan OTA dirancang untuk memperbarui firmware perangkat, Anda dapat menggunakannya untuk mengirim file apa pun ke satu atau lebih perangkat yang terdaftar AWS IoT. Saat Anda mengirim pembaruan melalui udara, kami sarankan Anda menandatangani 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 dengan alat penandatanganan kode Anda sendiri.

Saat Anda membuat pembaruan OTA, [Layanan Manajer Pembaruan OTA](#) membuat [AWS IoT tugas](#) untuk memberi tahu perangkat Anda bahwa pembaruan tersedia. Aplikasi demo OTA berjalan di perangkat Anda dan membuat tugas FreeRTOS yang berlangganan topik notifikasi untuk AWS IoT 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, menginstal pembaruan firmware. Jika Anda tidak menggunakan aplikasi demo Pembaruan FreeRTOS OTA, Anda harus mengintegrasikan [AWS IoT Perpustakaan Over the Air \(OTA\)](#) ke dalam aplikasi Anda sendiri untuk mendapatkan kemampuan pembaruan firmware.

over-the-air Pembaruan FreeRTOS memungkinkan Anda untuk:

- Tanda tangani firmware secara digital sebelum penyebaran.
- Terapkan gambar firmware baru ke satu perangkat, sekelompok perangkat, atau seluruh armada Anda.

- Terapkan firmware ke perangkat saat ditambahkan ke grup, disetel ulang, atau direprovisi.
- Verifikasi keaslian dan integritas firmware baru setelah digunakan ke perangkat.
- Memantau kemajuan penyebaran.
- Debug penyebaran gagal.

Menandai sumber daya OTA

Untuk membantu Anda mengelola sumber daya OTA Anda, Anda dapat secara opsional menetapkan metadata Anda sendiri untuk pembaruan dan streaming dalam bentuk tag. Tag memungkinkan Anda untuk mengkategorikan AWS IoT sumber daya Anda dengan cara yang berbeda (misalnya, dengan 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](#) atau [Prasyarat untuk pembaruan OTA menggunakan MQTT](#).
- [Buat bucket Amazon S3 untuk menyimpan pembaruan](#).
- [Membuat peran layanan Pembaruan OTA](#).
- [Membuat kebijakan pengguna OTA](#).
- [Membuat 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

File pembaruan OTA disimpan di bucket Amazon S3.

Jika Anda menggunakan Penandatanganan Kode AWS IoT, perintah yang Anda gunakan untuk membuat pekerjaan penandatanganan kode akan mengambil 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 GUID 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 Create bucket (Buat bucket).
3. Masukkan nama bucket.
4. Di bawah Pengaturan Bucket untuk Blokir Akses Publik tetap Blokir semua akses publik yang dipilih untuk menerima izin default.
5. Di bawah Pembuatan Versi Bucket, pilih Aktifkan untuk menyimpan semua versi dalam bucket yang sama.
6. Pilih Create bucket (Buat bucket).

Untuk informasi selengkapnya tentang Amazon S3, lihat [Panduan Pengguna Amazon Simple Storage Service](#).

Membuat 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 Create role (Buat peran).
4. Di Pilih jenis entitas tepercaya, pilih Layanan AWS.
5. Pilih IoT dari daftar AWS layanan.
6. Di bawah Pilih kasus penggunaan Anda, pilih IoT.
7. Pilih Next: Permissions (Selanjutnya: Izin).
8. Pilih Selanjutnya: Tag.
9. Pilih Selanjutnya: 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 bingung, Anda harus mengikuti instruksi dalam [AWS IoT Core](#) panduan 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 Pasang kebijakan.
3. Di kotak Pencarian, masukkan "AmazonFreeertosotUpdate", pilih AmazonFreeRtosotUpdate dari 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, lalu pilih Buat kebijakan.

Note

Prosedur berikut tidak diperlukan jika nama bucket Amazon S3 Anda dimulai dengan "afr-ota". Jika ya, kebijakan yang AWS dikelola 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 bucket Anda.

5. Pilih Tinjau kebijakan.
6. Masukkan nama untuk kebijakan, lalu pilih Buat kebijakan.

Membuat 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.
- Sertifikat akses yang disimpan diAWS Certificate Manager.
- Akses fitur pengiriman file AWS IoT berbasis MQTT.
- Akses pembaruan FreeRTOS OTA.
- Akses AWS IoT pekerjaan.
- Akses IAM.
- Penandatanganan Kode Akses untukAWS IoT. Lihat [Berikan akses ke penandatanganan kode untuk AWS IoT](#).
- Daftar platform perangkat keras FreeRTOS.
- Tag dan AWS IoT sumber daya untag.

Untuk memberikan izin yang diperlukan kepada pengguna Anda, lihat Kebijakan [IAM](#). Lihat juga [Mengizinkan pengguna dan layanan cloud untuk menggunakan AWS IoT](#) Lowongan Kerja.

Untuk menyediakan akses, tambahkan izin ke pengguna, grup, atau peran Anda:

- Pengguna dan grup diAWS IAM Identity Center:

Buat set izin. Ikuti petunjuk di [Buat set izin](#) di Panduan AWS IAM Identity Center Pengguna.

- Pengguna yang dikelola dalam IAM melalui penyedia identitas:

Buat peran untuk federasi identitas. Ikuti petunjuk dalam [Membuat peran untuk penyedia identitas pihak ketiga \(federasi\)](#) di Panduan Pengguna IAM.

- Pengguna IAM:

- Buat peran yang dapat diasumsikan pengguna Anda. Ikuti petunjuk dalam [Membuat peran untuk pengguna IAM di Panduan Pengguna IAM](#).
- (Tidak disarankan) Lampirkan kebijakan langsung ke pengguna atau tambahkan pengguna ke grup pengguna. Ikuti petunjuk dalam [Menambahkan izin ke pengguna \(konsol\)](#) di Panduan Pengguna IAM.

Membuat 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 CC3220SF-LAUNCHXL](#)
- [Membuat sertifikat penandatanganan kode untuk Espressif ESP32](#)
- [Membuat sertifikat penandatanganan kode untuk Nordic nrf52840-dk](#)
- [Membuat sertifikat penandatanganan kode untuk simulator FreeRTOS Windows](#)
- [Membuat sertifikat penandatanganan kode untuk perangkat keras khusus](#)

Membuat sertifikat penandatanganan kode untuk Texas Instruments CC3220SF-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](#)

SimpleLinkWi-Fi CC3220SF Wireless Microcontroller Launchpad Development Kit 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 ke dalam. AWS Certificate Manager Untuk informasi selengkapnya, lihat [Menginstal AWS CLI](#) di Panduan AWS Command Line Interface Pengguna.


Unduh dan instal versi terbaru [SimpleLinkCC3220](#) SDK. Secara default, file yang Anda butuhkan terletak 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 CC3220 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 komputer](#) Anda. Setelah Anda menginstal OpenSSL, pastikan bahwa `openssl` ditugaskan ke OpenSSL executable di command prompt atau lingkungan terminal Anda.

Membuat sertifikat penandatanganan kode yang ditandatangani sendiri

1. Buka prompt perintah 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 tsigner.key -out tsigner.csr
```

4. Konversi kunci pribadi CA akar taman bermain Texas Instruments dari format DER ke format PEM.

Kunci pribadi TI playground root CA 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. Konversi sertifikat CA akar taman bermain Texas Instruments dari format DER ke format PEM.

Sertifikat akar 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 Texas Instruments root CA:

```
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 Penandatanganan Kode untuk 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 tidak digunakan lagi, lihat. [Panduan Migrasi Repositori Github Amazon-freertos](#)

Papan Espressif ESP32 mendukung SHA-256 yang ditandatangani sendiri dengan sertifikat penandatanganan kode ECDSA.

Note

Untuk membuat sertifikat penandatanganan kode, instal [OpenSSL di komputer](#) Anda. Setelah Anda menginstal OpenSSL, pastikan bahwa `openssl` ditugaskan 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 ke dalam. 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 privat 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 Penandatanganan Kode untuk 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 tidak digunakan lagi, lihat [Panduan Migrasi Repositori Github Amazon-freertos](#)

Nordic nrf52840-dk mendukung SHA256 yang ditandatangani sendiri dengan sertifikat penandatanganan kode ECDSA.

Note

Untuk membuat sertifikat penandatanganan kode, instal [OpenSSL di komputer](#) Anda. Setelah Anda menginstal OpenSSL, pastikan bahwa `openssl` ditugaskan 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 ke dalam 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 privat 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 Penandatanganan Kode untuk AWS IoT direkomendasikan, Anda dapat menandatangani gambar firmware Anda secara manual.

Membuat sertifikat penandatanganan kode untuk simulator FreeRTOS Windows

Simulator FreeRTOS Windows 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 komputer Anda. Setelah Anda menginstal OpenSSL, pastikan bahwa `openssl` ditugaskan 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 ke dalam. 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 privat 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 Penandatanganan Kode untuk 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 ke dalam AWS Certificate Manager Untuk informasi tentang menginstal AWS CLI, lihat [Menginstal AWS CLI](#).

Setelah membuat sertifikat penandatanganan kode, Anda dapat menggunakan 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 memerlukan sertifikat untuk menggunakan algoritme dan ukuran kunci tertentu. Untuk informasi selengkapnya, lihat [Prasyarat](#) untuk Mengimpor Sertifikat. Untuk informasi selengkapnya tentang ACM, lihat [Mengimpor Sertifikat ke](#). AWS Certificate Manager

Anda harus menyalin, menempelkan, dan memformat konten sertifikat penandatanganan kode 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 menyediakan akses, tambahkan izin ke pengguna, grup, atau peran Anda:

- Pengguna dan grup di AWS IAM Identity Center:

Buat set izin. Ikuti petunjuk di [Buat set izin](#) di Panduan AWS IAM Identity Center Pengguna.

- Pengguna yang dikelola dalam IAM melalui penyedia identitas:

Buat peran untuk federasi identitas. Ikuti petunjuk dalam [Membuat peran untuk penyedia identitas pihak ketiga \(federasi\)](#) di Panduan Pengguna IAM.

- Pengguna IAM:

- Buat peran yang dapat diasumsikan pengguna Anda. Ikuti petunjuk dalam [Membuat peran untuk pengguna IAM di Panduan Pengguna](#) IAM.

- (Tidak disarankan) Lampirkan kebijakan langsung ke pengguna atau tambahkan pengguna ke grup pengguna. Ikuti petunjuk dalam [Menambahkan izin ke pengguna \(konsol\)](#) di 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 mengatur dan menjalankan aplikasi demo OTA, lihat [Over-the-air update aplikasi demo](#).

⚠ Important

- Dalam topik ini, jalur ke direktori unduhan FreeRTOS disebut sebagai *freertos*.
- Karakter spasi di *freertos* jalur dapat menyebabkan kegagalan build. Saat Anda mengkloning atau menyalin repositori, pastikan jalur yang Anda buat tidak mengandung karakter spasi.
- Panjang maksimum jalur file pada Microsoft Windows adalah 260 karakter. Jalur direktori unduhan FreeRTOS yang panjang dapat menyebabkan kegagalan build.
- Karena kode sumber mungkin berisi tautan simbolik, 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 [Symlink di Windows 10!](#).

Jika Anda menggunakan Git di bawah Windows, Anda harus mengaktifkan Mode Pengembang atau Anda harus:

- Atur `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`, `git 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 bila memungkinkan.

Konfigurasi

Dimulai dengan versi 201912.00, FreeRTOS OTA dapat menggunakan protokol HTTP atau MQTT untuk mentransfer gambar pembaruan firmware dari perangkat. AWS IoT Jika Anda menentukan kedua protokol saat membuat pembaruan OTA di FreeRTOS, setiap perangkat akan menentukan protokol yang digunakan untuk mentransfer gambar. Lihat [Prasyarat untuk pembaruan OTA menggunakan HTTP](#) untuk informasi selengkapnya.

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 diperlukan memori tambahan untuk koneksi MQTT karena digunakan bersama antara kontrol dan operasi data.

Kebijakan perangkat

Setiap perangkat yang menerima pembaruan OTA menggunakan MQTT harus terdaftar sebagai sesuatu AWS IoT dan hal tersebut harus memiliki kebijakan terlampir seperti yang tercantum di sini. Anda dapat menemukan informasi lebih lanjut tentang item di "Action" dan "Resource" objek di [Tindakan Kebijakan AWS IoT Inti](#) dan [Sumber Daya Tindakan AWS IoT Inti](#).

```
{
  "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 ke AWS IoT melalui MQTT.
- `iot:Subscribe` dan `iot:Publish` izin pada topik AWS IoT pekerjaan (`.../jobs/*`) memungkinkan perangkat yang terhubung untuk menerima pemberitahuan pekerjaan dan dokumen pekerjaan, dan untuk mempublikasikan status penyelesaian eksekusi pekerjaan.
- `iot:Subscribe` dan `iot:Publish` izin pada 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 pada 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 HTTP untuk melakukan pembaruan over-the-air (OTA). Dimulai dengan versi 201912.00, FreeRTOS OTA dapat menggunakan protokol HTTP atau MQTT untuk mentransfer gambar pembaruan firmware dari 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.
- Ketika Anda menentukan protokol MQTT dan HTTP untuk pekerjaan pembaruan OTA, pengaturan 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 (CoreMQTT Agent, HTTP, OTA Agent, 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 pembaruan OTA, Anda dapat menentukan kedua protokol sehingga protokol MQTT atau HTTP dapat digunakan., Anda dapat mengatur protokol utama yang digunakan oleh perangkat ke HTTP dengan mengubahnya. `configOTA_PRIMARY_DATA_PROTOCOL` ke `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 saat Anda mengaktifkan HTTP sebagai protokol data OTA. Dalam [vendors/espessif/boards/esp32/aws_demos/config_files/aws_iot_network_config.hfile](#), ubah `configENABLED_NETWORKS` menjadi `AWSIOT_NETWORK_TYPE_WIFI` hanya.

```
/**
 * @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 diperlukan memori heap tambahan untuk koneksi MQTT karena digunakan bersama antara kontrol dan operasi data. Namun, mengaktifkan data melalui HTTP memerlukan memori heap tambahan. Berikut ini adalah data penggunaan memori heap untuk semua platform yang didukung, dihitung menggunakan FreeRTOS `APIxPortGetFreeHeapSize`. Anda harus memastikan ada cukup RAM untuk menggunakan perpustakaan OTA.

Texas Instruments CC3220SF-LAUNCHXL

Operasi kontrol (MQTT): 12 KB

Operasi data (HTTP): 10 KB

Note

TI menggunakan RAM secara signifikan lebih sedikit karena SSL pada perangkat keras, sehingga tidak menggunakan perpustakaan mbedtls.

Keingintahuan Microchip PIC32MZEF

Operasi kontrol (MQTT): 65 KB

Operasi data (HTTP): 43 KB

Espressif

Operasi kontrol (MQTT): 65 KB

Operasi data (HTTP): 45 KB

Note

BLE pada ESP32 membutuhkan waktu sekitar 87 KB RAM. Tidak ada cukup RAM untuk mengaktifkan semuanya, yang disebutkan dalam konfigurasi khusus perangkat di atas.

Simulator jendela

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 itu harus memiliki kebijakan terlampir seperti yang tercantum di sini. Anda dapat menemukan informasi lebih lanjut tentang item di "Action" dan "Resource" objek di [Tindakan Kebijakan AWS IoT Inti](#) dan [Sumber Daya Tindakan AWS IoT Inti](#).

```
{
  "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 ke AWS IoT melalui MQTT.
- `iot:Subscribe` dan `iot:Publish` izin pada topik AWS IoT pekerjaan (`.../jobs/*`) memungkinkan perangkat yang terhubung untuk menerima pemberitahuan pekerjaan dan dokumen pekerjaan, dan untuk mempublikasikan status penyelesaian eksekusi pekerjaan.

- `iot:Receivelzin` memungkinkan AWS IoT Core untuk mempublikasikan pesan pada 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 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 Partner [FreeRTOS](#).

Untuk membuat pembaruan OTA

1. Terapkan versi awal firmware Anda ke satu atau beberapa perangkat.
2. Verifikasi bahwa firmware berjalan dengan benar.
3. Saat pembaruan firmware diperlukan, buat perubahan kode dan buat gambar baru.
4. Jika Anda menandatangani firmware secara manual, tandatangi lalu unggah gambar firmware yang ditandatangani ke bucket Amazon S3 Anda. Jika Anda menggunakan Penandatanganan Kode untuk AWS IoT, unggah gambar 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 tentukan 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 CC3220SF-LAUNCHXL](#)
- [Instal versi awal firmware pada Espressif ESP32](#)
- [Instal versi awal firmware pada Nordic NRF52840 DK](#)
- [Firmware awal pada simulator Windows](#)
- [Instal versi awal firmware pada papan khusus](#)



Instal versi awal firmware pada Texas Instruments CC3220SF-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](#)

Langkah-langkah ini ditulis dengan asumsi bahwa Anda telah membangun `aws_demos` proyek, seperti yang dijelaskan dalam. [Unduh, buat, flash, dan jalankan demo FreeRTOS OTA di Texas Instruments CC3220SF-LAUNCHXL](#)

1. Pada Texas Instruments CC3220SF-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 Mulai Pembuat Gambar.
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. Lepaskan emulator terminal Anda.
7. Di sisi kanan jendela aplikasi Uniflash, pilih Connect.
8. Di bawah Lanjutan, File, pilih File Pengguna.
9. Di panel pemilih File, pilih ikon  Add
File.
10. Jelajahi `/Applications/Ti/simplelink_cc32xx_sdk_<i>version</i>/tools/cc32xx_tools/certificate-playground` direktori, pilih `dummy-root-ca-cert`, pilih Buka, lalu pilih Tulis.
11. Di panel pemilih File, pilih ikon  Add
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 Jelajahi 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 `kemcuflashing.bin`.
 - b. Pilih kotak centang Penjual.
 - c. Di bawah File Token, 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_<i>version</i>/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 Terpercaya.
17. Kosongkan kotak centang Gunakan Katalog Root-Certificate Terpercaya default.

18. **Di bawah File Sumber, pilih Jelajahi, pilih simplelink_cc32xx_sdk_versi /tools/cc32xx_tools/certificate-playground/ 20160911.lst, lalu pilih Buka. certcatalogPlayGround**
19. **Di bawah File Sumber Tanda Tangan, pilih Jelajahi, pilih simplelink_cc32xx_sdk_versi /tools/cc32xx_tools/certificate-playground/ 20160911.lst.signed_3220.bin, lalu pilih Buka. certcatalogPlayGround**
20. Pilih  tombol untuk menyimpan proyek Anda.
21. Pilih  tombolnya.
22. Pilih Gambar Program (Buat dan Program).
23. Setelah proses pemrograman selesai, tempatkan 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 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 Pembaruan ESP-WROVER-KIT dan Over-the-Air.](#)

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`, komentari `#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED`, dan tentukan `CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED` atau `CONFIG_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 Ini harus diformat dengan cara berikut.

```
#define otapalconfigCODE_SIGNING_CERTIFICATE \
"-----BEGIN CERTIFICATE-----\n" \
"...base64 data...\n" \
"-----END CERTIFICATE-----\n";
```

3. Dengan demo Pembaruan OTA dipilih, ikuti langkah-langkah yang sama yang diuraikan dalam [Memulai dengan ESP32](#) untuk membangun dan mem-flash gambar. Jika sebelumnya Anda telah membangun dan mem-flash proyek, Anda mungkin perlu menjalankan `make clean` terlebih dahulu. Setelah Anda berlarimake `flash monitor`, Anda akan melihat sesuatu seperti berikut ini. Pemesanan beberapa pesan mungkin berbeda, karena aplikasi demo menjalankan beberapa 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 formatting: 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] vDHCPPProcess: 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] vDHCPPProcess: 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] [privSubscribeToJobNotificationTopics] OK: $aws/things/
<Your_Thing_Name>/jobs/$next/get/accepted
I (12963) ota_pal: privPAL_GetPlatformImageState
I (12963) esp_ota_ops: [0] aflags/seq:0x2/0x1, pflags/seq:0xffffffff/0x0
9 1285 [OTA Task] [privSubscribeToJobNotificationTopics] OK: $aws/things/
<Your_Thing_Name>/jobs/notify-next
10 1286 [OTA Task] [OTA_CheckForUpdate] Request #0
11 1289 [OTA Task] [privParseJSONbyModel] Extracted parameter [ clientToken:
  0:<Your_Thing_Name> ]
12 1289 [OTA Task] [privParseJSONbyModel] parameter not present: execution
13 1289 [OTA Task] [privParseJSONbyModel] parameter not present: jobId
14 1289 [OTA Task] [privParseJSONbyModel] parameter not present: jobDocument
15 1289 [OTA Task] [privParseJSONbyModel] parameter not present: afr_ota
16 1289 [OTA Task] [privParseJSONbyModel] parameter not present: streamname
17 1289 [OTA Task] [privParseJSONbyModel] parameter not present: files
18 1289 [OTA Task] [privParseJSONbyModel] parameter not present: filepath
19 1289 [OTA Task] [privParseJSONbyModel] parameter not present: filesize
20 1289 [OTA Task] [privParseJSONbyModel] parameter not present: fileid
21 1289 [OTA Task] [privParseJSONbyModel] parameter not present: certfile
22 1289 [OTA Task] [privParseJSONbyModel] parameter not present: sig-sha256-ecdsa
23 1289 [OTA Task] [privParseJobDoc] Ignoring job without ID.
24 1289 [OTA Task] [privOTA_Close] Context->0x3ffb4a8
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. Papan ESP32 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 GNU

Screen) untuk mendengarkan output serial papan. Ketahuilah bahwa menghubungkan ke port serial papan dapat menyebabkannya reboot.

Instal versi awal firmware pada 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 tidak digunakan lagi, lihat. [Panduan Migrasi Repositori Github Amazon-freertos](#)

Panduan ini ditulis dengan asumsi bahwa Anda telah melakukan langkah-langkah [Memulai dengan Nordic NRF52840-DK](#) dan Prasyarat Pembaruan [Over-the-Air](#). Sebelum Anda mencoba pembaruan OTA, Anda mungkin ingin menjalankan proyek demo MQTT yang dijelaskan dalam [Memulai dengan FreeRTOS](#) untuk memastikan bahwa papan dan toolchain 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 dipilih, ikuti langkah yang sama yang diuraikan [Memulai dengan Nordic NRF52840-DK](#) untuk membangun dan mem-flash gambar.

Anda akan melihat output yang serupa dengan yang berikut.

```
9 1285 [OTA Task] [privSubscribeToJobNotificationTopics] OK: $aws/things/your-thing-name/jobs/notify-next
10 1286 [OTA Task] [OTA_CheckForUpdate] Request #0
11 1289 [OTA Task] [privParseJSONbyModel] Extracted parameter [ clientToken: 0:your-thing-name ]
12 1289 [OTA Task] [privParseJSONbyModel] parameter not present: execution
13 1289 [OTA Task] [privParseJSONbyModel] parameter not present: jobId
14 1289 [OTA Task] [privParseJSONbyModel] parameter not present: jobDocument
15 1289 [OTA Task] [privParseJSONbyModel] 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, membangun aws_demos proyek, pastikan untuk menyertakan perpustakaan OTA. Untuk informasi selengkapnya tentang struktur kode sumber FreeRTOS, lihat [Demo Demo demo FreeRTOS](#).

Pastikan untuk menyertakan sertifikat penandatanganan kode, kunci pribadi, dan rantai kepercayaan sertifikat Anda baik dalam proyek FreeRTOS atau di perangkat Anda.

Menggunakan alat yang sesuai, membakar 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 `.rsu` yang merupakan jenis file berpemilik untuk Renesas. Alat ini dapat ditemukan di [Github](#). Anda dapat menggunakan contoh perintah 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 dalamnya [Buat bucket Amazon S3 untuk menyimpan pembaruan](#). Nama file yang perlu Anda salin ke Amazon S3 bergantung pada platform perangkat keras yang Anda gunakan:

- Texas Instrumen CC3220SF-LAUNCHXL: `vendors/ti/boards/cc3220_launchpad/aws_demos/ccs/debug/aws_demos.bin`
- Espressif ESP32: `vendors/espressif/boards/esp32/aws_demos/make/build/aws_demos.bin`

Membuat pembaruan OTA (AWS IoTkonsol)


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 sekelompok perangkat. Di bawah Perangkat untuk diperbarui, pilih satu atau beberapa hal atau grup benda dari menu tarik-turun.
6. Di bawah Pilih protokol untuk transfer file, pilih HTTP atau MQTT, atau pilih keduanya untuk memungkinkan setiap perangkat menentukan protokol yang akan digunakan.

7. Di bawah Masuk dan pilih file Anda, pilih Tandatangani 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 memenuhi syarat sepenuhnya 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

Di Texas Instruments CC3220SF-LAUNCHXL, jangan sertakan garis miring (/) terdepan 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 Jelajahi 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 PIC32MZEZ menghasilkan dua gambar biner dengan nama default dan `mp1lab.production.bin` `mp1lab.production.ota.bin` Gunakan file kedua saat Anda mengunggah gambar untuk pembaruan OTA.

11. Di bawah Pathname file pada perangkat, masukkan nama jalur yang memenuhi syarat ke lokasi di perangkat Anda di mana pekerjaan OTA akan menyalin gambar firmware. Lokasi ini bervariasi berdasarkan platform.

 Important

Pada Texas Instruments CC3220SF-LAUNCHXL, karena pembatasan keamanan, nama jalur gambar firmware harus `/sys/mcuflashing.bin`

12. Buka Jenis File dan masukkan nilai integer di kisaran 0-255. Jenis file yang Anda masukkan akan ditambahkan ke dokumen Job yang dikirimkan ke MCU. Pengembang firmware/software MCU memiliki kepemilikan penuh tentang apa yang harus dilakukan dengan nilai ini. Skenario yang mungkin termasuk MCU yang memiliki prosesor sekunder yang firmware-nya 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 [Membuat peran layanan Pembaruan OTA](#).
14. Pilih Selanjutnya.
15. Masukkan ID dan deskripsi untuk pekerjaan pembaruan OTA Anda.
16. Di bawah Jenis pekerjaan, pilih Pekerjaan Anda akan selesai setelah menerapkan ke perangkat/grup yang dipilih (snapshot).
17. Pilih konfigurasi opsional yang sesuai untuk pekerjaan Anda (Peluncuran eksekusi pekerjaan, Batalkan pekerjaan, batas waktu eksekusi pekerjaan, dan Tag).
18. Pilih Create (Buat).

Menggunakan image firmware yang ditandatangani sebelumnya

1. Di bawah Pilih dan tandatangi gambar firmware Anda, pilih Pilih gambar firmware yang telah ditandatangani sebelumnya.
2. Di bawah Pathname gambar firmware di perangkat, masukkan nama jalur yang memenuhi syarat ke lokasi di perangkat Anda di mana pekerjaan OTA akan menyalin gambar firmware. Lokasi ini bervariasi berdasarkan platform.
3. Di bawah Pekerjaan penandatanganan kode sebelumnya, pilih Pilih, lalu pilih tugas penandatanganan kode sebelumnya yang digunakan untuk menandatangani image firmware yang Anda gunakan untuk pembaruan OTA.

Menggunakan gambar firmware yang ditandatangani khusus

1. Di bawah Pilih dan tandatangi gambar firmware Anda, pilih Gunakan gambar firmware yang ditandatangani khusus.
2. Di bawah Nama jalur sertifikat penandatanganan kode di perangkat, masukkan nama jalur yang memenuhi syarat sepenuhnya 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 Pathname gambar firmware di perangkat, masukkan nama jalur yang memenuhi syarat ke lokasi di perangkat Anda di mana pekerjaan OTA akan menyalin gambar firmware. Lokasi ini bervariasi berdasarkan 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-nama.
- Alamat IP.
- Alamat email.
- Lokasi.
- Rincian bank.
- Informasi medis.

1. Di bawah Jenis pekerjaan, 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 mengunduh firmware yang diperbarui.

Setelah perangkat mengunduh firmware yang diperbarui, perangkat akan dimulai ulang dan kemudian menginstal firmware. Anda dapat melihat apa yang terjadi di terminal UART.

Untuk tutorial yang menunjukkan kepada Anda cara menggunakan konsol untuk membuat pembaruan OTA, lihat [Over-the-air update aplikasi demo](#).


```
--destination 's3={bucketName=your_destination_bucket}' \  
--profile-name your_profile_name
```

Note

your-source-bucket-name dan *your-destination-bucket-name* bisa menjadi ember Amazon S3 yang sama.

Ini adalah parameter untuk `put-signing-profile` dan `start-signing-job` perintah:

source

Menentukan lokasi firmware unsigned dalam ember 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 samping Versi, memilih Tampilkan.

destination

Tujuan pada perangkat tempat firmware yang ditandatangani di bucket S3 akan disalin. Format parameter ini sama dengan `source` parameternya.

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 `platformId` perangkat keras tempat Anda mendistribusikan pembaruan OTA.

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 gambar firmware yang ditandatangani adalah GUID. Anda memerlukan nama file ini saat 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, refresh browser Anda.

Perintah menampilkan ARN pekerjaan dan ID pekerjaan. Anda perlu nilai-nilai ini nanti. Untuk informasi selengkapnya tentang Penandatanganan KodeAWS 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 bucket Amazon S3 Anda.

Membuat aliran pembaruan firmware Anda

Aliran adalah antarmuka abstrak ke data yang dapat dikonsumsi oleh perangkat. Stream 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 gambar firmware Anda yang ditandatangani. File JSON harus berisi 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 gambar firmware Anda.

s3Location

Bucket dan kunci untuk firmware untuk streaming.

bucket

Bucket Amazon S3 tempat penyimpanan gambar 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 konten bucket Anda.

Jika Anda menggunakan Penandatanganan Kode untukAWS IoT, nama file adalah GUID yang dihasilkan oleh Penandatanganan Kode untukAWS IoT.

Gunakan create-stream AWS CLI perintah 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 sewenang-wenang 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 sewenang-wenang.

s3Location

Nama bucket tempat gambar firmware yang ditandatangani disimpan dan kunci (nama file) dari gambar firmware yang ditandatangani.

bucket

Bucket Amazon S3 tempat gambar firmware yang ditandatangani disimpan.

key

Kunci (nama file) dari gambar firmware yang ditandatangani.

Bila 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 `my-job-id` mana ID pekerjaan

ditampilkan oleh `create-signing-job` AWS CLI perintah. 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-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-jsonformat

```
{  
  "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-jsonbidang

Nama	Tipe	Deskripsi
otaUpdateId	string (maks: 128 mnt:1)	ID pembaruan OTA yang akan dibuat.
description	string (maks: 2028)	Penjabaran dari pembaruan OTA.
targets	Daftar	Perangkat yang ditargetkan untuk menerima pembaruan OTA.
protocols	Daftar	Protokol yang digunakan untuk mentransfer gambar pembaruan OTA. Nilai yang valid adalah [HTTP], [MQTT], [HTTP, MQTT]. Ketika kedua HTTP dan MQTT ditentuka

Nama	Tipe	Deskripsi
		n, perangkat target dapat memilih protokol.
targetSelection	string	<p>Menentukan apakah pembaruan akan terus berjalan (CONTINUOUS), atau akan selesai setelah semua hal yang ditentukan sebagai target telah menyelesaikan pembaruan (SNAPSHOT). Jika terus menerus, pembaruan juga dapat dijalankan pada suatu hal ketika perubahan terdeteksi dalam target. Misalnya, pembaruan akan berjalan pada suatu hal ketika benda ditambahkan ke grup target, bahkan setelah pembaruan diselesaikan oleh semua hal yang awalnya dalam grup. Nilai yang valid: BERKELANJUTAN SNAPSHOT.</p> <p>enum: TERUS MENERUS SNAPSHOT</p>
awsJobExecutionsRolloutConfig		Konfigurasi untuk peluncuran pembaruan OTA.
maximumPerMinute	bilangan bulat (maks: 1000 menit: 1)	Jumlah maksimum eksekusi pekerjaan pembaruan OTA dimulai per menit.

Nama	Tipe	Deskripsi
exponentialRate		Tingkat kenaikan untuk peluncuran pekerjaan. Parameter ini memungkinkan Anda untuk menentukan kenaikan tarif eksponensial untuk peluncuran pekerjaan.
baseRatePerMinute	bilangan bulat (maks: 1000 menit: 1)	Jumlah minimum hal-hal yang akan diberitahu tentang pekerjaan yang tertunda, per menit, pada awal peluncuran pekerjaan. Ini adalah tingkat awal peluncuran.
rateIncreaseCriteria		Kriteria untuk memulai peningkatan tingkat peluncuran untuk pekerjaan. AWS IoT mendukung hingga satu digit setelah desimal (misalnya, 1,5, tetapi tidak 1,55).
numberOfNotifiedThings	bilangan bulat (mnt:1)	Ketika jumlah hal ini telah diberitahukan, itu akan memulai peningkatan tingkat peluncuran.
numberOfSucceededThings	bilangan bulat (mnt:1)	Ketika jumlah hal ini telah berhasil dalam pelaksanaan pekerjaan mereka, itu akan memulai peningkatan tingkat peluncuran.

Nama	Tipe	Deskripsi
awsJobPresignedUrl Config		Informasi konfigurasi untuk URL yang telah ditandatangani sebelumnya.
expiresInSec	long	Berapa lama (dalam detik) URL yang telah ditandatangani sebelumnya valid. Nilai yang valid adalah 60 - 3600, nilai defaultnya adalah 1800 detik. URL pra-ditandatangani dihasilkan ketika permintaan untuk dokumen pekerjaan diterima.
awsJobAbortConfig		Kriteria yang menentukan kapan dan bagaimana penghentian pekerjaan terjadi.
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

Nama	Tipe	Deskripsi
<code>minNumberOfExecute dThings</code>	bilangan bulat (mnt:1)	Jumlah minimum hal yang harus menerima pemberitahuan eksekusi pekerjaan sebelum pekerjaan dapat dihentikan.
<code>awsJobTimeoutConfig</code>		Menentukan jumlah waktu setiap perangkat harus menyelesaikan pelaksanaannya dari pekerjaan. Sebuah timer dimulai ketika status eksekusi pekerjaan diatur ke <code>IN_PROGRESS</code> . Jika status eksekusi pekerjaan tidak diatur ke status terminal lain sebelum timer berakhir, maka akan secara otomatis diatur ke <code>TIMED_OUT</code>

Nama	Tipe	Deskripsi
inProgressTimeoutInMinutes	long	Menentukan jumlah waktu, dalam hitungan menit, perangkat ini harus menyelesaikan pelaksanaan pekerjaan ini. Interval batas waktu dapat di mana saja 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 tersebut. Setiap kali eksekusi pekerjaan tetap dalam status IN_PROGRESS lebih lama dari interval ini, eksekusi pekerjaan akan gagal dan beralih ke status terminalTIMED_OUT .
files	Daftar	File yang akan dialirkan oleh pembaruan OTA.
fileName	string	Nama file.
fileType	bilangan bulat 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 mnt:1)	ID stream.
fileId	bilangan bulat (maks: 255 menit: 0)	ID file yang terkait dengan aliran.
s3Location		Lokasi firmware yang diperbarui di S3.
bucket	string (mnt:1)	Ember S3.
key	string (mnt:1)	Kunci S3.
version	string	Versi bucket S3.
codeSigning		Metode penandatanganan kode file.
awsSignerJobId	string	ID AWSSignerJob yang dibuat untuk menandatangani file.
startSigningJobParameter		Menjelaskan pekerjaan penandatanganan kode.
signingProfileParameter		Menjelaskan profil penandatanganan kode.
certificateArn	string	Sertifikat ARN.

Nama	Tipe	Deskripsi
<code>platform</code>	string	Platform perangkat keras perangkat Anda.
<code>certificatePathOnDevice</code>	string	Lokasi sertifikat penandatanganan kode di perangkat Anda.
<code>signingProfileName</code>	string	Nama profil penandatanganan kode.
<code>destination</code>		Lokasi untuk menulis file yang ditandatangani kode.
<code>s3Destination</code>		Menjelaskan lokasi di S3 dari firmware yang diperbarui.
<code>bucket</code>	string (mnt:1)	Bucket S3 yang berisi firmware yang diperbarui.
<code>prefix</code>	string	Awalan S3.
<code>customCodeSigning</code>		Sebuah metode kustom untuk kode menandatangani file.
<code>signature</code>		Tanda tangan untuk file.
<code>inlineDocument</code>	blob	Representasi biner yang dikodekan base64 dari tanda tangan penandatanganan kode.
<code>certificateChain</code>		Rantai sertifikat.
<code>certificateName</code>	string	Nama sertifikat.

Nama	Tipe	Deskripsi
<code>inlineDocument</code>	string	Representasi biner yang dikodekan base64 dari rantai sertifikat penandatanganan kode.
<code>hashAlgorithm</code>	string	Algoritma hash yang digunakan untuk kode menandatangani file.
<code>signatureAlgorithm</code>	string	Algoritma tanda tangan yang digunakan untuk kode menandatangani file.
<code>attributes</code>	map	Daftar pasangan nama/atribut.
<code>roleArn</code>	string (maks: 2048 mnt:20)	Peran IAM yang memberikan akses ke Amazon S3, AWS IoT pekerjaan, dan sumber daya Penandatangan AWS Kode untuk membuat pekerjaan pembaruan OTA.
<code>additionalParameters</code>	map	Daftar parameter pembaruan OTA tambahan yang merupakan pasangan nama-nilai.
<code>tags</code>	Daftar	Metadata yang dapat digunakan untuk mengelola pembaruan.
<code>Key</code>	string (maks: 128 mnt:1)	Kunci tag.

Nama	Tipe	Deskripsi
Value	string (maks: 256 mnt:1)	Nilai tag.

Output

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

AWS CLI bidang output

Nama	Tipe	Deskripsi
otaUpdateId	string (maks: 128 mnt: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 dari file JSON dilewatkan ke create-ota-update perintah yang menggunakan Kode Penandatanganan untuk 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 kustom.

```
[
  {
    "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-nilai yang dikembalikan untuk `otaUpdateStatus` meliputi yang berikut:

CREATE_PENDING

Pembuatan pembaruan OTA tertunda.

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 [Jelaskan Eksekusi Pekerjaan](#).

Menghapus data terkait OTA

Saat ini, Anda tidak dapat menggunakan AWS IoT konsol untuk menghapus aliran atau pembaruan OTA. Anda dapat menggunakan 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 potongan-potongan 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-update` Perintah menghapus entri dalam 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 diatur dan pekerjaan dalam `In_Progress` keadaan, pekerjaan tidak dihapus.

Menghapus pekerjaan IoT yang dibuat untuk pembaruan OTA

FreeRTOS membuat AWS IoT pekerjaan saat Anda membuat 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 menetapkan bahwa hanya pekerjaan yang berada dalam keadaan terminal (`COMPLETED` atau `CANCELLED`) dapat dihapus. Anda dapat menghapus pekerjaan yang berada dalam keadaan non-terminal dengan melewati `force` parameter. Untuk informasi selengkapnya, lihat [DeleteJobAPI](#).

Note

Menghapus tugas dengan status `IN_PROGRESS` mengganggu eksekusi pekerjaan apa pun yang `IN_PROGRESS` di perangkat Anda dan dapat mengakibatkan perangkat dibiarkan dalam keadaan 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 dan faktor lainnya, diperlukan beberapa menit untuk menghapus pekerjaan. Sementara 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 `delete-job` AWS CLI perintah, Anda dapat melewati `--force` parameter `delete-job-execution` untuk memaksa penghapusan eksekusi pekerjaan. Untuk informasi selengkapnya, lihat [DeleteJobExecutionAPI](#).

Note

Menghapus eksekusi pekerjaan dengan status `IN_PROGRESS` mengganggu eksekusi pekerjaan apa pun yang `IN_PROGRESS` di perangkat Anda dan dapat mengakibatkan perangkat dibiarkan dalam keadaan nondeterministik. Pastikan bahwa setiap perangkat yang menjalankan pekerjaan yang telah dihapus dapat pulih ke keadaan yang diketahui.

Untuk informasi lebih lanjut tentang menggunakan aplikasi demo pembaruan OTA, lihat [Over-the-air update aplikasi demo](#).

Layanan Manajer Pembaruan OTA

Layanan Manajer Pembaruan 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.
- Cantumkan 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](#) pada 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 yang akan dikirim ke perangkat target.
- Peran IAM yang memberikan AWS IoT akses ke Amazon S3, AWS IoT pekerjaan, dan sumber daya Penandatanganan AWS Kode untuk membuat pekerjaan pembaruan OTA.
- Daftar opsional pasangan nama-nilai yang ditetapkan pengguna.

Pembaruan OTA dirancang untuk memperbarui firmware perangkat, tetapi Anda dapat menggunakannya untuk mengirim file apa pun yang Anda inginkan ke satu atau lebih perangkat yang terdaftar AWS IoT. Saat Anda mengirim pembaruan firmware melalui udara, kami sarankan Anda menandatangani secara digital sehingga perangkat yang menerimanya dapat memverifikasi bahwa perangkat tersebut 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 dengan [Penandatanganan Kode untuk FreeRTOS](#) atau Anda dapat menggunakan alat penandatanganan kode Anda sendiri.

Untuk kontrol lebih lanjut atas proses, Anda dapat menggunakan [CreateStream](#) API 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 OTA Manager 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 callback khusus untuk menanggapi pesan acara Agen OTA. Selama inisialisasi OS, MQTT, HTTP (jika HTTP digunakan untuk mengunduh file) dan antarmuka implementasi spesifik 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 hal-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 dan fungsi penyambungan kembali) 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, kita mengkonfigurasi demo untuk menggunakan callback aplikasi OTA default dan mengembalikan beberapa statistik sekali per detik. Singkatnya, kami meninggalkan beberapa detail dari demo ini.

Demo OTA juga menunjukkan manajemen koneksi MQTT dengan memantau callback putus dan membangun kembali koneksi. Ketika pemutusan terjadi, demo pertama-tama menangguhkan operasi Agen OTA dan kemudian mencoba untuk membangun kembali koneksi MQTT. Upaya rekoneksi MQTT ditunda oleh waktu yang secara eksponensial meningkat 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 `demos/ota` direktori.

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.
- Hubungkan ke AWS IoT titik akhir Anda.
- Inisialisasi Agen OTA.
- Loop yang memungkinkan pekerjaan pembaruan OTA dan statistik output sekali dalam satu detik.
- Jika MQTT terputus, hentikan operasi Agen OTA.
- Coba hubungkan lagi dengan penundaan eksponensial dan jitter.
- Jika terhubung kembali, lanjutkan operasi Agen OTA.
- Jika Agen berhenti, tunda satu detik, lalu coba sambungkan kembali.

Menggunakan callback aplikasi untuk acara Agen OTA

Contoh sebelumnya digunakan `privOtaAppCallback` sebagai handler callback untuk peristiwa Agen OTA. (Lihat parameter keempat untuk panggilan `OTA_Init` API). Jika Anda ingin menerapkan penanganan kustom 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 acara 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;
```

Agan OTA dapat menerima pembaruan di latar belakang selama pemrosesan aktif aplikasi utama. Tujuan penyampaian acara ini adalah untuk memungkinkan aplikasi untuk memutuskan apakah tindakan dapat diambil segera atau apakah harus ditangguhkan sampai setelah selesai beberapa

pemrosesan khusus aplikasi lainnya. Ini mencegah gangguan perangkat Anda yang tidak terduga selama pemrosesan aktif (misalnya, penyedotan debu) yang akan disebabkan oleh pengaturan ulang setelah pembaruan firmware. Ini adalah event pekerjaan yang diterima oleh handler callback:

OtaJobEventActivate

Saat pengendali callback menerima peristiwa ini, Anda dapat segera mengatur ulang perangkat atau menjadwalkan panggilan untuk mengatur ulang perangkat nanti. Hal ini memungkinkan Anda untuk menunda reset perangkat dan fase self-test, jika perlu.

OtaJobEventFail

Ketika handler callback menerima acara ini, pemutakhiran telah gagal. Anda tidak perlu melakukan apapun 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 itu 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 callback 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 melakukan firmware sebagai gambar permanen baru dengan memanggil fungsi. `OTA_SetImageState(OtaImageStateAccepted)`

OtaJobEventProcessed

Acara OTA yang diantrian oleh `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 kembali ke gambar sebelumnya.

OtaJobEventUpdateComplete

Acara notifikasi 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](#).

Agan OTA, yang berjalan di perangkat Anda, melakukan pemeriksaan integritas pada firmware saat tiba di perangkat.

Keamanan operator

Setiap panggilan API yang dilakukan melalui control plane API mengalami otentikasi dan otorisasi IAM Signature Version 4 standar. Untuk membuat penyebaran, Anda harus memiliki izin untuk memanggil `CreateDeployment`, `CreateJob`, dan `API.CreateStream`. Selain itu, dalam kebijakan bucket Amazon S3 atau ACL, Anda harus memberikan izin baca kepada prinsipal 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.

Kode — sertifikat penandatanganan menggunakan X.509 versi 3 dan ekstensi. Key Usage Extended Key Usage Key UsageEkstensi diatur ke Digital Signature dan Extended Key Usage ekstensi diatur keCode Signing. Untuk informasi selengkapnya tentang penandatanganan image kode Anda, lihat [Panduan Penandatanganan Kode untuk AWS IoT Pengembang](#) 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

- [Mengatur CloudWatch Log untuk pembaruan OTA](#)
- [Log panggilan API AWS IoT OTA dengan AWS CloudTrail](#)
- [Dapatkan rincian kegagalan createOtaUpdate menggunakan AWS CLI](#)
- [Dapatkan kode kegagalan OTA dengan AWS CLI](#)
- [Memecahkan masalah pembaruan OTA dari beberapa perangkat](#)
- [Memecahkan masalah pembaruan OTA dengan Launchpad Texas Instruments CC3220SF](#)

Mengatur CloudWatch Log untuk pembaruan OTA

Layanan Pembaruan OTA mendukung pencatatan dengan AmazonCloudWatch. 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 mengkonfigurasi pencatatan 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 informasi, lihat [Autentikasi dan Kontrol Akses untuk CloudWatch Log Amazon](#).

Buat peran logging dan aktifkan logging

Gunakan [AWS IoTkonsol](#) untuk membuat peran logging dan mengaktifkan logging.

1. Dari panel navigasi, pilih Pengaturan.
2. Di bawah Log, pilih Edit.
3. Di bawah Level verbositas, pilih Debug.
4. Di bawah Tetapkan peran, pilih Buat baru untuk membuat peran IAM untuk dicatat.
5. Di bawah Nama, masukkan nama unik untuk peran Anda. Peran Anda akan dibuat dengan semua izin yang diperlukan.
6. Pilih Update (Perbarui).

Log pembaruan OTA

Layanan Pembaruan OTA menerbitkan log ke akun Anda ketika salah satu dari 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 [CloudWatchkonsol](#).

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 di mana log dihasilkan.

`ActionType`

Tindakan yang dihasilkan 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`: 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 dihasilkan log.

`clientId`

ID klien MQTT yang membuat permintaan yang dihasilkan log.

`ClientToken`

Token klien yang terkait dengan permintaan yang dihasilkan log.

`detail`

Informasi lebih lanjut tentang operasi yang dihasilkan log.

`LogLevel`

Tingkat logging log. Untuk log pembaruan OTA, ini selalu diatur ke `DEBUG`.

`otaUpdateId`

ID pembaruan OTA yang menghasilkan log.

`protokol`

Protokol yang digunakan untuk membuat permintaan yang dihasilkan log.

status

Status operasi yang dihasilkan log. Nilai yang valid adalah:

- Berhasil
- Kegagalan

StreamID

ID AWS IoT aliran yang dihasilkan log.

timestamp

Waktu ketika log dihasilkan.

Nama Topik

Topik MQTT yang digunakan untuk membuat permintaan yang dihasilkan 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 saat 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 dengan CloudTrail, layanan yang menangkap panggilan API AWS IoT OTA dan mengirimkan file log ke bucket Amazon S3 yang Anda tentukan. CloudTrail menangkap panggilan API dari kode Anda ke API AWS IoT OTA. Menggunakan informasi yang dikumpulkan oleh CloudTrail, Anda dapat menentukan permintaan yang dibuat untuk AWS IoT OTA, alamat IP sumber dari mana permintaan dibuat, yang membuat permintaan, ketika dibuat, dan sebagainya.

Untuk informasi selengkapnya tentang 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 yang ditulis dengan catatan AWS layanan lainnya. CloudTrail menentukan kapan membuat dan menulis ke berkas baru berdasarkan periode waktu dan ukuran berkas.

Tindakan bidang kontrol AWS IoT OTA berikut dicatat oleh CloudTrail:

- [CreateStream](#)
- [DescribeStream](#)
- [ListStreams](#)
- [UpdateStream](#)
- [DeleteStream](#)
- [BuatOtaUpdate](#)
- [getOtaUpdate](#)
- [ListotaUpdates](#)
- [Deleteotaupdate](#)

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:

- Jika permintaan tersebut dibuat dengan kredensial pengguna IAM atau akar.
- Jika permintaan tersebut dibuat dengan kredensial keamanan sementara untuk peran atau pengguna federasi.
- Bahwa permintaan dibuat oleh layanan AWS 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 Pemberitahuan 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 berkas log FreeRTOS

CloudTrailfile log dapat berisi satu atau lebih entri log. Setiap entri berisi beberapa peristiwa yang diformat JSON. Entri log mewakili permintaan tunggal dari sumber manapun 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 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/mcuflashing.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 rincian kegagalan createOtaUpdate menggunakan AWS CLI

Jika proses pembuatan pekerjaan pembaruan OTA gagal, mungkin ada tindakan yang dapat Anda lakukan untuk memperbaiki masalah. 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, AWS IoT aliran, objek Amazon S3). Kesalahan yang dihadapi dapat menyebabkan proses gagal tanpa membuat AWS IoT pekerjaan. Di bagian pemecahan masalah ini kami memberikan instruksi tentang cara mengambil rincian kegagalan.

1. Instal dan konfigurasi [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 mana *ota_update_job_001* adalah 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 membuat 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 konfigurasi [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 mana *JobID* adalah string ID pekerjaan lengkap untuk pekerjaan yang statusnya ingin kita dapatkan (itu terkait dengan pekerjaan pembaruan OTA saat dibuat) dan *ThingName* merupakan nama AWS IoT hal 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 output ini, "" di "reasondetailsmap" memiliki dua bidang: bidang yang ditampilkan sebagai "0xEEEEEEEEEeeeeEEEEEE" berisi kode kesalahan generik dari Agen OTA; bidang yang ditampilkan sebagai "0xffffffff" berisi sub-kode. Kode kesalahan generik tercantum di 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 generik.

Memecahkan masalah pembaruan OTA dari beberapa perangkat

Untuk melakukan OTA pada beberapa perangkat (benda) menggunakan image firmware yang sama, terapkan fungsi (misalnya `getThingName()`) yang mengambil `clientcredentialIOT_THING_NAME` dari memori non-volatile. Pastikan bahwa fungsi ini membaca nama hal dari bagian memori non-volatile yang tidak ditimpa oleh OTA, dan bahwa nama hal disediakan sebelum menjalankan pekerjaan pertama. Jika Anda menggunakan alur JITP, Anda dapat membaca nama benda dari nama umum sertifikat perangkat Anda.

Memecahkan masalah pembaruan OTA dengan Launchpad Texas Instruments CC3220SF

Platform Launchpad CC3220SF 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 asynchronous event handler di `network_if.c`

Perpustakaan FreeRTOS

Library FreeRTOS menyediakan fungsionalitas tambahan ke kernel FreeRTOS dan pustaka internalnya. Anda dapat menggunakan pustaka FreeRTOS untuk jaringan dan keamanan dalam aplikasi tertanam. Library FreeRTOS juga memungkinkan aplikasi Anda berinteraksi dengan AWS IoT layanan. FreeRTOS menyertakan pustaka yang memungkinkan untuk:

- Sambungkan perangkat ke AWS IoT Cloud dengan aman menggunakan MQTT dan bayangan perangkat.
- Temukan dan sambungkan ke AWS IoT Greengrass core.
- Kelola koneksi Wi-Fi.
- Dengarkan dan proses [Pembaruan FreeRTOS Over-the-Air](#).

`libraries` Direktori berisi kode sumber pustaka FreeRTOS. Ada fungsi pembantu yang membantu dalam mengimplementasikan fungsionalitas perpustakaan. Kami tidak menyarankan Anda untuk mengubah fungsi pembantu ini.

Perpustakaan porting FreeRTOS

Pustaka porting berikut disertakan dalam konfigurasi FreeRTOS yang tersedia untuk diunduh di konsol FreeRTOS. Perpustakaan ini bergantung pada platform. Konten mereka berubah sesuai dengan platform perangkat keras Anda. Untuk informasi tentang mem-porting pustaka ini ke perangkat, lihat [Panduan Porting FreeRTOS](#).

Perpustakaan porting FreeRTOS

Perpustakaan	Referensi API	Deskripsi
Energi Rendah Bluetooth	Referensi API Energi Rendah Bluetooth	Menggunakan perpustakaan FreeRTOS Bluetooth Low Energy, mikrokontroler Anda dapat berkomunikasi dengan broker AWS IoT MQTT melalui perangkat gateway. Untuk informasi selengkapnya, lihat Perpustakaan Bluetooth Energi Rendah .
Pembaruan lewat udara	AWS IoT Referensi API over-the-air pembaruan O	Pustaka pembaruan FreeRTOS AWS IoT Over-the-air (OTA) memungkinkan Anda mengelola pemberitahuan pembaruan, mengunduh pembaruan, dan melakukan verifikasi kriptografi pembaruan firmware di perangkat FreeRTOS Anda. Untuk informasi selengkapnya, lihat AWS IoT Perpustakaan Over the Air (OTA) .
FreeRTOS	FreeRTOS+POSIX API Referensi	Anda dapat menggunakan pustaka FreeRTOS+POSIX untuk

Perpustakaan	Referensi API	Deskripsi
		<p>memindahkan aplikasi yang sesuai dengan POSIX ke ekosistem FreeRTOS.</p> <p>Untuk informasi selengkapnya, lihat FreeRtos+POSIX.</p>
Soket Aman	Referensi API Soket Aman	Untuk informasi selengkapnya, lihat Perpustakaan Secure Sockets .
FreeRTOS	Referensi API FreeRTOS+TCP	<p>FreeRTOS+TCP adalah scalable, open source dan thread aman TCP/IP stack untuk FreeRTOS.</p> <p>Untuk informasi selengkapnya, lihat FreeRTOS+TCP.</p>
Wi-Fi	Referensi API Wi-Fi	<p>Perpustakaan Wi-Fi FreeRTOS memungkinkan Anda untuk berinteraksi dengan tumpukan nirkabel tingkat rendah mikrokontroler Anda.</p> <p>Untuk informasi lain, lihat Perpustakaan Wi-Fi.</p>
CorePKCS11		<p>Pustaka CorePKCS11 adalah implementasi referensi dari Public Key Cryptography Standard #11, untuk mendukung penyediaan dan otentikasi klien TLS.</p> <p>Untuk informasi lain, lihat Pustaka CorePKCS11.</p>
TLS		Untuk informasi selengkapnya, lihat Keamanan Lapisan Pengangkutan .

Perpustakaan	Referensi API	Deskripsi
I/O yang umum	Referensi API I/O yang umum	Untuk informasi selengkapnya, lihat I/O .
Antarmuka Seluler	Referensi API Antarmuka Seluler	Perpustakaan Antarmuka Seluler memaparkan kemampuan beberapa modem seluler populer melalui API yang seragam. Untuk informasi lain, lihat perpustakaan Antarmuka seluler perpustakaan antarmuka seluler .

Perpustakaan aplikasi FreeRTOS

Anda dapat secara opsional menyertakan pustaka aplikasi mandiri berikut dalam konfigurasi FreeRTOS Anda untuk berinteraksi dengan AWS IoT layanan di cloud.

Note

Beberapa pustaka aplikasi memiliki API yang sama dengan pustaka di SDK AWS IoT Perangkat untuk Embedded C. Untuk pustaka ini, lihat [Referensi API AWS IoT Device SDK C](#). Untuk informasi selengkapnya tentang SDK AWS IoT Perangkat untuk Embedded C, lihat [AWS IoT SDK Perangkat untuk Embedded C](#).

Perpustakaan aplikasi FreeRTOS

Perpustakaan	Referensi API	Deskripsi
AWS IoT Device Defender	Referensi API SDK Defender C Perangkat	<p>AWS IoT Device Defender Library FreeRTOS menghubungkan perangkat FreeRTOS Anda ke AWS IoT Device Defender.</p> <p>Untuk informasi selengkapnya, lihat AWS IoT Device Defender perpustakaan.</p>

Perpustakaan	Referensi API	Deskripsi
AWS IoT Greengrass	Referensi Greengrass API	<p>AWS IoT GreengrassLibrary FreeRTOS menghubungkan perangkat FreeRTOS Anda keAWS IoT Greengrass.</p> <p>Untuk informasi selengkapnya, lihat AWS IoT GreengrassPustaka penemuan.</p>
MQTT	Referensi API Perpustakaan MQTT (v1.x.x) Referensi API Agen MQTT (v1) Referensi API SDK MQTT (v2.x) C	<p>Pustaka CoreMQTT menyediakan klien untuk perangkat FreeRTOS Anda untuk mempublikasikan dan berlangganan topik MQTT. MQTT adalah protokol yang digunakan perangkat untuk berinteraksiAWS IoT.</p> <p>Untuk informasi selengkapnya tentang pustaka CoreMQTT versi 3.0.0Perpustakaan CoreMQTT</p>

Perpustakaan	Referensi API	Deskripsi
Agen CoreMQTT	Referensi API Perpustakaan Agen CoreMQTT	<p>Pustaka CoreMQTT Agent adalah API tingkat tinggi yang menambahkan keamanan thread 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. Library menyediakan ekuivalen aman thread ke API CoreMQTT, sehingga dapat digunakan di lingkungan multi-threaded.</p> <p>Untuk informasi selengkapnya tentang pustaka CoreMQTT Agen lihat Perpustakaan Agen CoreMQTT.</p>
AWS IoT Device Shadow	Referensi API SDK Device Shadow C	<p>Pustaka AWS IoT Device Shadow memungkinkan perangkat FreeRTOS Anda berinteraksi dengan bayangan AWS IoT perangkat.</p> <p>Untuk informasi selengkapnya, lihat AWS IoT Pustaka Device Shadow.</p>

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 kompilator 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 include 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](#).

perpustakaan BackOffAlgorithm

Note

Konten di halaman ini mungkin tidak up-to-date. Silakan merujuk ke [Halaman perpustakaan Freertos.org](#) untuk update terbaru.

Pengantar

The [BackoffAlgoritma](#) pustaka adalah perpustakaan 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 jitter](#) algoritma

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.

Perpustakaan ini dapat digunakan secara bebas dan didistribusikan di bawah [Lisensi sumber terbuka 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

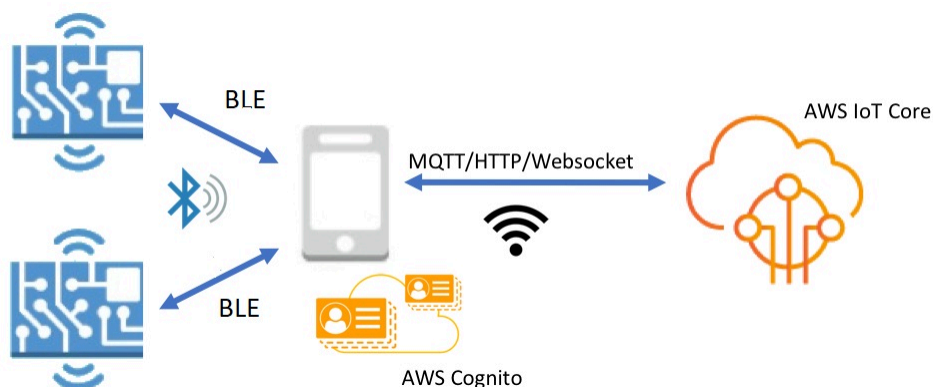
Perpustakaan Bluetooth Energi Rendah

⚠ 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 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 AWS IoT MQTT.

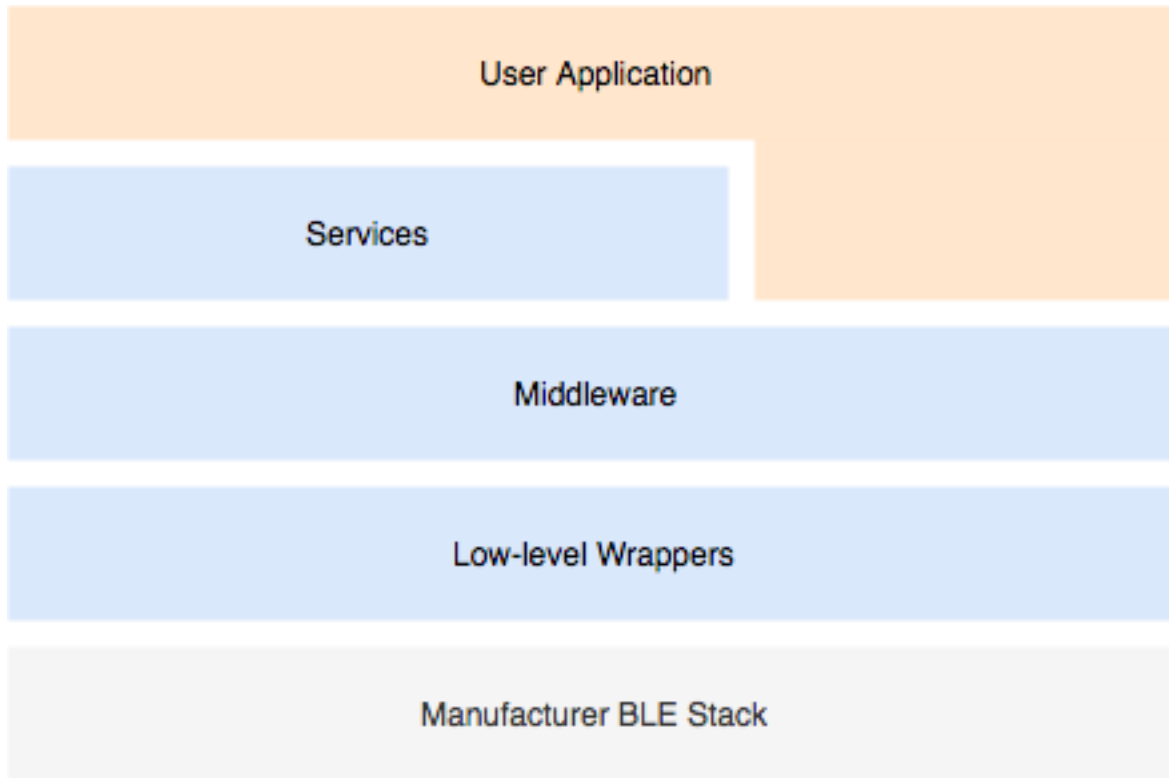


Menggunakan SDK Seluler untuk Perangkat Bluetooth FreeRTOS, Anda dapat menulis aplikasi seluler asli yang berkomunikasi dengan aplikasi yang disematkan pada mikrokontroler Anda melalui BLE. Untuk informasi selengkapnya tentang SDKs seluler, lihat [SDK Seluler untuk perangkat Bluetooth FreeRTOS](#).

Perpustakaan FreeRTOS BLE mencakup layanan untuk mengkonfigurasi jaringan Wi-Fi, mentransfer data dalam jumlah besar, dan menyediakan abstraksi jaringan melalui BLE. Perpustakaan FreeRTOS BLE juga mencakup middleware dan API tingkat 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 API middleware:

- Informasi perangkat
- Penyediaan Wi-Fi
- Abstraksi jaringan
- Transfer objek besar 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 (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 kredensi jaringan ke memori flash.
- Tetapkan prioritas jaringan.
- Hapus jaringan dan kredensi jaringan dari memori flash.

Abstraksi jaringan

Layanan abstraksi jaringan abstrak 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 transfer objek besar

Layanan Large Object Transfer 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 atas BLE

MQTT lebih BLE berisi profil GATT untuk membuat layanan proxy MQTT lebih 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 ke AWS 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 lebih profil BLE GATT](#).

Middleware

Middleware FreeRTOS Bluetooth Low Energy adalah abstraksi dari API tingkat rendah. API middleware membentuk antarmuka yang lebih ramah pengguna ke tumpukan Bluetooth Low Energy.

Menggunakan API middleware, Anda dapat mendaftarkan beberapa callback, di beberapa lapisan, ke satu peristiwa. Menginisialisasi middleware Bluetooth Low Energy juga menginisialisasi layanan dan mulai beriklan.

Langganan panggilan balik fleksibel berlangganan panggilan balik yang fleksibel

Misalkan perangkat keras Bluetooth Low Energy Anda terputus, dan layanan MQTT melalui Bluetooth Low Energy perlu mendeteksi pemutusan. 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

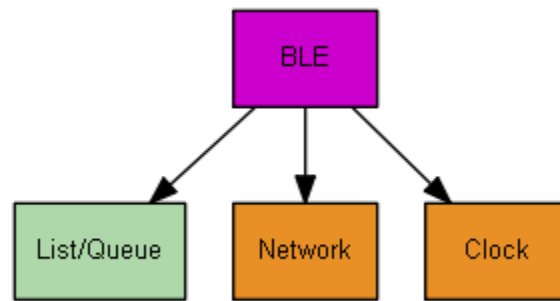
Pembungkus FreeRTOS Bluetooth Low Energy tingkat rendah adalah abstraksi dari tumpukan Bluetooth Low Energy pabrikan. Pembungkus tingkat rendah menawarkan serangkaian API umum untuk kontrol langsung atas perangkat keras. API tingkat rendah mengoptimalkan penggunaan RAM, tetapi fungsionalitas terbatas.

Gunakan API layanan Bluetooth Low Energy untuk berinteraksi dengan layanan Bluetooth Low Energy. API layanan menuntut lebih banyak sumber daya daripada API tingkat rendah.

Dependensi 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	Perpustakaan Wi-Fi

Untuk berkomunikasi dengan broker AWS IoT MQTT, Anda harus memiliki AWS akun dan Anda harus mendaftarkan perangkat Anda sebagai AWS IoT sesuatu. Untuk informasi selengkapnya tentang pengaturan, lihat [Panduan AWS IoT Pengembang](#).

FreeRTOS Bluetooth Low Energy menggunakan Amazon Cognito untuk autentikasi pengguna di perangkat seluler Anda. Untuk menggunakan layanan proxy MQTT, Anda harus membuat identitas Amazon Cognito dan kumpulan pengguna. Setiap Amazon Cognito Identity harus memiliki kebijakan yang sesuai yang melekat padanya. Untuk informasi selengkapnya, lihat [Panduan Developer Amazon Cognito](#).

File konfigurasi perpustakaan file konfigurasi perpustakaan

Aplikasi yang menggunakan layanan FreeRTOS MQTT melalui Bluetooth Low Energy harus menyediakan file `iot_ble_config.h` header, di mana parameter konfigurasi ditentukan. Parameter konfigurasi yang tidak terdefinisi mengambil nilai default yang ditentukan dalam `iot_ble_config_defaults.h`.

Beberapa parameter konfigurasi penting meliputi:

IOT_BLE_ADD_CUSTOM_SERVICES

Mengizinkan pengguna 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 [Bluetooth Low Energy API Referensi](#).

Pengoptimalan

Saat mengoptimalkan performa papan Anda, pertimbangkan hal berikut:

- API tingkat rendah menggunakan lebih sedikit RAM, tetapi menawarkan fungsionalitas terbatas.
- Anda dapat mengatur `IOT_BLE_CONFIG_MAX_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 pembatasan penggunaan

Secara default, perpustakaan FreeRTOS Bluetooth Low Energy menetapkan `BT_PROPERTY_SECURE_CONNECTION_ONLY` 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 dalam mode Secure Connections Only, mode keamanan LE tertinggi level 1, 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 level 1 level 4, perangkat harus memiliki kemampuan input dan output untuk perbandingan numerik.

Berikut adalah mode yang didukung, dan properti terkait mereka:

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 terotentikasi dengan enkripsi)

Mode ini tidak didukung.

Mode 1, Level 4 (Sambungan Aman LE yang Dotentikasi berpasangan 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 mengurus inisialisasi lapisan bawah tumpukan.

Middleware

Untuk menginisialisasi middleware

1. Inisialisasi driver perangkat keras Bluetooth Low Energy apa pun 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`.

API tingkat rendah

Jika Anda tidak ingin menggunakan layanan GATT FreeRTOS Bluetooth Low Energy, Anda dapat melewati middleware dan berinteraksi langsung dengan API tingkat rendah untuk menghemat sumber daya.

Untuk menginisialisasi API tingkat rendah

1. Inisialisasi driver perangkat keras Bluetooth Low Energy apa pun sebelum Anda memanggil API. Inisialisasi driver bukan bagian dari API tingkat rendah Bluetooth Low Energy.
2. Bluetooth Low Energy low-level API menyediakan panggilan aktifkan/nonaktifkan ke tumpukan Bluetooth Low Energy untuk mengoptimalkan daya dan sumber daya. Sebelum memanggil API, Anda harus mengaktifkan Bluetooth Low Energy.

```
const BTInterface_t * pXiface = BTGetBluetoothInterface();
xStatus = pXiface->pxEnable( 0 );
```

3. Manajer Bluetooth berisi API 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.pxBtGattServerInterface = ( BTGattServerInterface_t * )
xBTInterface.pxBTLeAdapterInterface->ppvGetGattServerInterface();
xBTInterface.pxBtGattServerInterface->pxGattServerInit( &xBTGattServerCb );
```

Setelah menginisialisasi adaptor Bluetooth Low Energy, Anda dapat menambahkan server GATT. Anda hanya dapat mendaftarkan satu server GATT pada 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 lengkap, lihat Referensi [Bluetooth Low Energy API](#).

Contoh penggunaan penggunaan penggunaan penggunaan:

Contoh di bawah ini menunjukkan cara menggunakan perpustakaan Bluetooth Low Energy untuk mengiklankan dan membuat layanan baru. Untuk aplikasi demo penuh FreeRTOS Bluetooth Low Energy, lihat [Aplikasi Demo Bluetooth Low Energy](#).

Periklanan

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 ke 1. Ini memicu IotBle_SetCustomAdvCb callback.

Penambahan layanan baru

Untuk contoh lengkap layanan, lihat [freertos/.../ble/services](#).

1. Buat UUID 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 `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` akan dipicu ketika `xCharCounterUUID_TYPE` diakses, dan `vWriteCommand` akan dipicu ketika `xCharControlUUID_TYPE` diakses, mendefinisikan 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,
    .pushHandlesBuffer = usHandlesBuffer,
    .pxBLEAttributes = (BTAttribute_t *)pxAttributeTable
};
```

6. Panggil `APIotBle_CreateService` dengan struktur yang Anda buat pada langkah sebelumnya. Middleware menyinkronkan pembuatan semua layanan, sehingga setiap layanan baru harus sudah didefinisikan ketika `IotBle_AddCustomServicesCb` callback dipicu.
 - a. Atur `IOT_BLE_ADD_CUSTOM_SERVICES` ke 1 dalam `vendors/vendor/boards/board/aws_demos/config_files/iot_ble_config.h`.
 - b. Buat `IotBle_AddCustomServicesCb` dalam aplikasi Anda:

```
void IotBle_AddCustomServicesCb(void)
{
    BTStatus_t xStatus;
    /* Select the handle buffer. */
    xStatus = IotBle_CreateService( (BTService_t *)&xGattDemoService,
    (IotBleAttributeEventCallback_t *)pxCallbackArray );
}
```

Porting

Perangkat input pengguna dan output periferal input pengguna dan output perangkat

Koneksi yang aman membutuhkan input dan output untuk perbandingan numerik.

eBLENumericComparisonCallbackAcara ini dapat didaftarkan menggunakan manajer acara:

```
xEventCb.pxNumericComparisonCb = &prvNumericComparisonCb;  
xStatus = BLE_RegisterEventCb( eBLENumericComparisonCallback, xEventCb );
```

Periferal harus menampilkan passkey numerik dan mengambil hasil perbandingan sebagai input.

Porting implementasi API

Untuk mem-port FreeRTOS ke target baru, Anda harus menerapkan beberapa API untuk layanan Penyediaan Wi-Fi dan fungsionalitas Bluetooth Low Energy.

Bluetooth API Energi Rendah

Untuk menggunakan middleware FreeRTOS Bluetooth Low Energy, Anda harus menerapkan beberapa API.

API umum antara GAP untuk Bluetooth Classic dan GAP untuk Bluetooth Low Energy

- pxBtManagerInit
- pxEnable
- pxDisable
- pxGetDeviceProperty
- pxSetDeviceProperty(Semua opsi adalah harapan wajib eBTpropertyRemoteRssi dan eBTpropertyRemoteVersionInfo)
- pxPair
- pxRemoveBond
- pxGetConnectionState
- pxPinReply
- pxSspReply
- pxGetTxpower
- pxGetLeAdapter

- pxDeviceStateChangedCb
- pxAdapterPropertiesCb
- pxSspRequestCb
- pxPairingStateChangedCb
- pxTxPowerCb

API khusus untuk GAP untuk Bluetooth Low Energy

- pxRegisterBleApp
- pxUnregisterBleApp
- pxBleAdapterInit
- pxStartAdv
- pxStopAdv
- pxSetAdvData
- pxConnParameterUpdateRequest
- pxRegisterBleAdapterCb
- pxAdvStartCb
- pxSetAdvDataCb
- pxConnParameterUpdateRequestCb
- pxCongestionCb


Server GATT

- pxRegisterServer
- pxUnregisterServer
- pxGattServerInit
- pxAddService
- pxAddIncludedService
- pxAddCharacteristic
- pxSetVal
- pxAddDescriptor
- pxStartService

- pxStopService
- pxDeleteService
- pxSendIndication
- pxSendResponse
- pxMtuChangedCb
- pxCongestionCb
- pxIndicationSentCb
- pxRequestExecWriteCb
- pxRequestWriteCb
- pxRequestReadCb
- pxServiceDeletedCb
- pxServiceStoppedCb
- pxServiceStartedCb
- pxDescriptorAddedCb
- pxSetValCallbackCb
- pxCharacteristicAddedCb
- pxIncludedServiceAddedCb
- pxServiceAddedCb
- pxConnectionCb
- pxUnregisterServerCb
- pxRegisterServerCb

Untuk informasi selengkapnya tentang mem-porting library Bluetooth FreeRTOS Low Energy ke platform Anda, lihat [Memindahkan Perpustakaan Bluetooth Energi Rendah](#) di Panduan Porting FreeRTOS.

SDK Seluler 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 ada berdasarkan repositori Amazon-freertos yang sekarang tidak digunakan lagi, lihat [Panduan Migrasi Repositori Github Amazon-freertos](#).

Anda dapat menggunakan SDK Seluler untuk Perangkat Bluetooth FreeRTOS untuk membuat aplikasi seluler yang berinteraksi dengan mikrokontroler Anda melalui Bluetooth Low Energy. Mobile SDK juga dapat berkomunikasi dengan AWS layanan, menggunakan Amazon Cognito untuk autentikasi pengguna.

Android SDK untuk perangkat Bluetooth FreeRTOS

Gunakan Android SDK untuk Perangkat Bluetooth FreeRTOS untuk membuat 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 menyiapkan dan menjalankan aplikasi seluler demo yang disertakan dengan SDK, lihat [Prasyarat](#) dan [Aplikasi demo SDK Seluler FreeRTOS Bluetooth Rendah Energi](#).

SDK iOS untuk perangkat Bluetooth FreeRTOS

Gunakan SDK iOS untuk Perangkat Bluetooth FreeRTOS untuk membuat 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 menggunakan `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 menyiapkan dan menjalankan aplikasi seluler demo yang disertakan dengan SDK, lihat [Prasyarat](#) dan [Aplikasi demo SDK Seluler FreeRTOS Bluetooth Rendah Energi](#).

Lampiran A: MQTT lebih profil BLE GATT

Detail Layanan GATT

MQTT over BLE menggunakan instance dari layanan GATT transfer data untuk mengirim pesan MQTT Concise Binary Object Representation (CBOR) antara perangkat FreeRTOS dan perangkat proxy. Layanan transfer data mengekspos karakteristik tertentu yang membantu mengirim dan menerima data mentah melalui protokol BLE GATT. Ini juga menangani fragmentasi dan perakitan muatan lebih besar dari ukuran unit transfer maksimum BLE (MTU).

Layanan UUID

A9D7-166A-D72E-40A9-A002-4804-4CC3-FF00

Instans layanan Instans layanan 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 berdasarkan 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 wajib	Properti Opsional	Izin Keamanan	Deskripsi Singkat	UUID
Kontrol	M	Tulis	Tidak ada	Menulis Kebutuhan Enkripsi	Digunakan untuk memulai dan	A9D7-166A - D72E-40A 9-

Karakteristik	Persyaratan	Properti Wajib	Properti Opsional	Izin Keamanan	Deskripsi Singkat	UUID
					menghentikan proxy MQTT.	A002-4804-4CC3-FF01
TxMessage	M	Baca, Pemberitahuan	Tidak ada	Baca Kebutuhan Enkripsi	Digunakan untuk mengirim pemberitahuan yang berisi pesan ke broker melalui proxy.	A9D7-166A- - D72E-40A9- A002-4804-4CC3- FF02
RxMessage	M	Baca, Tulis Tanpa Respon	Tidak ada	Baca, Tulis kebutuhan enkripsi Baca, Tulis Kebutuhan	Digunakan untuk menerima pesan dari broker melalui proxy.	A9D7-166A- - D72E-40A9- A002-4804-4CC3- FF03

Karakteristik	Persyaratan	Properti Wajib Properti wajib	Properti Opsional	Izin Keamanan	Deskripsi Singkat	UUID
TXLargeMessage	M	Baca, Pemberitahuan	Tidak ada	Baca Kebutuhan Enkripsi	Digunakan untuk mengirim pesan besar (Pesan > BLE MTU Size) ke broker melalui proxy.	A9D7-166A - D72E-40A 9- A002-48 04-4CC3- FF04
RXLargeMessage	M	Baca, Tulis Tanpa Respon	Tidak ada	Baca, Tulis kebutuhan enkripsi Baca, Tulis Kebutuhan	Digunakan untuk menerima pesan besar (Pesan > BLE MTU Size) dari broker melalui proxy.	A9D7-166A - D72E-40A 9- A002-48 04-4CC3- FF05

Persyaratan Prosedur GATT 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
Notifikasi	Wajib
Indikasi	Wajib

Jenis Pesan

Jenis pesan berikut dipertukarkan.

Jenis Pesan	Message	Peta dengan pasangan kunci/nilai pasangan ini
0x01	MENGHUBUNG	<ul style="list-style-type: none"> • Key = “w”, value = Type 0 Integer, Jenis pesan (1) • Key = “d”, value = Type 3, Text String, Client Identifier untuk sesi • Key = “a”, value = Type 3, Text String, Broker endpoint untuk sesi • Kunci = “c”, Nilai = Jenis Nilai Sederhana Benar/Salah
0x02	CONNACK	<ul style="list-style-type: none"> • Key = “w, value = Type 0 Integer, Jenis pesan (2) • Key = “s”, Nilai = Tipe 0 Integer, Kode status
0x03	MENERBITKAN	<ul style="list-style-type: none"> • Key = “w”, value = Type 0 Integer, Jenis pesan (3)

Jenis Pesan	Message	Peta dengan pasangan kunci/nilai pasangan ini
		<ul style="list-style-type: none"> • Key = “u”, value = Type 3, Text String, Topik untuk mempublikasikan • Key = “n”, value = Type 0, Integer, QoS untuk mempublikasikan • Key = “i”, value = Type 0, Integer, Message Identifier, Hanya untuk QoS 1 Publishes • Key = “k”, Value = Type 2, Byte String, Payload untuk mempublikasikan
0x04	PUBACK	<ul style="list-style-type: none"> • Dikirim Hanya untuk pesan QoS 1. • Key = “w”, value = Type 0 Integer, Jenis pesan (4) • Key = “i”, value = Type 0, Integer, Message Identifier
0x08	BERLANGGANAN	<ul style="list-style-type: none"> • Key = “w”, value = Type 0 Integer, Jenis pesan (8) • Key = “v”, value = Type 4, Array string teks, topik untuk berlangganan • Key = “o”, value = Type 4, Array of Integer, QoS untuk berlangganan • Key = “i”, value = Type 0, Integer, Message Identifier

Jenis Pesan	Message	Peta dengan pasangan kunci/nilai pasangan ini
0x09	SUBACK	<ul style="list-style-type: none"> • Key = "w", value = Type 0 Integer, Jenis pesan (9) • Key = "i", value = Type 0, Integer, Message Identifier • Key = "s", value = Type 0, Integer, Kode status untuk Berlangganan
0X0A	UNSUBSCRIBE (BERHENTI BERLANGGANAN)	<ul style="list-style-type: none"> • Key = "w", value = Type 0 Integer, Jenis pesan (10) • Key = "v", value = Type 4, Array string teks, topik untuk berhenti berlangganan • Key = "i", value = Type 0, Integer, Message Identifier
0x0B	UNSUBACK	<ul style="list-style-type: none"> • Key = "w", value = Type 0 Integer, Jenis pesan (11) • Key = "i", value = Type 0, Integer, Message Identifier • Key = "s", value = Type 0, Integer, kode Status untuk UnSubscription
0X0C	PINGREQ	<ul style="list-style-type: none"> • Key = "w", value = Type 0 Integer, Jenis pesan (12)
0x0D	PINGRESP	<ul style="list-style-type: none"> • Key = "w", value = Type 0 Integer, Jenis pesan (13)
0x0E	DISCONNECT	<ul style="list-style-type: none"> • Key = "w", value = Type 0 Integer, Jenis pesan (14)

Karakteristik Transfer Muatan Besar

TXLargeMessage

TXLargeMessage 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 sisa byte muatan, mana yang kurang. Setiap kali, itu meningkatkan offset dibaca 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 tertentu, transfer gagal dan proxy dan gateway melepaskan buffer.
- Jika proxy tidak mendapatkan respons baca dalam batas waktu tertentu, transfer gagal dan proxy melepaskan buffer.

RXLargeMessage

RXLargeMessage 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 buffer pesan sampai menerima permintaan tulis dengan panjang nol atau panjang kurang dari ukuran MTU.
- Jika perangkat tidak mendapatkan permintaan tulis dalam waktu tunggu 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 perpustakaan antarmuka seluler

Note

Konten pada halaman ini mungkin tidak up-to-date. Silakan lihat [halaman perpustakaan FreeRtos.org](#) untuk pembaruan terbaru.

Pengantar

Perpustakaan 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 lebih atau kurang dari perintah AT yang ditentukan oleh standar [3GPP TS v27.007](#). Proyek ini menyediakan [implementasi](#) seperti perintah AT standar dalam [komponen umum dapat digunakan kembali](#). Tiga perpustakaan Antarmuka Seluler dalam proyek ini semua mengambil keuntungan dari kode umum itu. Pustaka untuk setiap modem hanya mengimplementasikan perintah AT khusus vendor, kemudian mengekspos API library 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
- MISRA C: 2012 standar pengkodean. Setiap penyimpangan dari standar didokumentasikan dalam komentar kode sumber yang ditandai dengan “coverity”.

Dependensi dan persyaratan

Tidak ada ketergantungan langsung untuk pustaka 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 mem-porting library Antarmuka Seluler ke platform Anda, lihat [Memindahkan 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
cellular_3gpp_api.c	6.3K	5.7K
cellular_3gpp_urc_handler.c	0.9K	0.8K
cellular_at_core.c	1.4K	1.2K
cellular_common_api.c	0.5K	0.5K
cellular_common.c	1.6K	1.4K
cellular_pkthandler.c	1.4K	1.2K
cellular_pktio.c	1.8K	1.6K
Total perkiraan	13.9K	12,4K

Mulai

Unduh kode sumber sumber sumber sumber sumber sumber sumber sumber sumber

Kode sumber dapat diunduh sebagai bagian dari pustaka FreeRTOS atau dengan sendirinya.

Untuk mengkloning perpustakaan 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 folder struktur folder

Di root repositori ini Anda akan melihat folder ini:

- `source`: kode umum yang dapat digunakan kembali yang mengimplementasikan perintah AT standar yang ditentukan oleh 3GPP TS v27.007
- `doc`: dokumentasi
- `test`: unit test dan cbmc
- `tools`: alat untuk analisis statis Coverity dan CMock

Mengkonfigurasi dan membangun Pustaka

Pustaka Antarmuka Seluler perpustakaan Antarmuka Seluler harus dibangun sebagai bagian dari aplikasi. Untuk melakukannya, Anda harus memberikan konfigurasi tertentu. Proyek [FreeRtos_Cellular_Interface_Windows_Simulator](#) memberikan [contoh](#) cara mengkonfigurasi build. Informasi selengkapnya dapat ditemukan di Referensi API Seluler dapat ditemukan di [Referensi API Seluler](#).

Silakan lihat halaman [Antarmuka Seluler](#) untuk informasi lebih lanjut.

Integrasikan perpustakaan Antarmuka Seluler dengan platform MCU

Perpustakaan Antarmuka Seluler berjalan pada MCU menggunakan antarmuka yang disarankan, [Antarmuka Komunikasi](#), untuk berkomunikasi dengan modem seluler. Antarmuka Comm harus diimplementasikan pada platform MCU juga. Implementasi yang paling umum dari Comm Interface berkomunikasi melalui perangkat keras UART, tetapi dapat diimplementasikan melalui antarmuka fisik lainnya, seperti SPI, juga. Dokumentasi untuk Comm Interface dapat ditemukan di [Cellular Library API Referensi](#). Contoh implementasi berikut dari Comm Interface tersedia:

- [FreeRTOS antarmuka komunikasi simulator Windows](#)
- [FreeRTOS antarmuka komunikasi IO UART umum](#)
- [Antarmuka comm papan penemuan STM32 L475](#)
- [Antarmuka komunikasi papan Sierra Sensor Hub](#)

I/O

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 ada berdasarkan repositori Amazon-freertos yang sekarang tidak digunakan lagi, lihat [Panduan Migrasi Repositori Github Amazon-freertos](#).

Ikhtisar

Secara umum, driver perangkat tidak tergantung pada sistem operasi yang mendasarinya dan khusus untuk konfigurasi perangkat keras yang diberikan. Lapisan abstraksi perangkat keras (HAL) menyediakan antarmuka umum antara driver dan kode aplikasi tingkat yang lebih tinggi. HAL mengabstraksi detail tentang cara kerja driver tertentu dan menyediakan API yang seragam untuk mengontrol perangkat tersebut. Anda dapat menggunakan API yang sama untuk mengakses berbagai driver perangkat di beberapa papan referensi berbasis mikrokontroler (MCU).

FreeRTOS [umum I/O](#) bertindak sebagai lapisan abstraksi perangkat keras ini. Ini menyediakan satu set API standar untuk mengakses perangkat serial umum pada papan referensi yang didukung. API umum ini berkomunikasi dan berinteraksi dengan periferal ini dan memungkinkan kode Anda berfungsi di seluruh platform. Tanpa umum I/O, menulis kode untuk bekerja dengan perangkat tingkat rendah adalah silikon-vendor tertentu.

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.

Khusus 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.

Porting

Untuk informasi selengkapnya, lihat [Panduan Porting FreeRTOS](#).

AWS IoT Device Defenderperpustakaan

Note

Konten di halaman ini mungkin tidak up-to-date. Silakan merujuk ke [Halaman perpustakaan Freertos.org](#) untuk update terbaru.

Pengantar

Anda dapat menggunakanAWS IoT Device Defenderlibrary untuk mengirim metrik keamanan dari perangkat IoT Anda keAWS IoT Device Defender. Anda dapat menggunakanAWS IoT Device Defenderuntuk 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 Defendermengirimkan peringatan sehingga Anda dapat mengambil tindakan untuk memperbaiki masalah. Interaksi denganAWS IoT Device Defendermenggunakan[MQTT](#), protokol berlangganan publikasi yang ringan. Pustaka ini menyediakan API untuk menyusun dan mengenali string topik MQTT yang digunakan olehAWS IoT Device Defender.

Untuk informasi lebih lanjut, lihat [AWS IoT Device Defender](#) dalam Panduan DeveloperAWS IoT.

Perpustakaan 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. Itu juga tidak memiliki dependensi platform apa pun, seperti threading atau sinkronisasi. Ini dapat digunakan dengan perpustakaan MQTT apapun[JSON](#)atau[CBOR](#)perpustakaan. Perpustakaan memiliki**bukti**menunjukkan penggunaan memori yang aman dan tidak ada alokasi heap, sehingga cocok untuk mikrokontroler IoT, tetapi juga sepenuhnya portabel ke platform lain.

TheAWS IoT Device Defenderperpustakaan dapat digunakan secara bebas dan didistribusikan di bawah[Lisensi sumber terbuka 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
Total perkiraan	1.1K	0.6K

AWS IoT Greengrass Pustaka penemuan

Note

Konten pada halaman ini mungkin tidak up-to-date. Silakan lihat [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. Dengan menggunakan API AWS IoT Greengrass Discovery, perangkat Anda dapat mengirim pesan ke inti Greengrass setelah menemukan titik akhir inti.

Dependensi dan persyaratan

Untuk menggunakan perpustakaan Greengrass Discovery, Anda harus membuat sesuatu AWS IoT, termasuk sertifikat dan 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

Titik akhir AWS IoT Anda.

clientcredentialIOT_THING_NAME

Nama IoT Anda

clientcredentialWIFI_SSID

SSID untuk jaringan Wi-Fi Anda.

clientcredentialWIFI_PASSWORD

Kata sandi Wi-Fi Anda.

clientcredentialWIFI_SECURITY

Jenis keamanan yang digunakan 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 hal Anda.

keyCLIENT_PRIVATE_KEY_PEM

Kunci pribadi PEM terkait dengan hal Anda.

Anda harus menyiapkan grup Greengrass dan perangkat inti di konsol. Untuk informasi lebih lanjut, 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 lengkap, lihat [Referensi Greengrass API](#).

Contoh penggunaan**Alur kerja Greengrass**

Perangkat MCU memulai proses penemuan dengan meminta dari AWS IoT file JSON yang berisi parameter konektivitas inti Greengrass. Ada dua metode untuk mengambil parameter konektivitas inti Greengrass dari file JSON:

- Pemilihan otomatis iterasi melalui semua inti Greengrass yang tercantum dalam file JSON dan terhubung ke yang pertama yang 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, hubungi `GGD_GetGGCIPandCertificate` untuk meminta file JSON dengan informasi konektivitas inti Greengrass. Ketika `GGD_GetGGCIPandCertificate` dikembalikan, `pcBuffer` parameter berisi teks file JSON. `pxHostAddressDataParameter` berisi alamat IP dan port inti Greengrass yang dapat Anda hubungkan.

Untuk opsi penyesuaian lainnya, seperti mengalokasikan sertifikat secara dinamis, Anda harus memanggil API berikut:

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 respon HTTP.

GGD_JSONRequestGetFile

Mendapat string objek JSON. `GGD_JSONRequestGetSize` dan `GGD_JSONRequestGetFile` gunakan `GD_SecureConnect_Read` untuk mendapatkan data JSON dari soket. `GGD_JSONRequestStart`, `GD_SecureConnect_Send`, dan `GGD_JSONRequestGetSize` harus dipanggil untuk menerima data JSON dari AWS IoT.

GGD_GetIPandCertificateFromJSON

Ekstrak alamat IP dan sertifikat inti Greengrass dari data JSON. Anda dapat mengaktifkan pemilihan otomatis dengan mengatur `pxAutoSelectFlag` ke `True`. Pemilihan otomatis menemukan perangkat inti pertama yang dapat dihubungkan oleh perangkat FreeRTOS Anda. Untuk terhubung ke inti Greengrass, panggil `GD_SecureConnect_Connect` fungsi, teruskan alamat IP, port, dan sertifikat perangkat inti. Untuk menggunakan seleksi manual, atur bidang `HostParameters_t` parameter berikut:

pcGroupName

ID dari kelompok Greengrass yang menjadi milik inti. Anda dapat menggunakan `aws greengrass list-groups` CLI perintah 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 update terbaru.

Pustaka klien HTTP C untuk perangkat IoT kecil (MCU atau MPU kecil)

Pengantar

Pustaka CoreHTTP adalah implementasi klien dari subset [HTTP/1.1](#) Memstandarkan. Standar HTTP menyediakan protokol stateless yang berjalan di atas TCP/IP dan sering digunakan dalam sistem informasi hiperteks terdistribusi, kolaboratif.

Pustaka CoreHTTP mengimplementasikan subset [HTTP/1.1](#) standar protokol. 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 [Memversikan LTS \(v12.19.1\) dari parser http](#) dari Node.js. Perpustakaan memiliki [bukti](#) 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 [Demo otentikasi timbal balik CoreHTTP](#).

Perpustakaan ini dapat digunakan secara bebas dan didistribusikan di bawah [Lisensi sumber terbuka 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 pada halaman ini mungkin tidak up-to-date. Silakan lihat [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 API, seperti GitHub REST API. JSON dipertahankan sebagai standar oleh Ecma International.

Pustaka CoreJSON menyediakan parser yang mendukung pencarian kunci sekaligus menegakkan [sintaks Pertukaran Data JSON Standar ECMA-404](#). Perpustakaan ditulis dalam C dan dirancang untuk mematuhi 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 diawetkan. Ukuran tumpukan dapat ditentukan dengan mendefinisikan makroJSON_MAX_DEPTH,, yang default ke 32 tingkat. Setiap tingkat mengkonsumsi satu byte.

Ukuran Kode CoreJson (contoh yang 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](https://www.freertos.org) untuk update terbaru.

Pengantar

Pustaka CoreMQTT adalah implementasi klien dari [MQTT](https://mqtt.org/) (Transportasi Telemetri Antrian Pesan) standar. Standar MQTT menyediakan penerbitan/langganan ringan (atau [PubSub](https://pubsub.dev/) Protokol pesan 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 [MQTT 3.1.1](https://mqtt.org/) standar protokol. 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 di C dan dirancang agar sesuai dengan [ISO C90](https://www.iso.org/standard/62453.html) dan [MISRA C:2012](https://www.misra-c.org/). 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 API serializer/deserializer tingkat rendah. API ini dapat digunakan untuk membangun aplikasi IoT sederhana yang hanya terdiri dari subset fungsionalitas MQTT yang diperlukan, tanpa overhead lainnya. API serializer/deserializer 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. Perpustakaan ini 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 sumber terbuka 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 update terbaru.

Pengantar

Pustaka Agen CoreMQTT adalah API tingkat tinggi yang menambahkan keamanan utas ke [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 yang setara dengan API CoreMQTT, sehingga dapat digunakan di lingkungan multi-utas.

Agan 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 antrean pesan aman utas (atau mekanisme komunikasi antar-proses lainnya) untuk membuat serial semua permintaan untuk memanggil API MQTT. 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 apa pun selain [Perpustakaan CoreMQTT](#) dan perpustakaan 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.

Perpustakaan ini dapat digunakan secara bebas dan didistribusikan di bawah [Lisensi sumber terbuka 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_commands_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 IoTPerustakaan Over the Air (OTA)

 Note

Konten pada halaman ini mungkin tidak up-to-date. Silakan lihat [halaman perpustakaan FreeRtos.org](#) untuk pembaruan terbaru.

Pengantar

[Pustaka pembaruanAWS IoT Over-the-air \(OTA\)](#) memungkinkan Anda mengelola notifikasi, 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, perpustakaan 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 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, 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 terpapar kerentanan keamanan untuk jangka waktu yang lebih lama. Mengingat perangkat ini untuk memperbarui juga akan mahal dan dapat menyebabkan gangguan yang signifikan bagi konsumen karena downtime.

Pembaruan Over the Air (OTA) memungkinkan untuk memperbarui firmware perangkat tanpa penarikan atau kunjungan teknisi yang mahal. Metode ini menambahkan manfaat berikut:

- Keamanan - Kemampuan untuk dengan cepat merespons kerentanan keamanan dan bug perangkat lunak yang ditemukan setelah perangkat dikerahkan di lapangan.
- Inovasi - Produk dapat diperbarui sering sebagai fitur baru yang dikembangkan, mendorong siklus inovasi. Pembaruan dapat berlaku dengan cepat dengan waktu henti minimum dibandingkan dengan metode pembaruan tradisional.
- Biaya - Pembaruan OTA dapat mengurangi biaya perawatan 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 dari sumber tepercaya, bersama dengan validasi lain seperti memeriksa versi dan kompatibilitas.

Untuk informasi selengkapnya tentang pengaturan pembaruan OTA dengan FreeRTOS, lihat [Pembaruan FreeRTOS Over-the-Air](#).

AWS IoT Perpustakaan Over the Air (OTA)

Perpustakaan AWS IoT OTA memungkinkan Anda mengelola pemberitahuan pembaruan yang baru tersedia, mengunduhnya, dan melakukan verifikasi kriptografi pembaruan firmware. Dengan menggunakan perpustakaan 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. Perpustakaan 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 dengan jenis dan kondisi jaringan Anda.

API library ini 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.
- Memperbarui status perangkat.

Perpustakaan ini menggunakan AWS layanan untuk mengelola berbagai fungsi terkait cloud seperti mengirim pembaruan firmware, memantau sejumlah besar perangkat di beberapa wilayah, mengurangi radius ledakan penyebaran yang salah, dan memverifikasi keamanan pembaruan. Perpustakaan ini dapat digunakan dengan perpustakaan MQTT atau HTTP.

Demo untuk perpustakaan ini menunjukkan over-the-air pembaruan lengkap menggunakan Perpustakaan dan AWS Layanan Core MQTT pada perangkat FreeRTOS.

Fitur

Berikut adalah antarmuka Agen OTA lengkap:

OTA_Init

Menginisialisasi mesin OTA dengan memulai OTA Agent (“OTA Task”) dalam sistem. Hanya satu Agen OTA yang mungkin ada.

OTA_Shutdown

Sinyal ke Agen OTA untuk dimatikan. 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 keadaan 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 peristiwa untuk 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 untuk konversi string untuk kesalahan OTA.

OTA_JobParse_strerror

Mengkonversi kode kesalahan OTA Job Parsing ke 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 mendorong Agen OTA dengan menggunakan urutan panggilan API berikut.

1. Connect ke Agen AWS IoT Core MQTT. Untuk informasi selengkapnya, lihat [Perpustakaan Agen CoreMQTT](#).
2. Inisialisasi Agen OTA dengan menelepon `OTA_Init`, termasuk buffer, antarmuka ota yang diperlukan, nama benda dan callback 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 berikut: `accepted`, `rejected`, atau `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 mem-porting fungsionalitas OTA ke platform Anda, lihat [Memindahkan 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

Pustaka CorePKCS11

Note

Konten pada halaman ini mungkin tidak up-to-date. Silakan lihat [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 abstrak penyimpanan kunci, mendapatkan/mengatur properti untuk objek kriptografi, dan semantik sesi. Ini banyak digunakan untuk memanipulasi objek

kriptografi umum, dan itu 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 AWS referensi FreeRTOS menggunakan subset kecil API PKCS #11 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' kuncinya.

Pustaka CorePKCS11 berisi implementasi tiruan berbasis perangkat lunak dari antarmuka PKCS #11 (API) 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 cryptoprocessors aman, seperti Trusted Platform Module (TPM), Hardware Security Module (HSM), Secure Element, atau jenis kantong perangkat keras aman lainnya, mendistribusikan implementasi PKCS #11 dengan perangkat keras. Tujuan dari perangkat lunak CorePKCS11 hanya perpustakaan tiruan karena itu untuk menyediakan implementasi PKCS #11 khusus perangkat keras non yang memungkinkan untuk prototyping cepat dan pengembangan sebelum beralih ke cryptoprocessor tertentu PKCS #11 implementasi dalam perangkat produksi.

Hanya subset dari standar PKCS #11 yang diimplementasikan, dengan fokus pada operasi yang melibatkan kunci asimetris, pembuatan angka acak, dan hashing. Kasus penggunaan yang ditargetkan mencakup manajemen sertifikat dan kunci untuk otentikasi TLS, dan verifikasi tanda tangan 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, penyediaan dan otentikasi klien TLS, memerlukan implementasi hanya sebagian kecil dari standar antarmuka PKCS #11.

Fitur

Subset berikut dari PKCS #11 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.

Pengaturan umum dan meruntuhkan API

- `C_Initialize`

- C_Finalize
- C_GetFunctionList
- C_GetSlotList
- C_GetTokenInfo
- C_OpenSession
- C_CloseSession
- C_Login

API Penyediaan API 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 kriptografi 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 AWS CLI dan OpenSSL berikut (atau yang 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. Kirimkan 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 oleh 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 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 untuk hal 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 pustaka CorePkcCS11 ke platform Anda, lihat [Memindahkan Pustaka CorePkcCS11 di Panduan Porting FreeRTOS](#).

Penggunaan memori

Ukuran Kode CorePKCS11 (contoh yang 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](#)

Ikhtisar

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

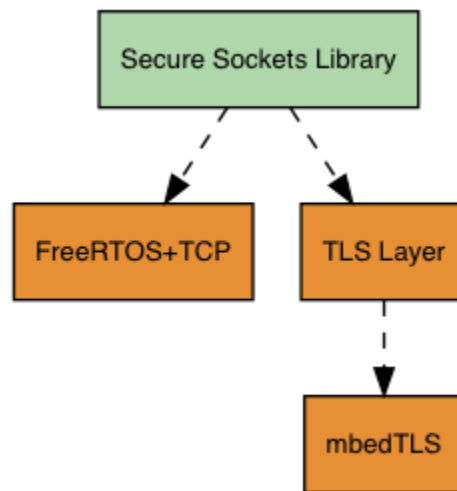
Saat ini, hanya API klien, ditambah implementasi [IP \(LWiP\) ringan](#) dari API Bind sisi server, yang didukung untuk FreeRTOS Secure Sockets.

Ketertanggung 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](#)
- [FreeTos+TCP dan lapisan FreeRTOS TLS dengan mbedTLS](#)

Diagram ketertanggung 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 selengkapny tentang layer FreeRTOS TLS, lihat. [Keamanan Lapisan Pengangkutan](#)



Fitur

Fitur perpustakaan FreeRTOS Secure Sockets meliputi:

- Antarmuka standar berbasis soket Berkeley
- Thread-safe API untuk mengirim dan menerima data
- E asy-to-enable TLS

Memecahkan masalah

Kode error

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.

API server tidak didukung oleh pustaka FreeRTOS Secure Sockets, kecuali untuk implementasi [IP \(LWiP\) ringan dari API sisi](#) server. Bind API 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/0wua2eiedgPySjAKBggqhkJOPQQDAjA5\n"
"MQswCQYDVQQGEwJVUzEPMA0GA1UEChMGQW1hem9uMRkwFwYDVQQDExBBbWF6b24g\n"
"Um9vdCBDQSAzMB4XDTE1MDUyNjAwMDAwMFoXDTE1MDUyNjAwMDAwMFowOTELMAkG\n"
"A1UEBHMCMVVMxDzANBgNVBAoTBkFtYXpvcjEzMBcGA1UEAxMQQW1hem9uIFJvb3Qg\n"
"Q0EgMzBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABCMXp8ZBf8ANm+gBG1bG81K1\n"
"ui2yEujSLtf6ycXYqm0fc4E705hr0XwzpcV0ho6AF2hiRVd9RFgdszflZwjrz6j\n"
"QjBAMA8GA1UdEwEB/wQFMAMBAf8wDgYDVR0PAAQH/BAQDAgGGMB0GA1UdDgQWBBSr\n"
"ttvXBp43rDCGB5Fwx5zEGbF4wDAKBggqhkJOPQQDAgNJADBGAiEA4IWSoxe3jfk\n"
"BqWTrBqYaGfY+uGh0PscGCM5nFuMQCIQCcAu/xlJyzlvnrXir4tiz+0pAUFteM\n"
"YyRIHN8wfdVo0w==\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, lihat [Soket 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 [Pustaka CorePKCS11](#)
- Sebuah [Keamanan Lapisan Pengangkutan](#)

Untuk informasi selengkapnya tentang porting, lihat [Porting Perpustakaan Soket Aman](#) di Panduan Porting FreeRTOS.

AWS IoT Pustaka Device Shadow

Note

Konten pada halaman ini mungkin tidak up-to-date. Silakan lihat [halaman perpustakaan FreeRtos.org](#) untuk pembaruan terbaru.

Pengantar

Anda dapat menggunakan library AWS IoT Device Shadow untuk menyimpan dan mengambil status saat ini (bayangan) dari setiap perangkat yang terdaftar. Bayangan perangkat adalah representasi virtual persisten dari perangkat Anda yang dapat berinteraksi dengan 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. Bayangan setiap perangkat diidentifikasi secara unik 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 dalam [AWS IoT dokumentasi](#).

Pustaka AWS IoT Device Shadow tidak memiliki dependensi pada pustaka tambahan selain pustaka C standar. Ini juga tidak memiliki dependensi platform apa pun, seperti threading atau sinkronisasi. Hal ini dapat digunakan dengan perpustakaan MQTT dan perpustakaan JSON.

Perpustakaan ini dapat digunakan secara bebas dan didistribusikan di bawah [lisensi open source MIT](#).

Ukuran Kode AWS IoT Device Shadow (contoh dibuat 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

Note

Konten di halaman ini mungkin tidak up-to-date. Silakan merujuk ke [Halaman perpustakaan Freertos.org](#) untuk update terbaru.

Pengantar

AWS IoT Jobs adalah layanan yang memberi tahu satu atau beberapa perangkat yang terhubung tentang penundaan pekerjaan. 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 AWS IoT menggunakan layanan pekerjaan MQTT, protokol berlangganan publikasi yang ringan. Pustaka ini menyediakan API untuk menyusun dan mengenali string topik MQTT yang digunakan oleh AWS IoT Layanan pekerjaan.

The AWS IoT Perpustakaan pekerjaan 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. Ini dapat digunakan dengan perpustakaan MQTT dan perpustakaan JSON apa pun. Perpustakaan memiliki [bukti](#) 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 sumber terbuka MIT](#).

Ukuran KodeAWS IoTPekerjaan (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 Pengangkutan

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 mengabstrak detail implementasi kriptografi jauh 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` repositori kode sumber.

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 tumpukan penuh TLS dan transportasi jaringan.

Untuk informasi selengkapnya tentang porting antarmuka TLS, lihat Memindahkan [Pustaka TLS](#) di Panduan Porting FreeRTOS.

Perpustakaan 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](#)

Ikhtisar

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.

Ketergantungan dan persyaratan

[Pustaka Wi-Fi FreeRTOS membutuhkan inti Freertos+TCP.](#)

Fitur

Pustaka Wi-Fi mencakup fitur-fitur berikut:

- Support untuk otentikasi WEP, WPA, WPA2, dan 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 menelepon `WIFI_GetMode`. Anda dapat mengatur mode wi-fi perangkat dengan menelepon `WIFI_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 kredensial 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

API Wi-Fi mendukung jenis keamanan WEP, WPA, WPA2, dan WPA3. 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 kredensial jaringan. Anda dapat memutuskan sambungan perangkat Wi-Fi dari jaringan dengan menelepon `WiFi_Disconnect`. Untuk informasi selengkapnya tentang pemindaian dan penghubung, lihat [Contoh penggunaan](#) dan [Referensi 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](#) dan [Referensi 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 )
{
    configPRINTF( ( "WiFi library initialized.\n" ) );
}
else
{
    configPRINTF( ( "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 )
{
    configPRINTF( ( "WiFi Connected to AP.\n" ) );
    // IP Stack will receive a network-up event on success
}
else
{
    configPRINTF( ( "WiFi failed to connect to AP.\n" ) );
    // Handle connection failure
}
```

Memindai AP terdekat

```
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 Demo demo FreeRTOS

FreeRTOS menyertakan beberapa aplikasi demo di `demos` folder, di bawah direktori FreeRTOS utama. Semua contoh yang dapat dijalankan oleh FreeRTOS muncul di `common` folder, di bawah `demos`. Ada juga folder untuk setiap platform yang memenuhi syarat FreeRTOS di bawah `demos` folder.

Sebelum Anda mencoba aplikasi demo, kami sarankan Anda menyelesaikan tutorial di [Memulai dengan FreeRTOS](#). Ini menunjukkan cara menyiapkan dan menjalankan demo Agen CoreMQTT.

Menjalankan demo FreeRTOS

Topik berikut ini menunjukkan cara menyiapkan dan menjalankan demo FreeRTOS

- [Aplikasi demo Bluetooth Low Energy](#)
- [Bootloader demo untuk Microchip Curiosity PIC32MZEF](#)
- [AWS IoT Device Defender Demo](#)
- [AWS IoT Greengrass Aplikasi demo penemuan V1](#)
- [AWS IoT Greengrass V2](#)
- [Demo inti HTTP](#)
- [AWS IoT Lowongan kerja perpustakaan demo](#)
- [Demo CoreMQTT CoreMQTT](#)
- [Over-the-air update 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 menelepon dan memulai demo Agen CoremQTT. Bergantung pada konfigurasi yang Anda pilih saat mengunduh FreeRTOS, dan di mana 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. Tergantung pada kombinasi, perangkat lunak mungkin tidak berjalan pada target yang dipilih karena kendala memori. Kami menyarankan Anda menjalankan satu demo pada satu waktu.

Konfigurasi demo

Demo demo telah dikonfigurasi untuk membuat 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 Low Energy

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 Rendah](#) demo


Aplikasi ini menunjukkan cara menggunakan MQTT melalui layanan Bluetooth Low Energy.

- [Penyediaan Wi-Fi](#) demo

Aplikasi ini menunjukkan cara menggunakan layanan Penyediaan Wi-Fi Energi Rendah Bluetooth.

- [Server Atribut Generik](#) demo

Aplikasi ini menunjukkan bagaimana menggunakan FreeRTOS Bluetooth Low Energy middleware API untuk membuat server GATT sederhana.

 Note

Untuk mengatur dan menjalankan demo FreeRTOS, ikuti langkah-langkah di dalamnya [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](#) atau [Android SDK untuk perangkat Bluetooth FreeRTOS](#).

Mengatur AWS IoT dan Amazon Cognito untuk FreeRTOS Bluetooth Low Energy

Untuk menghubungkan perangkat Anda ke AWS IoT seluruh MQTT, Anda perlu mengatur AWS IoT dan Amazon 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 Hal-hal.
3. Pilih Buat, lalu pilih Buat satu hal.
4. Masukkan nama untuk perangkat Anda, lalu pilih Selanjutnya.
5. Jika Anda menghubungkan mikrokontroler Anda ke cloud melalui perangkat seluler, pilih Buat hal tanpa sertifikat. Karena SDK Seluler menggunakan Amazon Cognito untuk autentikasi perangkat, Anda tidak perlu membuat sertifikat perangkat untuk demo yang menggunakan Bluetooth Low Energy.

Jika Anda menghubungkan mikrokontroler ke cloud secara langsung melalui Wi-Fi, pilih Buat sertifikat, pilih Aktifkan, lalu unduh sertifikat, kunci publik, dan kunci pribadi.

6. Pilih hal yang baru saja Anda buat dari daftar barang terdaftar, lalu pilih Interact dari halaman barang Anda. Buat catatan tentang titik akhir AWS IoT REST API.

Untuk informasi lebih lanjut tentang pengaturan, lihat [Cara memulai AWS IoT](#).

Untuk membuat kolam pengguna Amazon Cognito

1. Buka konsol Amazon Cognito, lalu pilih Kelola Kolam Pengguna.
2. Pilih Buat kolam pengguna.
3. Beri nama pangkalan 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 pangkalan.

Catat ID pool yang muncul di halaman Pengaturan Umum kumpulan pengguna Anda.

7. Dari panel navigasi, pilih Klien aplikasi, lalu pilih Tampilkan detail. Catat rahasia ID klien aplikasi dan klien aplikasi.

Untuk membuat kolam identitas Amazon Cognito

1. Buka konsol Amazon Cognito, lalu 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 terautentikasi dan tidak terautentikasi untuk mengakses Amazon Cognito.
6. Pilih Edit kolam identitas. Perhatikan ID kolam identitas. Itu harus dari formulir - west-2:12345678-1234-1234-1234-123456789012.

Untuk informasi selengkapnya tentang menyiapkan 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. Cari dan pilih peran identitas terautentikasi, pilih Lampirkan kebijakan, lalu pilih Tambahkan kebijakan sebaris.
3. Pilih tab JSON dan tempelkan 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, lalu pilih Buat kebijakan.

Simpan informasi Anda AWS IoT dan Amazon Cognito di tangan. Anda memerlukan endpoint dan ID untuk mengautentikasi aplikasi seluler Anda dengan AWS Cloud.

Siapkan lingkungan FreeRTOS Anda untuk Bluetooth Low Energy

Untuk mengatur lingkungan Anda, Anda perlu mengunduh FreeRTOS dengan mikrokontroler Anda, dan mengunduh serta mengonfigurasi SDK Seluler untuk Perangkat Bluetooth FreeRTOS di perangkat seluler Anda. [Perpustakaan Bluetooth Energi Rendah](#)

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 yang memenuhi syarat FreeRTOS, lihat panduan untuk papan Anda di [Memulai dengan FreeRTOS](#).

Note

Anda dapat menjalankan demo pada mikrokontroler Bluetooth Low Energy-enabled apa pun dengan FreeRTOS dan pustaka FreeRTOS Bluetooth Low Energy yang di-porting. Saat ini, proyek [MQTT melalui Bluetooth Energi Rendah](#) demo FreeRTOS sepenuhnya di-porting ke perangkat berkemampuan Bluetooth Low Energy berikut:

- [Espressif ESP32-DevKit C dan ESP-WROVER-KIT](#)
- [Nordik NRF52840-dk](#)

Komponen umum

Aplikasi demo FreeRTOS memiliki dua komponen umum:

- Network Manager
- Aplikasi demo Bluetooth Low Energy Mobile SDK

Network Manager

Network Manager mengelola koneksi jaringan mikrokontroler Anda. Hal ini terletak di direktori FreeRTOS Anda `aws_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 diaktifkan Wi-Fi, Network Manager 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 *papan* 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 didukung saat ini, lihat baris yang dimulai dengan `#define AWSIOT_NETWORK_TYPE` in `aws_iot_network.h`.

Aplikasi demo SDK Seluler FreeRTOS Bluetooth Rendah Energi

Aplikasi demo SDK FreeRTOS Bluetooth Low Energy Mobile terletak GitHub di [Android SDK untuk Perangkat Bluetooth FreeRTOS](#) di bawah `amazon-freertos-ble-android-sdk/app` dan [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 mengkonfigurasi aplikasi demo iOS SDK

Saat Anda menentukan variabel konfigurasi, gunakan format nilai placeholder yang disediakan dalam file konfigurasi.

1. Mengonfirmasi [SDK iOS untuk perangkat Bluetooth FreeRTOS](#) bahwa telah terpasang.
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 terautentikasi 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 dalam 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 Create (Buat).

5. Buka `amazon-freertos-ble-ios-sdk/Example/AmazonFreeRTOSDemo/AmazonFreeRTOSDemo/Amazon/AmazonConstants.swift`, dan mendefinisikan kembali 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 bawah `CognitoIdentity`, mendefinisikan kembali variabel berikut:

- `PoolId`: ID kolam identitas Amazon Cognito.
- `Region`: AWS Wilayah Anda.


Di bawah `CognitoUserPool`, mendefinisikan kembali variabel berikut:

- `PoolId`: ID kolam pengguna Amazon Cognito.
- `AppClientId`: ID klien aplikasi Anda.
- `AppClientSecret`: Rahasia klien aplikasi Anda.
- `Region`: AWS Wilayah Anda.

Untuk mengkonfigurasi aplikasi demo Android SDK

Saat Anda menentukan variabel konfigurasi, gunakan format nilai placeholder yang disediakan dalam file konfigurasi.

1. Mengonfirmasi [Android SDK untuk perangkat Bluetooth FreeRTOS](#) bahwa telah terpasang.
2. Buat AWS IoT kebijakan di wilayah Anda (jika Anda belum melakukannya).

 Note

Kebijakan ini berbeda dengan kebijakan IAM yang dibuat untuk identitas terautentikasi 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 dalam 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 Create (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 kembali 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 kembali variabel berikut:

- `PoolId`: ID kolam identitas Amazon Cognito.
- `Region`: AWS Wilayah Anda.

Di bawah `CognitoUserPool`, mendefinisikan kembali variabel berikut:

- `PoolId`: ID kolam pengguna Amazon Cognito.
- `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 memasang mikrokontroler dan perangkat seluler Anda dengan aman (langkah 6), Anda memerlukan emulator terminal serial dengan kemampuan input dan output (seperti TeraTerm). Konfigurasi terminal untuk terhubung ke papan Anda dengan koneksi serial seperti yang diinstruksikan [Memasang 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 Perangkat di aplikasi demo Bluetooth Low Energy Mobile SDK.

11:20 AM Thu Nov 8 Wi-Fi 34%

Devices Logout

ESP32

2796386F-3940-BEBA-7730-3C51DA88922F

Note

Semua perangkat dengan FreeRTOS dan layanan informasi perangkat (*freertos*/.../device_information) yang berada dalam jangkauan muncul dalam daftar.

- Pilih mikrokontroler Anda dari daftar perangkat. Aplikasi membuat koneksi dengan papan, dan garis hijau muncul di sebelah perangkat yang terhubung.

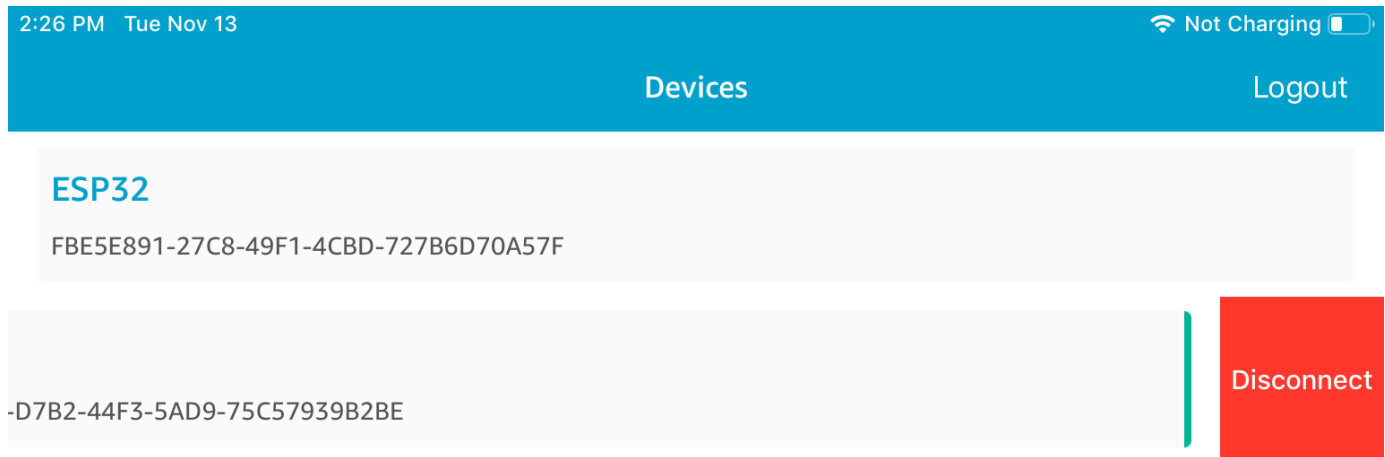
2:24 PM Tue Nov 13 Wi-Fi Not Charging

Devices Logout

ESP32

8F8FD9DF-D7B2-44F3-5AD9-75C57939B2BE

Anda dapat memutuskan sambungan dari mikrokontroler Anda dengan menyeret garis ke kiri.

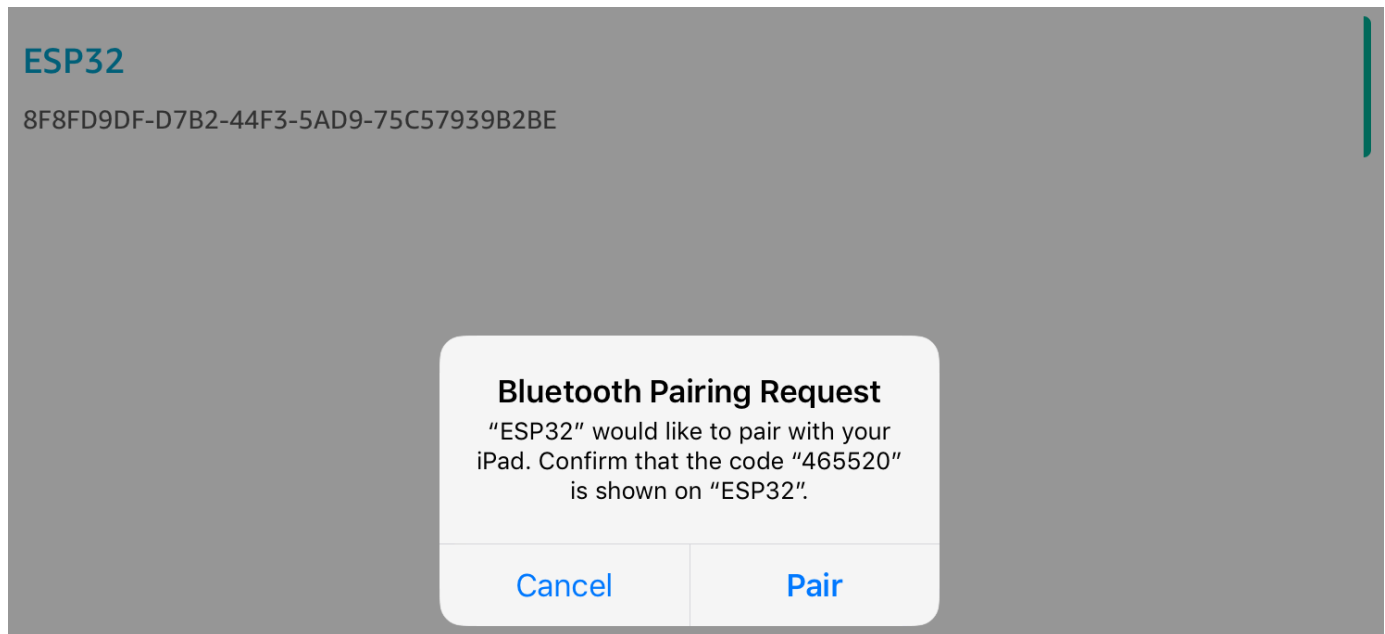


6. Jika diminta, pasang mikrokontroler dan perangkat seluler Anda.

```

24 261107 [Btc_task]   Disconnected from BLE device. Stopping the counter update
25 261108 [Btc_task]   Disconnect received for MQTT service instance 0
26 261108 [Btc_task]   BLE disconnected with remote device, start advertisement
27 261108 [Btc_task]   Started advertisement. Listening for a BLE Connection.
28 261289 [Btc_task]   BLE Connected to remote device, connId = 0
E (2614110) BT_GATT: gatts_write_attr_perm_check - GATT_INSUF_AUTHENTICATION: MITM required
E (2614420) BT_SMP: Value for numeric comparison = 465520
29 261412 [uTask]     Numeric comparison:465520
30 261412 [uTask]     Press 'y' to confirm

```



Jika kode untuk perbandingan numerik sama pada kedua perangkat, pasang perangkat.

Note

Aplikasi demo Bluetooth Low Energy Mobile SDK menggunakan Amazon Cognito untuk autentikasi pengguna. Pastikan Anda telah menyiapkan kumpulan pengguna dan identitas Amazon Cognito, dan Anda telah melampirkan kebijakan IAM ke identitas terautentikasi.

MQTT melalui Bluetooth Energi Rendah

Dalam demo MQTT over 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 uji MQTT untuk membuka klien MQTT.
3. Di Topik berlangganan ***thing-name/example/topic1***, masukkan, lalu pilih Berlangganan topik.

Jika Anda menggunakan Bluetooth Low Energy untuk memasang mikrokontroler dengan perangkat seluler Anda, pesan MQTT akan dialihkan melalui aplikasi demo Bluetooth Low Energy Mobile SDK di perangkat seluler Anda.

Untuk mengaktifkan demo melalui Bluetooth Low Energy

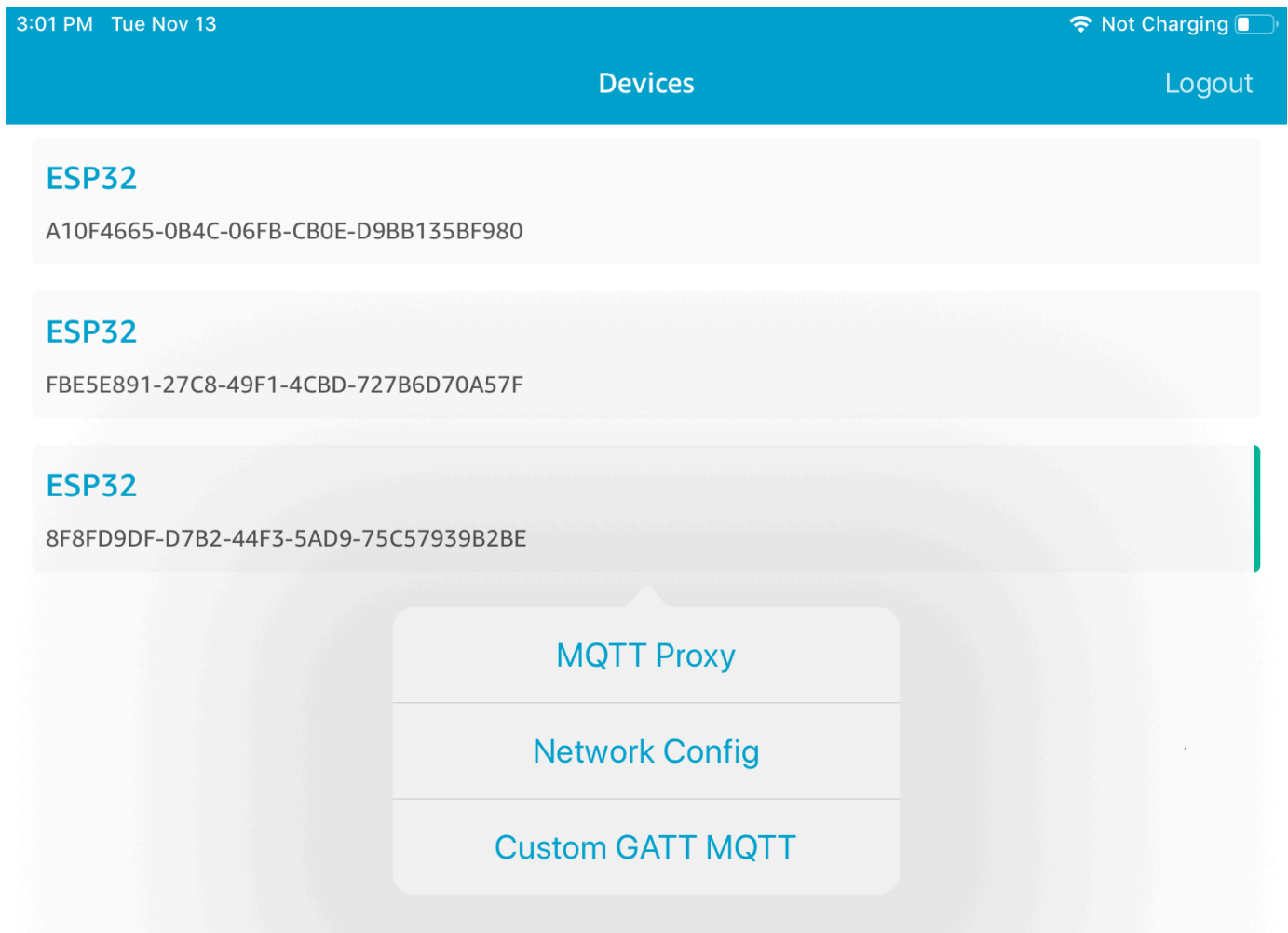
1. Bukavendors/***vendor***/boards/***board***/aws_demos/config_files/aws_demo_config.h, dan tentukan `CONFIG_MQTT_BLE_TRANSPORT_DEMO_ENABLED`.
2. Bukademos/include/aws_clientcredential.h, dan konfigurasi `clientcredentialMQTT_BROKER_ENDPOINT` dengan titik akhir AWS IoT broker. Konfigurasi `clientcredentialIOT_THING_NAME` dengan nama benda untuk perangkat pengontrol mikro BLE. Endpoint 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 harus berada di wilayah yang sama di mana identitas kognito dan kumpulan pengguna dikonfigurasi.

Untuk menjalankan demo

1. Membangun dan menjalankan proyek demo pada mikrokontroler Anda.
2. Pastikan bahwa Anda telah memasang papan Anda dan perangkat mobile Anda menggunakan [Aplikasi demo SDK Seluler FreeRTOS Bluetooth Rendah Energi](#).
3. Dari daftar Perangkat di aplikasi seluler demo, pilih mikrokontroler Anda, lalu pilih Proksi MQTT 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

Wi-Fi Provisioning adalah layanan FreeRTOS Bluetooth Low Energy yang memungkinkan Anda mengirim kredensi 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 pada Espressif ESP32-DevKit C.

Untuk mengaktifkan demo

1. Aktifkan layanan Penyediaan Wi-Fi. Bukavendors/*vendor*/boards/*board*/aws_demos/config_files/iot_ble_config.h, dan atur `#define IOT_BLE_ENABLE_WIFI_PROVISIONING ke1` (di mana *vendor* adalah nama vendor dan *papan* adalah nama papan yang Anda gunakan untuk menjalankan demo).

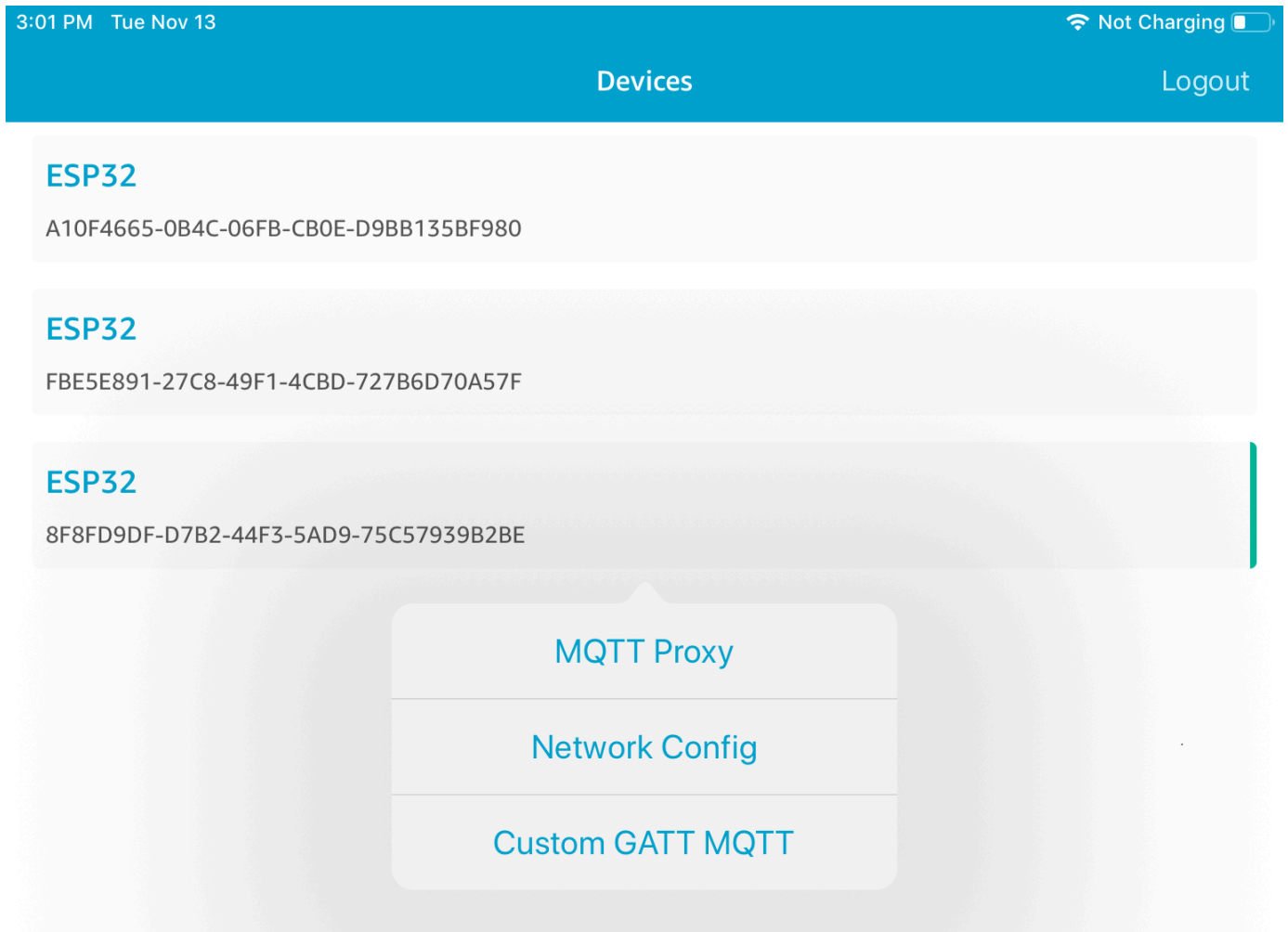
Note

Layanan Penyediaan Wi-Fi dinonaktifkan secara default.

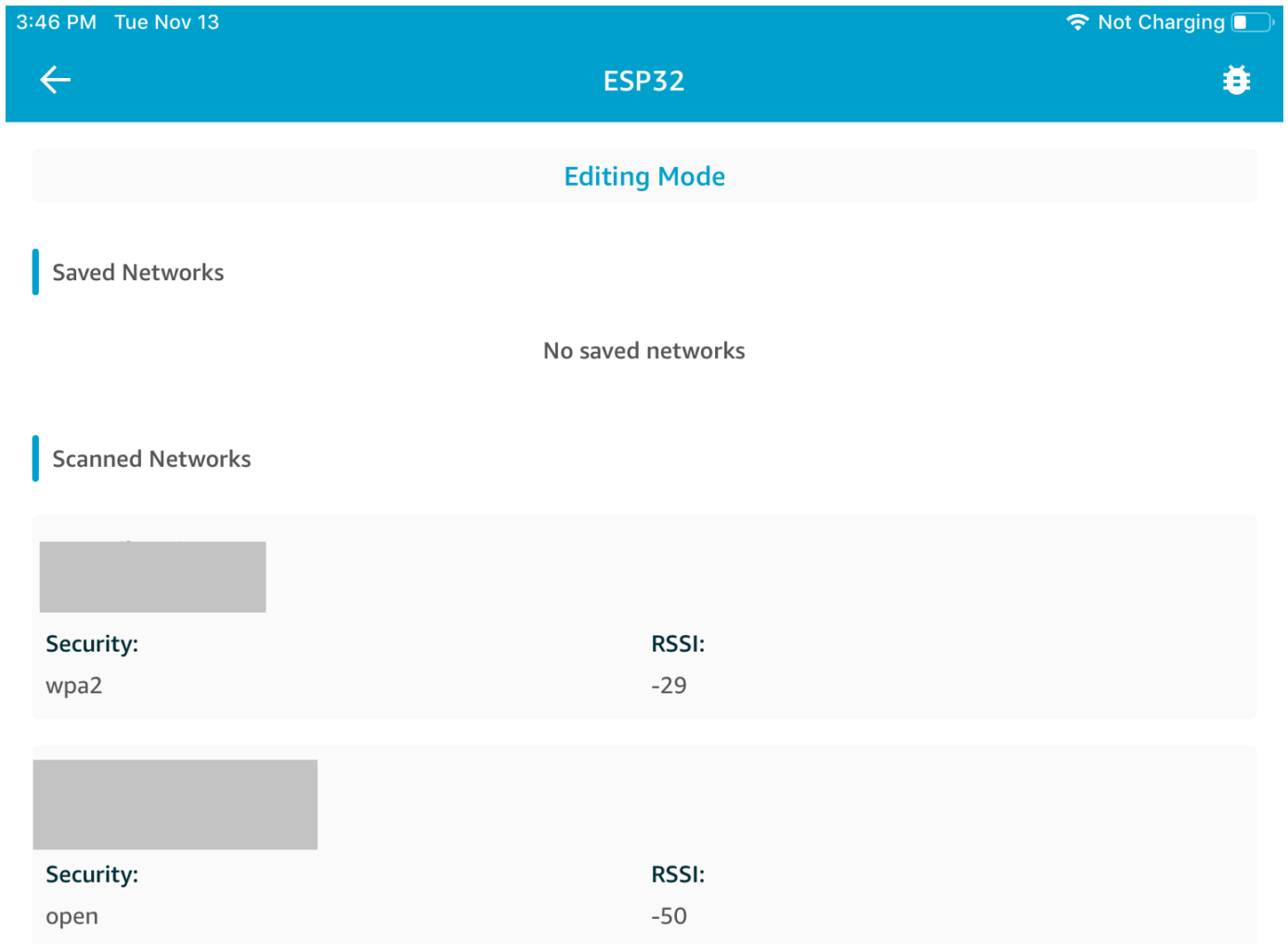
2. Konfigurasi [Network Manager](#) untuk mengaktifkan Bluetooth Low Energy dan Wi-Fi.

Untuk menjalankan demo

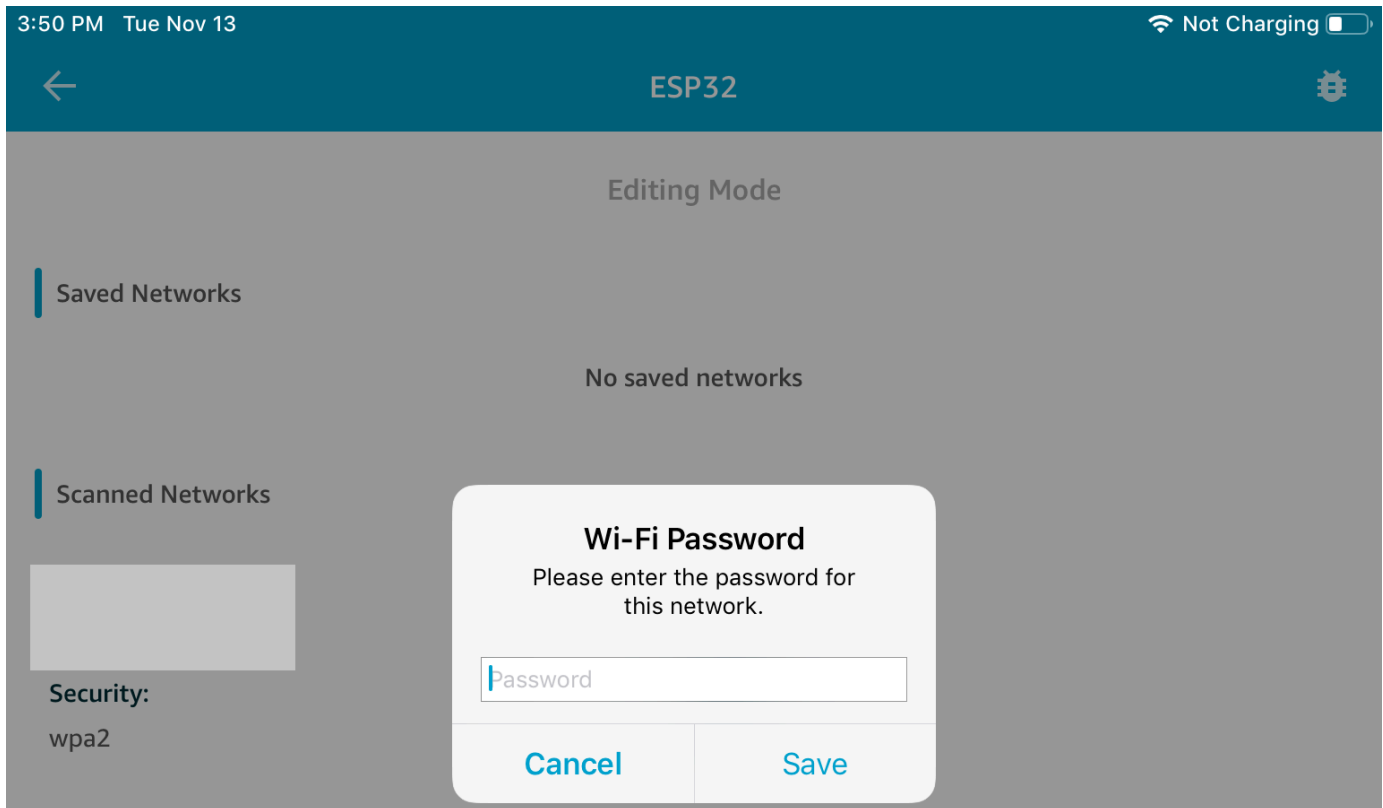
1. Membangun dan menjalankan proyek demo pada mikrokontroler Anda.
2. Pastikan bahwa Anda telah memasang mikrokontroler Anda dan perangkat mobile Anda menggunakan [Aplikasi demo SDK Seluler FreeRTOS Bluetooth Rendah Energi](#).
3. Dari daftar Perangkat di aplikasi seluler demo, pilih mikrokontroler Anda, lalu pilih Network Config untuk membuka pengaturan konfigurasi jaringan.



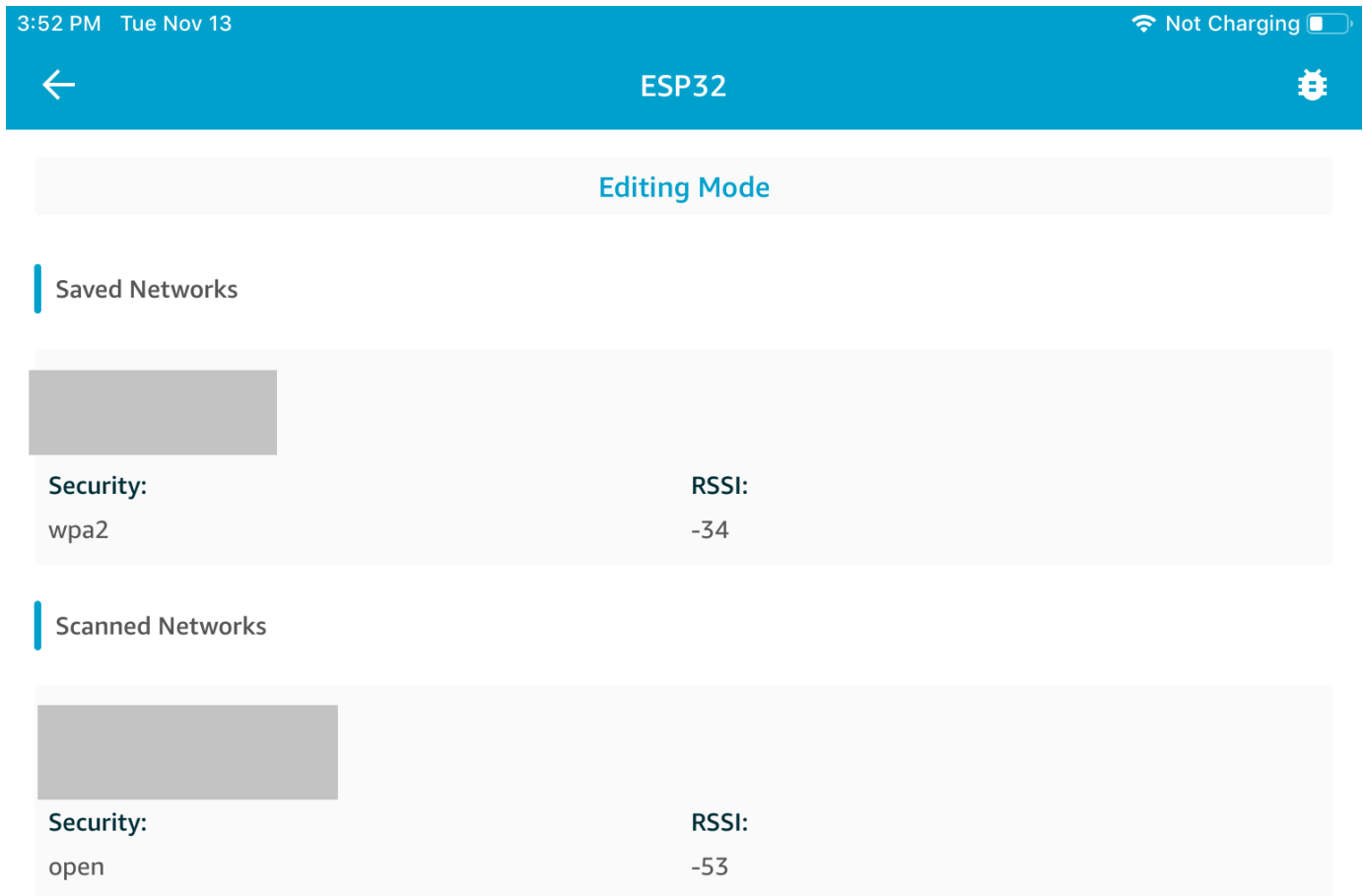
4. Setelah Anda memilih Network Config untuk papan Anda, mikrokontroler mengirimkan daftar jaringan di sekitarnya ke perangkat seluler. Jaringan Wi-Fi yang tersedia muncul dalam daftar di bawah Jaringan yang dipindai.



Dari daftar Jaringan yang dipindai, pilih jaringan Anda, lalu masukkan SSID dan kata sandi, jika diperlukan.

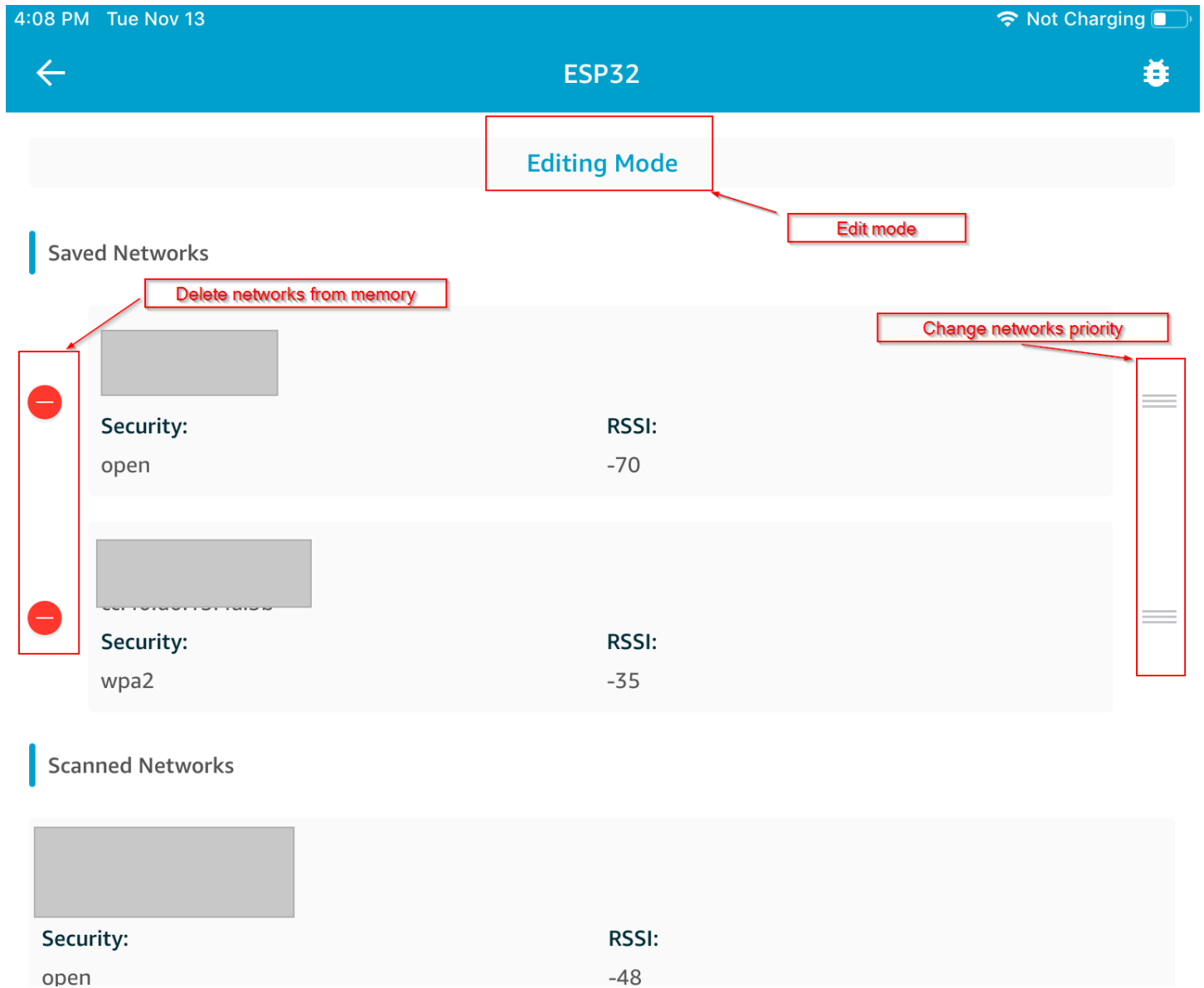


Mikrokontroler terhubung ke dan menyimpan jaringan. Jaringan muncul di bawah Jaringan Tersimpan.



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 kembali, 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, demo Generic Attributes (GATT) Server aplikasi pada mikrokontroler Anda mengirimkan nilai counter sederhana untuk [Aplikasi demo SDK Seluler FreeRTOS Bluetooth Rendah Energi](#).

Menggunakan Bluetooth Low Energy Mobile SDK, Anda dapat membuat klien GATT Anda sendiri untuk perangkat mobile yang terhubung ke server GATT pada mikrokontroler Anda dan berjalan secara parallel dengan aplikasi mobile 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 *papan* adalah nama papan yang Anda gunakan untuk menjalankan demo), tambahkan `#define IOT_BLE_ADD_CUSTOM_SERVICES (1)` ke daftar pernyataan `define`.

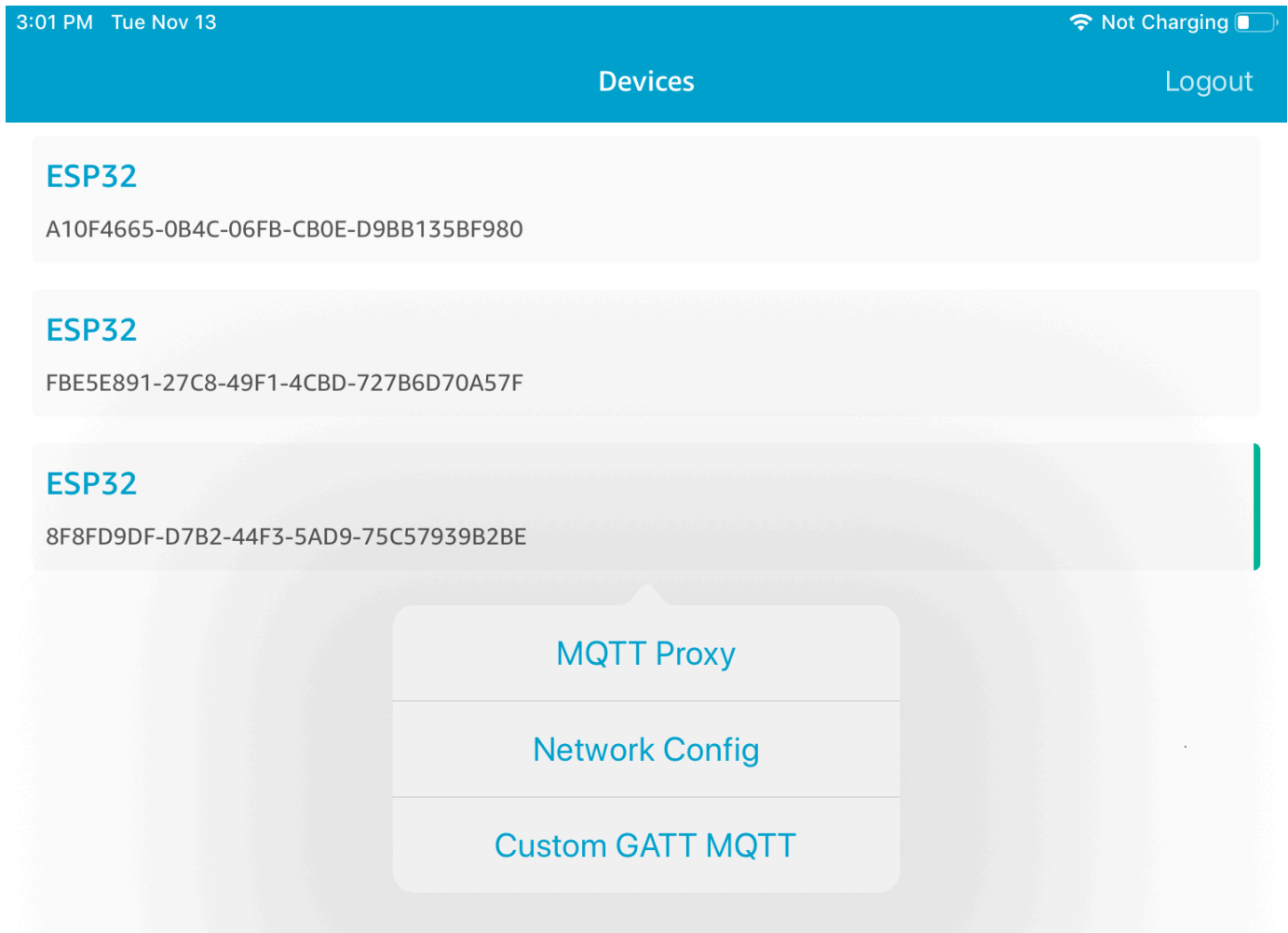
Note

Demo GATT Bluetooth Low Energy dinonaktifkan secara default.

2. Buka `freertos/vendors/vendor/boards/board/aws_demos/config_files/aws_demo_config.h`, komentari `#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED`, dan tentukan `CONFIG_BLE_GATT_SERVER_DEMO_ENABLED`.

Untuk menjalankan demo

1. Membangun dan menjalankan proyek demo pada mikrokontroler Anda.
2. Pastikan bahwa Anda telah memasang papan Anda dan perangkat mobile Anda menggunakan [Aplikasi demo SDK Seluler FreeRTOS Bluetooth Rendah Energi](#).
3. Dari daftar Perangkat di aplikasi, pilih papan Anda, lalu pilih Proksi MQTT untuk membuka opsi proxy MQTT.



4. Kembali ke daftar Perangkat, pilih papan Anda, lalu pilih Kustom GATT MQTT untuk membuka opsi layanan GATT khusus.
5. Pilih Start Counter untuk mulai menerbitkan data ke topik ***your-thing-name/example/topic*** MQTT.

Setelah Anda mengaktifkan proxy MQTT, Hello World dan incrementing pesan counter muncul pada ***your-thing-name/example/topic*** topik.

Bootloader demo untuk Microchip Curiosity PIC32MZEF

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 PIC32MZE (DM320104) dari cabang utama repositori FreeRTOS Reference Integration dan tidak akan lagi membawanya dalam rilis baru. Microchip telah mengeluarkan [pemberitahuan resmi](#) bahwa PIC32MZE (DM320104) tidak lagi direkomendasikan untuk desain baru. Proyek PIC32MZE dan kode sumber masih dapat diakses melalui tag rilis sebelumnya. Microchip merekomendasikan agar pelanggan menggunakan Curiosity [PIC32MZ-EF-2.0 Development board \(DM320209\)](#) untuk desain baru. Platform Pic32Mzv1 masih dapat ditemukan di [v202012.00](#) dari repositori Integrasi Referensi FreeRTOS. Namun, platform ini tidak lagi didukung oleh [v202107.00](#) dari Referensi FreeRTOS.

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 menggunakan algoritma tanda tangan digital kurva elips. 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:

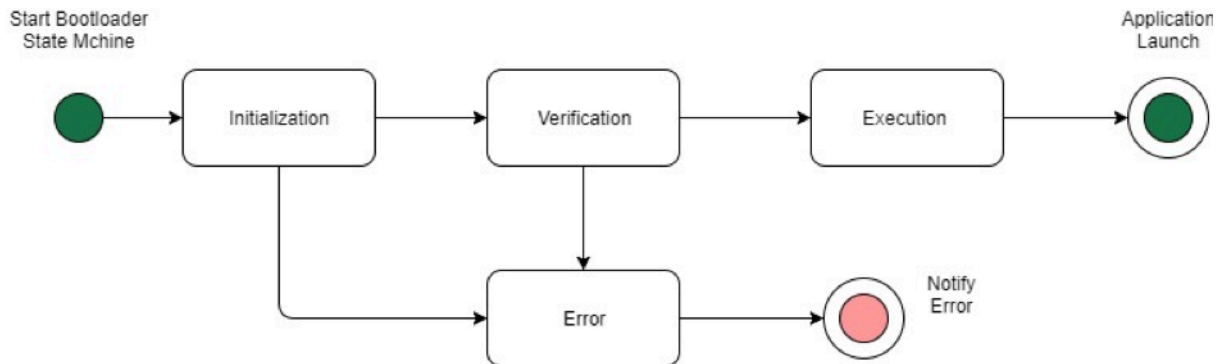
- Mempertahankan gambar aplikasi pada perangkat dan beralih di antara keduanya.
- Memungkinkan eksekusi self-test dari gambar OTA yang diterima dan rollback pada kegagalan.
- Memeriksa tanda tangan dan versi gambar pembaruan OTA.

Note

Untuk mengatur dan menjalankan demo FreeRTOS, ikuti langkah-langkah di dalamnya [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.
Mengeksekusi Default	Bootloader meluncurkan gambar default.
Kesalahan	Bootloader dalam keadaan kesalahan.

Dalam diagram sebelumnya, keduanya `Execute Image` dan `Execute Default` ditampilkan sebagai `Execution` negara.

Status Eksekusi Bootloader

Bootloader dalam `Execution` keadaan dan siap untuk 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

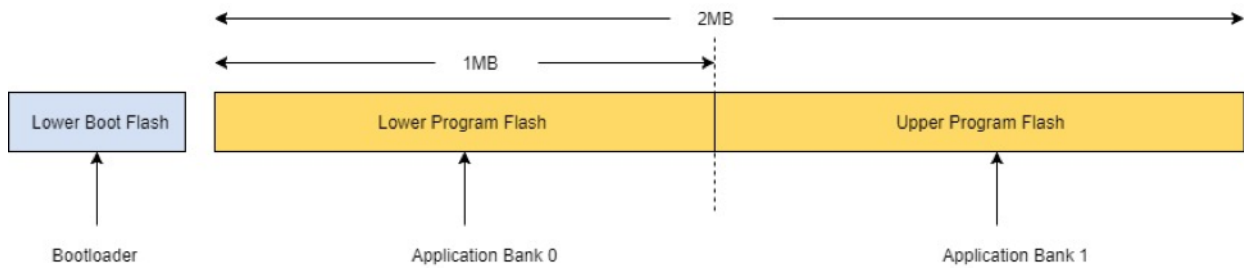
Jika opsi konfigurasi untuk meluncurkan gambar default diaktifkan, bootloader meluncurkan aplikasi dari alamat eksekusi default. Opsi ini harus dinonaktifkan kecuali saat debugging.

Status Kesalahan Bootloader

Bootloader dalam keadaan kesalahan dan tidak ada gambar yang valid yang ada di perangkat. Bootloader harus memberi tahu pengguna. Implementasi default mengirimkan pesan log ke konsol dan mengedipkan cepat LED di papan tanpa batas waktu.

Perangkat flash

Platform Microchip Curiosity PIC32MZEF berisi flash program internal dua megabyte (MB) yang dibagi menjadi dua bank. Ini mendukung pertukaran peta memori antara dua bank ini dan pembaruan langsung. Bootloader demo diprogram di wilayah flash boot bawah yang terpisah.



Struktur gambar aplikasi

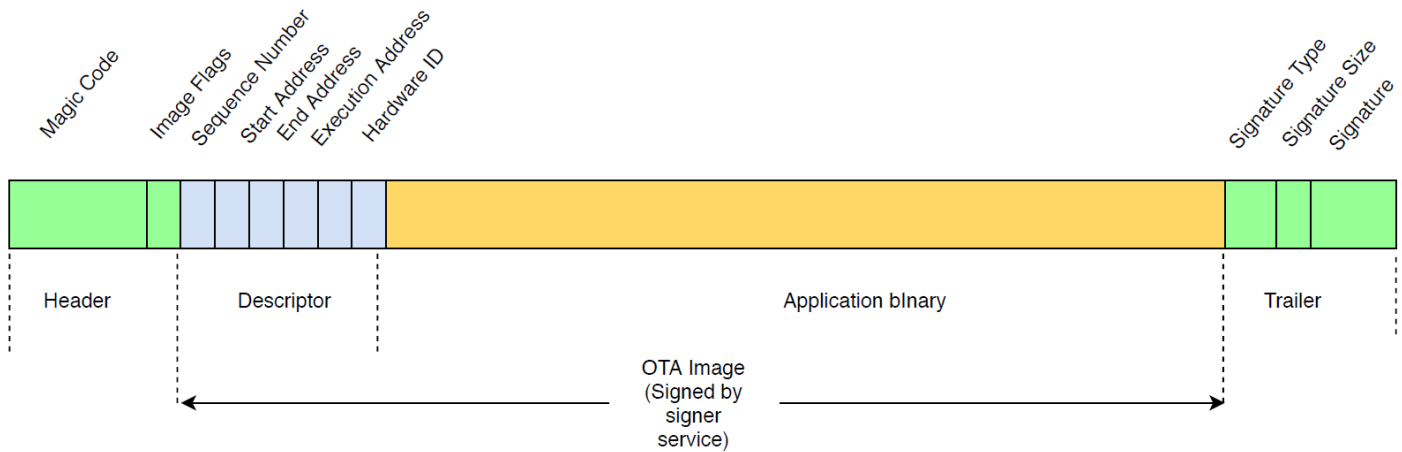


Diagram menunjukkan komponen utama dari gambar aplikasi yang disimpan di setiap bank perangkat.

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 pelaksana ditandai sebagai komit tertunda, itu berarti perangkat berada dalam fase uji mandiri OTA. Bahkan jika gambar pada perangkat ditandai valid, mereka pergi melalui langkah-langkah verifikasi yang sama pada setiap boot. Jika gambar ditandai sebagai baru, bootloader menandainya sebagai commit pending dan meluncurkannya untuk self-test setelah verifikasi. Bootloader juga menginisialisasi dan memulai pengatur waktu pengawas sehingga jika gambar OTA baru gagal melakukan self-test, perangkat reboot dan bootloader menolak gambar dengan menghapusnya dan mengeksekusi gambar valid sebelumnya.

Perangkat hanya dapat memiliki satu gambar yang valid. Gambar lainnya bisa berupa gambar OTA baru atau commit pending (self-test). Setelah pembaruan OTA yang 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. Descriptor gambar dihasilkan oleh utilitas pasca-build yang menggunakan file konfigurasi (`ota-descriptor.config`) untuk menghasilkan deskriptor yang sesuai dan menambahkannya 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
Mulai Alamat	4 byte
Alamat	4 byte
Alamat Eksekusi	4 byte
ID Perangkat	4 byte
Reserved	4 byte

Nomor Urutan

Nomor urut harus bertambah 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 sampai 4294967295.

Mulai Alamat

Alamat awal gambar aplikasi pada perangkat. Saat deskriptor gambar ditambahkan ke biner aplikasi, alamat ini adalah awal dari deskriptor gambar.

Alamat

Alamat akhir gambar aplikasi pada perangkat, tidak termasuk trailer gambar.

Alamat Eksekusi

Alamat eksekusi gambar.

ID Perangkat

ID perangkat keras unik yang digunakan oleh bootloader untuk kebenaran gambar OTA dibuat untuk platform yang benar.

Reserved

Ini disimpan untuk penggunaan future.

Gambar Trailer

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	32 byte
Ukuran	4 byte
Tanda tangan	256 byte

Jenis Tanda

Jenis tanda tangan adalah string yang mewakili algoritma kriptografi yang digunakan dan berfungsi sebagai penanda untuk trailer. Bootloader mendukung algoritma tanda tangan digital kurva elips. Standarnya adalah sig-sha256-ecdsa.

Ukuran

Ukuran tanda tangan kriptografi, dalam byte.

Tanda tangan

Tanda tangan kriptografi dari aplikasi biner prepended 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 untuk tujuan debugging saja.

Aktifkan Mulai Default

Memungkinkan eksekusi aplikasi dari alamat default dan harus diaktifkan untuk debugging saja. Gambar dijalankan 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 untuk tujuan debugging saja dan harus tetap diaktifkan dalam produksi.

Menghapus Gambar yang Tidak

Memungkinkan penghapusan bank penuh jika verifikasi gambar di bank tersebut gagal. Opsi ini disediakan untuk debugging dan harus tetap diaktifkan dalam produksi.

Aktifkan Verifikasi ID Perangkat Keras

Mengaktifkan verifikasi ID perangkat keras dalam 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 awal, akhir, dan alamat eksekusi dalam deskriptor gambar OTA. Kami menyarankan agar Anda tetap mengaktifkan opsi ini.

Membangun bootloader

Bootloader demo disertakan sebagai proyek loadable dalam `aws_demos` proyek yang terletak di `freertos/vendors/microchip/boards/curiosity_pic32mzef/aws_demos/mp1ab/` dalam repositori kode sumber FreeRTOS. Ketika `aws_demos` proyek dibangun, itu membangun bootloader pertama, diikuti oleh aplikasi. Output akhir adalah gambar hex terpadu termasuk bootloader dan aplikasi. `factory_image_generator.py` utilitas disediakan untuk menghasilkan gambar hex terpadu dengan tanda tangan kriptografi. Skrip utilitas bootloader terletak di `freertos/demos/ota/bootloader/utility/`.

Langkah pra-bangun bootloader

Langkah pra-bangun 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 dikodekan Abstrak 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 cara menggunakan library AWS IoT Device Defender untuk terhubung [AWS IoT Device Defender](#). Demo menggunakan pustaka CoreMQTT untuk membuat koneksi

MQTT melalui TLS (otentikasi bersama) ke BrokerAWS IoT MQTT dan pustaka CoreJSON untuk memvalidasi dan mengurai tanggapan yang diterima dariAWS IoT Device Defender layanan. Demo menunjukkan cara membuat laporan berformat JSON menggunakan metrik yang dikumpulkan dari perangkat, dan cara mengirimkan laporan yang dibuat keAWS IoT Device Defender layanan. Demo juga menunjukkan cara mendaftarkan fungsi callback dengan pustaka CoreMQTT untuk menangani respons dariAWS IoT Device Defender layanan untuk mengonfirmasi apakah laporan terkirim diterima atau ditolak.

Note

Untuk mengatur dan menjalankan demo FreeRTOS, ikuti langkah-langkah di dalamnya[Memulai dengan FreeRTOS](#).

Fungsionalitas

Demo ini membuat tugas aplikasi tunggal yang menunjukkan cara mengumpulkan metrik, membuat laporan pembela perangkat dalam format JSON, dan mengirimkannya keAWS IoT Device Defender layanan melalui koneksi MQTT yang aman ke BrokerAWS IoT MQTT. Demo ini mencakup metrik jaringan standar serta metrik khusus. Untuk metrik khusus, demo mencakup:

- Metrik bernama `task_numbers` yang merupakan daftar ID tugas FreeRTOS. Jenis metrik ini adalah “daftar angka”.
- Metrik bernama `stack_high_water_mark` yang merupakan tanda air tinggi tumpukan 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 dalamAWS IoT Device Defender laporan. 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 angka 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 [GitHub](#)web.

Untuk ESP32, konfigurasi LWIP default tidak menggunakan penguncian inti dan oleh karena itu demo akan menggunakan metrik berduri. Jika Anda ingin menggunakan implementasi koleksi metrik LWIP referensi, tentukan makro berikut di `lwipopts.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 sebuah contoh output.


```
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,"u": "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 [GitHub](#) situs web.

Berlangganan AWS IoT Device Defender topik

Fungsi [subscribeToDefenderTopic](#) berlangganan topik MQTT di mana tanggapan terhadap laporan Device Defender yang dipublikasikan 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

[collectDeviceMetrics](#) Fungsi 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 oleh AWS IoT Device Defender dan menuliskannya ke buffer yang disediakan. Format dokumen JSON yang diharapkan AWS IoT Device Defender ditentukan dalam [metrik sisi perangkat](#) di Panduan AWS IoT Pengembang.

Menerbitkan AWS IoT Device Defender laporan

AWS IoT Device Defender Laporan ini dipublikasikan pada topik MQTT untuk menerbitkan AWS IoT Device Defender laporan JSON. Laporan ini dibuat menggunakan makro `DEFENDER_API_JSON_PUBLISH`, seperti yang ditunjukkan dalam [cuplikan kode](#) ini di GitHub situs web.

Callback untuk menangani respons

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 AWS IoT Device Defender layanan. Jika pesan berasal dari AWS IoT Device Defender layanan, pesan akan mengurai 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 menjalankan demo AWS IoT Greengrass Discovery untuk FreeRTOS, Anda perlu menyiapkan AWS, AWS IoT Greengrass, dan AWS IoT. Untuk mengatur AWS, ikuti petunjuk di [Menyiapkan AWS akun dan izin Anda](#). Untuk mengatur AWS IoT Greengrass, Anda perlu membuat grup Greengrass dan kemudian menambahkan inti Greengrass. Untuk informasi lebih lanjut 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 Greengrass_ServiceRole.
3. Pilih Lampirkan kebijakan, pilih AmazonS3FullAccess AWSIoTFullAccess, lalu pilih Lampirkan kebijakan.
4. Jelajahi [AWS IoT konsol](#).
5. Di panel navigasi, pilih Greengrass, lalu pilih grup Greengrass yang telah Anda buat sebelumnya.
6. Pilih Pengaturan, lalu pilih Tambahkan peran.
7. Pilih Greengrass_ServiceRole, lalu pilih Simpan.

Connect papan Anda AWS IoT dan konfigurasi 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.

Membuat AWS IoT Greengrass kebijakan baru

1. Jelajahi [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 tempelkan JSON berikut ke dalam jendela editor kebijakan:

```
{
  "Effect": "Allow",
  "Action": [
    "greengrass:*"
  ],
  "Resource": "*"
}
```

Kebijakan ini memberikan AWS IoT Greengrass izin untuk semua sumber daya.

5. Pilih Create (Buat).

Untuk melampirkan AWS IoT Greengrass kebijakan ke sertifikat perangkat Anda

1. Jelajahi [AWS IoT konsol](#).
2. Di panel navigasi, pilih Kelola, lalu pilih hal yang telah Anda buat sebelumnya.
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 telah Anda buat sebelumnya, lalu pilih Lampirkan.

2. [Mengunduh FreeRTOS](#)

Note

Jika Anda mengunduh FreeRTOS dari konsol FreeRTOS, pilih Connect ke AWS IoT Greengrass - **Platform**, bukan Connect ke AWS IoT - **Platform**.

3. [Mengkonfigurasi demo FreeRTOS](#).

Buka `freertos/vendors/vendor/boards/board/aws_demos/config_files/aws_demo_config.h`, komentari `#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED`, dan tentukan `CONFIG_GREENGRASS_DISCOVERY_DEMO_ENABLED`.

Setelah Anda mengatur AWS IoT dan AWS IoT Greengrass, dan setelah mengunduh dan mengonfigurasi FreeRTOS, Anda dapat membuat, mem-flash, dan menjalankan demo Greengrass di perangkat Anda. Untuk mengatur lingkungan pengembangan perangkat keras dan perangkat lunak papan Anda, ikuti petunjuk di halaman [Panduan memulai khusus papan](#).

Demo Greengrass menerbitkan serangkaian pesan ke inti Greengrass, dan ke klien AWS IoT MQTT. Untuk melihat pesan di klien AWS IoT MQTT, buka [AWS IoT konsol](#), pilih Uji, pilih klien uji MQTT, lalu 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 instans Amazon EC2

Jika Anda bekerja dengan instans Amazon EC2

1. Temukan DNS Publik (IPv4) yang terkait dengan instans Amazon EC2 Anda — buka konsol Amazon EC2, dan di panel navigasi kiri, pilih Instans. Pilih instans Amazon EC2 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 instans Amazon EC2 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	-
Semua ICMP - IPv4	ICMP	Semua	0.0.0.0/0	-
Semua ICMP - IPv4	ICMP	Semua	::/0	-

4. DiAWS IoT konsol pilih Greengrass, lalu Grup, dan pilih grup Greengrass yang sebelumnya Anda buat. Pilih Pengaturan. Ubah Deteksi koneksi lokal ke Kelola informasi koneksi secara manual.
5. Di panel navigasi, pilih Core lalu pilih 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 sewaktu-waktu). Pilihan terbaik adalah menggunakan DNS Publik (IPv4) yang Anda catat di langkah pertama.
7. Tambahkan benda FreeRTOS IoT yang Anda buat ke grup GG.
 - a. Pilih panah belakang untuk kembali ke halamanAWS IoT Greengrass grup. Di panel navigasi, pilih Perangkat lalu pilih Tambah Perangkat.
 - b. Pilih Pilih Hal IoT. 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 deployment AWS IoT Greengrass grup Anda dan pastikan bahwa deployment berhasil. Anda sekarang dapat 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 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 merelai pesan (termasuk fungsi Lambda) antara perangkat klien, layanan AWS IoT Core cloud, dan komponen Greengrass, deploy dan konfigurasi [Komponen jembatan MQTT](#).
- Terapkan [komponen detektor IP](#) untuk secara otomatis mendeteksi informasi konektivitas, 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 AWS IoT Greengrass V2](#).

Demo inti HTTP

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 perpustakaan CoreHTTP.

Topik

- [Demo otentikasi timbal balik CoreHTTP](#)
- [Demo unggahan Amazon S3 dasar CoreHTTP](#)
- [Demo unduh S3 dasar S3](#)
- [Inti HTTP 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 cara membuat koneksi ke server HTTP menggunakan TLS dengan otentikasi timbal balik antara klien dan server. Demo ini menggunakan implementasi antarmuka transportasi berbasis MbedTLS untuk membuat koneksi TLS server dan klien yang diautentikasi, dan menunjukkan alur kerja respons permintaan di HTTP.

Note

Untuk mengatur dan menjalankan demo FreeRTOS, ikuti langkah-langkah di dalamnya [Memulai dengan FreeRTOS](#).

Fungsionalitas

Demo ini membuat tugas aplikasi tunggal dengan contoh-contoh yang menunjukkan bagaimana untuk menyelesaikan berikut:

- Connect ke server HTTP di AWS IoT titik akhir.
- Mengirim permintaan POST.
- Menerima respon.
- Putuskan hubungan 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' (35.166.192.88) 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 2842 [iot_thread] [INFO][HTTPDemo][http_demo_mutual_auth.c:393] 22 2842 [iot_thread] Sending HTTP POST request to a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com/topics/topic?qos=1...23 2842 [iot_thread]
24 2882 [iot_thread] [INFO][HTTPDemo][http_demo_mutual_auth.c:418] 25 2882 [iot_thread] Received HTTP response from a2zk5tjv9x07ct-ats.iot.us-west-2.amazonaws.com/topics/topic?qos=1...
26 2882 [iot_thread]
27 2882 [iot_thread] [INFO][HTTPDemo][http_demo_mutual_auth.c:283] 28 2882 [iot_thread] Demo completed successfully.29 2882 [iot_thread]
30 2882 [iot_thread] [INFO][DEMO][2882] memory_metrics::freertos_heap::before::bytes::2088152
31 2882 [iot_thread] [INFO][DEMO][2882] memory_metrics::freertos_heap::after::bytes::1990104
32 2882 [iot_thread] [INFO][DEMO][2882] memory_metrics::demo_task_stack::before::bytes::1908
33 2882 [iot_thread] [INFO][DEMO][2882] memory_metrics::demo_task_stack::after::bytes::1908
34 3882 [iot_thread] [INFO][DEMO][3882] Demo completed successfully.
35 3884 [iot_thread] [INFO][INIT][3884] SDK cleanup done.
36 3884 [iot_thread] [INFO][DEMO][3884] -----DEMO FINISHED-----

```

AWS IoT Konsol menghasilkan output yang terlihat seperti berikut:

The screenshot shows the AWS IoT console interface. At the top, there is a 'Publish' section with a text input field containing '#' and a 'Publish to topic' button. Below this is a terminal window showing the output of the demo, which includes the message: "Hello from AWS IoT console". At the bottom, there is a 'topic' section with a date and time stamp (November 20, 2020, 19:09:09 (UTC-0800)) and a 'Export Hide' button. The message content is displayed as: {"message": "Hello, world"}.

Organisasi sumber kode sumber

File sumber demo diberi nama `http_demo_mutual_auth.c` dan dapat ditemukan di [freertos/demos/coreHTTP/](#) direktori dan di [GitHub](#) situs web.

Menghubungkan ke server AWS IoT HTTP

[connectToServerWithBackoffRetries](#) Fungsi ini mencoba untuk membuat koneksi TLS yang saling diautentikasi ke server AWS IoT HTTP. Jika koneksi gagal, ia akan mencoba ulang 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 timeout yang meningkat secara eksponensial dan kembali `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 bagaimana mengirim permintaan POST ke server AWS IoT HTTP. Untuk informasi selengkapnya tentang membuat permintaan ke REST API AWS IoT, lihat [Protokol komunikasi perangkat - HTTPS](#). Respon diterima dengan panggilan API CoreHTTP yang sama, `HTTPClient_Send`.

Demo unggahan Amazon S3 dasar 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

Contoh ini menunjukkan cara mengirim permintaan PUT ke server HTTP Amazon Simple Storage Service (Amazon S3) dan mengunggah file kecil. Tindakan ini juga melakukan permintaan GET untuk memverifikasi ukuran file setelah unggahan. Contoh ini menggunakan [antarmuka transport 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-langkah di dalamnya [Memulai dengan FreeRTOS](#).

Berulir tunggal versus multi ulir

Ada dua model penggunaan CoreHTTP, single threaded dan multithreaded (multitasking). Meskipun demo di bagian ini menjalankan pustaka HTTP di thread, itu sebenarnya menunjukkan bagaimana menggunakan CoreHTTP dalam lingkungan berulir tunggal. Hanya satu tugas dalam demo ini menggunakan HTTP API. Meskipun aplikasi berulir tunggal harus berulang kali memanggil pustaka HTTP, aplikasi multithreaded malah dapat mengirim permintaan HTTP di latar belakang dalam tugas agen (atau daemon).

Organisasi sumber kode sumber

File sumber demo diberi nama `http_demo_s3_upload.c` dan dapat ditemukan di `freertos/demos/coreHTTP/` direktori dan di [GitHub](#) situs web.

Mengonfigurasi koneksi server HTTP Amazon S3 S3

Demo ini menggunakan URL yang telah ditandatangani sebelumnya untuk terhubung ke server HTTP Amazon S3 dan mengotorisasi akses ke objek yang akan 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. Menyiapkan AWS akun:
 - a. Jika belum, [buat AWS akun](#).
 - b. Akun dan izin diatur 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 dan AWS IoT dengan menambahkan kebijakan ini:
 - AmazonS3FullAccess

2. Buat bucket di Amazon S3 dengan mengikuti langkah-langkah di [Bagaimana cara membuat bucket S3?](#) dalam Panduan Pengguna Amazon Simple Storage Service.
3. Unggah file ke Amazon S3 dengan mengikuti langkah-langkah di [Bagaimana cara mengunggah file dan folder ke bucket S3? Mengunggah file ke bucket S3? Mengunggah file ke bucket S3? Mengunggah file ke bucket S3? Mengunggah file ke bucket S](#) .
4. Buat 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 kali terhubung ke server HTTP Amazon S3 dengan otentikasi server TLS. Kemudian, itu menciptakan permintaan HTTP untuk meng-upload 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 [GitHub](#) web.

Menghubungkan ke server HTTP Amazon S3

[connectToServerWithBackoffRetries](#) Fungsi ini mencoba untuk membuat koneksi TCP ke server HTTP. Jika koneksi gagal, ia akan mencoba ulang 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.

`privConnectToServer` Fungsi ini menunjukkan cara membuat koneksi ke server HTTP Amazon S3 dengan hanya menggunakan otentikasi server. Menggunakan antarmuka transportasi berbasis MbedTLS yang diimplementasikan dalam `FreeRTOS-Plus/Source/Application-Protocols/network_transport/freertos_plus_tcp/using_mbedtls/using_mbedtls.c` file. Definisi `privConnectToServer` dapat ditemukan di situs [GitHub](#) web.

Unggah data

`privUploadS3ObjectFile` 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 `privUploadS3objectFile()` dapat ditemukan di situs [GitHub](#) web.

Memverifikasi unggahan

`privVerifyS3objectFileSize` Fungsi panggilan `privGetS3objectFileSize` untuk mengambil ukuran objek dalam bucket S3. Server HTTP Amazon S3 saat ini tidak mendukung permintaan HEAD menggunakan URL yang telah ditandatangani sebelumnya, sehingga byte ke-0 diminta. Ukuran file yang terkandung dalam bidang `Content-Range` header respon ini. `206 Partial Content` Respons diharapkan dari server. Kode status respons lainnya adalah kesalahan.

Kode sumber untuk `privGetS3objectFileSize()` dapat ditemukan di situs [GitHub](#) web.

Demo unduh S3 dasar 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 jangkauan](#) untuk mengunduh file dari server HTTP Amazon S3. Permintaan rentang didukung secara native di CoreHTTP API saat Anda menggunakan `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 socket jaringan. Rentang permintaan menurunkan risiko memiliki paket jatuh, yang memerlukan transmisi ulang pada koneksi TCP, dan sehingga mereka meningkatkan konsumsi daya perangkat.

Contoh ini menggunakan [antarmuka transport 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-langkah di dalamnya [Memulai dengan FreeRTOS](#).

Berulir tunggal versus multi ulir

Ada dua model penggunaan CoreHTTP, single threaded dan multithreaded (multitasking). Meskipun demo di bagian ini menjalankan pustaka HTTP di thread, itu sebenarnya menunjukkan bagaimana menggunakan CoreHTTP dalam lingkungan berulir tunggal (hanya satu tugas yang menggunakan API HTTP dalam demo). Meskipun aplikasi berulir tunggal harus berulang kali memanggil pustaka HTTP, aplikasi multithreaded malah dapat mengirim permintaan HTTP di latar belakang dalam tugas agen (atau daemon).

Organisasi sumber kode sumber

Proyek demo diberi nama `http_demo_s3_download.c` dan dapat ditemukan di `freertos/demos/coreHTTP/` direktori dan di [GitHub](#) situs web.

Mengonfigurasi koneksi server HTTP Amazon S3 S3

Demo ini menggunakan URL yang telah ditandatangani sebelumnya untuk terhubung ke server HTTP Amazon S3 dan mengotorisasi akses ke objek yang akan 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. Menyiapkan AWS akun:
 - a. Jika belum, [buat dan aktifkan AWS akun](#).
 - b. Akun dan izin diatur 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 dan AWS IoT dengan menambahkan kebijakan ini:
 - AmazonS3FullAccess

Fungsi `privGetS3objectFileSize()` mengambil ukuran file dalam bucket S3. Connection: keep-aliveHeader 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 yang terkandung dalam bidangContent-Range header respon ini. 206 Partial ContentRespons diharapkan dari server; kode status respons lainnya yang diterima adalah kesalahan.

Kode sumber untuk `privGetS3objectFileSize()` dapat ditemukan di [GitHub](#).

Setelah mengambil ukuran file, demo ini menciptakan permintaan rentang baru untuk setiap rentang byte file yang akan diunduh. Menggunakan `HTTPClient_AddRangeHeader()` untuk setiap bagian dari file.

Mengirim permintaan jangkauan dan menerima tanggapan

Fungsi `privDownloadS3objectFile()` mengirimkan permintaan rentang dalam satu lingkaran sampai seluruh file diunduh. Fungsi `APIHTTPClient_Send()` mengirimkan permintaan dan menerima respons secara sinkron. Ketika fungsi kembali, respon diterima dalam `xResponse`. Status-code kemudian diverifikasi menjadi 206 Partial Content dan jumlah byte download sejauh ini bertambah dengan nilaiContent-Length header.

Kode sumber untuk `privDownloadS3objectFile()` dapat ditemukan di [GitHub](#).

Inti HTTP 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 [antrean aman thread FreeRTOS](#) untuk menahan permintaan dan tanggapan yang menunggu untuk diproses. Dalam demo ini, ada tiga tugas yang harus diperhatikan.

- Tugas utama menunggu permintaan muncul di antrian permintaan. Ini akan mengirim permintaan tersebut melalui jaringan, kemudian menempatkan respons ke dalam antrian respons.

- Tugas permintaan menciptakan HTTP objek permintaan perpustakaan untuk mengirim ke server dan menempatkan mereka ke dalam antrian permintaan. Setiap objek permintaan menentukan rentang byte dari file S3 yang telah dikonfigurasi aplikasi untuk diunduh.
- Tugas respons menunggu respons muncul di antrian respons. Ini log setiap respon 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 [query URL yang telah ditandatangani sebelumnya](#).

Organisasi sumber kode sumber

Proyek demo diberi nama `http_demo_s3_download_multithreaded.c` dan dapat ditemukan di [freertos/demos/coreHTTP/](#) direktori dan situs [GitHub](#) web.

Membangun proyek demo

Proyek demo menggunakan [edisi komunitas gratis Visual Studio](#). Untuk membangun demo:

1. Buka file `solusimqtt_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 sebelumnya, maka Anda harus memilih Platform Toolset kompatibel dengan versi Anda: Project -> RtosDemos Properties -> Platform Toolset.

Mengkonfigurasi proyek demo

Demo menggunakan [tumpukan FreeRtos+TCP TCP/IP](#), jadi ikuti instruksi yang disediakan untuk [proyek starter TCP/IP](#) untuk:

1. Instal [komponen pra-syarat](#) (seperti WinPCap).
2. Opsional [mengatur alamat IP statis atau dinamis, alamat gateway dan netmask](#).
3. Opsional [mengatur alamat MAC](#).

4. [Pilih antarmuka jaringan Ethernet](#) pada mesin host Anda.
5. Yang penting [uji koneksi jaringan Anda](#) sebelum mencoba menjalankan demo HTTP.

Mengonfigurasi koneksi server HTTP Amazon S3 S3

Ikuti petunjuk untuk [Mengonfigurasi koneksi server HTTP Amazon S3 S3](#) di CoreHTTP dasar download demo.

Fungsionalitas

Demo menciptakan tiga tugas secara total:

- Salah satu yang mengirimkan permintaan dan menerima tanggapan melalui jaringan.
- Salah satu yang menciptakan permintaan untuk mengirim.
- Salah satu yang memproses tanggapan yang diterima.

Dalam demo ini, tugas utama:

1. Menciptakan permintaan dan respon antrian.
2. Menciptakan koneksi ke server.
3. Menciptakan permintaan dan respon tugas.
4. Menunggu antrian permintaan untuk mengirim permintaan melalui jaringan.
5. Menempatkan tanggapan yang diterima melalui jaringan ke dalam antrian respons.

Permintaan tugas:

1. Menciptakan masing-masing permintaan rentang.

Tanggapannya tugas:

1. Memproses masing-masing tanggapan yang diterima.

Typedefs

Demo mendefinisikan struktur berikut untuk mendukung multithreading.

Minta item

Struktur berikut menentukan item permintaan untuk menempatkan ke dalam antrian permintaan. Item permintaan disalin ke dalam 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 mendefinisikan item respon untuk menempatkan ke dalam antrian respon. Item respons disalin ke 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 ditandatangani sebelumnya untuk alamat host untuk membuat koneksi dengan server HTTP Amazon S3.

2. Mem-parsing URL yang telah ditandatangani sebelumnya untuk path ke objek di bucket S3.
3. Menghubungkan ke server HTTP Amazon S3 menggunakan TLS dengan otentikasi server.
4. Menciptakan permintaan dan respon antrian.
5. Menciptakan permintaan dan respon tugas.

Fungsi `privHTTPDemoTask()` ini mengatur, dan memberikan status demo. Kode sumber untuk fungsi ini dapat ditemukan di [Github](#).

Dalam fungsi `privDownloadLoop()`, blok tugas utama dan menunggu permintaan dari antrian permintaan. Ketika menerima permintaan itu mengirimkannya menggunakan fungsi `APIHTTPClient_Send()`. Jika fungsi API berhasil, maka menempatkan respons ke dalam antrian respons.

Kode sumber untuk `privDownloadLoop()` dapat ditemukan di [Github](#).

Tugas permintaan HTTP

Tugas permintaan ditentukan dalam fungsi `privRequestTask`. Kode sumber untuk fungsi ini dapat ditemukan di [Github](#).

Tanggapannya mengambil ukuran file dalam bucket Amazon S3 Hal ini dilakukan dalam fungsi `privGetS3ObjectFileSize`. Header “Connection: keep-alive” ditambahkan ke permintaan ini ke Amazon S3 untuk menjaga koneksi tetap terbuka setelah respons dikirim. Server HTTP Amazon S3 saat ini tidak mendukung permintaan HEAD menggunakan URL yang telah ditandatangani sebelumnya, sehingga byte ke-0 diminta. Ukuran file yang terkandung dalam bidang `Content-Range` header respon ini. `206 Partial Content` Respons diharapkan dari server; kode status respons lainnya yang diterima adalah kesalahan.

Kode sumber untuk `privGetS3ObjectFileSize` 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 untuk dikirim. Rentang file dikonfigurasi oleh pengguna demo di `makrodemoconfigRANGE_REQUEST_LENGTH`. Permintaan rentang didukung secara native di API pustaka klien HTTP menggunakan fungsi `HTTPClient_AddRangeHeader`. Fungsi `privRequestS3ObjectRange` menunjukkan bagaimana menggunakan `HTTPClient_AddRangeHeader()`.

Kode sumber untuk fungsi `privRequestS3ObjectRange` dapat ditemukan di [Github](#).

Tanggapannya HTTP

Tugas respons menunggu antrian respons untuk respons yang diterima melalui jaringan. Tugas utama mengisi antrian respons ketika berhasil menerima respons HTTP. Tugas ini memproses respons dengan mencatat kode status, header, dan isi. 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 respon ditentukan dalam fungsi `privResponseTask`. Kode sumber untuk fungsi ini dapat ditemukan di [Github](#).

AWS IoT Lowongan kerja perpustakaan 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 perpustakaan AWS IoT Pekerjaan menunjukkan kepada Anda cara terhubung ke [layanan AWS IoT Pekerjaan](#) melalui koneksi MQTT, mengambil pekerjaan dari AWS IoT, dan memprosesnya di perangkat. Proyek demo AWS IoT Jobs menggunakan [port FreeRTOS Windows](#), sehingga dapat dibangun dan dievaluasi dengan versi [Visual Studio Community](#) di Windows. Tidak diperlukan perangkat keras mikrokontroler. Demo membuat 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-langkah di dalamnya [Memulai dengan FreeRTOS](#).

Organisasi kode sumber sumber sumber sumber sumber sumber sumber sumber

Kode demo ada di `jobs_demo.c` file dan dapat ditemukan di [GitHub](#) situs web atau di `freertos/demos/jobs_for_aws/` direktori.

Konfigurasi koneksi broker AWS IoT MQTT

Dalam demo ini, Anda menggunakan koneksi MQTT ke broker AWS IoT MQTT. Koneksi ini dikonfigurasi dengan cara yang sama seperti [Demo otentikasi timbal balik CoreMQTT](#).

Fungsionalitas

Demo menunjukkan alur kerja yang digunakan untuk menerima pekerjaan dari AWS IoT dan memrosesnya 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 Referensi AWS CLI Perintah. Demo mengharuskan dokumen pekerjaan untuk memiliki set `action` kunci `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 buat sebuah tugas seperti pada contoh berikut perintah.

```
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 ditetapkan `publish` untuk mempublikasikan ulang pesan ke topik. Lihat format berikut untuk dokumen pekerjaan ini.

```
{
  "action": "publish",
  "message": "ADD_MESSAGE_HERE",
  "topic": "topic/name/here"
}
```

Demo loop sampai menerima dokumen pekerjaan dengan `action` kunci yang ditetapkan `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. Membangun koneksi MQTT menggunakan fungsi pembantu `dimqtt_demo_helpers.c`.
2. Berlangganan topik MQTT untuk `NextJobExecutionChanged` API, menggunakan fungsi pembantu `dimqtt_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 `dimqtt_demo_helpers.c`. String topik dirakit sebelumnya, menggunakan makro yang ditentukan oleh pustaka AWS IoT Jobs.
4. Berulang kali menelepon `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 `mqtt_demo_helpers.c` file.

Callback untuk pesan MQTT yang diterima

[prvEventCallback](#) Fungsi panggilan `Jobs_MatchTopic` dari perpustakaan 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.

Mengirim pembaruan untuk pekerjaan yang sedang berjalan

Fungsi [prvSendUpdateForJob\(\)](#) memanggil `Jobs_Update()` dari pustaka Jobs untuk mengisi string topik yang digunakan dalam operasi penerbitan MQTT yang segera mengikuti.

Demo CoreMQTT 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 library 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 bersama 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 berlangganan-publish MQTT di tingkat [QoS 1](#). Ini berlangganan filter topik, kemudian menerbitkan 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-langkah di dalamnya [Memulai dengan FreeRTOS](#).

Kode sumber

File sumber demo diberi nama `mqtt_demo_mutual_auth.c` dan dapat ditemukan di [freertos/demos/coreMQTT/](#) direktori dan situs [GitHub](#) web.

Fungsionalitas

Demo menciptakan tugas aplikasi tunggal yang loop melalui serangkaian contoh yang menunjukkan cara terhubung ke broker, berlangganan topik pada broker, mempublikasikan topik pada broker, lalu akhirnya, putus sambungan dari broker. Aplikasi demo keduanya berlangganan dan menerbitkan topik yang sama. Setiap kali demo menerbitkan pesan ke broker MQTT, broker mengirimkan pesan yang sama kembali ke aplikasi demo.

Penyelesaian demo yang berhasil akan menghasilkan output yang mirip dengan gambar berikut.

```

89 1548 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1798] 40 1548 [iot_thread] MQTT connection established with the broker.41 1548 [iot_thread]
42 1548 [iot_thread] [INFO] [MQTT_MutualAuth_Demo] [mqtt_demo_mutual_auth.c:675] 43 1548 [iot_thread] An MQTT connection is established with a3c4bx1snc0lp8-ats.iot.us-west-2
.amazonaws.com.44 1548 [iot_thread]
45 1548 [iot_thread] [INFO] [MQTT_MutualAuth_Demo] [mqtt_demo_mutual_auth.c:747] 46 1548 [iot_thread] Attempt to subscribe to the MQTT topic MyIOTThingTest5/example/topic.47
1548 [iot_thread]
48 1548 [iot_thread] [INFO] [MQTT_MutualAuth_Demo] [mqtt_demo_mutual_auth.c:761] 49 1548 [iot_thread] SUBSCRIBE sent for topic MyIOTThingTest5/example/topic to broker.50 154
8 [iot_thread]
51 1588 [iot_thread] [INFO] [MQTT] [core_mqtt.c:855] 52 1588 [iot_thread] Packet received. ReceivedBytes=3.53 1588 [iot_thread]
54 1588 [iot_thread] [INFO] [MQTT_MutualAuth_Demo] [mqtt_demo_mutual_auth.c:907] 55 1588 [iot_thread] Subscribed to the topic MyIOTThingTest5/example/topic with maximum QoS
1.56 1588 [iot_thread]
57 2188 [iot_thread] [INFO] [MQTT_MutualAuth_Demo] [mqtt_demo_mutual_auth.c:458] 58 2188 [iot_thread] Publish to the MQTT topic MyIOTThingTest5/example/topic.59 2188 [iot_th
read]
60 2188 [iot_thread] [INFO] [MQTT_MutualAuth_Demo] [mqtt_demo_mutual_auth.c:465] 61 2188 [iot_thread] Attempt to receive publish message from broker.62 2188 [iot_thread]
63 2248 [iot_thread] [INFO] [MQTT] [core_mqtt.c:855] 64 2248 [iot_thread] Packet received. ReceivedBytes=2.65 2248 [iot_thread]
66 2248 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1132] 67 2248 [iot_thread] Ack packet deserialized with result: MQTTSuccess.68 2248 [iot_thread]
69 2248 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1145] 70 2248 [iot_thread] State record updated. New state=MQTTPublishDone.71 2248 [iot_thread]
72 2248 [iot_thread] [INFO] [MQTT_MutualAuth_Demo] [mqtt_demo_mutual_auth.c:888] 73 2248 [iot_thread] PUBACK received for packet Id 2.74 2248 [iot_thread]
75 2248 [iot_thread] [INFO] [MQTT] [core_mqtt.c:855] 76 2248 [iot_thread] Packet received. ReceivedBytes=45.77 2248 [iot_thread]
78 2248 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1015] 79 2248 [iot_thread] De-serialized incoming PUBLISH packet: DeserializerResult=MQTTSuccess.80 2248 [iot_thread]
81 2248 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1028] 82 2248 [iot_thread] State record updated. New state=MQTTPubAckSend.83 2248 [iot_thread]
84 2248 [iot_thread] [INFO] [MQTT_MutualAuth_Demo] [mqtt_demo_mutual_auth.c:958] 85 2248 [iot_thread] Incoming QoS : 1
86 2248 [iot_thread]
87 2248 [iot_thread] [INFO] [MQTT_MutualAuth_Demo] [mqtt_demo_mutual_auth.c:969] 88 2248 [iot_thread] Incoming Publish Topic Name: MyIOTThingTest5/example/topic matches subs
cribed topic.Incoming Publish Message : Hello World!89 2248 [iot_thread]
90 2848 [iot_thread] [INFO] [MQTT_MutualAuth_Demo] [mqtt_demo_mutual_auth.c:478] 91 2848 [iot_thread] Keeping Connection Idle...92 2848 [iot_thread]
93 4848 [iot_thread] [INFO] [MQTT_MutualAuth_Demo] [mqtt_demo_mutual_auth.c:458] 94 4848 [iot_thread] Publish to the MQTT topic MyIOTThingTest5/example/topic.95 4848 [iot_th
read]
96 4848 [iot_thread] [INFO] [MQTT_MutualAuth_Demo] [mqtt_demo_mutual_auth.c:465] 97 4848 [iot_thread] Attempt to receive publish message from broker.98 4848 [iot_thread]
99 4888 [iot_thread] [INFO] [MQTT] [core_mqtt.c:855] 100 4888 [iot_thread] Packet received. ReceivedBytes=2.101 4888 [iot_thread]
102 4888 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1132] 103 4888 [iot_thread] Ack packet deserialized with result: MQTTSuccess.104 4888 [iot_thread]
105 4888 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1145] 106 4888 [iot_thread] State record updated. New state=MQTTPublishDone.107 4888 [iot_thread]
108 4888 [iot_thread] [INFO] [MQTT_MutualAuth_Demo] [mqtt_demo_mutual_auth.c:888] 109 4888 [iot_thread] PUBACK received for packet Id 3.110 4888 [iot_thread]
111 4928 [iot_thread] [INFO] [MQTT] [core_mqtt.c:855] 112 4928 [iot_thread] Packet received. ReceivedBytes=45.113 4928 [iot_thread]
114 4928 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1015] 115 4928 [iot_thread] De-serialized incoming PUBLISH packet: DeserializerResult=MQTTSuccess.116 4928 [iot_thread]
117 4928 [iot_thread] [INFO] [MQTT] [core_mqtt.c:1028] 118 4928 [iot_thread] State record updated. New state=MQTTPubAckSend.119 4928 [iot_thread]
120 4928 [iot_thread] [INFO] [MQTT_MutualAuth_Demo] [mqtt_demo_mutual_auth.c:958] 121 4928 [iot_thread] Incoming QoS : 1
122 4928 [iot_thread]
123 4928 [iot_thread] [INFO] [MQTT_MutualAuth_Demo] [mqtt_demo_mutual_auth.c:969] 124 4928 [iot_thread] Incoming Publish Topic Name: MyIOTThingTest5/example/topic matches su
bscribed topic.Incoming Publish Message : Hello World!125 4928 [iot_thread]
126 5528 [iot_thread] [INFO] [MQTT_MutualAuth_Demo] [mqtt_demo_mutual_auth.c:478] 127 5528 [iot_thread] Keeping Connection Idle...128 5528 [iot_thread]

```

AWS IoTKonsol akan menghasilkan output yang serupa dengan gambar berikut.

Publish
Specify a topic and a message to publish with a QoS of 0.

Publish to topic

```

1 | "message": "Hello from AWS IoT console"
2 |
3 |

```

MyIoTThingTest5/example/topic November 03, 2020, 13:03:57 (UTC-0800) Export Hide

We cannot display the message as JSON, and are instead displaying it as UTF-8 String.

Hello World!

MyIoTThingTest5/example/topic November 03, 2020, 13:03:52 (UTC-0800) Export Hide

We cannot display the message as JSON, and are instead displaying it as UTF-8 String.

Hello World!

MyIoTThingTest5/example/topic November 03, 2020, 13:03:47 (UTC-0800) Export Hide

We cannot display the message as JSON, and are instead displaying it as UTF-8 String.

Hello World!

MyIoTThingTest5/example/topic November 03, 2020, 13:03:43 (UTC-0800) Export Hide

Coba lagi logika dengan backoff eksponensial dan jitter eksponensial

Fungsi [prvBackoffForCoba Ulang](#) menunjukkan bagaimana operasi jaringan gagal dengan server, misalnya, koneksi TLS atau permintaan berlangganan MQTT, dapat dicoba ulang dengan backoff eksponensial dan jitter. Fungsi menghitung periode backoff untuk percobaan ulang berikutnya, dan melakukan penundaan backoff jika upaya coba lagi belum habis. Karena perhitungan periode backoff membutuhkan pembuatan angka acak, fungsi tersebut menggunakan modul PKCS11 untuk menghasilkan angka acak. Penggunaan modul PKCS11 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

[prvConnectToServerWithBackoffRetries](#) Fungsi ini mencoba untuk membuat koneksi TLS yang saling diautentikasi ke broker MQTT. Jika koneksi gagal, ia akan mencoba ulang setelah periode backoff. Periode backoff akan meningkat secara eksponensial hingga jumlah upaya maksimum tercapai atau periode backoff maksimum tercapai.

`BackoffAlgorithm_GetNextBackoff` Fungsi ini memberikan nilai backoff yang meningkat secara eksponensial dan kembali `RetryUtilsRetriesExhausted` ketika jumlah maksimum upaya telah tercapai. `prvConnectToServerWithBackoffRetries` Fungsi mengembalikan status kegagalan jika koneksi TLS ke broker tidak dapat dibuat setelah jumlah upaya yang dikonfigurasi.

`ConnectionWithBroker` Fungsi [PRVCreateMQTT](#) 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 menetapkan detik yang masih hidup untuk `brokerxConnectInfo`.

Fungsi berikutnya menunjukkan bagaimana antarmuka transportasi TLS dan fungsi waktu diatur dalam konteks MQTT menggunakan `MQTT_Init` fungsi. Hal ini juga menunjukkan bagaimana acara callback fungsi pointer (`prvEventCallback`) diatur. Callback ini digunakan untuk melaporkan pesan masuk.

Berlangganan topik MQTT

`SubscribeWithBackoffRetries` Fungsi [PrvMqtt](#) 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 backoff eksponensial, untuk `RETRY_MAX_ATTEMPTS`.

Menerbitkan ke topik

`PublishToTopic` Fungsi [PrvMqtt](#) menunjukkan cara mempublikasikan ke topik di broker MQTT.

Menerima pesan masuk

Aplikasi mendaftarkan fungsi callback event 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 event callback yang didaftarkan oleh aplikasi. [prvEventCallback](#) Fungsi ini adalah contoh dari fungsi event callback tersebut. `prvEventCallback` memeriksa jenis paket masuk dan panggilan handler yang sesuai. Pada contoh di bawah ini, fungsi baik panggilan `prvMQTTProcessIncomingPublish()` untuk menangani pesan publikasi masuk atau `prvMQTTProcessResponse()` untuk menangani ucapan terima kasih (ACK).

Memproses paket penerbitan MQTT yang masuk

ProcessIncomingPublishFungsi [PrvMqtt](#) menunjukkan bagaimana memproses paket penerbitan 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 [PrvMqttUnsubscribeFromTopic](#).

Mengubah CA akar yang digunakan dalam demo

Secara default, demo FreeRTOS menggunakan sertifikat Amazon Root CA 1 (kunci bit RSA 2048) untuk mengautentikasi dengan AWS IoT Core server. Dimungkinkan untuk menggunakan [sertifikat CA lain untuk otentikasi server](#), termasuk sertifikat Amazon Root CA 3 (kunci ECC 256 bit). Untuk mengubah CA root untuk demo otentikasi bersama CoreMQTT:

1. Di 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 dan kemudian tempelkan dalam `"...insert here..."` teks. Hasilnya akan terlihat seperti contoh berikut ini.

```
#define democonfigROOT_CA_PEM "-----BEGIN CERTIFICATE-----\n"\n
"MIIBtjCCAVugAwIBAgITBmyf1XSXNmY/0wua2eiedgPySjAKBggqhkJOPQQDAjA5\n"\n
"MQswCQYDVQQGEwJVUzEPMA0GA1UEChMGQW1hem9uMRkwFwYDVQQDEwBBbWF6b24g\n"\n
"Um9vdCBDQSAzMB4XDTE1MDUyNjAwMDAwMFoXDTE1MDUyNjAwMDAwMFowOjEw\n"\n
"A1UEBHMCMVVMxZDZANBgNVBAoTBkFtYXpvcjEzMBcGA1UEAxMQW1hem9uIFJvb3Qg\n"\n
"Q0EgMzBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABCMxp8ZBf8ANm+gBG1bG81K1\n"\n
"ui2yEujSLtf6ycXYqm0fc4E705hr0XwzpcV0ho6AF2hiRVd9RFgdszflZwjrzT6j\n"\n
"QjBAMA8GA1UdEwEB/wQFMAMBAf8wDgYDVR0PAQH/BAQDAgGGMB0GA1UdDgQWBBSr\n"\n
"ttvXBp43rDCBG5Fwx5zEGbF4wDAKBggqhkJOPQQDAgNJADBGAiEA4IWSoxe3jfk\n"\n
"BqWTirBqYaGFy+uGh0PscGCM05nFuMQCIQCcAu/xlJyzlvnrXir4tiz+OpAUFteM\n"\n
"YyRIHN8wfdVo0w==\n"\n
"-----END CERTIFICATE-----\n"
```

4. (Opsional) Anda dapat mengubah CA root untuk demo lainnya. Ulangi langkah 1 sampai 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 bagaimana menggunakan aplikasi multithreaded untuk membuat koneksi ke brokerAWS 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 berlangganan-publish MQTT di 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-langkah di dalamnya [Memulai dengan FreeRTOS](#).

Demo ini menggunakan antrian aman thread untuk menahan perintah untuk berinteraksi dengan API MQTT. Ada dua tugas yang harus 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, kemudian 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 memasukkan 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 callback tunggal. Demo ini juga mencakup manajer berlangganan yang memungkinkan tugas untuk menentukan callback untuk memanggil pesan publikasi masuk tentang topik yang telah mereka langgani. Callback publikasi masuk agen dalam demo ini memanggil manajer langganan untuk fan out menerbitkan 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 seperti sesi sebelumnya.

Berulir tunggal vs multithreaded

Ada dua model penggunaan CoreMQTT, single threaded dan multithreaded (multitasking). Model berulir tunggal menggunakan pustaka CoreMQTT hanya dari satu thread, dan mengharuskan Anda untuk melakukan panggilan eksplisit berulang di perpustakaan 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. Ketika Anda menjalankan protokol MQTT dalam tugas agen Anda tidak harus secara eksplisit mengelola status MQTT atau memanggil fungsi MQTT_ProcessLoop API. Selain itu, saat Anda menggunakan tugas agen, beberapa tugas aplikasi dapat berbagi koneksi MQTT tunggal tanpa perlu sinkronisasi primitif seperti mutex.

Kode sumber

File sumber demo diberi nama `mqtt_agent_task.simple_sub_pub_demo.c` dan dapat ditemukan di [freertos/demos/coreMQTT_Agent/](https://github.com/FreeRTOS/FreeRTOS-Demos/tree/master/coreMQTT_Agent) direktori dan situs [GitHub](https://github.com) web.

Fungsionalitas

Demo ini membuat setidaknya dua tugas: tugas utama yang memproses permintaan untuk panggilan API MQTT, dan sejumlah subtugas yang dapat dikonfigurasi yang membuat permintaan tersebut. Dalam demo ini, tugas utama membuat subtask, memanggil loop pemrosesan, dan membersihkan sesudahnya. 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 utama

Tugas aplikasi utama, [RunCoreMQTTAgentDemo](#), menetapkan sesi MQTT, membuat subtugas, dan menjalankan loop pemrosesan [MQTTAgent_CommandLoop](#) sampai 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 dihentikan, ia terputus dari broker.

Perintah

Ketika Anda memanggil CoreMQTT Agent API itu menciptakan perintah yang dikirim ke antrian tugas agen, yang diproses di `MQTTAgent_CommandLoop()`. Pada saat perintah dibuat, callback penyelesaian opsional dan parameter konteks dapat dilewatkan. Setelah perintah yang sesuai selesai, callback penyelesaian akan dipanggil dengan konteks berlalu dan nilai-nilai kembali 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 MQTTAgentCommandContext](#).

Perintah dianggap selesai saat:

- Berlangganan, berhenti berlangganan, dan menerbitkan dengan QoS > 0: Setelah paket pengakuan yang sesuai telah diterima.
- Semua operasi lainnya: Setelah CoreMQTT API 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

pemanggilan tidak boleh menggunakan kembali struktur perintah sebelum pemanggilan callback penyelesaian. Perhatikan bahwa karena callback penyelesaian dipanggil oleh Agen MQTT, itu 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

Perintah diproses terus menerus di `MQTTAgent_CommandLoop()`. Jika tidak ada perintah untuk diproses, loop akan menunggu maksimum untuk `MQTT_AGENT_MAX_EVENT_QUEUE_WAIT_TIME` untuk ditambahkan ke antrian, dan, jika tidak ada perintah ditambahkan, itu akan menjalankan iterasi tunggal `MQTT_ProcessLoop()`. Ini memastikan bahwa MQTT Keep-Alive dikelola, dan bahwa setiap penerbitan yang masuk diterima bahkan ketika tidak ada perintah dalam antrian.

Fungsi command loop perintah akan kembali karena alasan berikut:

- Sebuah perintah mengembalikan kode status selain `MQTTSuccess`. Status kesalahan dikembalikan oleh loop perintah, sehingga Anda dapat memutuskan bagaimana untuk 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 putus (dari `MQTTAgent_Disconnect`) diproses. Loop perintah keluar sehingga TCP dapat diputuskan.
- Perintah terminate (from `MQTTAgent_Terminate`) diproses. Perintah ini juga menandai perintah apapun masih dalam antrian atau menunggu paket pengakuan sebagai kesalahan, dengan kode kembali dari `MQTTRecvFailed`.

Manajer berlangganan berlangganan berl

Karena demo menggunakan beberapa topik, manajer berlangganan adalah cara mudah untuk mengaitkan topik berlangganan dengan callback atau tugas unik. Manajer berlangganan dalam demo ini adalah single-threaded, sehingga tidak boleh digunakan oleh beberapa tugas secara bersamaan. Dalam demo ini, fungsi manajer langganan hanya dipanggil dari fungsi callback yang diteruskan ke agen MQTT, dan hanya dijalankan dengan konteks thread tugas agen.

Sederhana berlangganan-mempublikasikan Tugas

Setiap instance [prvSimpleSubscribePublishTask](#) membuat langganan ke topik MQTT, dan menciptakan operasi publikasi untuk topik itu. Untuk menunjukkan beberapa jenis publikasi, bahkan tugas bernomor menggunakan QoS 0 (yang selesai setelah paket publikasi dikirim) dan tugas ganjil menggunakan QoS 1 (yang lengkap setelah menerima paket PUBACK).

Over-the-air update aplikasi demo

FreeRTOS menyertakan aplikasi demo yang menunjukkan fungsionalitas perpustakaan over-the-air (OTA). 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 melatih perpustakaan OTA menggunakan `vrRunOTAUpdateDemo()`.
3. Menciptakan klien MQTT menggunakan `_establishMqttConnection()`.
4. Menghubungkan ke broker AWS IoT MQTT menggunakan `IotMqtt_Connect()` dan mendaftarkan callback putus MQTT: `prvNetworkDisconnectCallback`.
5. Panggilan `OTA_AgentInit()` untuk membuat tugas OTA dan mendaftarkan callback yang akan digunakan saat tugas OTA selesai.
6. Menggunakan kembali koneksi MQTT dengan `xOTAConnectionCtx.pvControlClient = _mqttConnection;`
7. Jika MQTT terputus, aplikasi menanggukhan agen OTA, mencoba menyambung kembali menggunakan penundaan eksponensial dengan jitter, dan kemudian melanjutkan agen OTA.

Sebelum Anda dapat menggunakan pembaruan OTA, selesaikan 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. Petunjuk demo khusus perangkat tersedia untuk perangkat yang memenuhi syarat Freertos berikut:

- [Texas Instruments CC3220SF-LAUNCHXL](#)
- [Keingintahuan Microchip PIC32MZEF](#)

- [Espressif](#)
- [Unduh, bangun, flash, dan jalankan demo FreeRTOS OTA di Renesas RX65N](#)

Setelah Anda membangun, flash, dan menjalankan aplikasi demo OTA pada 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. Membuat catatan dari setiap kesalahan yang dihasilkan selama proses.

Pekerjaan pembaruan OTA yang sukses menampilkan output seperti berikut. 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-
a3b0cf83ddb
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-a3b0cf83ddb]
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-a3b0cf83ddb/update to br63 971 [MQTT Agent Task] [core_mqtt.c:1045] [INFO] [MQTT]
De-serialized incoming PUBLISH packet: DeserializerResult=MQTTSuccess.
65 973 [MQTT Agent Task] [core_mqtt.c:1058] [INFO] [MQTT] State record updated. New
state=MQTTPublishDone.
64 973 [OTA Agent Task] [ota_demo_core_mqtt.c:902] [INFO] [MQTT] Successfully
updated with the new image.
```

O konfigurasi ver-the-air demo

Konfigurasi demo OTA adalah opsi konfigurasi khusus demo yang disediakan di `aws_iot_ota_update_demo.c`. Konfigurasi ini berbeda dari konfigurasi perpustakaan OTA yang disediakan dalam file konfigurasi perpustakaan OTA.

OTA_DEMO_KEEP_ALIVE_DETIK

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 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 dua kali lipat 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 reconnect 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 CC3220SF-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](#).

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, konfigurasi `AWS IoT` titik akhir, SSID Wi-Fi dan kata sandi Anda, serta kunci dan sertifikat pribadi untuk papan Anda.
2. Buka `freertos/vendors/vendor/boards/board/aws_demos/config_files/aws_demo_config.h`, komentari `#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 solusi dan pastikan itu dibangun tanpa kesalahan.
4. Mulai emulator terminal dan gunakan pengaturan berikut untuk menyambung 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 pesan `AWS IoT MQTT`.

Saat dijalankan, emulator terminal harus menampilkan teks seperti berikut ini:

```
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, bangun, flash, dan jalankan demo FreeRTOS OTA di Microchip Curiosity PIC32MZEZ

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 PIC32MZEZ (DM320104) dari cabang utama repositori FreeRTOS Reference Integration dan tidak akan lagi membawanya dalam rilis baru. Microchip telah mengeluarkan [pemberitahuan resmi](#) bahwa PIC32MZEZ (DM320104) tidak lagi direkomendasikan untuk desain baru. Proyek PIC32MZEZ dan kode sumber masih dapat diakses melalui tag rilis sebelumnya. Microchip merekomendasikan

agar pelanggan menggunakan Curiosity [PIC32MZ-EF-2.0 Development board \(DM320209\)](#) untuk desain baru. Platform Pic32Mzv1 masih dapat ditemukan di [v202012.00](#) dari repositori Integrasi Referensi FreeRTOS. Namun, platform ini tidak lagi didukung oleh [v202107.00](#) dari Referensi FreeRTOS.

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 IDE MPLAB X, konfigurasi AWS IoT titik akhir, 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 isi 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 dilampirkan dalam tanda kutip.

Misalnya, sertifikat Anda akan terlihat seperti berikut:

```
"-----BEGIN CERTIFICATE-----\n"
"MIIBXTCCAQ0gAwIBAgIJAM4DeybZcTwKMAoGCCqGSM49BAMCMCEXHzAdBgNVBAMM\n"
"FnR1c3Rf621nbmVyQGftYXpvbi5jb20wHhcNMTcxMTAzMTkxODM1WhcNMTgxMTAz\n"
"MTkxODM2WjAhMR8wHQYDVQBBZZZ0ZXN0X3NpZ251ckBhbWF6b24uY29tMFkwEwYH\n"
"KoZIZj0CAQYIKoZIZj0DAQcDQgAERavZfVwL1X+E4dIF7dbkVMUn4IrJ1CAsFkc8\n"
"gzXpzn683H40XMK1tDZPEwr9ng78w9+QYQg7ygnr2stz8yhh06MkMCIwCwYDVR0P\n"
"BAQDAgeAMBGA1UdJQQMMAoGCCsGAQUFBwMDMAoGCCqGSM49BAMCA0gAMEUCIF0R\n"
"r5cb7rEUNtW0vGd05Macrg0ABfSoVYvB0K9fP63WAqt5h3BaS123coKSGg84twlq\n"
"Tk0/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 sertifikat `aws_ota_codesigner_certificate.pem`.
7. Buka `freertos/vendors/vendor/boards/board/aws_demos/config_files/aws_demo_config.h`, komentari `#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 solusi dan pastikan itu dibangun tanpa kesalahan.
9. Mulai emulator terminal dan gunakan pengaturan berikut untuk menyambung 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 itu dapat terhubung ke Wi-Fi dan broker pesan AWS IoT MQTT.

Ketika Anda menjalankan proyek, IDE MPLAB X harus membuka jendela output. Pastikan tab ICD4 dipilih. Anda akan melihat output berikut.

```
Bootloader version 00.09.00
[prvB00T_Init] Watchdog timer initialized.
[prvB00T_Init] Crypto initialized.

[prvValidateImage] Validating image at Bank : 0
[prvValidateImage] No application image or magic code present at: 0xbd000000
[prvB00T_ValidateImages] Validation failed for image at 0xbd000000

[prvValidateImage] Validating image at Bank : 1
[prvValidateImage] No application image or magic code present at: 0xbd100000
[prvB00T_ValidateImages] Validation failed for image at 0xbd100000
```

```

[privBOOT_ValidateImages] Booting default image.

>0 36246 [IP-task] vDHCPPProcess: offer ac140a0eip
                                     1 36297 [IP-task] vDHCPPProcess: 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] [privSubscribeToJobNotificationTopics] OK: $aws/things/devthingota/
jobs/$next/get/accepted
11 38863 [OTA Task] [privSubscribeToJobNotificationTopics] 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] [privParseJSONbyModel] Extracted parameter [ clientToken:
0:devthingota ]
15 38973 [OTA Task] [privParseJSONbyModel] parameter not present: execution
16 38973 [OTA Task] [privParseJSONbyModel] parameter not present: jobId
17 38973 [OTA Task] [privParseJSONbyModel] parameter not present: jobDocument
18 38973 [OTA Task] [privParseJSONbyModel] parameter not present: streamname
19 38973 [OTA Task] [privParseJSONbyModel] parameter not present: files
20 38975 [OTA Task] [privParseJSONbyModel] parameter not present: filepath
21 38975 [OTA Task] [privParseJSONbyModel] parameter not present: filesize
22 38975 [OTA Task] [privParseJSONbyModel] parameter not present: fileid
23 38975 [OTA Task] [privParseJSONbyModel] parameter not present: certfile
24 38975 [OTA Task] [privParseJSONbyModel] parameter not present: sig-sha256-ecdsa
25 38975 [OTA Task] [privParseJobDoc] Ignoring job without ID.
26 38975 [OTA Task] [privOTA_Close] Context->0x8003b620
27 38975 [OTA Task] [privPAL_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 ini:

```
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] vDHCPPProcess: offer c0a804beip
3 7230 [IP-task] vDHCPPProcess: 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 PIC32MZEF 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 dalam [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`, komentari `#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED`, dan tentukan `CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED` atau `CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED`.
4. Membangun proyek demo dengan berjalan `make` di `vendors/esp8266/boards/esp32/aws_demos` direktori. Anda dapat mem-flash program demo dan memverifikasi outputnya dengan menjalankan `make 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`, komentari `#define CONFIG_CORE_MQTT_MUTUAL_AUTH_DEMO_ENABLED`, dan tentukan `CONFIG_OTA_MQTT_UPDATE_DEMO_ENABLED` atau `CONFIG_OTA_HTTP_UPDATE_DEMO_ENABLED`.
 - Buka `vendors/vendor/boards/board/aws_demos/config_files/ota_demo_config.h`, dan salin sertifikat penandatanganan kode SHA-256/ECDSA Anda di:

```
#define ota_palconfig_CODE_SIGNING_CERTIFICATE [] = "your-certificate-key";
```

Unduh, bangun, flash, dan jalankan demo FreeRTOS OTA di Renesas RX65N

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 caranya download, membangun, flash dan menjalankan aplikasi demo FreeRTOS OTA pada Renesas RX65N.

Topik

- [Menyiapkan lingkungan pengoperasian](#)

- [Menyiapkan AWS sumber daya](#)
- [Impor, konfigurasi file header dan buat aws_demos dan boot_loader](#)

Menyiapkan lingkungan pengoperasian

Prosedur di bagian ini menggunakan lingkungan berikut:

- IDE: e² studio 7.8.0, dan e² studio 2020-07
- Toolchain: CCRX Compiler v3.0.1
- Perangkat target: RSKRX65N-2MB
- Debugger: E², E² Lite emulator
- Perangkat lunak: Renesas Flash Programmer, Renesas Secure Flash Programmer.exe, Tera Term

Menyiapkan perangkat keras

1. Connect emulator E² Lite dan port serial USB ke papan dan PC RX65N Anda.
2. Connect sumber daya ke RX65N.

Menyiapkan AWS sumber daya

1. Untuk menjalankan demo FreeRTOS, Anda harus memiliki AWS akun dengan pengguna IAM yang memiliki izin untuk mengakses AWS IoT layanan. Jika belum, ikuti langkah-langkah di [Menyiapkan AWS akun dan izin Anda](#).
2. Untuk mengatur pembaruan OTA, ikuti langkah-langkah di [Prasyarat pembaruan OTA](#). Secara khusus, ikuti langkah-langkah di [Prasyarat untuk pembaruan OTA menggunakan MQTT](#).
3. Buka [konsol AWS IoT](#).
4. Di panel navigasi kiri, pilih Kelola, lalu pilih Hal-hal untuk membuat sesuatu.

Suatu hal 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 instance 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 hal yang membantu Anda mengelola hal-hal 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.

Download these files and save them in a safe place. Certificates can be retrieved at any time, but the private and public keys cannot be retrieved after you close this page.

In order to connect a device, you need to download the following:

A certificate for this thing	9dba40d984.cert.pem	Download
A public key	9dba40d984.public.key	Download
A private key	9dba40d984.private.key	Download

You also need to download a root CA for AWS IoT:
A root CA for AWS IoT [Download](#)

[Activate](#)

[Cancel](#) [Done](#) [Attach a policy](#)

- f. Pilih kebijakan yang Anda buat [Kebijakan perangkat](#).

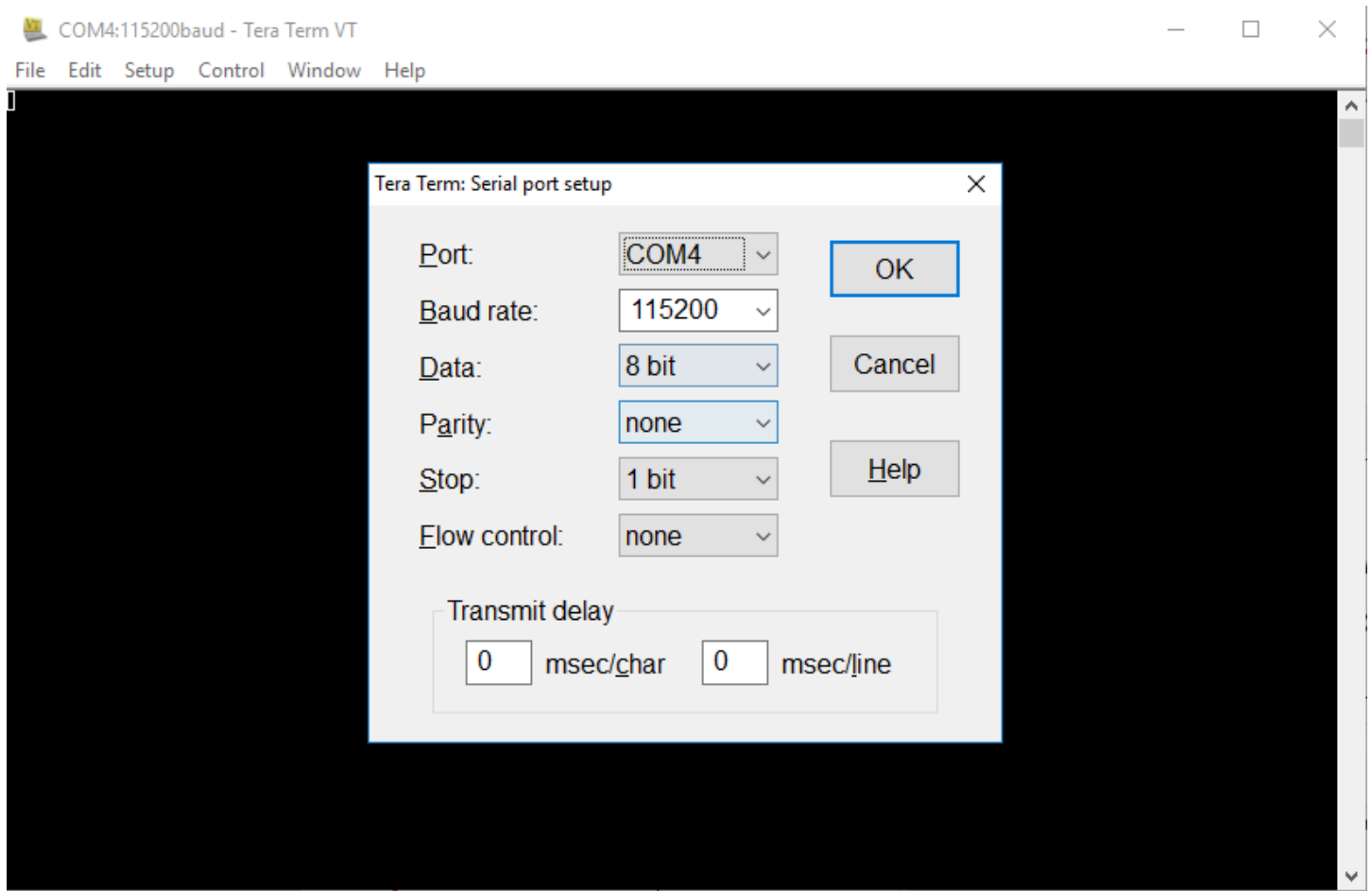
Setiap perangkat yang menerima pembaruan OTA menggunakan MQTT harus terdaftar sebagai sesuatu AWS IoT dan hal tersebut harus memiliki kebijakan terlampir seperti yang terdaftar. Anda dapat menemukan informasi lebih lanjut tentang item di "Action" dan "Resource" objek di [Tindakan Kebijakan AWS IoT Inti](#) dan [Sumber Daya Tindakan AWS IoT Inti](#).

Catatan

- `iot:Connect` izin ini memungkinkan perangkat untuk terhubung AWS IoT ke MQTT.
- `iot:Subscribe` dan `iot:Publish` izin pada topik AWS IoT pekerjaan (`.../jobs/*`) memungkinkan perangkat yang terhubung untuk menerima pemberitahuan pekerjaan dan dokumen pekerjaan, dan untuk mempublikasikan status penyelesaian eksekusi pekerjaan.
- `iot:Subscribe` dan `iot:Publish` izin pada 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:Receivelzin` memungkinkan AWS IoT Core untuk mempublikasikan pesan pada topik tersebut ke perangkat yang terhubung. Izin ini diperiksa pada setiap 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 AWS.
 - a. Untuk membuat kunci dan sertifikasi, lihat bagian 7.3 “Membuat Pasangan Kunci ECDSA-SHA256 dengan OpenSSL” di [Kebijakan Desain Pembaruan Firmware Renesas MCU](#).
 - b. Buka [konsol AWS IoT](#). Di panel navigasi kiri, pilih Kelola, lalu Pekerjaan. Pilih Buat Job lalu Buat pembaruan OTA Pekerjaan.
 - c. Di bawah Pilih perangkat untuk memperbarui pilih Pilih kemudian pilih hal yang Anda buat sebelumnya. Pilih Selanjutnya.
 - d. Pada halaman pekerjaan pembaruan Create a FreeRTOS OTA, lakukan hal berikut:
 - i. Untuk Pilih protokol untuk transfer gambar firmware, pilih MQTT.
 - ii. Untuk Pilih dan tandatangi gambar firmware Anda, pilih Tandatangi 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. Jelajahi untuk memilih sertifikat (`secp256r1.crt`), kunci pribadi 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-langkah di [Berikan akses ke penandatanganan kode untuk AWS IoT](#).

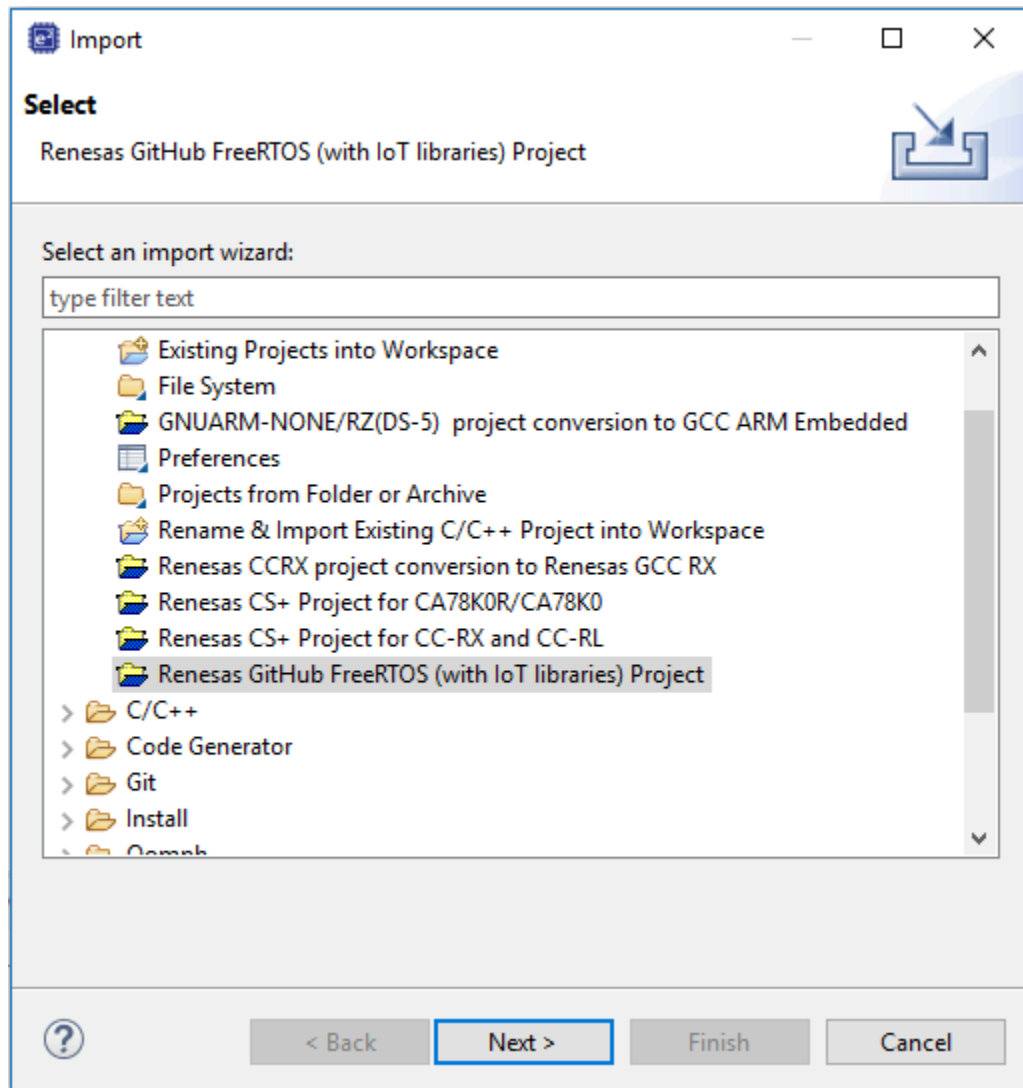
Jika tidak memiliki Tera Term diinstal pada PC Anda, Anda dapat men-download dari <https://ttssh2.osdn.jp/index.html.en> dan mengaturnya seperti yang ditunjukkan di sini. Pastikan Anda mencolokkan port Serial USB dari perangkat ke PC.



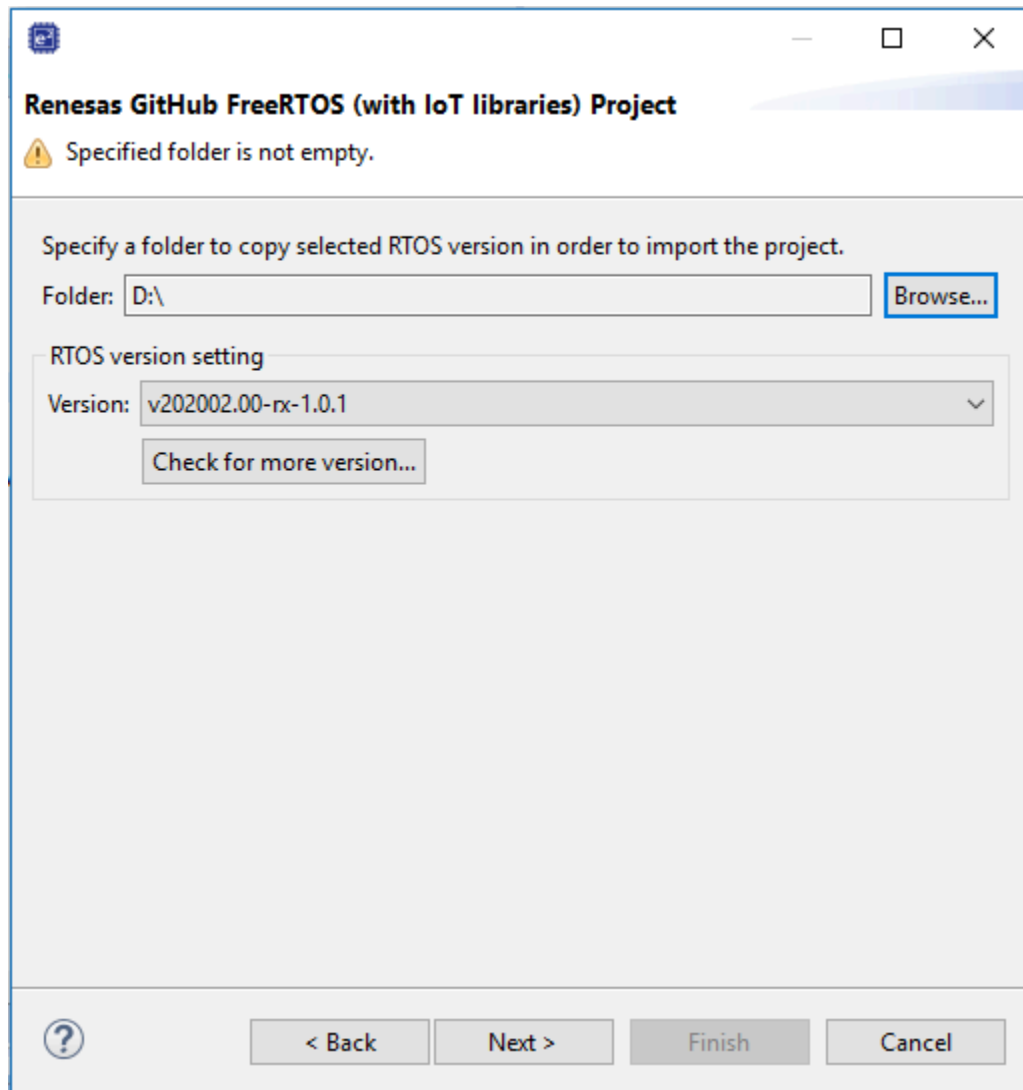
Impor, konfigurasi file header dan buat `aws_demos` dan `boot_loader`

Untuk memulai, Anda memilih versi terbaru dari paket FreeRTOS, dan ini akan diunduh dari GitHub dan diimpor secara otomatis ke dalam proyek. Dengan cara ini Anda dapat fokus pada konfigurasi FreeRTOS dan menulis kode aplikasi.

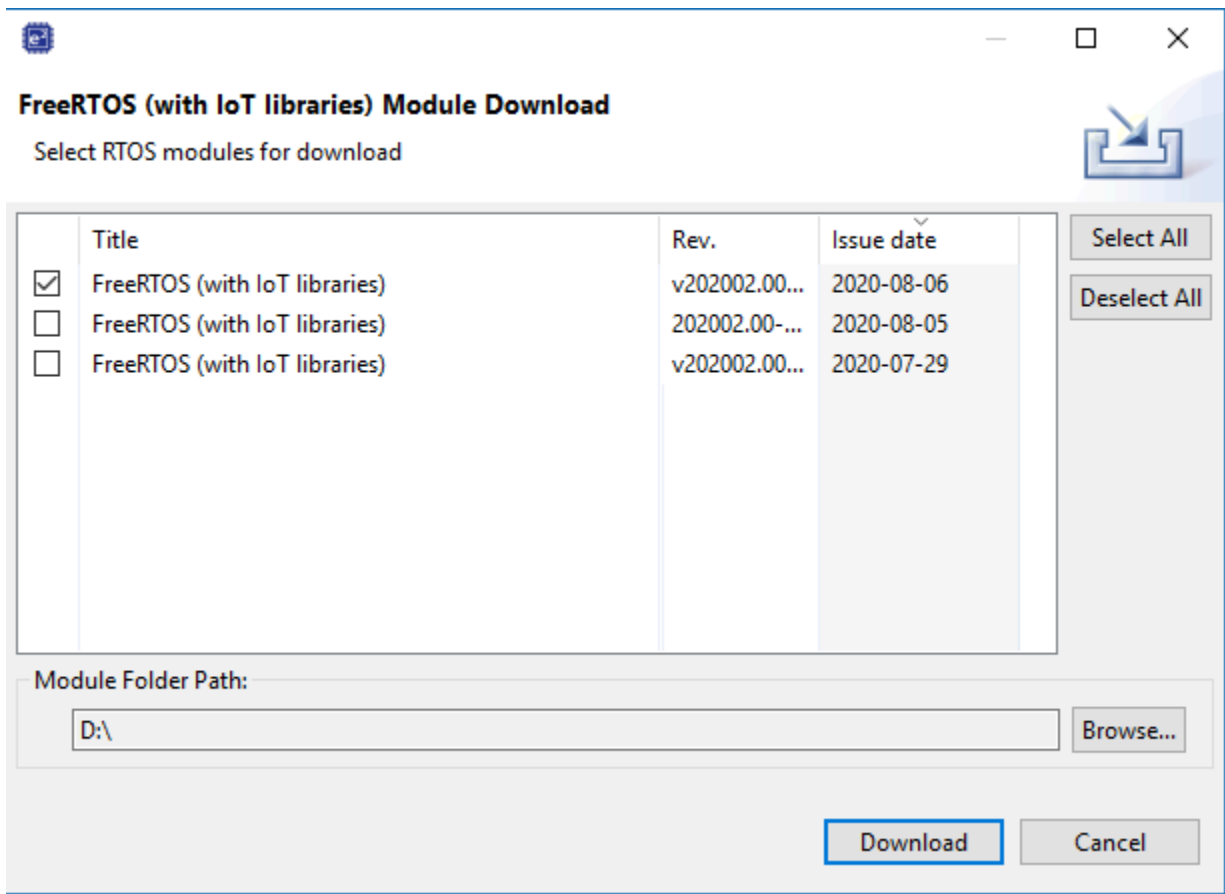
1. Peluncuran e² studio.
2. Pilih File, lalu pilih Impor...
3. Pilih Renesas GitHub FreeRTOS (dengan perpustakaan IoT) Proyek.



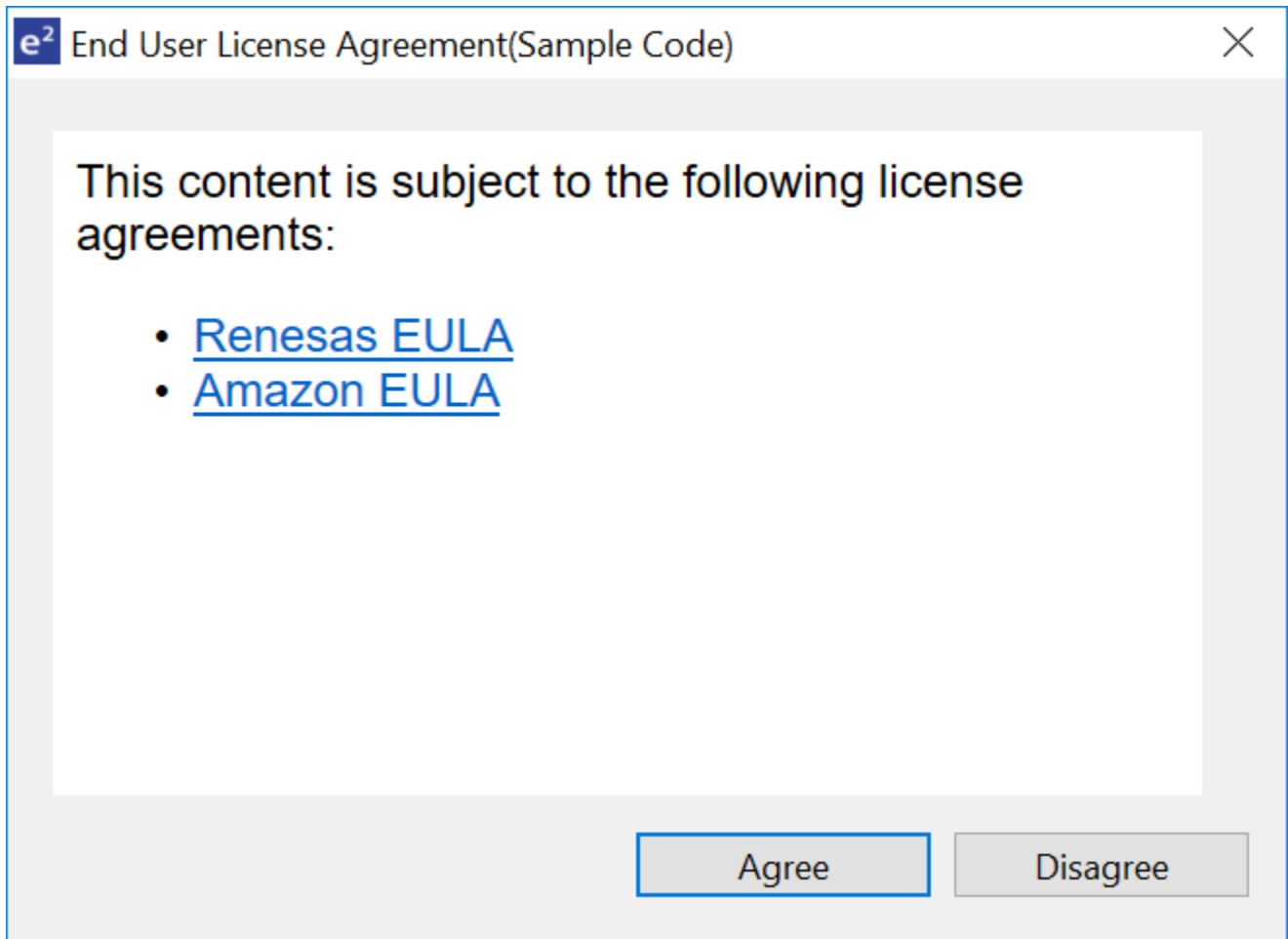
4. Pilih Periksa versi lainnya... untuk menampilkan kotak dialog unduhan.



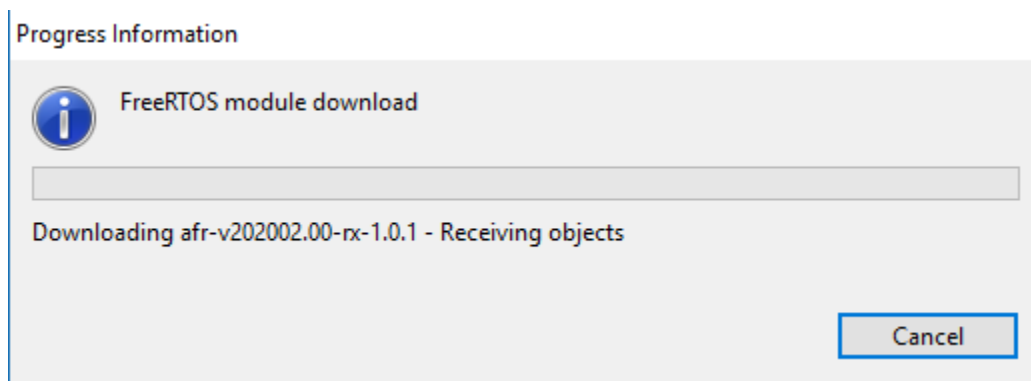
5. Pilih paket terbaru.



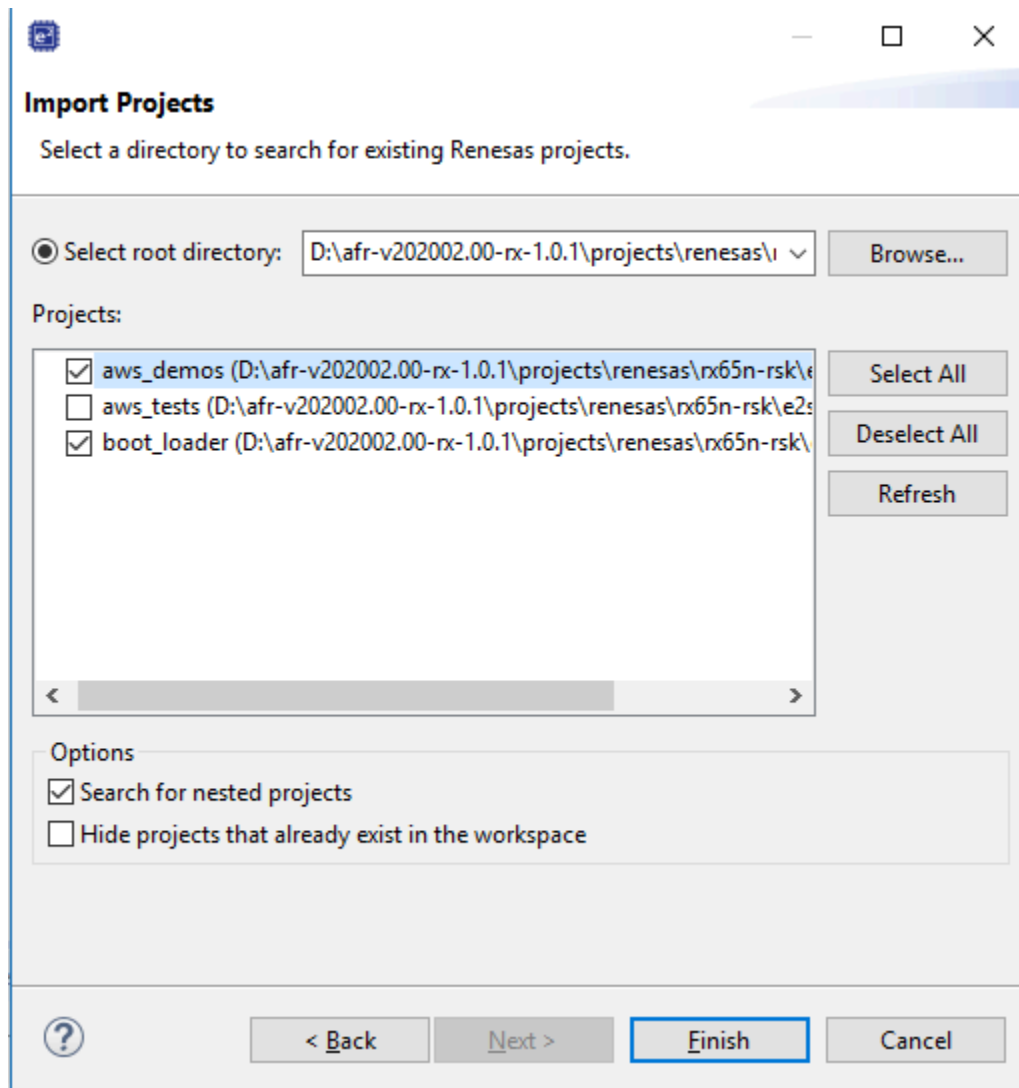
6. Pilih Setuju untuk menerima perjanjian lisensi pengguna akhir.



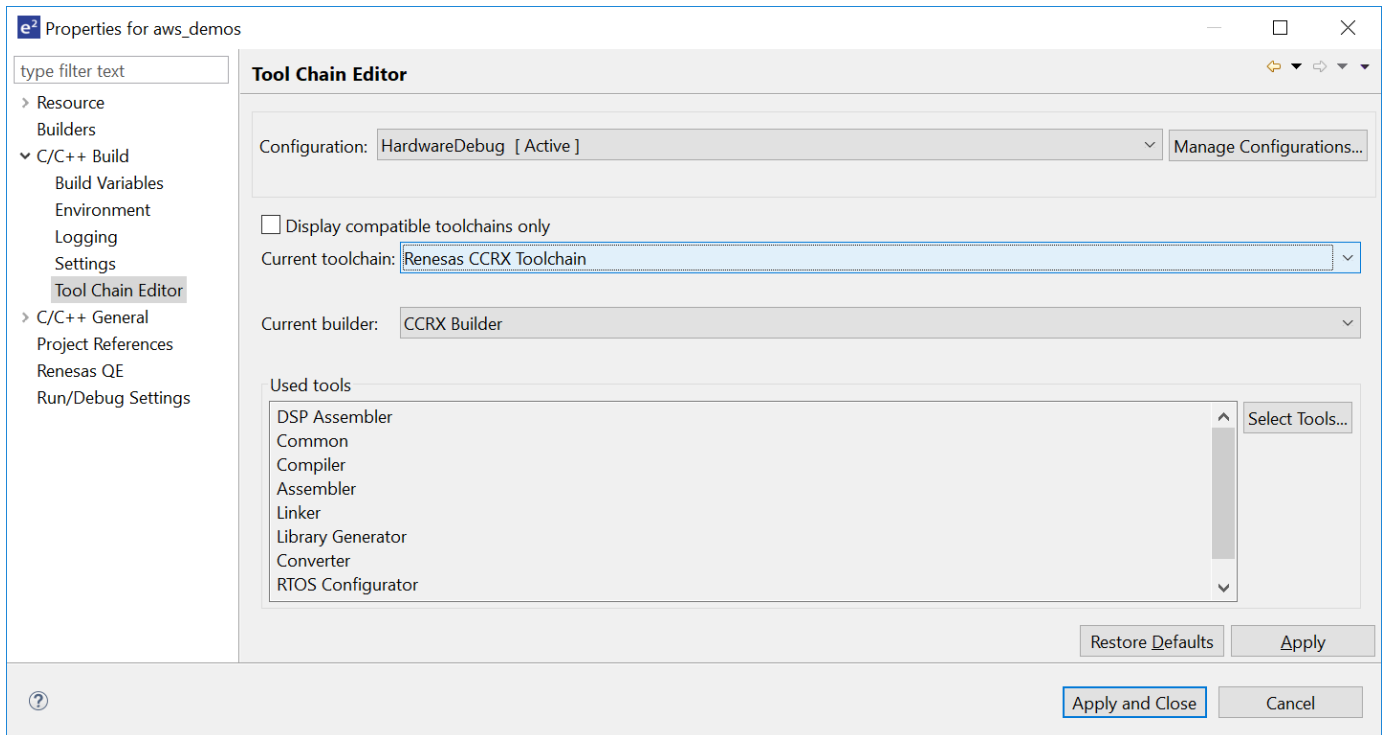
7. Tunggu hingga unduhan selesai.



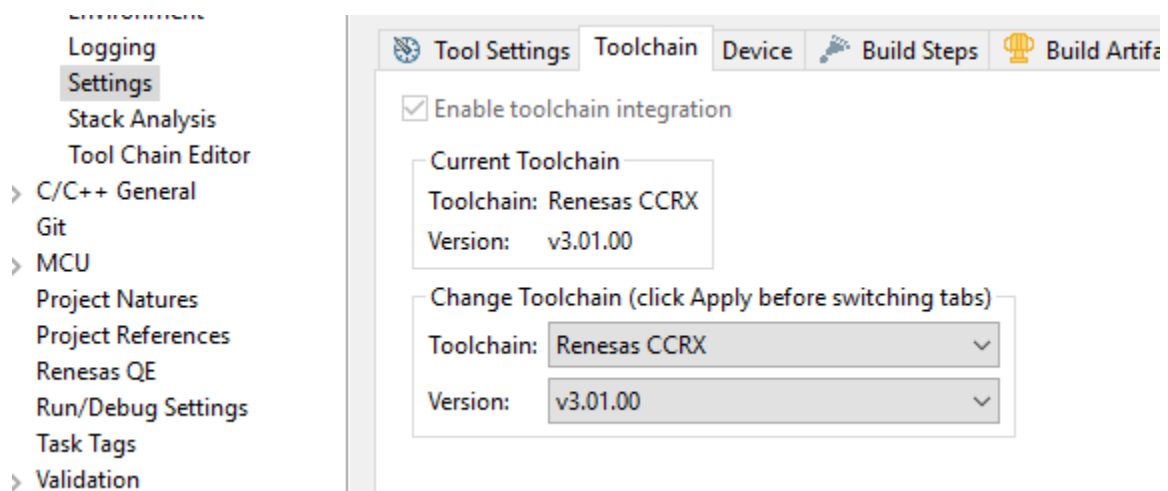
8. Pilih proyek `aws_demos` dan `boot_loader`, lalu pilih Finish untuk mengimpornya.



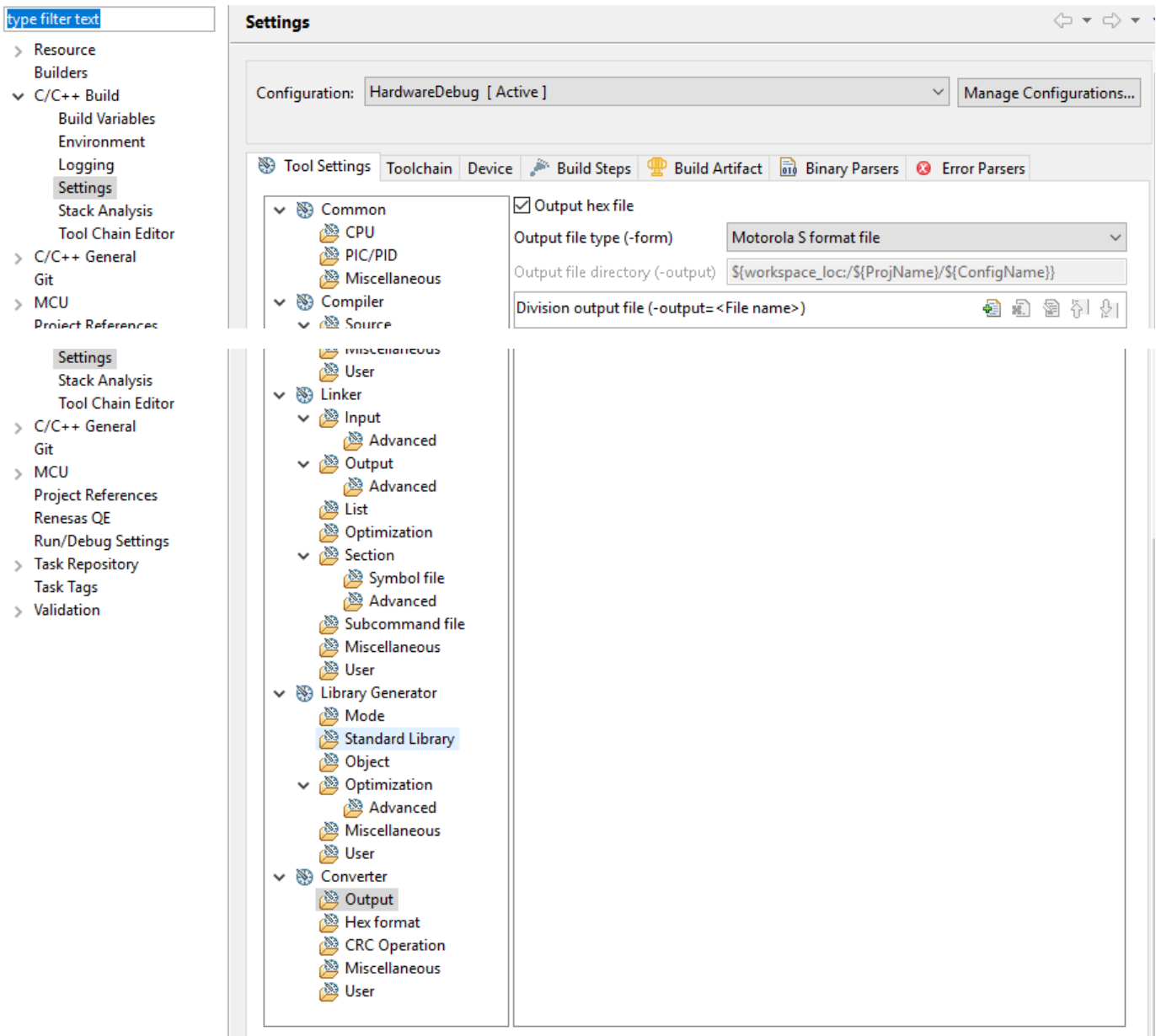
9. Untuk kedua proyek, buka properti proyek. Di panel navigasi, pilih Tool Chain Editor.
 - a. Pilih toolchain saat ini.
 - b. Pilih pembangun saat ini.



10. Di panel navigasi, pilih Pengaturan. Pilih tab Toolchain, dan kemudian pilih versi toolchain.



Pilih tab Pengaturan Alat, luaskan Konverter dan kemudian pilih Keluaran. Di jendela utama, pastikan File hex keluaran dipilih, lalu pilih jenis file Output.



11. Dalam proyek bootloader, bukaprojects\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 mengimplementasikan FreeRTOS OTA dengan menggunakan Amazon Web Services di RX65N](#) dan bagian 7.3 “Membuat Pasangan Kunci ECDSA-SHA256 dengan OpenSSL” di [Renesas MCU Firmware Update Design Policy](#).

```

2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20  * * DISCLAIMER
21  * * File Name      : code_signer_public_key.h
22  * * History       : DD.MM.YYYY Version Description
23
24
25
26
27
28  #ifndef CODE_SIGNER_PUBLIC_KEY_H
29  #define CODE_SIGNER_PUBLIC_KEY_H
30
31
32  /*
33  * PEM-encoded code signer public key.
34  *
35  * Must include the PEM header and footer:
36  * "-----BEGIN CERTIFICATE-----\n"
37  * "...base64 data...\n"
38  * "-----END CERTIFICATE-----"
39  */
40  // #define CODE_SIGNER_PUBLIC_KEY_PEM "Paste code signer public key here."
41  #define CODE_SIGNER_PUBLIC_KEY_PEM \
42  "-----BEGIN PUBLIC KEY-----\n"
43  "MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEVxqVltTUZ5LXrmurImTTQz1jLtQ"\
44  "sz9cj31BZ189ny+m813UkaolY4/aEwa6fTu8PVeaiyEwJeQJ7Y8pYGC9IA=="\
45  "-----END PUBLIC KEY-----\n"
46
47  extern const uint8_t code_signer_public_key[];
48  extern const uint32_t code_signer_public_key_length;
49
50  #endif /* CODE_SIGNER_PUBLIC_KEY_H */

```

Kemudian membangun proyek untuk membuat `boot_loader.mot`.

12. Buka `aws_demos` proyek.

- Buka [konsol AWS IoT](#).
- Di panel navigasi di sebelah kiri, pilih Pengaturan. Catat titik akhir kustom Anda di kotak teks titik akhir data Perangkat.
- Pilih Kelola, lalu pilih Hal-hal. Catat AWS IoT nama papan Anda.
- Dalam `aws_demos` proyek, bukalah `demo/include/aws_clientcredential.h` dan tentukan nilai berikut.

```

#define clientcredentialMQTT_BROKER_ENDPOINT[] = "Your AWS IoT endpoint";
#define clientcredentialIOT_THING_NAME "The AWS IoT thing name of your board"

```

```

28
29
30  /*
31  * @brief MQTT Broker endpoint.
32  *
33  * @todo Set this to the fully-qualified DNS name of your MQTT broker.
34  */
35  #define clientcredentialMQTT_BROKER_ENDPOINT "xxxxx-ats.iot.ap-northeast-1.amazonaws.com"
36
37  /*
38  * @brief Host name.
39  *
40  * @todo Set this to the unique name of your IoT Thing.
41  */
42  #define clientcredentialIOT_THING_NAME "thingname"

```

- Buka file `tools/certificate_configuration/CertificateConfigurator.html`.

- f. Impor file PEM sertifikat dan file PEM Kunci Pribadi yang Anda unduh sebelumnya.
- g. Pilih Buat dan simpan `aws_clientcredential_keys.h` dan ganti file ini `didemos/include/` direktori.

Certificate Configuration Tool

FreeRTOS Developer Demos

Provide client certificate and private key PEM files downloaded from the AWS IoT Console.

Certificate PEM file:

Choose File
No file chosen

Private Key PEM file:

Choose File
No file chosen

⬇ Generate and save `aws_clientcredential_keys.h`

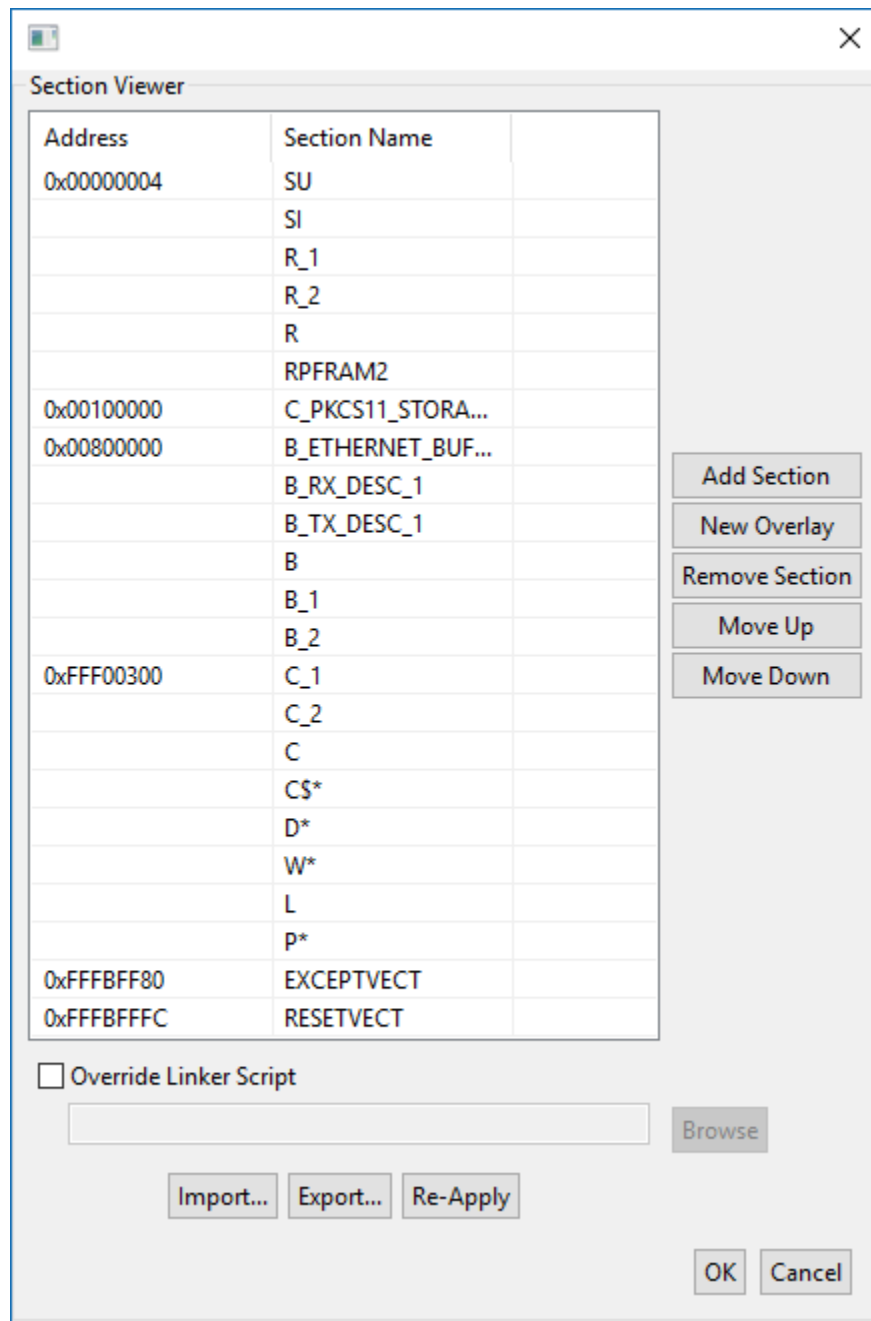
▲ Save the generated header file to the `demos/common/include` folder of the demo project.

Copyright (C) 2017 Amazon.com, Inc. or its affiliates. All Rights Reserved.

- h. Bukavendors/renesas/boards/rx65n-rsk/aws_demos/config_files/ota_demo_config.h file, dan tentukan nilai-nilai ini.

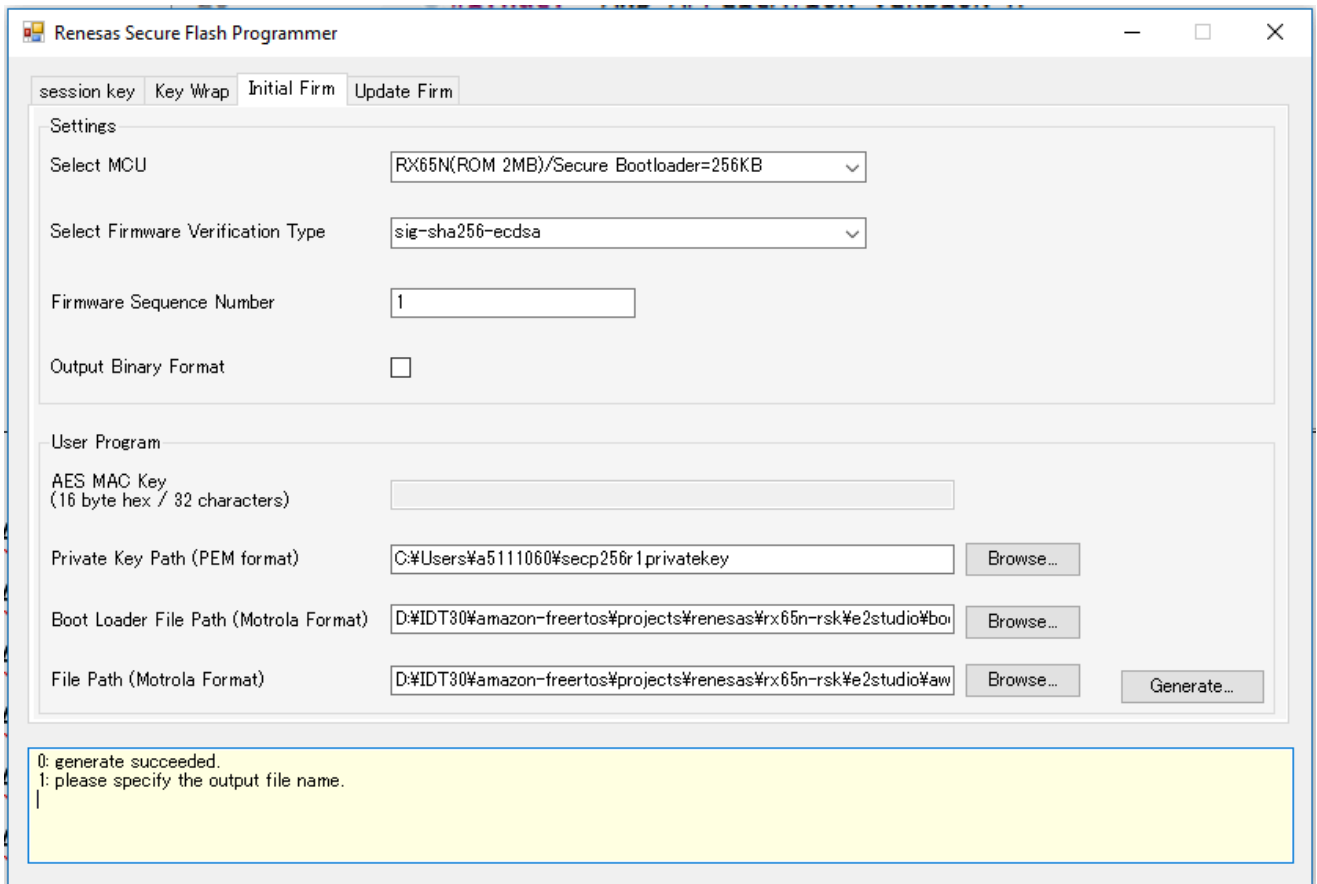
```
#define otapalconfigCODE_SIGNING_CERTIFICATE [] = "your-certificate-key";
```

Di *your-certificate-key* mana nilai dari file tersebut `secp256r1.crt`. Ingatlah untuk menambahkan “\” setelah setiap baris dalam sertifikat. Untuk informasi selengkapnya tentang membuat `secp256r1.crt` file, lihat [Cara mengimplementasikan FreeRTOS OTA dengan menggunakan Amazon Web Services di RX65N](#) dan bagian 7.3 “Membuat Pasangan Kunci ECDSA-SHA256 dengan OpenSSL” di [Renesas MCU Firmware Update Design Policy](#).



- d. Pilih Bangun 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 RX65N-RSK untuk menginstal firmware awal.
- a. Unduh <https://github.com/renesas/Amazon-FreeRTOS-Tools> dan buka `Renesas Secure Flash Programmer.exe`.
 - b. Pilih tab Initial Firm dan kemudian atur parameter berikut:

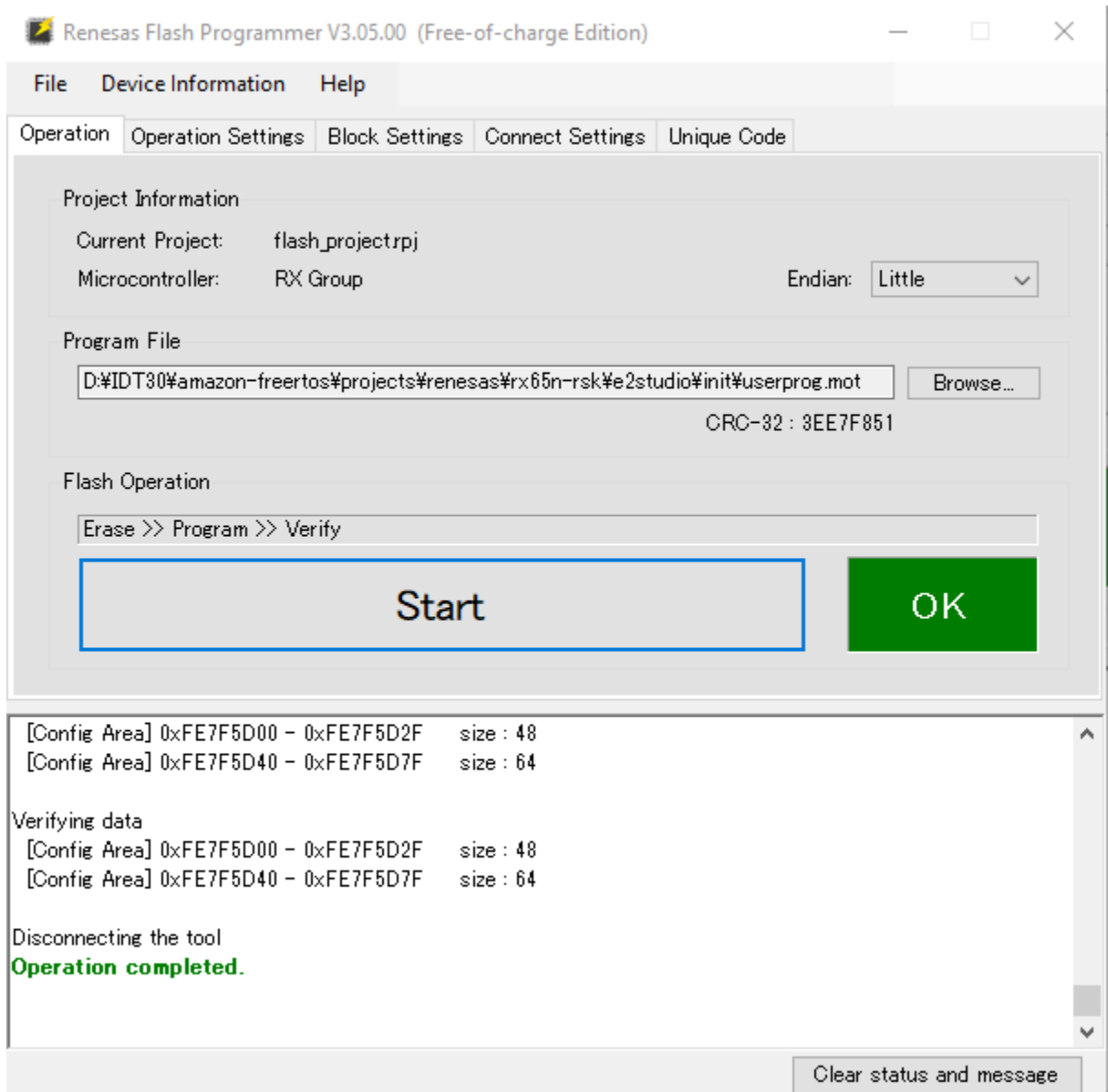
- Private Key Path - Lokasisecp256r1.privatekey.
- Boot Loader File Path - Lokasiboot_loader.mot (projects\renesas\rx65n-rsk\e2studio\boot_loader\HardwareDebug).
- File Path - Lokasiaws_demos.mot (projects\renesas\rx65n-rsk\e2studio\aws_demos\HardwareDebug).



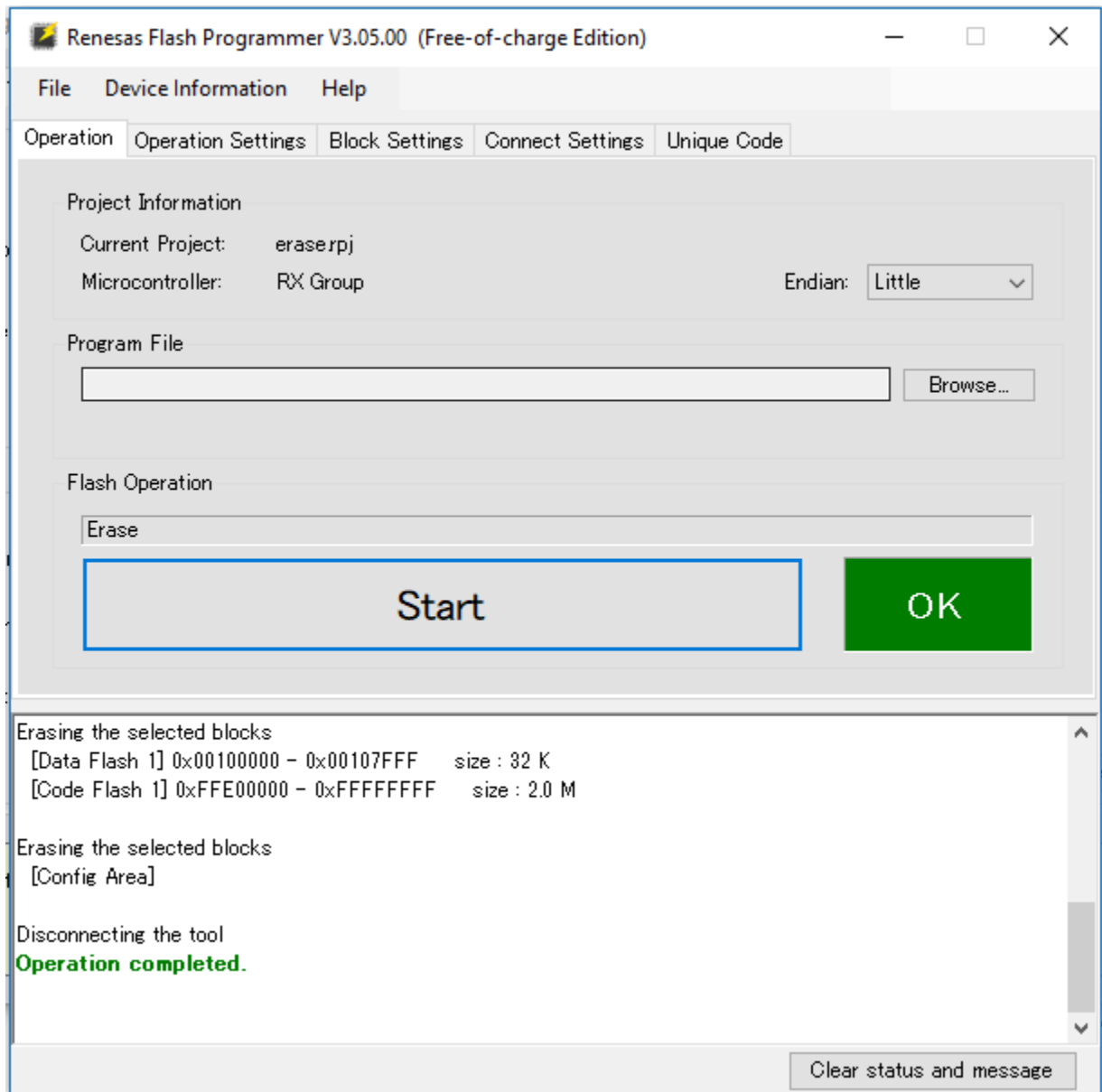
- Buat direktori bernamainit_firmware, Generateuserprog.mot, dan simpan keinit_firmware direktori. Verifikasi bahwa menghasilkan berhasil.

15. Flash firmware awal pada RX65N-RSK.

- Download versi terbaru dari Renesas Flash Programmer (Programming GUI) dari <https://www.renesas.com/tw/en/products/software-tools/tools/programmer/renesas-flash-programmer-programming-gui.html>.
- Bukavendors\renesas\rx_mcu_boards\boards\rx65n-rsk\aws_demos\flash_project\erase_from_bank\ erase.rpj file untuk menghapus data di bank.
- Pilih Mulai untuk menghapus bank.



- d. Untuk mem-flash `userprog.mot`, pilih Jelajahi... dan arahkan ke `init_firmware` direktori, pilih `userprog.mot` file dan pilih Mulai.



16. Versi 0.9.2 (versi awal) firmware diinstal ke RX65N-RSK Anda. Papan RX65N-RSK sekarang mendengarkan pembaruan OTA. Jika Anda telah membuka Tera Term pada 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] vDHCPPProcess: offer c0a80a09ip
14 3597 [ETHER_RECEI] Heap: current 233200 lowest 233000
15 3597 [IP-task] vDHCPPProcess: 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.
```

```

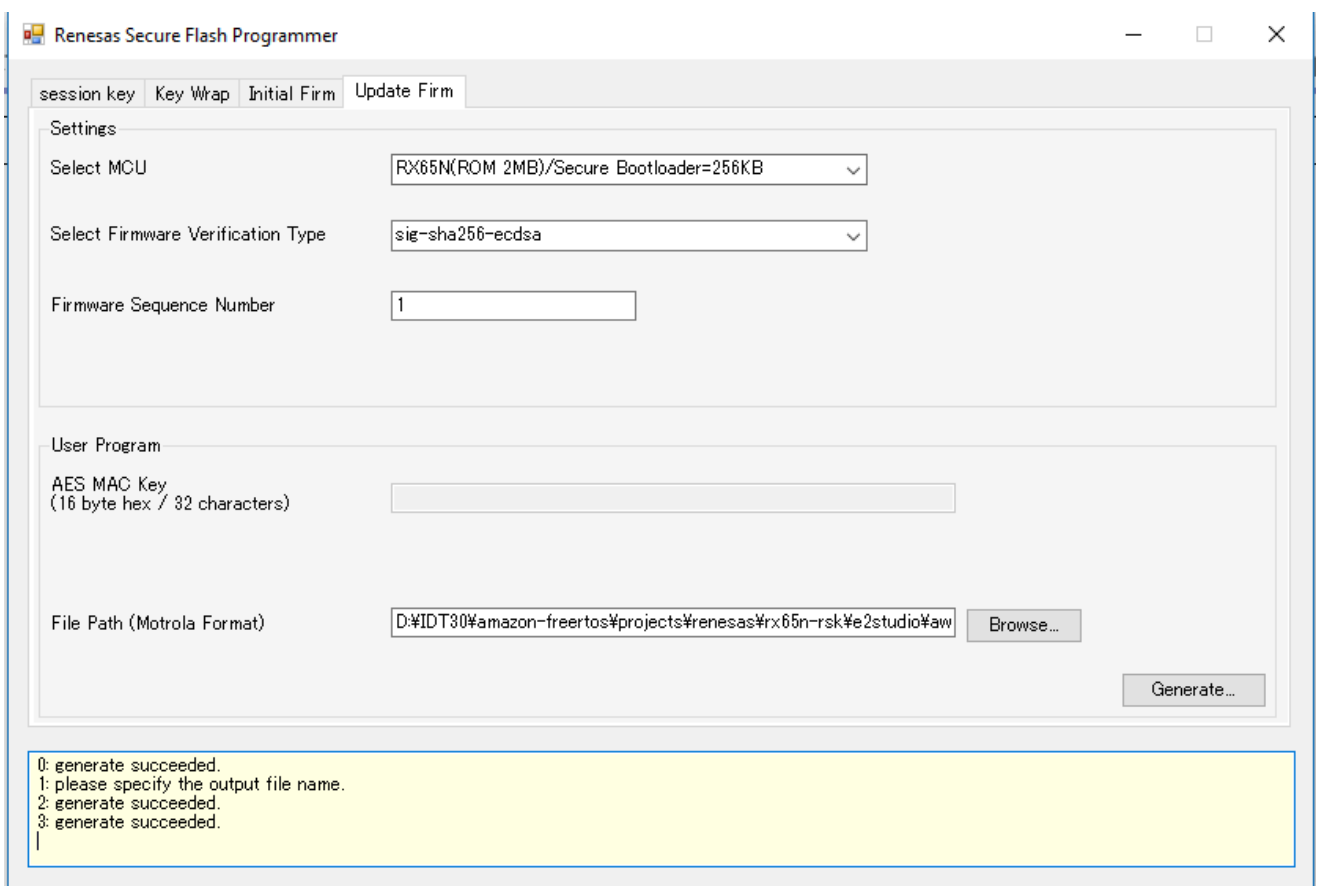
73 7530 [OTA Agent T] [INFO ][MQTT][7530] (MQTT connection 81cfc8, SUBSCRIBE
operation 818c48) Wait complete with result SUCCESS.
74 7530 [OTA Agent T] [privSubscribeToJobNotificationTopics] OK: $aws/things/rx65n-
gr-rose/jobs/notify-next
75 7530 [OTA Agent T] [privRequestJob_Mqtt] Request #0
76 7532 [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] [privOTAAgentTask] Called handler. Current State
[RequestingJob] Event [RequestJobDocument] New state [WaitingForJob]
80 7552 [OTA Agent T] [privParseJSONbyModel] Extracted parameter [ clientToken:
0:rx65n-gr-rose ]
81 7552 [OTA Agent T] [privParseJSONbyModel] parameter not present: execution
82 7552 [OTA Agent T] [privParseJSONbyModel] parameter not present: jobId
83 7552 [OTA Agent T] [privParseJSONbyModel] parameter not present: jobDocument
84 7552 [OTA Agent T] [privParseJSONbyModel] parameter not present: afr_ota
85 7552 [OTA Agent T] [privParseJSONbyModel] parameter not present: protocols
86 7552 [OTA Agent T] [privParseJSONbyModel] parameter not present: files
87 7552 [OTA Agent T] [privParseJSONbyModel] 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

```

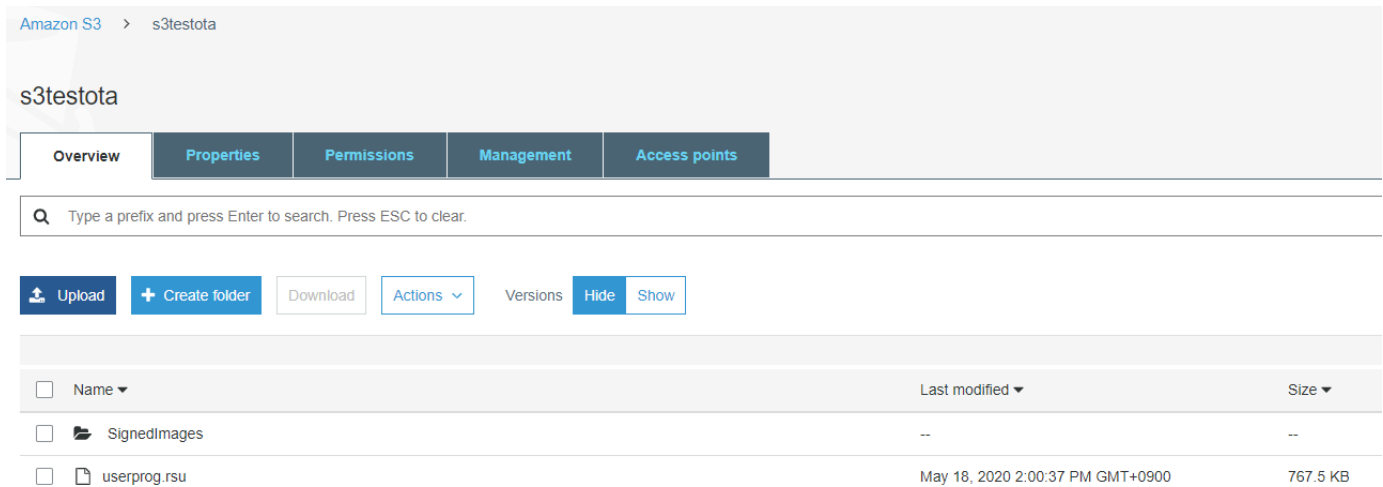
17. Tugas B: Perbarui versi firmware Anda

- a. Bukademos/include/aws_application_version.h file dan naikkan nilai APP_VERSION_BUILD token ke 0.9.3.

- b. Membangun kembali proyek.
18. Buatkan `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 Update Firm 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 menghasilkan berhasil.



19. Unggah pembaruan firmware, `userproj.rsu`, ke bucket Amazon S3 seperti yang dijelaskan dalam [Buat bucket Amazon S3 untuk menyimpan pembaruan](#).



20. Buat pekerjaan untuk memperbarui firmware pada RX65N-RSK.

AWS IoTJobs adalah layanan yang memberitahukan satu atau lebih perangkat yang terhubung dari [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.

- Masuk ke [konsol AWS IoT](#) tersebut. Di panel navigasi, pilih Kelola, lalu pilih Pekerjaan.
- Pilih Buat pekerjaan, lalu pilih Buat pekerjaan Pembaruan OTA. Pilih sesuatu, lalu pilih Berikutnya.
- Buat pekerjaan pembaruan FreeRTOS OTA sebagai berikut:
 - Pilih MQTT.
 - Pilih profil penandatanganan kode yang Anda buat di bagian sebelumnya.
 - Pilih image firmware yang Anda unggah ke bucket Amazon S3.
 - Untuk Pathname gambar firmware di perangkat, masukkan **test**.
 - Pilih peran IAM yang Anda buat di bagian sebelumnya.
- Pilih Selanjutnya.

MQTT

Select and sign your firmware image

Code signing ensures that devices only run code published by trusted authors and that the code has not been altered or corrupted since it was signed. You have three options for code signing. [Learn more](#)

Sign a new firmware image for me
 Select a previously signed firmware image
 Use my custom signed firmware image

Code signing profile [Learn more](#)

ota_signing	SHA256	ECDSA	aaaaaaaa	Clear	Change
-------------	--------	-------	----------	-----------------------	------------------------

Select your firmware image in S3 or upload it

userprog.rsu	Change
--------------	------------------------

Pathname of firmware image on device [Learn more](#)

IAM role for OTA update job

Choose a role which grants AWS IoT access to the S3, AWS IoT jobs and AWS Code signing resources to create an OTA update job. [Learn more](#)

Role (requires S3 access)

ota_test_beginner	Select
-------------------	------------------------

[Cancel](#) [Back](#) [Next](#)

e. Masukkan ID dan kemudian pilih Buat.

21. Buka kembali Tera Term untuk memverifikasi bahwa firmware telah berhasil diperbarui ke demo OTA versi 0.9.3.

```

21 3000 [tmr_svc] the network is up and running
22 10710 [Tmr Svc] Write certificate...
23 10752 [ETHER_RECEI] Heap: current 232336 lowest 232136
24 11652 [ETHER_RECEI] Heap: current 226352 lowest 225952
25 12405 [iot_thread] [INFO ][DEMO][12405] -----STARTING DEMO-----
26 12405 [iot_thread] [INFO ][INIT][12405] SDK successfully initialized.
27 12405 [iot_thread] [INFO ][DEMO][12405] Successfully initialized the demo. Network type for the demo: 4
28 12405 [iot_thread] [INFO ][MQTT][12405] MQTT library successfully initialized.
29 12405 [iot_thread] [INFO ][DEMO][12405] OTA demo version 0.9.3
30 12405 [iot_thread] [INFO ][DEMO][12405] Connecting to broker...
31 12405 [iot_thread] [INFO ][DEMO][12405] MQTT demo client identifier is rx65n-gr-rose (length 13).

```

22. DiAWS IoT konsol, verifikasi bahwa status pekerjaan Berhasil.

Jobs > AFR_OTA-demo_test

JOB

AFR_OTA-demo_test

COMPLETED Actions ▾

Overview Last updated Jun 3, 2020 4:48:38 PM +0900 [All Statuses](#) [Refresh](#)

Details

Resource Tags

0	0	0	0	1	0	0	0
Queued	In progress	Timed out	Failed	Succeeded	Rejected	Canceled	Removed

Resource	Last updated	Status
> rx65n-gr-rose	Jun 3, 2020 4:48:33 PM +0900	Succeeded ⋮

Tutorial: Lakukan pembaruan OTA di Espressif ESP32 menggunakan 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 mikrokontroler Espressif ESP32 yang terhubung ke proxy MQTT Bluetooth Low Energy pada perangkat Android. Ini memperbarui perangkat menggunakan pekerjaan pembaruan AWS IoT Over-the-air (OTA). Perangkat terhubung ke AWS IoT menggunakan kredensi Amazon Cognito yang dimasukkan dalam aplikasi demo Android. Operator resmi memulai pembaruan OTA dari cloud. Saat perangkat terhubung melalui aplikasi demo Android, pembaruan OTA dimulai dan firmware diperbarui pada perangkat.

Versi FreeRTOS 2019.06.00 Major dan yang lebih baru menyertakan dukungan proxy MQTT Bluetooth Low Energy yang dapat digunakan untuk penyediaan Wi-Fi dan koneksi aman ke AWS IoT layanan. 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 SDK yang menggunakan profil akses generik (GAP) dan profil atribut generik (GATT).

Berikut adalah langkah-langkah yang akan kami ikuti untuk memungkinkan pembaruan OTA melalui Bluetooth Low Energy:

1. Konfigurasi penyimpanan: Buat bucket dan kebijakan Amazon S3 dan konfigurasi pengguna yang dapat melakukan pembaruan.
2. Buat sertifikat penandatanganan kode: Buat sertifikat penandatanganan dan izinkan pengguna menandatangani pembaruan firmware.
3. Mengkonfigurasi autentikasi Amazon Cognito: Buat penyedia kredensi, kumpulan pengguna, dan akses aplikasi ke pangkalan pengguna.
4. Konfigurasi FreeRTOS: Siapkan Bluetooth Low Energy, kredensi klien, dan sertifikat publik penandatanganan kode.
5. Mengkonfigurasi aplikasi Android: Siapkan penyedia kredensi, kumpulan pengguna, dan deploy aplikasi ke perangkat Android.
6. Jalankan skrip pembaruan OTA: Untuk memulai pembaruan OTA, gunakan skrip pembaruan OTA.

Untuk informasi selengkapnya tentang cara kerja pembaruan, lihat [Pembaruan FreeRTOS Over-the-Air](#). Untuk informasi tambahan tentang cara mengatur fungsionalitas proxy Bluetooth Low Energy MQTT, lihat postingan [Menggunakan Bluetooth Low Energy dengan FreeRTOS di Espressif ESP32](#) oleh Richard Kang.

Prasyarat

Untuk melakukan langkah-langkah di tutorial ini, Anda memerlukan sumber daya berikut:

- Papan pengembangan ESP32.
- 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.

Pada komputer pengembangan Anda yang Anda butuhkan:

- Ruang disk yang cukup (~ 500 Mb) untuk toolchain Xtensa dan kode sumber FreeRTOS dan contoh.
- Android Studio diinstal.
- [AWS CLI](#) terinstal.
- Python3 diinstal.
- [Boto3 AWS Software Developer Kit \(SDK\) untuk Python](#).

Langkah-langkah dalam tutorial ini mengasumsikan bahwa Xtensa toolchain, ESP-IDF, dan FreeRTOS kode diinstal dalam/esp direktori di direktori home Anda. Anda harus menambahkan~/esp/xtensa-esp32-elf/bin ke\$PATH variabel Anda.

Langkah 1: Konfigurasi penyimpanan

1. [Buat bucket Amazon S3 untuk menyimpan pembaruan](#) dengan versi diaktifkan untuk menahan gambar firmware.
2. [Membuat peran layanan Pembaruan OTA](#) dan tambahkan kebijakan terkelola berikut ke peran:
 - AWSIoTLogging
 - 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 pada entitas tepercaya.
4. Ikuti langkah-langkah untuk [Membuat kebijakan pengguna OTA](#) dan melampirkannya ke pengguna Anda.

Langkah 2: Buat sertifikat penandatanganan kode

1. Buat bucket Amazon S3 dengan versi diaktifkan untuk menahan citra firmware.
2. Buat sertifikat penandatanganan kode yang dapat digunakan untuk menandatangani firmware. Catat sertifikat Amazon Resource Name (ARN) ketika sertifikat diimpor.

```
aws acm import-certificate --profile=ota-update-user --certificate file://  
ecdsasigner.crt --private-key file://ecdsasigner.key
```

Contoh keluaran:

```
{  
  "CertificateArn": "arn:aws:acm:us-east-1:<account>:certificate/<certid>"  
}
```

Anda akan menggunakan ARN nanti untuk membuat profil penandatanganan. Jika Anda ingin, 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 keluaran:

```
{  
  "arn": "arn:aws:signer::<account>:/signing-profiles/esp32Profile"  
}
```

Langkah 3: Konfigurasi otentikasi Amazon Cognito

Membuat 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. Di panel navigasi kiri, pilih Pengaturan. Di Titik akhir data Perangkat, buat catatan dari nilai titik akhir. Endpoint harus sesuatu seperti `xxxxxxxxxxxxxxxxx.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 akan melihat pesan "Anda belum memiliki kebijakan apa pun" dan Anda dapat memilih Buat kebijakan.
5. Masukkan nama kebijakan Anda, misalnya "esp32_mqtt_proxy_iot_policy".

6. Di bagian Tambahkan pernyataan, pilih Mode lanjutan. Salin dan tempelkan 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 Create (Buat).

Buat objek AWS IoT

1. Masuk ke [konsol AWS IoT](#) tersebut.
2. Di panel navigasi kiri, pilih Kelola, lalu pilih Hal-hal.
3. Di sudut kanan atas, pilih Buat. Jika Anda tidak memiliki hal-hal 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 one thing.

5. Pada Tambahkan perangkat Anda ke halaman registri hal, masukkan nama untuk barang Anda (misalnya, “esp32-ble”). Hanya karakter alfanumerik, tanda hubung (-), dan garis bawah (_) yang diizinkan. Pilih Selanjutnya.
6. Pada halaman Tambahkan sertifikat untuk barang 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 autentikasi dan otorisasi, tidak diperlukan sertifikat perangkat.

Membuat Klien Aplikasi Amazon Cognito

1. Masuk ke [konsol Amazon Cognito](#).
2. Di spanduk navigasi kanan atas, pilih Buat kolam 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 Menghasilkan rahasia klien dipilih.
8. Pilih Buat klien aplikasi.
9. Pilih Kembali ke perincian kolam.
10. Pada halaman Tinjau pangkalan pengguna, pilih Buat pangkalan. Anda akan melihat pesan yang bertuliskan “Kolam pengguna Anda berhasil dibuat.” Perhatikan 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 pangkalan 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 terautentikasi dan tidak terautentikasi, pilih Izinkan.
9. Catat ID kumpulan identitas, yang ada dalam format `east-1:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx`.

Melampirkan kebijakan IAM ke identitas yang diautentikasi

1. Buka Amazon Cognito [konsol](#).
2. Pilih pangkalan identitas yang baru saja Anda buat (misalnya, "mqtt_proxy_identity_pool").
3. Pilih Edit kolom 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 peran tersebut.
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: Amazon FreeRTOS

1. Unduh versi terbaru kode Amazon FreeRTOS dari [GitHub repo FreeRTOS](#).
2. Untuk mengaktifkan demo pembaruan OTA, ikuti langkah-langkah di [Memulai dengan Espressif ESP32- DevKit C dan ESP-WROVER-KIT](#).
3. Buat modifikasi tambahan ini di file-file berikut:
 - a. Bukavendors/espressif/boards/esp32/aws_demos/config_files/aws_demo_config.h dan tentukanCONFIG_OTA_UPDATE_DEMO_ENABLED.
 - b. Bukavendors/espressif/boards/esp32/aws_demos/common/config_files/aws_demo_config.h dan ubahdemoconfigNETWORK_TYPES keAWSIOT_NETWORK_TYPE_BLE.
 - c. Bukademos/include/aws_clientcredential.h dan masukkan URL endpoint Anda untukclientcredentialMQTT_BROKER_ENDPOINT.

Masukkan nama benda Anda untukclientcredentialIOT_THING_NAME (misalnya, “esp32-ble”). Sertifikat tidak harus ditambahkan saat Anda menggunakan kredensi Amazon Cognito.

- d. Bukavendors/espressif/boards/esp32/aws_demos/config_files/aws_iot_network_config.h dan ubahconfigSUPPORTED_NETWORKS danconfigENABLED_NETWORKS sertakan sajaAWSIOT_NETWORK_TYPE_BLE.
- e. Bukavendors/*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: Konfigurasi aplikasi Android

1. Unduh Android Bluetooth Low Energy SDK dan contoh aplikasi dari GitHub repo [amazon-freertos-ble-android-sdk](#).
2. Buka file `app/src/main/res/raw/awsconfiguration.json` dan isi Pool Id, Region AppClientId, dan AppClientSecret gunakan petunjuk 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 Region (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 penggeser untuk perangkat yang ditemukan ke Aktif.
 - h. Tekan y pada konsol debug port serial untuk ESP32.
 - i. Pilih Pasangkan & Connect.
5. Semakin... tautan 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 konsol ESP32. 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 memasang prasyarat, jalankan perintah berikut:

```
pip3 install boto3
```

```
pip3 install pathlib
```

2. Kenaikan versi aplikasi FreeRTOS didemos/include/aws_application_version.h.
3. Bangun file.bin baru.
4. Unduh skrip python [start_ota.py](#). Untuk melihat konten bantuan untuk skrip, jalankan perintah berikut di jendela terminal:

```
python3 start_ota.py -h
```

Anda akan melihat tampilan seperti berikut ini:

```

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
--otasingningprofile 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 templat 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 --
otasingningprofile abcd --signingcertificateid certificateid

```

Anda akan melihat pembaruan dimulai di konsol debug ESP32:

```

38 2462 [OTA Task] [prvParseJobDoc] Job was accepted. Attempting to start transfer.
---
49 2867 [OTA Task] [prvIngestDataBlock] Received file block 1, size 1024
50 2867 [OTA Task] [prvIngestDataBlock] Remaining: 1290
51 2894 [OTA Task] [prvIngestDataBlock] Received file block 2, size 1024
52 2894 [OTA Task] [prvIngestDataBlock] Remaining: 1289
53 2921 [OTA Task] [prvIngestDataBlock] Received file block 3, size 1024
54 2921 [OTA Task] [prvIngestDataBlock] Remaining: 1288
55 2952 [OTA Task] [prvIngestDataBlock] Received file block 4, size 1024
56 2953 [OTA Task] [prvIngestDataBlock] Remaining: 1287
57 2959 [iot_thread] State: Active  Received: 5  Queued: 5  Processed: 5
Dropped: 0

```

6. Ketika pembaruan OTA selesai, perangkat akan dimulai ulang seperti yang dipersyaratkan oleh proses pembaruan OTA. Kemudian mencoba untuk terhubung menggunakan firmware yang diperbarui. Jika upgrade 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 library 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 AWS Shadow. 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 AWS IoT Device Shadow.

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-langkah di dalamnya [Memulai dengan FreeRTOS](#).

Fungsionalitas

Demo membuat tugas aplikasi tunggal yang loop melalui serangkaian contoh yang menunjukkan bayangan/`update` dan/`update/delta` callback untuk mensimulasikan toggling status perangkat jarak jauh. Ini mengirimkan pembaruan bayangan dengand`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 BrokerAWS IoT MQTT, dan mengasumsikan adapower`On` keadaan dalam bayangan perangkat.

Demo melakukan operasi berikut:

1. Membangun koneksi MQTT dengan menggunakan fungsi pembantu `dshadow_demo_helpers.c`.
2. Kumpulkan string topik MQTT untuk operasi bayangan perangkat, menggunakan makro yang ditentukan oleh pustakaAWS IoT Device Shadow.
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` menggunakan fungsi pembantu `dshadow_demo_helpers.c`.
5. Publikasikan keadaan yang diinginkanpower`On` menggunakan fungsi pembantu `dshadow_demo_helpers.c`. Ini akan menyebabkan/`update/delta` pesan yang akan dikirim ke perangkat.
6. Tangani pesan MQTT masuk`privEventCallback`, dan tentukan apakah pesan terkait dengan bayangan perangkat dengan menggunakan fungsi yang ditentukan oleh libraryAWS IoT Device Shadow (`Shadow_MatchTopic`). Jika pesan adalah/`update/delta` pesan bayangan perangkat, maka fungsi demo utama akan menerbitkan pesan kedua untuk memperbarui status yang dilaporkan kepower`On`. Jika/`update/accepted` pesan diterima, verifikasi bahwa pesan tersebut`clientToken` sama dengan yang dipublikasikan sebelumnya di pesan pembaruan. Itu akan menandai akhir demo.

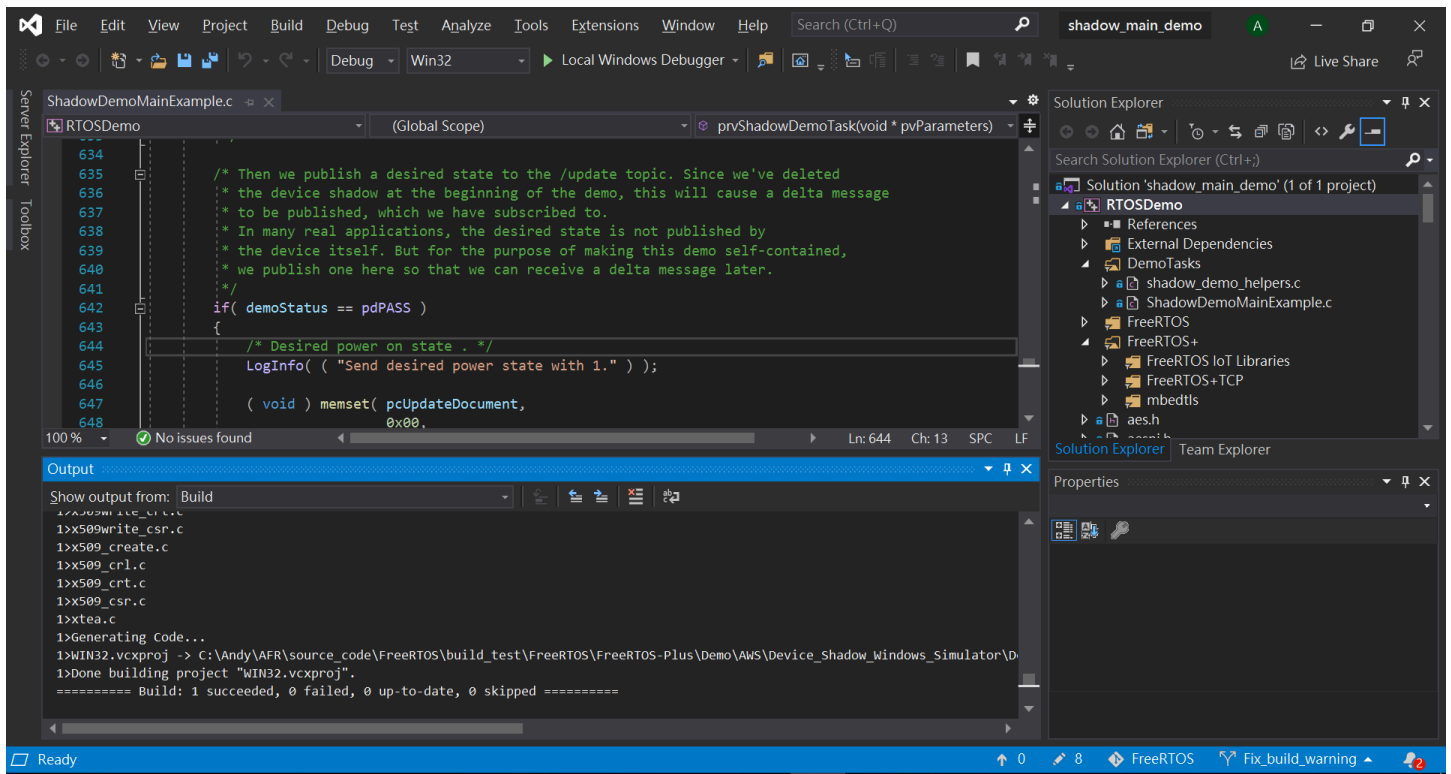
```

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:$aws/things/testClient16:34:41/shadow/update/delta.87 9296 [ShadowDemo]
88 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateDeltaHandler:256] 89 9296 [ShadowDemo] /update/delta json payload:{"version":1,"timestamp":1602751002,"state":{"powerOn":1},"metadata":{"powerOn":{"timestamp":1602751002},"clientToken":"009136"}}.90 9296 [ShadowDemo]
91 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateDeltaHandler:298] 92 9296 [ShadowDemo] version: 193 9296 [ShadowDemo]
94 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateDeltaHandler:308] 95 9296 [ShadowDemo] version:1, uICurrentVersion:0
96 9296 [ShadowDemo]
97 9296 [ShadowDemo] [INFO] [SHADOW] [prvUpdateDeltaHandler:342] 98 9296 [ShadowDemo] The new power on state newState:1, uICurrentPowerOnState:0
99 9296 [ShadowDemo]
100 9296 [ShadowDemo] [INFO] [SHADOW] [prvEventCallback:482] 101 9296 [ShadowDemo] pPublishInfo->pTopicName:$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":{"desired":{"powerOn":{"timestamp":1602751003},"version":2,"timestamp":1602751003,"clientToken":"009696"}}},"reported":{"powerOn":{"timestamp":1602751003},"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"}}},"reported":{"powerOn":{"timestamp":1602751003},"version":1,"timestamp":1602751002,"clientToken":"009136"}}}.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 clientToken=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:730] 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 saat demo berhasil.



Connect ke brokerAWS IoT MQTT

Untuk terhubung ke brokerAWS IoT MQTT, kami menggunakan metode yang sama sepertiMQTT_Connect() di[Demo otentikasi timbal balik CoreMQTT](#).

Hapus dokumen bayangan

Untuk menghapus dokumen bayangan, panggilxPublishToTopic dengan pesan kosong, menggunakan makro yang ditentukan oleh pustakaAWS IoT Device Shadow. Ini digunakanMQTT_Publish untuk mempublikasikan ke/delete topik. Bagian kode berikut menunjukkan bagaimana hal ini dilakukan dalam fungsiprvShadowDemoTask.

```
/* 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 dariAWS IoT broker tentang perubahan bayangan. Topik Device Shadow dirakit oleh makro yang didefinisikan di perpustakaan Device Shadow. Bagian kode berikut menunjukkan bagaimana hal ini dilakukan dalamprvShadowDemoTask 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 Pemangan angan angan angan angan angan

Untuk mengirim pembaruan bayangan, panggilan `demoxPublishToTopic` 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 [Perpustakaan Klien CoreMQTT](#) menggunakan `MQTT_Init` fungsi, akan memberi tahu kami tentang acara paket yang masuk. Lihat fungsi callback [prvEventCallback](#) aktif GitHub.

Fungsi callback mengonfirmasi paket masuk adalah tipeMQTT_PACKET_TYPE_PUBLISH, dan menggunakan Device Shadow Library APIShadow_MatchTopic untuk mengonfirmasi bahwa pesan yang masuk adalah pesan bayangan.

Jika pesan yang masuk adalah pesan bayangan dengan tipeShadowMessageTypeUpdateDelta, maka kita memanggil [prvUpdateDeltaHandler](#) untuk menangani pesan ini.

PenanganprvUpdateDeltaHandler menggunakan pustaka CoreJSON untuk mengurai pesan guna mendapatkan nilai delta untukpowerOn status dan membandingkannya dengan status perangkat saat ini yang dikelola secara lokal. Jika berbeda, status perangkat lokal diperbarui untuk mencerminkan nilai barupowerOn status dari dokumen bayangan.

Jika pesan yang masuk adalah pesan bayangan dengan tipeShadowMessageTypeUpdateAccepted, maka kita memanggil [prvUpdateAcceptedHandler](#) untuk menangani pesan ini. HandlerprvUpdateAcceptedHandler mem-parsing pesan menggunakan pustaka CoreJSON untuk mendapatkan pesanclientToken dari pesan. Fungsi handler ini memeriksa bahwa token klien dari pesan JSON cocok dengan token klien yang digunakan oleh aplikasi. Jika tidak cocok, fungsi log 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 didemos/tcp/aws_tcp_echo_client_single_task.c.

Sebelum Anda memulai, pastikan Anda telah mengunduh FreeRTOS ke mikrokontroler Anda dan buat dan jalankan 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-langkah di dalamnya [Memulai dengan FreeRTOS](#).

Server TCP dan demo klien saat ini tidak didukung pada Cypress CYW943907AEVAL1F dan CYW954907AEVAL1F Development Kit.

1. Ikuti petunjuk dalam [Menyiapkan Server Echo TLS](#) di Panduan Porting FreeRTOS.

Sebuah server gema TLS harus berjalan dan mendengarkan pada port 9000.

Selama penyiapan, Anda seharusnya menghasilkan empat file:

- `client.pem`(sertifikat klien)
- `client.key`(kunci pribadi klien)
- `server.pem`(sertifikat server)
- `server.key`(kunci pribadi server)

2. Gunakan alat `initools/certificate_configuration/CertificateConfigurator.html` untuk menyalin sertifikat klien (`client.pem`) dan kunci pribadi klien (`client.key`) ke `aws_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 Server Echo TLS berjalan.
5. Mengatur `configTCP_ECHO_CLIENT_PORT` variabel untuk `9000`, port di mana TLS Echo Server mendengarkan.
6. Mengatur `configTCP_ECHO_TASKS_SINGLE_TASK_TLS_ENABLED` variabel untuk `1`.
7. Gunakan alat `tools/certificate_configuration/PEMfileToCString.html` untuk menyalin sertifikat server (`server.pem`) ke `TlsECHO_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`, komentari `#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.