



Panduan Developer

Integrasi terkelola untuk AWS IoT Device Management



Integrasi terkelola untuk AWS IoT Device Management: Panduan Developer

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

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

Table of Contents

Untuk apa integrasi terkelola AWS IoT Device Management	1
Wilayah yang Didukung	1
Apakah Anda pengguna integrasi terkelola pertama kali?	1
Ikhtisar integrasi terkelola	1
Terminologi integrasi terkelola	2
Terminologi integrasi terkelola umum	2
Cloud-to-cloud terminologi	2
Terminologi model data	3
Siapkan integrasi terkelola	4
Mendaftar untuk Akun AWS	4
Buat pengguna dengan akses administratif	4
Memulai	7
Jenis perangkat	7
Konfigurasi kunci enkripsi	8
Teknik orientasi	9
Orientasi perangkat yang terhubung langsung	9
Orientasi hub	9
Orientasi perangkat yang terhubung dengan hub	9
Cloud-to-cloud onboarding perangkat	9
Penyediaan perangkat	10
Mengelola siklus hidup dan profil perangkat	12
Perangkat	12
Profil perangkat	13
Model data	14
Model data integrasi terkelola	14
AWS implementasi model data Matter	16
Skema model data	17
Skema kemampuan	18
Skema definisi tipe	18
Skema untuk definisi kemampuan	19
Skema untuk definisi tipe	36
Membangun dan menggunakan definisi tipe dalam dokumen skema kemampuan	41
Perintah dan acara perangkat	55
Perintah perangkat	55

Acara Perangkat	57
Memberi tanda pada sumber daya	59
Dasar-dasar tag	59
Pembatasan dan batasan tanda	60
Tag dengan kebijakan IAM	60
Pemberitahuan integrasi terkelola	64
Siapkan Amazon Kinesis untuk notifikasi	64
Langkah 1: Buat aliran data Amazon Kinesis	64
Langkah 2: Buat kebijakan izin	65
Langkah 3: Arahkan ke dasbor IAM dan pilih Peran	65
Langkah 4: Gunakan kebijakan kepercayaan khusus	65
Langkah 5: Terapkan kebijakan izin Anda	66
Langkah 6: Masukkan nama peran	67
Siapkan notifikasi integrasi terkelola	67
Langkah 1: Berikan izin pengguna untuk memanggil API CreateDestination	67
Langkah 2: Panggil CreateDestination API	68
Langkah 3: Panggil CreateNotificationConfiguration API	68
Jenis acara dipantau dengan integrasi terkelola	69
Cloud-to-Cloud Konektor (C2C)	74
Apa itu konektor cloud-to-cloud (C2C)?	74
Katalog konektor	74
AWS Lambda berfungsi sebagai konektor C2C	75
Alur kerja konektor integrasi terkelola	75
Pedoman dalam menggunakan konektor C2C () cloud-to-cloud	76
Bangun konektor C2C (Cloud-to-Cloud)	76
Prasyarat	76
Persyaratan konektor C2C	77
OAuth 2.0 persyaratan untuk penautan akun	78
Menerapkan operasi antarmuka konektor C2C	85
Panggil konektor C2C Anda	107
Tambahkan izin ke Peran IAM Anda	108
Uji konektor C2C Anda secara manual	108
Gunakan konektor C2C (Cloud-to-Cloud)	109
SDK Hub	120
Arsitektur SDK Hub	120
Orientasi perangkat	120

Komponen orientasi perangkat	120
Alur orientasi perangkat	121
Kontrol perangkat	122
Aliran kontrol perangkat	123
Komponen SDK	123
Instal dan validasi integrasi terkelola Hub SDK	124
Instal SDK menggunakan AWS IoT Greengrass	125
Menerapkan Hub SDK dengan skrip	127
Menyebarkan Hub SDK dengan systemd	130
Onboard hub Anda	133
Subsistem orientasi hub	134
Pengaturan untuk orientasi	135
Perangkat onboard dan mengoperasikannya di hub	144
Pengaturan sederhana untuk onboard dan mengoperasikan perangkat	144
Penyiapan yang dipandu pengguna untuk onboard dan mengoperasikan perangkat	151
Penangan sertifikat kustom	159
Definisi dan komponen API	160
Contoh membangun	162
Penggunaan	166
Plugin protokol kustom	167
Klien Hub SDK	168
Dapatkan integrasi terkelola Hub SDK	168
Tentang toolkit Hub SDK	168
Buat aplikasi kustom Anda dengan klien Hub SDK	168
Menjalankan aplikasi kustom Anda	170
API klien SDK Hub	171
Jenis Data	176
Kontrol hub	178
Prasyarat	178
Akhiri komponen SDK perangkat	179
Integrasikan dengan SDK perangkat Akhir	179
Contoh: Membangun kontrol hub	182
Contoh yang didukung	183
Platform yang didukung	183
Aktifkan CloudWatch Log	183
Prasyarat	184

Pengaturan konfigurasi log SDK Hub	184
Jenis perangkat Zigbee dan Z-Wave yang didukung	186
Jalankan integrasi terkelola pada Raspberry Pi	188
Flash firmware Sonoff Zigbee	189
Integrasi terkelola gambar Hub SDK pada Raspberry Pi	190
Integrasi terkelola wadah Hub SDK Docker di Raspberry Pi	194
Aplikasi demo integrasi terkelola	199
Hub integrasi terkelola offboard	201
Ikhtisar proses offboard Hub SDK	201
Prasyarat	202
Proses offboard Hub SDK	202
Setelah offboarding Hub SDK	205
Middleware khusus protokol	207
Arsitektur middleware	207
End-to-end contoh alur perintah middleware	208
Organisasi kode middleware	208
Integrasikan middleware dengan SDK	214
SDK perangkat akhir	217
Apa itu SDK Perangkat Akhir?	217
Arsitektur dan komponen	218
Provisioning	219
Alur kerja penyediaan	219
Tetapkan variabel lingkungan	220
Daftarkan titik akhir kustom	220
Buat profil penyediaan	220
Buat hal yang dikelola	221
Penyediaan Wi-Fi pengguna SDK	222
Penyediaan armada dengan klaim	222
Kemampuan Hal yang Dikelola	222
Pembaruan OTA	223
Ikhtisar arsitektur OTA	223
Prasyarat	223
Melaksanakan tugas Over-the-Air (OTA)	224
Pengaturan konfigurasi tugas OTA	226
Terapkan pengaturan konfigurasi ke tugas OTA	227
Pantau pemberitahuan OTA	228

Memproses dokumen pekerjaan	229
Menerapkan agen OTA	230
Generator kode model data	231
Proses pembuatan kode	231
Pengaturan lingkungan	234
Hasilkan kode untuk perangkat	235
Fungsi C tingkat rendah APIs	237
OnOff API kluster	238
Interaksi layanan-perangkat	240
Menangani perintah jarak jauh	241
Menangani peristiwa yang tidak diminta	241
Memulai dengan End device SDK	242
Port SDK perangkat Akhir	254
Referensi teknis	257
Keamanan	260
Perlindungan data	261
Enkripsi data saat istirahat untuk integrasi terkelola	262
Manajemen identitas dan akses	268
Audiens	268
Mengautentikasi dengan identitas	269
Mengelola akses menggunakan kebijakan	270
AWS kebijakan terkelola	272
Bagaimana integrasi terkelola bekerja dengan IAM	276
Contoh kebijakan berbasis identitas	281
Pemecahan masalah	284
Menggunakan Peran Terkait Layanan	286
Gunakan AWS Secrets Manager untuk perlindungan data untuk alur kerja C2C	290
Bagaimana integrasi terkelola menggunakan rahasia	291
Cara membuat rahasia	291
Berikan akses untuk integrasi terkelola AWS IoT Device Management untuk mengambil rahasia	291
Validasi kepatuhan	293
Gunakan integrasi terkelola dengan titik akhir VPC antarmuka	293
Pertimbangan titik akhir VPC	294
Membuat titik akhir VPC	295
Menguji titik akhir VPC	296

Kontrol akses	297
Penetapan harga	299
Batasan	299
Connect ke integrasi terkelola untuk endpoint AWS IoT Device Management FIPS	299
Titik akhir bidang kendali	299
Pemantauan	301
CloudTrail log	301
Acara manajemen di CloudTrail	303
Contoh acara	304
Riwayat dokumen	307
.....	cccviii

Untuk AWS IoT Device Management apa integrasi terkelola?

Integrasi terkelola untuk AWS IoT Device Management membantu penyedia solusi IoT menyatukan kontrol dan pengelolaan perangkat IoT dari ratusan produsen. Anda dapat menggunakan integrasi terkelola untuk mengotomatiskan alur kerja penyiapan perangkat dan mendukung interoperabilitas di banyak perangkat, terlepas dari vendor perangkat atau protokol konektivitas. Dengan integrasi terkelola, Anda dapat menggunakan satu antarmuka pengguna dan set APIs untuk mengontrol, mengelola, dan mengoperasikan berbagai perangkat.

Topik

- [Wilayah yang Didukung](#)
- [Apakah Anda pengguna integrasi terkelola pertama kali?](#)
- [Ikhtisar integrasi terkelola](#)
- [Terminologi integrasi terkelola](#)

Wilayah yang Didukung

Integrasi terkelola untuk AWS IoT Device Management didukung di Wilayah berikut:

- Kanada (Pusat)
- Eropa (Irlandia)

Apakah Anda pengguna integrasi terkelola pertama kali?

Jika Anda adalah pengguna pertama kali integrasi terkelola, kami sarankan Anda mulai dengan membaca bagian berikut:

- [Siapkan integrasi terkelola](#)
- [Memulai integrasi terkelola untuk AWS IoT Device Management](#)

Ikhtisar integrasi terkelola

Gambar berikut memberikan gambaran tingkat tinggi dari integrasi terkelola

Terminologi integrasi terkelola

Dalam integrasi terkelola, ada banyak konsep dan istilah yang penting untuk dipahami untuk mengelola implementasi perangkat Anda sendiri. Bagian berikut menguraikan konsep dan istilah kunci tersebut untuk memberikan pemahaman yang lebih baik tentang integrasi terkelola.

Terminologi integrasi terkelola umum

Konsep penting untuk dipahami untuk integrasi terkelola adalah Hal yang Dikelola dibandingkan dengan AWS IoT Core sesuatu.

- **AWS IoT Core Hal:** An AWS IoT Core Thing adalah AWS IoT Core konstruksi yang menyediakan representasi digital. Pengembang diharapkan untuk mengelola kebijakan, penyimpanan data, aturan, tindakan, topik MQTT, dan pengiriman status perangkat ke penyimpanan data. Untuk informasi selengkapnya tentang apa AWS IoT Core itu, lihat [Mengelola perangkat dengan AWS IoT](#).
- **Integrasi terkelola Hal Terkelola:** Dengan Hal yang Dikelola, kami menyediakan abstraksi untuk menyederhanakan interaksi perangkat dan tidak mengharuskan pengembang untuk membuat item seperti aturan, tindakan, Topik MQTT, dan kebijakan.

Cloud-to-cloud terminologi

Perangkat fisik yang terintegrasi dengan integrasi terkelola mungkin berasal dari penyedia cloud pihak ketiga. Untuk menghubungkan perangkat tersebut ke integrasi terkelola dan berkomunikasi dengan penyedia cloud pihak ketiga, terminologi berikut mencakup beberapa konsep kunci yang mendukung alur kerja tersebut:

- **Cloud-to-cloud Konektor (C2C):** Konektor C2C membuat koneksi antara integrasi terkelola dan penyedia cloud pihak ketiga.
- **Penyedia cloud pihak ketiga:** Untuk perangkat yang diproduksi dan dikelola di luar integrasi terkelola, penyedia cloud pihak ketiga memungkinkan kontrol perangkat ini untuk pengguna akhir dan integrasi terkelola berkomunikasi dengan penyedia cloud pihak ketiga untuk berbagai alur kerja seperti perintah perangkat.

Terminologi model data

Integrasi terkelola menggunakan model data untuk mengatur data dan end-to-end komunikasi antar perangkat Anda. Terminologi berikut mencakup beberapa konsep kunci untuk memahami dua model data tersebut:

- **Perangkat:** Entitas yang mewakili perangkat fisik (seperti bel pintu video) yang memiliki beberapa node yang bekerja sama untuk menyediakan satu set fitur lengkap.
- **Endpoint:** Endpoint merangkum fitur mandiri (dering, deteksi gerakan, pencahayaan dalam bel pintu video).
- **Kemampuan:** Entitas yang mewakili komponen yang diperlukan untuk membuat fitur tersedia di titik akhir (tombol atau lampu dan lonceng di fitur bel video).
- **Tindakan:** Entitas yang mewakili interaksi dengan kemampuan perangkat (membunyikan bel atau melihat siapa yang ada di pintu).
- **Peristiwa:** Entitas yang mewakili peristiwa dari kemampuan perangkat. Perangkat dapat mengirim acara untuk melaporkan incident/alarm, an activity from a sensor etc. (e.g. there is knock/ring di pintu).
- **Properti:** Entitas yang mewakili atribut tertentu dalam status perangkat (bel berdering, lampu teras menyala, kamera merekam).
- **Model Data:** Lapisan data sesuai dengan elemen data dan kata kerja yang membantu mendukung fungsionalitas aplikasi. Aplikasi beroperasi pada struktur data ini ketika ada maksud untuk berinteraksi dengan perangkat. Untuk informasi lebih lanjut, lihat [connectedhomeip](#) di situs web. GitHub
- **Skema:** Skema adalah representasi dari model data dalam format JSON.

Siapkan integrasi terkelola

Bagian berikut memandu Anda melalui penyiapan awal untuk menggunakan integrasi terkelola untuk AWS IoT Device Management.

Topik

- [Mendaftar untuk Akun AWS](#)
- [Buat pengguna dengan akses administratif](#)

Mendaftar untuk Akun AWS

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

Untuk mendaftar untuk Akun AWS

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

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

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

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

Buat pengguna dengan akses administratif

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

Amankan Anda Pengguna root akun AWS

1. Masuk ke [Konsol Manajemen AWS](#) sebagai pemilik akun dengan memilih pengguna Root dan memasukkan alamat Akun AWS email Anda. Di laman berikutnya, masukkan kata sandi.

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

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

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

Buat pengguna dengan akses administratif

1. Aktifkan Pusat Identitas IAM.

Untuk mendapatkan petunjuk, silakan lihat [Mengaktifkan AWS IAM Identity Center](#) di Panduan Pengguna AWS IAM Identity Center .

2. Di Pusat Identitas IAM, berikan akses administratif ke pengguna.

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

Masuk sebagai pengguna dengan akses administratif

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

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

Tetapkan akses ke pengguna tambahan

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

Untuk petunjuknya, lihat [Membuat set izin](#) di Panduan AWS IAM Identity Center Pengguna.

2. Tetapkan pengguna ke grup, lalu tetapkan akses masuk tunggal ke grup.

Untuk petunjuk, lihat [Menambahkan grup](#) di Panduan AWS IAM Identity Center Pengguna.

Memulai integrasi terkelola untuk AWS IoT Device Management

Bagian berikut menguraikan langkah-langkah yang perlu Anda ambil untuk mulai menggunakan integrasi terkelola.

Topik

- [Jenis perangkat](#)
- [Konfigurasi kunci enkripsi](#)
- [Teknik orientasi](#)

Jenis perangkat

Integrasi terkelola mengelola banyak jenis perangkat. Setiap perangkat berada dalam salah satu dari tiga kategori berikut:

- Perangkat yang terhubung langsung: Jenis perangkat ini langsung terhubung ke titik akhir integrasi terkelola. Biasanya, perangkat ini dibangun dan dikelola oleh produsen perangkat yang menyertakan SDK perangkat akhir integrasi terkelola untuk konektivitas langsung.
- Perangkat yang terhubung dengan hub: Perangkat ini terhubung ke integrasi terkelola melalui hub yang menjalankan integrasi terkelola Hub SDK, yang mengelola fungsi penemuan, orientasi, dan kontrol perangkat. Pengguna akhir dapat menggunakan perangkat ini menggunakan inisiasi penekanan tombol atau pemindaian kode batang.

Dua alur kerja berikut ini didukung untuk melakukan onboarding perangkat yang terhubung dengan hub:

- Tekan tombol yang dimulai oleh pengguna akhir untuk memulai penemuan perangkat
- Pemindaian berbasis barcode untuk melakukan asosiasi perangkat
- Cloud-to-cloud Perangkat (C2C): Ini adalah perangkat yang dirancang dan dikelola oleh vendor yang memelihara infrastruktur cloud mereka sendiri dan aplikasi seluler bermerek untuk kontrol perangkat. Integrasi terkelola pelanggan dapat mengakses katalog konektor C2C pra-bangun atau membuatnya sendiri, untuk mengembangkan solusi IoT yang bekerja dengan beberapa cloud vendor pihak ketiga melalui antarmuka terpadu.

Ketika pengguna akhir mengaktifkan perangkat C2C untuk pertama kalinya, perangkat tersebut harus disediakan dengan penyedia cloud pihak ketiga masing-masing untuk integrasi terkelola guna mendapatkan kemampuan dan metadata perangkatnya. Setelah menyelesaikan alur kerja penyediaan tersebut, integrasi terkelola dapat berkomunikasi dengan perangkat cloud dan penyedia cloud pihak ketiga atas nama pengguna akhir.

Note

Hub bukan jenis perangkat tertentu seperti yang tercantum di atas. Tujuannya adalah melayani peran sebagai pengontrol perangkat rumah pintar dan memfasilitasi koneksi antara integrasi terkelola dan penyedia cloud pihak ketiga. Ini dapat berfungsi sebagai jenis perangkat seperti yang tercantum di atas dan sebagai hub.

Konfigurasi kunci enkripsi

Keamanan sangat penting untuk data yang dirutekan antara pengguna akhir, integrasi terkelola, dan cloud pihak ketiga. Salah satu metode yang kami dukung untuk melindungi data perangkat Anda adalah end-to-end enkripsi yang memanfaatkan kunci enkripsi aman untuk merutekan data Anda.

Sebagai pelanggan integrasi terkelola, Anda memiliki dua opsi berikut untuk menggunakan kunci enkripsi:

- Gunakan kunci enkripsi terkelola integrasi terkelola default.
- Berikan AWS KMS key yang Anda buat.

Untuk informasi selengkapnya tentang layanan AWS KMS, lihat [Layanan manajemen kunci \(KMS\)](#)

Memanggil [PutDefaultEncryptionConfiguration](#) API di Panduan Referensi API Integrasi Terkelola memberi Anda akses untuk memperbarui opsi kunci enkripsi mana yang ingin Anda gunakan. Secara default, integrasi terkelola menggunakan kunci enkripsi terkelola integrasi terkelola default. Anda dapat memperbarui konfigurasi kunci enkripsi kapan saja menggunakan [PutDefaultEncryptionConfiguration](#) API.

Selain itu, memanggil perintah [GetDefaultEncryptionConfiguration](#) API mengembalikan informasi tentang konfigurasi enkripsi untuk AWS akun di wilayah default atau tertentu.

Teknik orientasi

Di bawah ini adalah jenis-jenis orientasi:

Orientasi perangkat yang terhubung langsung

Lihat langkah-langkah [Provisionee](#) untuk melakukan onboard perangkat yang terhubung langsung.

Orientasi hub

Lihat langkah-langkah [Onboard hub Anda ke integrasi terkelola](#) untuk melakukan onboard hub.

Orientasi perangkat yang terhubung dengan hub

Lihat langkah-langkah [Perangkat onboard dan mengoperasikannya di hub](#) untuk melakukan onboard pada perangkat yang terhubung dengan hub.

Cloud-to-cloud onboarding perangkat

Lihat langkah-langkah [Gunakan konektor C2C \(Cloud-to-Cloud\)](#) untuk melakukan onboard perangkat cloud dari vendor cloud pihak ketiga ke integrasi terkelola.

Penyediaan perangkat

Penyediaan perangkat memfasilitasi proses orientasi perangkat, mengawasi seluruh siklus hidup perangkat, dan menetapkan repositori terpusat untuk informasi perangkat yang dapat diakses oleh aspek lain dari integrasi terkelola. Integrasi terkelola menyediakan antarmuka terpadu untuk mengelola berbagai jenis perangkat, mengakomodasi perangkat pelanggan pihak pertama yang terhubung langsung melalui perangkat pengembangan perangkat lunak perangkat (SDK) atau commercial-off-the-shelf (COTS) perangkat yang ditautkan secara tidak langsung melalui perangkat hub.

Setiap perangkat, terlepas dari jenis perangkat, dalam integrasi terkelola memiliki pengidentifikasi unik global yang disebut `a.managedThingId`. Pengenal ini digunakan dalam orientasi dan pengelolaan perangkat untuk seluruh siklus hidup perangkat. Ini sepenuhnya dikelola oleh integrasi terkelola dan unik untuk perangkat tertentu di semua integrasi terkelola secara keseluruhan. Wilayah AWS Ketika perangkat awalnya ditambahkan ke integrasi terkelola, pengenal ini dibuat dan dilampirkan ke hal yang dikelola dalam integrasi terkelola. Hal yang dikelola adalah representasi digital dari perangkat fisik dalam integrasi terkelola untuk mencerminkan semua metadata perangkat dari perangkat fisik. Untuk perangkat pihak ketiga, mereka mungkin memiliki pengidentifikasi unik terpisah khusus untuk cloud pihak ketiga mereka selain `managedThingId` disimpan dalam integrasi terkelola yang mewakili perangkat fisik.

Perangkat yang disediakan dapat memiliki status yang berbeda tergantung pada tahap aliran orientasi mereka. Daftar berikut menjelaskan setiap status penyediaan:

- **DIAKTIFKAN:** Perangkat telah ditemukan dan perintah dan kontrol tersedia.
- **DITEMUKAN:** Perangkat telah ditemukan tetapi perintah dan kontrol belum tersedia.
- **UNASSOCIATED:** Hal yang dikelola telah dibuat tetapi membutuhkan tindakan lebih lanjut untuk ditemukan. Itu tidak dapat dijangkau dari AWS Cloud atau pengontrol integrasi AWS IoT Terkelola (hub)
- **PRE_ASSOCIATED:** Hal yang dikelola telah dibuat dan siap untuk penemuan otomatis setelah dinyalakan atau terhubung. Itu tidak dapat dijangkau dari AWS Cloud atau pengontrol integrasi AWS IoT Terkelola (hub).
- **DELETE_IN_PROGRESS:** Proses penghapusan asinkron dimulai.
- **DIHAPUS:** Perangkat telah dihapus dari file AWS Cloud.

- **TERISOLASI:** Hal terkelola yang sebelumnya ditemukan atau diaktifkan yang tidak lagi dapat dijangkau. Misalnya, perangkat untuk cloud pihak ketiga yang asosiasi konektornya semuanya telah dihapus.

Alur orientasi berikut adalah untuk menyediakan hub Anda dengan integrasi terkelola:

[Onboard hub Anda ke integrasi terkelola](#): Siapkan penyedia inti dan plugin khusus protokol yang bekerja sama untuk menangani otentikasi, komunikasi, dan penyiapan perangkat.

Alur orientasi berikut disediakan untuk menyediakan perangkat yang terhubung dengan hub Anda dengan integrasi terkelola:

- [Pengaturan sederhana \(SS\)](#): Pengguna akhir memberi daya pada perangkat IoT dan memindai kode QR-nya menggunakan aplikasi produsen perangkat. Perangkat ini kemudian terdaftar ke cloud integrasi terkelola dan terhubung ke hub IoT.
- [Pengaturan tanpa sentuhan \(ZTS\)](#): Perangkat ini pra-terkait di hulu dalam rantai pasokan. Misalnya, alih-alih pengguna akhir memindai kode QR perangkat, langkah ini diselesaikan lebih awal untuk menautkan perangkat ke akun pelanggan.
- [Pengaturan yang dipandu pengguna \(UGS\)](#): Pengguna akhir memberi daya pada perangkat dan mengikuti langkah-langkah interaktif untuk memasukkannya ke integrasi terkelola. Ini mungkin termasuk menekan tombol pada hub IoT, menggunakan aplikasi produsen perangkat, atau menekan tombol pada hub dan perangkat. Anda dapat menggunakan metode ini jika Setup sederhana gagal.

Note

Alur kerja penyediaan perangkat dalam integrasi terkelola adalah agnostik dari persyaratan orientasi untuk perangkat. Integrasi terkelola menyediakan antarmuka pengguna yang efisien untuk orientasi dan pengelolaan perangkat, terlepas dari jenis perangkat atau protokol perangkat.

Siklus hidup profil perangkat dan perangkat

Mengelola siklus hidup perangkat dan profil perangkat memastikan armada perangkat Anda aman dan berjalan secara efisien.

Topik

- [Perangkat](#)
- [Profil perangkat](#)

Perangkat

Selama orientasi awal, integrasi terkelola menciptakan kembaran digital perangkat fisik Anda yang disebut Hal Terkelola. The Managed Thing memiliki `managedThingID` yang menyediakan pengidentifikasi unik global untuk mengidentifikasi perangkat dalam integrasi terkelola di semua wilayah. Perangkat berpasangan dengan hub lokal selama penyediaan komunikasi real-time dengan integrasi terkelola atau cloud pihak ketiga untuk perangkat pihak ketiga. Perangkat juga dikaitkan dengan pemilik sebagaimana diidentifikasi oleh `owner` parameter di publik APIs untuk Hal Terkelola seperti `GetManagedThing`. Perangkat ditautkan ke profil perangkat yang sesuai berdasarkan jenis perangkat.

Note

Perangkat fisik mungkin memiliki beberapa catatan jika disediakan beberapa kali di bawah pelanggan yang berbeda.

Siklus hidup perangkat dimulai dengan pembuatan Hal Terkelola dalam integrasi terkelola menggunakan `CreateManagedThing` API dan berakhir saat pelanggan menghapus Hal Terkelola menggunakan `DeleteManagedThing` API. Siklus hidup perangkat dikelola oleh publik berikut: APIs

- `CreateManagedThing`
- `ListManagedThings`
- `GetManagedThing`
- `UpdateManagedThing`
- `DeleteManagedThing`

Profil perangkat

Profil perangkat mewakili jenis perangkat tertentu seperti bola lampu atau bel pintu. Ini terkait dengan produsen dan berisi kemampuan perangkat. Profil perangkat menyimpan materi otentikasi yang diperlukan untuk permintaan penyiapan konektivitas perangkat dengan integrasi terkelola. Bahan otentikasi yang digunakan adalah kode batang perangkat.

Selama proses pembuatan perangkat, pabrikan dapat mendaftarkan profil perangkat mereka dengan integrasi terkelola. Hal ini memungkinkan pabrikan untuk mendapatkan bahan yang diperlukan untuk perangkat dari integrasi terkelola selama alur kerja orientasi dan penyediaan. Metadata dari profil perangkat disimpan di perangkat fisik atau dicetak pada label perangkat. Siklus hidup profil perangkat berakhir saat pabrikan menghapusnya dalam integrasi terkelola.

Model data

Model data mewakili hierarki organisasi tentang bagaimana data diatur dalam suatu sistem. Selain itu, ini mendukung end-to-end komunikasi di seluruh implementasi perangkat Anda. Untuk integrasi terkelola, ada dua model data yang digunakan. Model data integrasi terkelola dan AWS implementasi Model Data Materi. Mereka memiliki kesamaan, tetapi juga memiliki perbedaan halus yang diuraikan dalam topik berikut.

Untuk perangkat pihak ketiga, kedua model data digunakan untuk komunikasi antara pengguna akhir, integrasi terkelola, dan penyedia cloud pihak ketiga. Untuk menerjemahkan pesan seperti perintah perangkat dan peristiwa perangkat dari dua model data, fungsionalitas Cloud-to-Cloud Konektor dimanfaatkan.

Topik

- [Model data integrasi terkelola](#)
- [AWS implementasi model data Matter](#)
- [Skema model data](#)

Model data integrasi terkelola

Model data integrasi terkelola mengelola semua komunikasi antara pengguna akhir dan integrasi terkelola.

Hirarki Perangkat

Elemen endpoint dan capability data digunakan untuk menggambarkan perangkat dalam model data integrasi terkelola.

endpoint

endpoint ini mewakili antarmuka logis atau layanan yang ditawarkan oleh fitur.

```
{
  "endpointId": { "type": "string" },
  "capabilities": Capability[]
}
```

Capability

capabilityIni mewakili kemampuan perangkat.

```
{
  "$id": "string",           // Schema identifier (e.g. /schema-versions/
  capability/matter.OnOff@1.4)
  "name": "string",         // Human readable name
  "version": "string",      // e.g. 1.0
  "properties": Property[],
  "actions": Action[],
  "events": Event[]
}
```

Untuk elemen capability data, ada tiga item yang terdiri dari item tersebut:property,action, danevent. Mereka dapat digunakan untuk berinteraksi dengan dan memantau perangkat.

- Properti: Status yang dipegang oleh perangkat, seperti atribut tingkat kecerahan saat ini dari cahaya yang dapat diredukan.

```
{
  "name":           // Property Name is outside of Property Entity
  "value": Value,   // value represented in any type e.g. 4, "A", []
  "lastChangedAt": Timestamp // ISO 8601 Timestamp upto milliseconds yyyy-MM-
  ddTHH:mm:ss.ssssssZ
  "mutable": boolean,
  "retrievable": boolean,
  "reportable": boolean
}
```

- Tindakan: Tugas yang dapat dilakukan, seperti mengunci pintu pada kunci pintu. Tindakan dapat menghasilkan respons dan hasil.

```
{
  "name": { "$ref": "/schema-versions/definition/aws.name@1.0" }, //required
  "parameters": Map<String name, JSONNode value>,
  "responseCode": HTTPResponseCode,
  "errors": {
    "code": "string",
    "message": "string"
  }
}
```

- Peristiwa: Pada dasarnya catatan transisi keadaan masa lalu. Sementara property mewakili keadaan saat ini, peristiwa adalah jurnal masa lalu, dan mencakup penghitung yang meningkat secara monoton, stempel waktu, dan prioritas. Mereka memungkinkan menangkap transisi status, serta pemodelan data yang tidak mudah dicapai dengan. property

```
{
  "name": { "$ref": "/schema-versions/definition/aws.name@1.0" },      //
  required
  "parameters": Map<String name, JSONNode value>
}
```

AWS implementasi model data Matter

AWS Implementasi Model Data Matter mengelola semua komunikasi antara integrasi terkelola dan penyedia cloud pihak ketiga.

Untuk informasi selengkapnya, lihat [Model Data Materi: Sumber Daya Pengembang](#).

Hirarki Perangkat

Ada dua elemen data yang digunakan untuk menggambarkan perangkat: endpoint dan cluster.

endpoint

endpoint ini mewakili antarmuka logis atau layanan yang ditawarkan oleh fitur.

```
{
  "id": { "type": "string" },
  "clusters": Cluster[]
}
```

cluster

cluster ini mewakili kemampuan perangkat.

```
{
  "id": "hexadecimalString",
  "revision": "string"          // optional
  "attributes": AttributeMap<String attributeId, JSONNode>,
  "commands": CommandMap<String commandId, JSONNode>,
}
```

```
"events": EventMap<String eventId, JsonNode>
}
```

Untuk elemen `cluster data`, ada tiga item yang terdiri dari item tersebut: `attribute`, `command`, dan `event`. Mereka dapat digunakan untuk berinteraksi dengan dan memantau perangkat.

- Atribut: Status yang dipegang oleh perangkat, seperti atribut tingkat kecerahan saat ini dari cahaya yang dapat diredupkan.

```
{
  "id" (hexadecimalString): (JsonNode) value
}
```

- Perintah: Tugas yang dapat dilakukan, seperti mengunci pintu pada kunci pintu. Perintah dapat menghasilkan respons dan hasil.

```
"id": {
  "fieldId": "fieldValue",
  ...
  "responseCode": HTTPResponseCode,
  "errors": {
    "code": "string",
    "message": "string"
  }
}
```

- Peristiwa: Pada dasarnya catatan transisi keadaan masa lalu. Sementara `attributes` mewakili keadaan saat ini, peristiwa adalah jurnal masa lalu, dan mencakup penghitung yang meningkat secara monoton, stempel waktu, dan prioritas. Mereka memungkinkan menangkap transisi status, serta pemodelan data yang tidak mudah dicapai dengan `attributes`

```
"id": {
  "fieldId": "fieldValue",
  ...
}
```

Skema model data

Integrasi terkelola mendukung dua jenis skema: kemampuan dan definisi tipe. Jika Anda membuat model data kustom, Anda menggunakan dokumen skema JSON untuk menentukan salah satu jenis skema. Setiap dokumen skema memiliki batas 50.000 karakter.

Skema kemampuan

Kemampuan adalah blok bangunan mendasar yang mewakili fungsionalitas tertentu dalam titik akhir. Dengan kemampuan, Anda dapat memodelkan status dan perilaku perangkat menggunakan properti, tindakan, dan peristiwa. Properti memungkinkan Anda memodelkan atribut status perangkat secara fleksibel dengan tipe data deklaratif apa pun. Tindakan dan peristiwa memodelkan perilaku perangkat, termasuk perintah yang dapat dijalankan dan sinyal yang dapat dilaporkan.

Berikut ini menampilkan struktur tingkat tinggi dari skema kemampuan.

```
Capability
|
|-- Action
|-- Event
|-- Property
```

Tindakan

Entitas yang mewakili interaksi dengan kemampuan perangkat. Misalnya, membunyikan bel atau melihat siapa yang ada di pintu.

Peristiwa

Entitas yang mewakili peristiwa dari kemampuan perangkat. Perangkat dapat mengirim peristiwa untuk melaporkan insiden, alarm, atau aktivitas dari sensor seperti ketukan di pintu.

Properti

Entitas yang mewakili atribut tertentu dalam keadaan perangkat. Misalnya, bel berdering atau lampu teras menyala

Setiap kemampuan mencakup pengenalan namespace unik, informasi versi, dan deskripsi tujuannya. Dokumen skema menggunakan versi semantik untuk mempertahankan kompatibilitas mundur sambil mengaktifkan fitur baru.

Untuk informasi selengkapnya, lihat [Skema untuk definisi kemampuan](#).

Skema definisi tipe

Definisi tipe adalah tipe data terstruktur deklaratif yang memungkinkan penggunaan kembali dan kemampuan komposisi. Ini mendefinisikan bagaimana informasi harus diformat dan dibatasi. Gunakan definisi tipe untuk membuat format data standar di seluruh solusi IoT Anda.

Setiap definisi jenis meliputi:

- Pengenal namespaced unik
- Judul
- Deskripsi
- Properti yang menentukan pemformatan dan kendala data

Jenis dapat berupa primitif sederhana, seperti bilangan bulat atau string dengan batas yang ditentukan, atau struktur kompleks seperti enumerasi atau objek khusus dengan beberapa bidang. Definisi tipe menggunakan sintaks skema JSON untuk menentukan batasan termasuk nilai minimum dan maksimum, panjang string, dan pola yang diijinkan.

Untuk informasi selengkapnya, lihat [Skema untuk definisi tipe](#).

Skema untuk definisi kemampuan

Kemampuan didokumentasikan menggunakan dokumen JSON deklaratif yang memberikan kontrak yang jelas tentang bagaimana kemampuan harus berfungsi dalam sistem.

Untuk kemampuan, elemen wajib adalah `$id`, `nameExtrinsicId`, `extrinsicVersion` dan setidaknya satu elemen dalam setidaknya satu bagian berikut:

- `properties`
- `actions`
- `events`

Elemen opsional dalam suatu kemampuan adalah `$ref`, `title`, `description`, `version`, `$defs` dan `extrinsicProperties`. Untuk kemampuan, `$ref` harus mengacu pada `aws.capability`.

Bagian berikut merinci skema yang digunakan untuk definisi kemampuan.

\$ id (wajib)

Elemen `$id` mengidentifikasi definisi skema. Itu harus mengikuti struktur ini:

- Mulai dengan awalan `/schema-versions/` URI
- Sertakan `capability` jenis skema
- Gunakan garis miring maju (`/`) sebagai pemisah jalur URI

- Sertakan identitas skema, dengan fragmen yang dipisahkan oleh periode () .
- Gunakan @ karakter untuk memisahkan ID skema dan versi
- Akhiri dengan versi semver, menggunakan period (.) untuk memisahkan fragmen versi

Identitas skema harus dimulai dengan namespace root yang panjangnya 3-12 karakter, diikuti oleh sub-namespace dan nama opsional.

Versi semver mencakup versi MAJOR (hingga 3 digit), versi MINOR (hingga 3 digit), dan versi PATCH opsional (hingga 4 digit).

Note

Anda tidak dapat menggunakan ruang nama aws yang dicadangkan atau matter

Example Contoh \$ id

```
/schema-version/capability/aws.Recording@1.0
```

\$ ref

\$refElemen mengacu pada kemampuan yang ada dalam sistem. Ini mengikuti kendala yang sama dengan elemen. \$id

Note

Definisi atau kemampuan tipe harus ada dengan nilai yang disediakan dalam \$ref file.

Example Contoh \$ ref

```
/schema-version/definition/aws.capability@1.0
```

nama (wajib)

Elemen nama adalah string yang mewakili nama entitas dalam dokumen skema. Ini sering berisi singkatan dan harus mengikuti aturan ini:

- Hanya berisi karakter alfanumerik, titik (.), garis miring maju (/), tanda hubung (-), dan spasi
- Mulailah dengan surat
- Maksimal 64 karakter

Elemen nama digunakan di UI dan dokumentasi konsol Amazon Web Services.

Example Contoh nama

```
Door Lock
On/Off
Wi-Fi Network Management
PM2.5 Concentration Measurement
RTCSessionController
Energy EVSE
```

title

Elemen judul adalah string deskriptif untuk entitas yang diwakili oleh dokumen skema. Ini dapat berisi karakter apa pun dan digunakan dalam dokumentasi. Panjang maksimum untuk judul kemampuan adalah 256 karakter.

Example Judul contoh

```
Real-time Communication (RTC) Session Controller
Energy EVSE Capability
```

deskripsi

`description`Elemen memberikan penjelasan rinci tentang entitas yang diwakili oleh dokumen skema. Ini dapat berisi karakter apa pun dan digunakan dalam dokumentasi. Panjang maksimum untuk deskripsi kemampuan adalah 2048 karakter

Example Deskripsi contoh

```
Electric Vehicle Supply Equipment (EVSE) is equipment used to charge an Electric
Vehicle (EV) or Plug-In Hybrid Electric Vehicle.
    This capability provides an interface to the functionality of Electric
Vehicle Supply Equipment (EVSE) management.
```

versi

Elemen `version` bersifat opsional. Ini adalah string yang mewakili versi dokumen skema. Hal ini memiliki batasan berikut:

- Menggunakan format semver, dengan fragmen versi berikut dipisahkan oleh `.` (periode).
 - MAJORversi, maksimal 3 digit
 - MINORversi, maksimal 3 digit
 - PATCHversi (opsional), maksimal 4 digit
- Panjangnya bisa antara 3 dan 12 karakter.

Example Contoh versi

1.0

1.12

1.4.1

Bekerja dengan versi kemampuan

Kemampuan adalah entitas berversi yang tidak dapat diubah. Setiap perubahan diharapkan untuk membuat versi baru. Sistem menggunakan versi semantik dengan format MAJOR.MINOR.PATCH, di mana:

- Versi MAJOR meningkat saat membuat perubahan API yang tidak kompatibel ke belakang
- Versi MINOR meningkat saat menambahkan fungsionalitas dengan cara yang kompatibel ke belakang
- Versi PATCH meningkat saat membuat penambahan kecil yang tidak berdampak pada kemampuan.

Kemampuan yang berasal dari cluster Matter didasarkan pada versi 1.4 dan setiap rilis Matter diharapkan akan diimpor ke dalam sistem. Karena versi Matter menggunakan level semver MAJOR dan MINOR, integrasi terkelola hanya dapat menggunakan versi PATCH.

Saat Anda menambahkan versi PATCH untuk Matter, pastikan untuk memperhitungkan bahwa Matter menggunakan revisi berurutan. Semua versi PATCH harus mematuhi revisi yang didokumentasikan dalam spesifikasi Matter dan ini harus kompatibel ke belakang.

Untuk memperbaiki masalah yang tidak kompatibel ke belakang, Anda harus bekerja dengan Connectivity Standards Alliance (CSA) untuk menyelesaikannya dalam spesifikasi dan mendapatkan revisi baru yang dirilis.

AWS Kemampuan -managed dirilis dengan versi awal. 1.0 Dengan ini, ketiga level versi dapat digunakan.

ExtrinsicVersion (wajib)

Ini adalah string yang mewakili versi yang dikelola di luar AWS IoT sistem. Untuk kemampuan Matter, `extrinsicVersion` petakan ke `revision`

Ini direpresentasikan sebagai nilai bilangan bulat yang dirangkai, dan panjangnya bisa dari 1 hingga 10 digit numerik.

Example Contoh versi

7

1567

ExtrinsicId (wajib)

`extrinsicId` elemen tersebut mewakili pengenal yang dikelola di luar sistem IoT Amazon Web Services. Untuk kemampuan Matter, ia memetakan `clusterId`, `attributeId`, `commandId`, `eventId`, atau `fieldId`, tergantung pada konteksnya.

`extrinsicId` dapat berupa bilangan bulat desimal stringifikasi (1-10 digit) atau bilangan bulat heksadesimal stringifikasi (awalan 0x atau 0X, diikuti oleh 1-8 digit heksadesimal).

Note

Untuk AWS, Vendor ID (VID) adalah 0x1577, dan untuk Matter, itu adalah 0. Sistem memastikan bahwa skema khusus tidak menggunakan ini yang dicadangkan VIDs untuk kemampuan.

Example Contoh ekstrinsicids

```
0018
0x001A
0x15771002
```

\$defs

`$defs` Bagian ini adalah peta sub-skema yang dapat direferensikan dalam dokumen skema sebagaimana diizinkan oleh skema JSON. Dalam peta ini, kuncinya digunakan dalam definisi referensi lokal dan nilainya memberikan skema JSON.

Note

Sistem hanya memberlakukan peta yang `$defs` valid dan bahwa setiap sub-skema adalah skema JSON yang valid. Tidak ada aturan tambahan yang diberlakukan.

Ikuti kendala ini saat bekerja dengan definisi:

- Gunakan hanya karakter ramah URI dalam nama definisi
- Pastikan setiap nilai adalah sub-skema yang valid
- Sertakan sejumlah sub-skema yang sesuai dengan batas ukuran dokumen skema

ExtrinsicProperties

`extrinsicProperties` Elemen berisi seperangkat properti yang didefinisikan dalam sistem eksternal tetapi dipertahankan dalam model data. Untuk kemampuan Matter, ia memetakan ke berbagai elemen yang tidak dimodelkan atau sebagian dimodelkan dalam cluster, atribut, perintah, atau peristiwa ZCL.

Properti Ekstrinsik harus mengikuti kendala ini:

- Nama properti harus alfanumerik tanpa spasi atau karakter khusus
- Nilai properti dapat berupa nilai skema JSON
- Maksimal 20 properti

Sistem ini mendukung berbagai `extrinsicProperties`, termasuk `access`, `apiMaturity`, `cli`, `cliFunctionName`, dan lain-lain. Properti ini memfasilitasi ACL untuk AWS (dan sebaliknya) transformasi model data.

Note

Properti ekstrinsik didukung untuk `action`, `eventproperty`, dan elemen `struct` bidang kemampuan, tetapi tidak untuk kemampuan atau cluster itu sendiri.

Properti ekstrinsik yang didukung sistem

Sistem melacak atribut cluster, atribut, perintah, atau peristiwa yang tidak dimodelkan atau sebagian dimodelkan berikut seperti `extrinsicProperties` selama transformasi ke atau dari ZCL:

`access`

Setiap objek akses berisi yang berikut:

- `op`- Operasi dimodelkan sebagai enum dengan nilai: `read`, `write`, atau `invoke`
- `privilege`- Hak istimewa dimodelkan sebagai enum dengan nilai: `view`, `proxy_viewoperate`, `manage` atau `administer`
- `role`- String tak terbatas yang mewakili peran operator

`apiMaturity`

String polos tak terbatas yang mewakili tingkat kematangan. Ini dimodelkan dalam ZCL sebagai enum dengan nilai: `stable`, `provisional`, `internal` atau `deprecated`

`side`

Dimodelkan sebagai enum dengan nilai: `either`, dan `server client`

Properti Boolean

Properti berikut adalah bendera boolean:

- `isFabricScoped`
- `isFabricSensitive`
- `mustUseAtomicWrite`
- `mustUseTimedInvoke`

Properti string

Properti berikut direpresentasikan sebagai string tak terbatas:

- `cli`
- `cliFunctionName`
- `functionName`
- `group`
- `introducedIn`
- `manufacturerCode`
- `noDefaultImplementation`
- `presentIf`
- `priority`
- `removedIn`
- `reportableChange`
- `reportMinInterval`
- `reportMaxInterval`
- `restriction`
- `storage`

Pertimbangan transformasi

Untuk transformasi ZCL, `extrinsicProperties` disimpan dalam peta tanpa pemrosesan. Skema khusus yang menggunakan penemuan tidak mengalami transformasi ZCL. Namun, jika Anda berencana untuk mengimplementasikan transformasi ZCL untuk skema kustom di masa mendatang, Anda harus memodelkan semua tipe string polos tak terbatas `extrinsicProperties` dan menentukan batasan seperti enum, pola (regex), dan panjang. Persiapan ini memastikan penanganan yang tepat dari sifat-sifat ini selama transformasi.

Sebaliknya, AWS untuk transformasi konektor, `extrinsicProperties` tidak termasuk sama sekali, karena rincian ini tidak diperlukan dalam format konektor.

Properti

Properti mewakili status kemampuan yang dikelola perangkat. Setiap negara didefinisikan sebagai pasangan kunci-nilai, di mana kunci menggambarkan nama negara dan nilai menggambarkan definisi negara.

Saat bekerja dengan properti, ikuti batasan ini:

- Gunakan hanya karakter alfanumerik dalam nama properti, tanpa spasi atau karakter khusus
- Sertakan sejumlah properti yang sesuai dengan batas ukuran dokumen skema

Bekerja dengan properti

Properti dalam kemampuan adalah elemen mendasar yang mewakili keadaan tertentu dari perangkat yang didukung oleh integrasi terkelola. Ini mewakili kondisi atau konfigurasi perangkat saat ini. Dengan menstandarisasi bagaimana properti ini didefinisikan dan terstruktur, sistem rumah pintar memastikan bahwa perangkat dari produsen yang berbeda dapat berkomunikasi secara efektif, menciptakan pengalaman yang mulus dan dapat dioperasikan.

Untuk properti kemampuan, elemen wajib adalah `extrinsicId` dan `value`. Elemen opsional dalam properti kemampuan adalah `description`, `retrievable`, `mutable`, `reportable` dan `extrinsicProperties`.

Nilai

Struktur tak terbatas yang memungkinkan pembangun untuk menempatkan batasan yang sesuai dengan skema JSON untuk menentukan tipe data properti ini.

Saat mendefinisikan nilai, ikuti batasan ini:

- Untuk tipe sederhana, gunakan `type` dan batasan skema JSON asli lainnya seperti `maxLength` `maximum`
- Untuk jenis komposit, gunakan `oneOf`, `allOf`, atau `anyOf`. Sistem tidak mendukung not kata kunci
- Untuk merujuk ke tipe global apa pun, gunakan `$ref` dengan referensi yang dapat ditemukan yang valid
- Untuk nullability, ikuti definisi skema tipe OpenAPI dengan menyediakan atribut nullable dengan flag boolean (jika null adalah nilai yang diizinkan) `true`

Contoh:

```
{
  "$ref": "/schema-versions/definition/matter.uint16@1.4",
  "nullable": true,
  "maximum": 4096
}
```

Dapat diambil

Boolean yang menjelaskan apakah status dapat dibaca atau tidak.

Aspek keterbacaan negara ditangguhkan pada implementasi kemampuan perangkat. Perangkat memutuskan apakah status tertentu dapat dibaca atau tidak. Aspek negara ini belum didukung untuk dilaporkan dalam laporan kemampuan dan karenanya tidak ditegakkan dalam sistem.

Contoh: `true` atau `false`

bisa berubah

Boolean yang menjelaskan apakah status dapat ditulis atau tidak.

Aspek kemampuan tulis negara ditangguhkan pada implementasi kemampuan perangkat. Perangkat memutuskan apakah status tertentu dapat ditulis atau tidak. Aspek negara ini belum didukung untuk dilaporkan dalam laporan kemampuan dan karenanya tidak ditegakkan dalam sistem.

Contoh: `true` atau `false`

Dilapor

Boolean yang menjelaskan jika status dilaporkan oleh perangkat ketika ada perubahan status.

Aspek reportabilitas negara ditangguhkan pada implementasi kemampuan perangkat. Perangkat memutuskan apakah status tertentu dapat dilaporkan atau tidak. Aspek negara ini belum didukung untuk dilaporkan dalam laporan kemampuan dan karenanya tidak ditegakkan dalam sistem.

Contoh: `true` atau `false`

Tindakan

Tindakan adalah operasi yang dikelola skema yang mengikuti model permintaan-respons. Setiap tindakan mewakili operasi yang diimplementasikan perangkat.

Ikuti kendala ini saat menerapkan tindakan:

- Sertakan hanya tindakan unik dalam larik tindakan

- Sertakan sejumlah tindakan yang sesuai dengan batas ukuran dokumen skema

Bekerja dengan tindakan

Tindakan adalah cara standar untuk berinteraksi dengan dan mengontrol kemampuan perangkat dalam sistem integrasi terkelola. Ini merupakan perintah atau operasi tertentu yang dapat dijalankan pada perangkat, lengkap dengan format terstruktur untuk memodelkan permintaan atau parameter respons yang diperlukan. Tindakan ini berfungsi sebagai jembatan antara niat pengguna dan operasi perangkat, memungkinkan kontrol yang konsisten dan andal di berbagai jenis perangkat pintar.

Untuk suatu tindakan, elemen wajib adalah `name` dan `extrinsicId`. Elemen opsional adalah `description`, `extrinsicProperties`, `request` dan `response`.

Deskripsi

Deskripsi memiliki batasan panjang maksimum 1536 karakter.

Permintaan

Bagian permintaan bersifat opsional dan dapat dihilangkan jika tidak ada parameter permintaan. Jika dihilangkan, sistem mendukung pengiriman permintaan tanpa muatan apa pun hanya dengan menggunakan nama. Action ini digunakan dalam tindakan sederhana, seperti menyalakan atau mematikan lampu.

Tindakan kompleks membutuhkan parameter tambahan. Misalnya, permintaan untuk melakukan streaming rekaman kamera mungkin mencakup parameter tentang protokol streaming yang akan digunakan atau apakah akan mengirim aliran ke perangkat tampilan tertentu.

Untuk permintaan tindakan, elemen wajibnya adalah `parameters`. Elemen opsional adalah `description`, `extrinsicId`, dan `extrinsicProperties`.

Minta deskripsi

Deskripsi mengikuti format yang sama dengan bagian 3.5, dengan panjang maksimum 2048 karakter.

Respons

Dalam integrasi terkelola, untuk permintaan tindakan apa pun yang dikirim melalui [SendManagedThingCommand](#) API, permintaan mencapai perangkat dan mengharapkan respons asinkron kembali. Respons tindakan mendefinisikan struktur respons ini.

Untuk permintaan tindakan, elemen wajibnya adalah `parameters`. Elemen opsional adalah `name`, `description`, `extrinsicId`, `extrinsicProperties`, `errors` dan `responseCode`.

Deskripsi respons

Deskripsi mengikuti format yang sama dengan [deskripsi](#), dan memiliki panjang maksimum 2048 karakter.

Nama respons

Nama mengikuti format yang sama seperti [nama \(wajib\)](#), dengan rincian tambahan ini:

- Nama konvensional respons diturunkan dengan Response menambahkan nama tindakan.
- Jika Anda ingin menggunakan nama yang berbeda, Anda dapat memberikannya dalam `name` elemen ini. Jika `name` diberikan dalam respons, maka nilai ini lebih diutamakan daripada nama konvensional.

Kesalahan

Array pesan unik tak terbatas yang disediakan dalam respons, jika ada kesalahan saat memproses permintaan.

Batasan:

- Item pesan dideklarasikan sebagai objek JSON dengan bidang berikut:
 - `code`: String yang berisi karakter alfanumerik dan `_` (garis bawah), dengan panjang antara 1 hingga 64 karakter
 - `message`: Nilai string tak terbatas

Example Contoh pesan kesalahan

```
"errors": [  
  {  
    "code": "AD_001",  
    "message": "Unable to receive signal from the sensor. Please check connection  
with the sensor."  
  }  
]
```

Kode respons

Kode integer yang menampilkan bagaimana permintaan ditangani. Sebaiknya kode perangkat mengembalikan kode menggunakan spesifikasi kode status respons server HTTP untuk memungkinkan keseragaman dalam sistem.

Kendala: Nilai integer mulai dari 100 hingga 599.

Parameter permintaan atau respons

Bagian parameter didefinisikan sebagai peta nama dan pasangan sub-skema. Sejumlah parameter dapat didefinisikan dalam parameter permintaan, jika mereka dapat masuk dalam dokumen skema.

Nama parameter hanya dapat berisi karakter alfanumerik. Spasi atau karakter lainnya tidak diperbolehkan.

bidang parameter

Elemen wajib dalam a parameter adalah `extrinsicId` dan `value`. Elemen opsional adalah `description` dan `extrinsicProperties`.

Elemen deskripsi mengikuti format yang sama dengan [deskripsi](#), dengan panjang maksimum 1024 karakter.

`extrinsicId` dan **`extrinsicProperties`** mengesampingkan

`extrinsicId` dan `extrinsicProperties` ikuti format yang sama dengan [Extrinsicid \(wajib\)](#) dan [ExtrinsicProperties](#), dengan detail tambahan ini:

- Jika `extrinsicId` diberikan dalam permintaan atau respons, nilai ini lebih diutamakan daripada nilai yang diberikan pada tingkat tindakan. Sistem harus menggunakan request/response level `extrinsicId` terlebih dahulu, jika hilang gunakan level tindakan `extrinsicId`
- Jika `extrinsicProperties` disediakan dalam permintaan atau respons, properti ini lebih diutamakan daripada nilai `value` yang diberikan pada tingkat tindakan. Sistem harus mengambil level tindakan `extrinsicProperties` dan mengganti pasangan kunci-nilai yang disediakan di level tersebut request/response `extrinsicProperties`

Example Contoh penggantian Extrinsicid dan ExtrinsicProperties

```
{
```

```
"name": "ToggleWithEffect",
"extrinsicId": "0x0001",

"extrinsicProperties": {
  "apiMaturity": "provisional",
  "introducedIn": "1.2"
},
"request": {
  "extrinsicProperties": {
    "apiMaturity": "stable",
    "manufacturerCode": "XYZ"
  },
  "parameters": {
    ...
  }
},
"response": {
  "extrinsicProperties": {
    "noDefaultImplementation": true
  },
  "parameters": {
{
    ...
  }
}
}
```

Dalam contoh di atas, nilai efektif untuk permintaan tindakan adalah:

```
# effective request
"name": "ToggleWithEffect",
"extrinsicId": "0x0001",
"extrinsicProperties": {
  "apiMaturity": "stable",
  "introducedIn": "1.2"
  "manufacturerCode": "XYZ"
},
"parameters": {
  ...
}

# effective response
"name": "ToggleWithEffectResponse",
```

```
"extrinsicId": "0x0001",
"extrinsicProperties": {
  "apiMaturity": "provisional",
  "introducedIn": "1.2"
  "noDefaultImplementation": true
},
"parameters": {
  ...
}
```

Tindakan bawaan

Untuk semua kemampuan, Anda dapat melakukan tindakan khusus menggunakan kata kunci `ReadState` dan `UpdateState`. Kedua kata kunci tindakan ini akan bertindak berdasarkan properti kemampuan yang didefinisikan dalam model data.

ReadState

Mengirim perintah ke `managedThing` untuk membaca nilai-nilai properti statusnya. Gunakan `ReadState` sebagai cara untuk memaksa status perangkat diperbarui.

UpdateState

Mengirim perintah untuk memperbarui beberapa properti.

Memaksa sinkronisasi status perangkat mungkin berguna dalam skenario berikut:

1. Perangkat itu offline untuk jangka waktu tertentu dan tidak memancarkan peristiwa.
2. Perangkat baru saja disediakan dan belum memiliki status apa pun yang dipertahankan di cloud.
3. Status perangkat tidak sinkron dengan keadaan perangkat yang sebenarnya.

ReadState contoh

Periksa apakah lampu menyala atau mati menggunakan [SendManagedThingCommandAPI](#):

```
{
  "Endpoints": [
    {
      "endpointId": "1",
      "capabilities": [
        {
```

```

    "id": "aws.OnOff",
    "name": "On/Off",
    "version": "1",
    "actions": [
      {
        "name": "ReadState",
        "parameters": {
          "propertiesToRead": [ "OnOff" ]
        }
      }
    ]
  }
]
}

```

Baca semua properti status untuk `matter.OnOff` kemampuan:

```

{
  "Endpoints": [
    {
      "endpointId": "1",
      "capabilities": [
        {
          "id": "aws.OnOff",
          "name": "On/Off",
          "version": "1",
          "actions": [
            {
              "name": "ReadState",
              "parameters": {
                "propertiesToRead": [ "*" ]
                // Use the wildcard operator to read ALL state properties for a
                capability
              }
            }
          ]
        }
      ]
    }
  ]
}

```

UpdateState contoh

Ubah lampu OnTime untuk menggunakan [SendManagedThingCommandAPI](#):

```
{
  "Endpoints": [
    {
      "endpointId": "1",
      "capabilities": [
        {
          "id": "matter.OnOff",
          "name": "On/Off",
          "version": "1",
          "actions": [
            {
              "name": "UpdateState",
              "parameters": {
                "OnTime": 5
              }
            }
          ]
        }
      ]
    }
  ]
}
```

Peristiwa

Peristiwa adalah sinyal searah yang dikelola skema yang diimplementasikan oleh perangkat.

Menerapkan acara sesuai dengan kendala ini:

- Sertakan hanya acara unik dalam array acara
- Sertakan sejumlah peristiwa yang sesuai dengan batas ukuran dokumen skema

Peristiwa dalam sistem integrasi terkelola

Bekerja dengan acara

Peristiwa adalah cara standar untuk secara proaktif belajar tentang perubahan pada perangkat atau lingkungannya. Ini merupakan peristiwa model yang akan dikirim perangkat ke cloud untuk

memberikan informasi tentang sesuatu yang dimodifikasi pada perangkat atau dirasakan di lingkungannya. Karena peristiwa ini dimodelkan, pelanggan dapat menggunakannya dalam aliran kontrol untuk bereaksi terhadap peristiwa tertentu dan detail yang disediakan di dalamnya.

Untuk suatu acara, elemen wajib adalah `name` dan `extrinsicId`. Elemen opsional adalah `description`, `extrinsicProperties`, dan `request`.

Deskripsi

Deskripsi mengikuti format yang sama seperti yang dijelaskan dalam [deskripsi](#), dengan panjang maksimum 512 karakter.

Permintaan

`request` Bagian ini opsional dan dapat dihilangkan jika tidak ada parameter permintaan. Jika dihilangkan, sistem mendukung perangkat yang mengirim permintaan acara tanpa muatan apa pun hanya dengan menggunakan nama Acara. Ini digunakan dalam peristiwa sederhana, seperti kegagalan sensor pada pompa atau jika alarm dimatikan pada alarm asap atau karbon monoksida.

Tindakan kompleks membutuhkan parameter tambahan. Misalnya, permintaan untuk melakukan streaming rekaman kamera mungkin mencakup parameter tentang protokol streaming yang akan digunakan atau apakah akan mengirim aliran ke perangkat tampilan tertentu.

Untuk permintaan acara, elemen wajibnya adalah `parameters`. Tidak ada elemen opsional.

Respons

Respons peristiwa saat ini tidak didukung.

Skema untuk definisi tipe

Bagian berikut merinci skema yang digunakan untuk definisi tipe.

\$ id

Elemen `$id` mengidentifikasi definisi skema. Itu harus mengikuti struktur ini:

- Mulai dengan awalan `/schema-versions/` URI
- Sertakan `definition` jenis skema
- Gunakan garis miring maju (`/`) sebagai pemisah jalur URI
- Sertakan identitas skema, dengan fragmen yang dipisahkan oleh periode (`.`) .

- Gunakan @ karakter untuk memisahkan ID skema dan versi
- Akhiri dengan versi semver, menggunakan period (.) untuk memisahkan fragmen versi

Identitas skema harus dimulai dengan namespace root yang panjangnya 3-12 karakter, diikuti oleh sub-namespace dan nama opsional.

Versi semver mencakup versi MAJOR (hingga 3 digit), versi MINOR (hingga 3 digit), dan versi PATCH opsional (hingga 4 digit).

Note

Anda tidak dapat menggunakan ruang nama aws yang dicadangkan atau matter

Example Contoh \$ id

```
/schema-version/capability/aws.Recording@1.0
```

\$ ref

Elemen \$ref mereferensikan definisi tipe yang ada dalam sistem. Ini mengikuti kendala yang sama dengan elemen. \$id

Note

Definisi atau kemampuan tipe harus ada dengan nilai yang disediakan dalam \$ref file.

Example Contoh \$ ref

```
/schema-version/definition/aws.capability@1.0
```

name

Elemen nama adalah string yang mewakili nama entitas dalam dokumen skema. Ini sering berisi singkatan dan harus mengikuti aturan ini:

- Hanya berisi karakter alfanumerik, titik (.), garis miring maju (/), tanda hubung (-), dan spasi
- Mulailah dengan surat

- Maksimal 192 karakter

Elemen nama digunakan di UI dan dokumentasi konsol Amazon Web Services.

Example Contoh nama

```
Door Lock
On/Off
Wi-Fi Network Management
PM2.5 Concentration Measurement
RTCSessionController
Energy EVSE
```

title

Elemen judul adalah string deskriptif untuk entitas yang diwakili oleh dokumen skema. Ini dapat berisi karakter apa pun dan digunakan dalam dokumentasi.

Example Judul contoh

```
Real-time Communication (RTC) Session Controller
Energy EVSE Capability
```

deskripsi

`description` Elemen memberikan penjelasan rinci tentang entitas yang diwakili oleh dokumen skema. Ini dapat berisi karakter apa pun dan digunakan dalam dokumentasi.

Example Deskripsi contoh

```
Electric Vehicle Supply Equipment (EVSE) is equipment used to charge an Electric
Vehicle (EV) or Plug-In Hybrid Electric Vehicle.
    This capability provides an interface to the functionality of Electric
Vehicle Supply Equipment (EVSE) management.
```

Ekstrinsicid

`extrinsicId` Elemen tersebut mewakili pengenal yang dikelola di luar sistem IoT Amazon Web Services. Untuk kemampuan Matter, ia memetakan `keclusterId`, `attributeId`, `commandId`, `eventId`, `ataufieldId`, tergantung pada konteksnya.

`extrinsicId` dapat berupa bilangan bulat desimal stringifikasi (1-10 digit) atau bilangan bulat heksadesimal stringifikasi (awalan `0x` atau `0X`, diikuti oleh 1-8 digit heksadesimal).

Note

Untuk AWS, Vendor ID (VID) adalah `0x1577`, dan untuk Matter, itu adalah `0`. Sistem memastikan bahwa skema khusus tidak menggunakan ini yang dicadangkan VIDs untuk kemampuan.

Example Contoh ekstrinsicids

```
0018
0x001A
0x15771002
```

ExtrinsicProperties

`extrinsicProperties` Elemen berisi seperangkat properti yang didefinisikan dalam sistem eksternal tetapi dipertahankan dalam model data. Untuk kemampuan Matter, ia memetakan ke berbagai elemen yang tidak dimodelkan atau sebagian dimodelkan dalam cluster, atribut, perintah, atau peristiwa ZCL.

Properti Ekstrinsik harus mengikuti kendala ini:

- Nama properti harus alfanumerik tanpa spasi atau karakter khusus
- Nilai properti dapat berupa nilai skema JSON
- Maksimal 20 properti

Sistem ini mendukung berbagai `extrinsicProperties`, termasuk `access`, `apiMaturity`, `cli`, `cliFunctionName`, dan lain-lain. Properti ini memfasilitasi ACL untuk AWS (dan sebaliknya) transformasi model data.

Note

Properti ekstrinsik didukung untuk `action`, `eventproperty`, dan elemen `struct` bidang kemampuan, tetapi tidak untuk kemampuan atau cluster itu sendiri.

Properti ekstrinsik yang didukung sistem

Sistem melacak atribut cluster, atribut, perintah, atau peristiwa yang tidak dimodelkan atau sebagian dimodelkan berikut seperti `extrinsicProperties` selama transformasi ke atau dari ZCL:

`access`

Setiap objek akses berisi yang berikut:

- `op`- Operasi dimodelkan sebagai enum dengan nilai: `read`, `write`, atau `invoke`
- `privilege`- Hak istimewa dimodelkan sebagai enum dengan nilai: `view`, `proxy_viewoperate`, `manage` atau `administer`
- `role`- String tak terbatas yang mewakili peran operator

`apiMaturity`

String polos tak terbatas yang mewakili tingkat kematangan. Ini dimodelkan dalam ZCL sebagai enum dengan nilai: `stable`, `provisional`, `internal` atau `deprecated`

`side`

Dimodelkan sebagai enum dengan nilai: `either`, dan `server client`

Properti Boolean

Properti berikut adalah bendera boolean:

- `isFabricScoped`
- `isFabricSensitive`
- `mustUseAtomicWrite`
- `mustUseTimedInvoke`

Properti string

Properti berikut direpresentasikan sebagai string tak terbatas:

- `cli`
- `cliFunctionName`
- `functionName`
- `group`
- `introducedIn`
- `manufacturerCode`

- `noDefaultImplementation`
- `presentIf`
- `priority`
- `removedIn`
- `reportableChange`
- `reportMinInterval`
- `reportMaxInterval`
- `restriction`
- `storage`

Pertimbangan transformasi

Untuk transformasi ZCL, `extrinsicProperties` disimpan dalam peta tanpa pemrosesan. Skema khusus yang menggunakan penemuan tidak mengalami transformasi ZCL. Namun, jika Anda berencana untuk mengimplementasikan transformasi ZCL untuk skema kustom di masa mendatang, Anda harus memodelkan semua tipe string polos tak terbatas `extrinsicProperties` dan menentukan batasan seperti enum, pola (regex), dan panjang. Persiapan ini memastikan penanganan yang tepat dari sifat-sifat ini selama transformasi.

Sebaliknya, AWS untuk transformasi konektor, `extrinsicProperties` tidak termasuk sama sekali, karena rincian ini tidak diperlukan dalam format konektor.

Membangun dan menggunakan definisi tipe dalam dokumen skema kemampuan

Semua elemen dalam skema memutuskan untuk mengetik definisi. Definisi tipe ini adalah definisi tipe primitif (seperti boolean, string, angka) atau definisi tipe namespaced (definisi tipe yang dibangun dari definisi tipe primitif untuk kenyamanan).

Saat Anda mendefinisikan skema khusus, Anda dapat menggunakan definisi primitif dan definisi tipe namespace.

Daftar Isi

- [Definisi tipe primitif](#)
 - [Boolean](#)

- [Dukungan tipe integer](#)
- [Nomor](#)
- [String](#)
- [Nulls](#)
- [Array](#)
- [Objek](#)
- [Definisi tipe namespaced](#)
 - [Jenis matter](#)
 - [Jenis aws](#)
 - [Definisi jenis bitmap](#)
 - [Definisi tipe enum](#)

Definisi tipe primitif

Definisi tipe primitif adalah blok bangunan untuk semua definisi tipe yang didefinisikan dalam integrasi terkelola. Semua definisi namespace, termasuk definisi tipe kustom, diselesaikan ke definisi tipe primitif baik melalui `$ref` kata kunci atau kata kunci `type`.

Semua tipe primitif dapat dibatalkan dengan menggunakan `nullable` kata kunci, dan Anda dapat mengidentifikasi semua tipe primitif dengan menggunakan kata kunci `type`.

Boolean

Jenis Boolean mendukung nilai default.

Definisi sampel:

```
{
  "type" : "boolean",
  "default" : "false",
  "nullable" : true
}
```

Dukungan tipe integer

Jenis integer mendukung hal berikut:

- Nilai default
- Nilai maximum
- Nilai minimum
- Nilai exclusiveMaximum
- Nilai exclusiveMinimum
- Nilai multipleOf

Jika nilai yang x divalidasi, berikut ini harus benar:

- $x \geq \text{minimum}$
- $x > \text{exclusiveMinimum}$
- $x < \text{exclusiveMaximum}$

Note

Angka dengan bagian pecahan nol dianggap bilangan bulat, tetapi angka floating point ditolak.

```
1.0 // Schema-Compliant
3.1415926 // NOT Schema-Compliant
```

Meskipun Anda dapat menentukan keduanya minimum dan exclusiveMinimum atau keduanya maximum dan exclusiveMaximum, kami tidak menyarankan untuk menggunakan keduanya secara bersamaan.

Contoh definisi:

```
{
  "type" : "integer",
  "default" : 2,
  "nullable" : true,
  "maximum" : 10,
  "minimum" : 0,
  "multipleOf": 2
}
```

Definisi alternatif:

```
{
  "type" : "integer",
  "default" : 2,
  "nullable" : true,
  "exclusiveMaximum" : 11,
  "exclusiveMinimum" : -1,
  "multipleOf": 2
}
```

Nomor

Gunakan jenis angka untuk jenis numerik apa pun, termasuk bilangan bulat dan angka floating point.

Jenis nomor mendukung hal berikut:

- Nilai default
- Nilai maximum
- Nilai minimum
- Nilai exclusiveMaximum
- Nilai exclusiveMinimum
- multipleOfnilai. Kelipatannya bisa menjadi nomor floating point.

Jika nilai yang x divalidasi, berikut ini harus benar:

- $x \geq \text{minimum}$
- $x > \text{exclusiveMinimum}$
- $x < \text{exclusiveMaximum}$

Meskipun Anda dapat menentukan keduanya minimum dan exclusiveMinimum atau keduanya maximum dan exclusiveMaximum, kami tidak menyarankan untuk menggunakan keduanya secara bersamaan.

Contoh definisi:

```
{
```

```
"type" : "number",
"default" : 0.4,
"nullable" : true,
"maximum" : 10.2,
"minimum" : 0.2,
"multipleOf": 0.2
}
```

Definisi alternatif:

```
{
  "type" : "number",
  "default" : 0.4,
  "nullable" : true,
  "exclusiveMaximum" : 10.2,
  "exclusiveMinimum" : 0.2,
  "multipleOf": 0.2
}
```

String

Jenis string mendukung hal berikut:

- Nilai default
- kendala panjang (harus angka non-negatif) termasuk dan nilai `maxLength` `minLength`
- `pattern` nilai untuk ekspresi reguler

Saat Anda mendefinisikan ekspresi reguler, string valid jika ekspresi cocok di mana saja di dalam string. Misalnya, ekspresi reguler `p` cocok dengan string apa pun yang mengandung `p`, seperti `"apple"`, bukan hanya string `"p"`. Untuk kejelasan, kami sarankan mengelilingi ekspresi reguler dengan `^...$` (misalnya, `^p$`), kecuali Anda memiliki alasan khusus untuk tidak melakukannya.

Definisi sampel:

```
{
  "type" : "string",
  "default" : "defaultString",
  "nullable" : true,
  "maxLength": 10,
  "minLength": 1,
}
```

```
"pattern" : "^[0-9a-fA-F]{2}+$"  
}
```

Nulls

Tipe nol hanya menerima satu nilai: `null`.

Definisi sampel:

```
{ "type": "null" }
```

Array

Jenis array mendukung hal berikut:

- `default`— daftar yang akan digunakan sebagai nilai default.
- `items`— Definisi tipe JSON dikenakan pada semua elemen array.
- Kendala panjang (harus berupa angka non-negatif)
 - `minItems`
 - `maxItems`
- `pattern` nilai untuk Regex
- `uniqueItems`— boolean yang menunjukkan jika elemen dalam array harus unik
- `prefixItems`— sebuah array di mana setiap item adalah skema yang sesuai dengan setiap indeks dari array dokumen. Artinya, array di mana elemen pertama memvalidasi elemen pertama dari array input, elemen kedua memvalidasi elemen kedua dari array input, dan seterusnya.

Definisi sampel:

```
{  
  "type": "array",  
  "default": ["1", "2"],  
  "items" : {  
    "type": "string",  
    "pattern": "^[a-zA-Z0-9_ -/]+$"  
  },  
  "minItems" : 1,  
  "maxItems": 4,  
  "uniqueItems" : true,  
}
```

```
}
```

Contoh validasi array:

```
//Examples:  
["1", "2", "3", "4"] // Schema-Compliant  
[] // NOT Schema-Compliant: minItems=1  
["1", "1"] // NOT Schema-Compliant: uniqueItems=true  
["{}"] // NOT Schema-Compliant: Does not match the RegEx pattern.
```

Definisi alternatif menggunakan validasi Tuple:

```
{  
  "type": "array",  
  "prefixItems": [  
    { "type": "number" },  
    { "type": "string" },  
    { "enum": ["Street", "Avenue", "Boulevard"] },  
    { "enum": ["NW", "NE", "SW", "SE"] }  
  ]  
}  
  
//Examples:  
[1600, "Pennsylvania", "Avenue", "NW"] // Schema-Compliant  
  
// And, by default, it's also okay to add additional items to end:  
[1600, "Pennsylvania", "Avenue", "NW", "Washington"] // Schema-Compliant
```

Objek

Jenis objek mendukung hal berikut:

- **Kendala properti**
 - **properties**— Tentukan properti (pasangan kunci-nilai) dari suatu objek dengan menggunakan kata kunci. `properties` Nilai `properties` adalah objek, di mana setiap kunci adalah nama properti dan setiap nilai adalah skema yang digunakan untuk memvalidasi properti itu. Properti apa pun yang tidak cocok dengan nama properti dalam `properties` kata kunci diabaikan oleh kata kunci ini.
 - **required**— Secara default, properti yang ditentukan oleh `properties` kata kunci tidak diperlukan. Namun, Anda dapat memberikan daftar properti yang diperlukan menggunakan

`required` kata kunci. `required` Kata kunci mengambil array nol atau lebih string. Masing-masing string ini harus unik.

- `propertyNames`— Kata kunci ini memungkinkan kontrol atas RegEx pola untuk nama properti. Misalnya, Anda mungkin ingin menegaskan bahwa semua properti objek memiliki nama mengikuti konvensi tertentu.
- `patternProperties`— Ini memetakan ekspresi reguler ke skema. Jika nama properti cocok dengan ekspresi reguler yang diberikan, nilai properti harus memvalidasi terhadap skema yang sesuai. Misalnya, gunakan `patternProperties` untuk menentukan bahwa nilai harus cocok dengan skema tertentu, mengingat jenis nama properti tertentu.
- `additionalProperties` Kata kunci ini mengontrol bagaimana properti tambahan ditangani. Properti tambahan adalah properti yang namanya tidak tercantum dalam kata kunci properti atau yang cocok dengan ekspresi reguler mana pun `patternProperties`. Secara default, properti tambahan diperbolehkan. Menyetel bidang ini `false` berarti tidak ada properti tambahan yang diizinkan.
- `unevaluatedProperties`— Kata kunci ini mirip dengan `additionalProperties` kecuali bahwa ia dapat mengenali properti yang dideklarasikan dalam subskema. `unevaluatedProperties` bekerja dengan mengumpulkan properti apa pun yang berhasil divalidasi saat memproses skema dan menggunakannya sebagai daftar properti yang diizinkan. Ini memungkinkan Anda melakukan hal-hal yang lebih kompleks seperti menambahkan properti secara kondisional. Lihat contoh di bawah ini untuk lebih jelasnya.
- `anyOf`— Nilai kata kunci ini HARUS berupa array yang tidak kosong. Setiap item dari array HARUS menjadi Skema JSON yang valid. Sebuah instance berhasil memvalidasi terhadap kata kunci ini jika berhasil memvalidasi terhadap setidaknya satu skema yang ditentukan oleh nilai kata kunci ini.
- `oneOf`— Nilai kata kunci ini HARUS berupa array yang tidak kosong. Setiap item dari array HARUS menjadi Skema JSON yang valid. Sebuah instance berhasil memvalidasi terhadap kata kunci ini jika berhasil memvalidasi dengan tepat satu skema yang ditentukan oleh nilai kata kunci ini.

Contoh yang diperlukan:

```
{
  "type": "object",
  "required": ["test"]
}
```

```
// Schema Compliant
{
  "test": 4
}

// NOT Schema Compliant
{}
```

PropertyNames contoh:

```
{
  "type": "object",
  "propertyNames": {
    "pattern": "^[A-Za-z_][A-Za-z0-9_]*$"
  }
}

// Schema Compliant
{
  "_a_valid_property_name_001": "value"
}

// NOT Schema Compliant
{
  "001 invalid": "value"
}
```

PatternProperties contoh:

```
{
  "type": "object",
  "patternProperties": {
    "^S_": { "type": "string" },
    "^I_": { "type": "integer" }
  }
}

// Schema Compliant
{ "S_25": "This is a string" }
{ "I_0": 42 }

// NOT Schema Compliant
{ "S_0": 42 } // Value must be a string
```

```
{ "I_42": "This is a string" } // Value must be an integer
```

AdditionalProperties contoh:

```
{
  "type": "object",
  "properties": {
    "test": {
      "type": "string"
    }
  },
  "additionalProperties": false
}

// Schema Compliant
{
  "test": "value"
}
OR
{}

// NOT Schema Compliant
{
  "notAllowed": false
}
```

UnevaluatedProperties contoh:

```
{
  "type": "object",
  "properties": {
    "standard_field": { "type": "string" }
  },
  "patternProperties": {
    "^@": { "type": "integer" } // Allows properties starting with '@'
  },
  "unevaluatedProperties": false // No other properties allowed
}

// Schema Compliant
{
  "standard_field": "some value",
  "@id": 123,
```

```
"@timestamp": 1678886400
}
// This passes because "standard_field" is evaluated by properties,
// "@id" and "@timestamp" are evaluated by patternProperties,
// and no other properties remain unevaluated.

// NOT Schema Compliant
{
  "standard_field": "some value",
  "another_field": "unallowed"
}
// This fails because "another_field" is unevaluated and doesn't match
// the @ pattern, leading to a violation of unevaluatedProperties: false
```

AnyOf contoh:

```
{
  "anyOf": [
    { "type": "string", "maxLength": 5 },
    { "type": "number", "minimum": 0 }
  ]
}

// Schema Compliant
"short"
12

// NOT Schema Compliant
"too long"
-5
```

OneOf contoh:

```
{
  "oneOf": [
    { "type": "number", "multipleOf": 5 },
    { "type": "number", "multipleOf": 3 }
  ]
}

// Schema Compliant
10
9
```

```
// NOT Schema compliant
2 // Not a multiple of either 5 or 3
15 // Multiple of both 5 and 3 is rejected.
```

Definisi tipe namespaced

Definisi tipe namespaced adalah tipe yang dibangun dari tipe primitif. Tipe ini harus mengikuti format Integrasi *namespace.typename*. terkelola menyediakan tipe yang telah ditentukan di bawah `aws` dan `matter` ruang nama. Anda dapat menggunakan namespace apa pun untuk jenis kustom kecuali ruang nama yang dicadangkan `aws` dan `matter`.

Untuk menemukan definisi tipe namespace yang tersedia, gunakan [ListSchemaVersions](#) API dengan Type filter disetel ke `definition`

Jenis `matter`

Temukan tipe data di bawah `matter` namespace menggunakan [ListSchemaVersions](#) API dengan Namespace filter disetel ke `matter` dan filter disetel ke `Type.definition`

Jenis `aws`

Temukan tipe data di bawah `aws` namespace menggunakan [ListSchemaVersions](#) API dengan Namespace filter disetel ke `aws` dan filter disetel ke `Type.definition`

Definisi jenis bitmap

Bitmap memiliki dua properti yang diperlukan:

- `type` harus menjadi objek
- `properties` harus berupa objek yang berisi setiap definisi bit. Setiap bit adalah objek dengan properti `extrinsicId` dan `value`. Nilai setiap bit harus berupa bilangan bulat dengan nilai minimum 0 dan nilai maksimum minimal 1.

Contoh definisi bitmap:

```
{
  "title" : "Sample Bitmap Type",
  "description" : "Type definition for SampleBitmap.",
```

```
    "$ref" : "/schema-versions/definition/aws.bitmap@1.0 ",
    "type" : "object",
    "additionalProperties" : false,
    "properties" : {
      "Bit1" : {
        "extrinsicId" : "0x0000",
        "value" : {
          "type" : "integer",
          "maximum" : 1,
          "minimum" : 0
        }
      },
      "Bit2" : {
        "extrinsicId" : "0x0001",
        "value" : {
          "type" : "integer",
          "maximum" : 1,
          "minimum" : 0
        }
      }
    }
  }
}

// Schema Compliant
{
  "Bit1": 1,
  "Bit1": 0
}

// NOT Schema Compliant
{
  "Bit1": -1,
  "Bit1": 0
}
```

Definisi tipe enum

Enum membutuhkan tiga properti:

- type harus menjadi objek
- enum harus berupa array string unik, dengan minimal satu item
- extrinsicIdMap adalah objek dengan properti yang merupakan nilai enum. Nilai masing-masing properti harus menjadi pengidentifikasi ekstrinsik yang sesuai dengan nilai enum.

Contoh definisi enum:

```
{
  "title" : "SampleEnum Type",
  "description" : "Type definition for SampleEnum.",
  "$ref" : "/schema-versions/definition/aws.enum@1.0",
  "type" : "string",
  "enum" : [
    "EnumValue0",
    "EnumValue1",
    "EnumValue2"
  ],
  "extrinsicIdMap" : {
    "EnumValue0" : "0",
    "EnumValue1" : "1",
    "EnumValue2" : "2"
  }
}

// Schema Compliant
"EnumValue0"
"EnumValue1"
"EnumValue2"

// NOT Schema Compliant
"NotAnEnumValue"
```

Kelola perintah dan acara perangkat IoT

Perintah perangkat memberikan kemampuan untuk mengelola perangkat fisik dari jarak jauh memastikan kontrol penuh atas perangkat selain melakukan pembaruan keamanan, perangkat lunak, dan perangkat keras yang kritis. Dengan armada perangkat yang besar, mengetahui kapan perangkat melakukan perintah memberikan pengawasan atas seluruh implementasi perangkat Anda. Perintah perangkat atau pembaruan otomatis akan memicu perubahan status perangkat, yang pada gilirannya akan membuat acara perangkat baru. Peristiwa perangkat ini akan memicu notifikasi yang dikirim secara otomatis ke tujuan yang dikelola pelanggan.

Topik

- [Perintah perangkat](#)
- [Acara Perangkat](#)

Perintah perangkat

Permintaan perintah adalah perintah yang dikirim ke perangkat. Permintaan perintah mencakup muatan yang menentukan tindakan yang akan diambil seperti menyalakan bola lampu. Untuk mengirim perintah perangkat, `SendManagedThingCommand` API dipanggil atas nama pengguna akhir dengan integrasi terkelola dan permintaan perintah dikirim ke perangkat.

Respons terhadap a `SendManagedThingCommand` adalah `traceId` dan Anda dapat menggunakan ini `traceId` untuk melacak pengiriman perintah dan alur kerja respons perintah terkait sedapat mungkin.

Untuk informasi selengkapnya tentang operasi `SendManagedThingCommand` API, lihat [SendManagedThingCommand](#).

UpdateStateaksi

Untuk memperbarui status perangkat seperti waktu lampu menyala, gunakan `UpdateState` tindakan saat memanggil `SendManagedThingCommand` API. Berikan properti model data dan nilai baru yang Anda perbaruiparameters. Contoh di bawah ini menggambarkan permintaan `SendManagedThingCommand` API yang `OnTime` memperbarui bola lampu ke5.

```
{
  "Endpoints": [
    {
```

```
"endpointId": "1",
"capabilities": [
  {
    "id": "matter.OnOff",
    "name": "On/Off",
    "version": "1",
    "actions": [
      {
        "name": "UpdateState",
        "parameters": {
          "OnTime": 5
        }
      }
    ]
  }
]
```

ReadStateaksi

Untuk mendapatkan status terbaru perangkat termasuk nilai saat ini dari semua properti model data, gunakan ReadState tindakan saat memanggil SendManagedThingCommand API. Di `propertiesToRead`, Anda dapat menggunakan opsi berikut:

- Berikan properti model data tertentu untuk mendapatkan nilai terbaru seperti `OnOff` menentukan apakah lampu menyala atau mati.
- Gunakan operator wildcard (*) untuk membaca semua properti status perangkat untuk kemampuan.

Contoh di bawah ini menggambarkan kedua skenario untuk permintaan `SendManagedThingCommand` API menggunakan ReadState tindakan:

```
{
  "Endpoints": [
    {
      "endpointId": "1",
      "capabilities": [
        {
          "id": "aws.OnOff",
```

```
    "name": "On/Off",
    "version": "1",
    "actions": [
      {
        "name": "ReadState",
        "parameters": {
          "propertiesToRead": [ "OnOff" ]
        }
      }
    ]
  }
]
```

```
{
  "Endpoints": [
    {
      "endpointId": "1",
      "capabilities": [
        {
          "id": "aws.OnOff",
          "name": "On/Off",
          "version": "1",
          "actions": [
            {
              "name": "ReadState",
              "parameters": {
                "propertiesToRead": [ "*" ]
              }
            }
          ]
        }
      ]
    }
  ]
}
```

Acara Perangkat

Peristiwa perangkat mencakup status perangkat saat ini. Ini bisa berarti perangkat telah berubah status, atau melaporkan statusnya bahkan jika statusnya tidak berubah. Ini termasuk laporan properti

dan peristiwa yang didefinisikan dalam model data. Suatu peristiwa bisa berupa siklus mesin cuci telah selesai atau termostat telah mencapai suhu yang ditargetkan yang ditetapkan oleh pengguna akhir.

Pemberitahuan acara perangkat

Pengguna akhir dapat berlangganan tujuan tertentu yang dikelola pelanggan yang mereka buat untuk pembaruan pada peristiwa perangkat tertentu. Untuk membuat tujuan yang dikelola pelanggan, panggil API. `CreateDestination` Saat peristiwa perangkat dilaporkan ke integrasi terkelola oleh perangkat, tujuan yang dikelola pelanggan akan diberi tahu jika ada.

Menandai sumber daya integrasi terkelola

Untuk membantu Anda mengelola dan mengatur sumber daya Anda, Anda dapat secara opsional menetapkan metadata Anda sendiri ke masing-masing sumber daya ini dalam bentuk tag. Bagian ini menjelaskan tag dan menunjukkan cara membuatnya.

Dasar-dasar tag

Anda dapat menggunakan tag untuk mengkategorikan sumber daya integrasi terkelola dengan berbagai cara (misalnya, berdasarkan tujuan, pemilik, atau lingkungan). Ini berguna ketika Anda memiliki banyak sumber daya dari jenis yang sama - Anda dapat dengan cepat mengidentifikasi sumber daya berdasarkan tag yang telah Anda tetapkan padanya. Setiap tanda terdiri dari kunci dan nilai opsional, yang keduanya Anda tentukan. Misalnya, Anda dapat menentukan satu set tag untuk jenis barang Anda yang membantu Anda melacak perangkat berdasarkan jenis. Kami menyarankan agar Anda merancang serangkaian kunci tanda yang memenuhi kebutuhan Anda untuk setiap jenis sumber daya. Penggunaan serangkaian kunci tanda akan mempermudah Anda dalam mengelola sumber daya Anda.

Anda dapat mencari dan memfilter sumber daya berdasarkan tag yang Anda tambahkan atau terapkan. Anda juga dapat menggunakan tag untuk mengontrol akses ke sumber daya Anda seperti yang dijelaskan dalam [Menggunakan tanda dengan kebijakan IAM](#).

Untuk kemudahan penggunaan, Editor Tag di AWS Management Console menyediakan cara terpusat dan terpadu untuk membuat dan mengelola tag Anda. Untuk informasi selengkapnya, lihat [Bekerja dengan Editor Tag](#) di [Bekerja dengan Konsol AWS Manajemen](#).

Anda juga dapat bekerja dengan tag menggunakan AWS CLI dan API integrasi terkelola. Anda dapat mengaitkan tag dengan hal-hal terkelola, profil penyedia, loker kredensial, dan tugas over-the-air (OTA) saat Anda membuatnya dengan menggunakan Tags bidang dalam perintah berikut:

- [CreateManagedThing](#)
- [CreateProvisioningProfile](#)
- [CreateCredentialLocker](#)
- [CreateOtaTask](#)
- [CreateAccountAssociation](#)

Anda juga dapat menambahkan, mengubah, atau menghapus tanda untuk sumber daya yang sudah ada yang mendukung penandaan dengan menggunakan perintah berikut:

- [TagResource](#)
- [ListTagsForResource](#)
- [UntagResource](#)

Anda dapat mengedit kunci dan nilai tanda, dan dapat menghapus tanda dari sumber daya kapan saja. Anda dapat mengatur nilai tanda ke string kosong, tetapi tidak dapat mengatur nilai tanda ke null. Jika Anda menambahkan tanda yang memiliki kunci yang sama dengan tanda yang ada pada sumber daya tersebut, nilai yang baru akan menimpa nilai yang lama. Jika Anda menghapus sebuah sumber daya, semua tanda yang terkait dengan sumber daya tersebut juga dihapus.

Pembatasan dan batasan tanda

Batasan dasar berikut berlaku untuk tag:

- Jumlah maksimum tag per sumber daya - 50
- Panjang kunci maksimum - 127 karakter Unicode di UTF-8
- Panjang nilai maksimum - 255 karakter Unicode di UTF-8
- Kunci dan nilai tanda peka huruf besar-kecil.
- Jangan gunakan `aws` : awalan dalam nama atau nilai tag Anda. Ini dicadangkan untuk AWS digunakan. Anda tidak dapat mengedit atau menghapus nama atau nilai tanda dengan awalan ini. Tag dengan awalan ini tidak dihitung terhadap tag Anda per batas sumber daya.
- Jika skema penandaan Anda digunakan di beberapa layanan dan sumber daya, harap perhatikan bahwa layanan lain mungkin memiliki pembatasan pada karakter yang diizinkan. Karakter yang diizinkan termasuk huruf, spasi, dan angka yang dapat direpresentasikan dalam UTF-8, dan karakter khusus berikut: `+ - = . _ : / @`.

Menggunakan tanda dengan kebijakan IAM

Anda dapat menerapkan izin tingkat sumber daya berbasis tag dalam kebijakan IAM yang Anda gunakan untuk tindakan API integrasi terkelola. Hal ini memberi Anda kontrol yang lebih baik atas sumber daya yang dapat dibuat, dimodifikasi, atau digunakan oleh pengguna. Anda menggunakan elemen `Condition` (juga disebut blok `Condition`) dengan kunci konteks syarat berikut dan nilai-nilai dalam kebijakan IAM untuk mengontrol akses pengguna (izin) berdasarkan tanda sumber daya:

- Gunakan `aws:ResourceTag/tag-key: tag-value` untuk mengizinkan atau menolak tindakan pengguna pada sumber daya dengan tag tertentu.
- Gunakan `aws:RequestTag/tag-key: tag-value` untuk mengharuskan tag tertentu digunakan (atau tidak digunakan) saat membuat permintaan API untuk membuat atau memodifikasi sumber daya yang memungkinkan tag.
- Gunakan `aws:TagKeys: [tag-key, ...]` untuk mengharuskan sekumpulan kunci tag tertentu digunakan (atau tidak digunakan) saat membuat permintaan API untuk membuat atau memodifikasi sumber daya yang memungkinkan tag.

Note

Kunci konteks kondisi dan nilai dalam kebijakan IAM hanya berlaku untuk tindakan integrasi terkelola tersebut di mana pengidentifikasi untuk sumber daya yang mampu diberi tag adalah parameter wajib. Misalnya, penggunaan [GetCustomEndpoint](#) tidak diizinkan atau ditolak berdasarkan kunci dan nilai konteks kondisi karena tidak ada sumber daya yang dapat diberi tag (hal-hal yang dikelola, profil penyedia, loker kredensi, over-the-air tugas) yang direferensikan dalam permintaan ini. Untuk informasi selengkapnya tentang sumber daya integrasi terkelola yang dapat diberi tag dan kunci kondisi yang didukungnya, baca [Tindakan, sumber daya, dan kunci kondisi untuk fitur integrasi AWS IoT terkelola](#). AWS IoT Device Management

Untuk informasi selengkapnya tentang penggunaan tag, lihat [Mengontrol Akses Menggunakan Tag](#) di Panduan AWS Identity and Access Management Pengguna. Bagian [Referensi Kebijakan IAM JSON](#) dari panduan itu memiliki sintaks terperinci, deskripsi, dan contoh elemen, variabel, dan logika evaluasi kebijakan JSON di IAM.

Kebijakan contoh berikut menerapkan dua batasan berbasis tag untuk `CreateManagedThing` tindakan tersebut. Pengguna IAM dibatasi oleh kebijakan ini:

- Tidak dapat membuat hal yang dikelola dengan tag “env=prod” (dalam contoh, lihat baris).
`"aws:RequestTag/env" : "prod"`
- Tidak dapat memodifikasi atau mengakses hal terkelola yang memiliki tag “env=prod” yang ada (dalam contoh, lihat baris). `"aws:ResourceTag/env" : "prod"`

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": "iotmanagedintegrations:CreateManagedThing",
      "Resource": "arn:aws:iotmanagedintegrations:us-east-1:123456789012:managed-thing/*",
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/env": "prod"
        }
      }
    },
    {
      "Effect": "Deny",
      "Action": [
        "iotmanagedintegrations:CreateManagedThing",
        "iotmanagedintegrations>DeleteManagedThing",
        "iotmanagedintegrations:GetManagedThing",
        "iotmanagedintegrations:UpdateManagedThing"
      ],
      "Resource": "arn:aws:iotmanagedintegrations:us-east-1:123456789012:managed-thing/*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/env": "prod"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "iotmanagedintegrations:CreateManagedThing",
        "iotmanagedintegrations>DeleteManagedThing",
        "iotmanagedintegrations:GetManagedThing",
        "iotmanagedintegrations:UpdateManagedThing"
      ],
      "Resource": "*"
    }
  ]
}
```

```
}
```

Anda juga dapat menentukan beberapa nilai tag untuk kunci tag yang diberikan dengan melampirkannya dalam daftar, seperti ini:

```
"StringEquals" : {  
  "aws:ResourceTag/env" : ["dev", "test"]  
}
```

Note

Jika Anda mengizinkan atau menolak akses para pengguna ke sumber daya berdasarkan tanda, maka Anda harus mempertimbangkan untuk menolak secara eksplisit memberikan kemampuan kepada pengguna untuk menambahkan atau menghapus tanda tersebut dari sumber daya yang sama. Jika tidak, pengguna dapat mengakali pembatasan Anda dan mendapatkan akses atas sumber daya dengan melakukan modifikasi pada tanda dari sumber daya tersebut.

Pemberitahuan integrasi terkelola

Pemberitahuan integrasi terkelola memberikan pembaruan dan wawasan utama dari perangkat. Pemberitahuan mencakup peristiwa konektor, perintah perangkat, peristiwa siklus hidup, pembaruan OTA (Over-the-Air), dan laporan kesalahan. Wawasan ini memberikan informasi yang dapat ditindaklanjuti untuk membuat alur kerja otomatis, mengambil tindakan segera, atau menyimpan data peristiwa untuk pemecahan masalah.

Saat ini, hanya aliran data Amazon Kinesis yang didukung sebagai tujuan notifikasi integrasi terkelola. Pertama-tama Anda harus menyiapkan aliran data Amazon Kinesis dan mengizinkan akses integrasi terkelola ke aliran data sebelum menyiapkan notifikasi.

Siapkan Amazon Kinesis untuk notifikasi

Langkah-langkah persiapan Amazon Kinesis

- [Langkah 1: Buat aliran data Amazon Kinesis](#)
- [Langkah 2: Buat kebijakan izin](#)
- [Langkah 3: Arahkan ke dasbor IAM dan pilih Peran](#)
- [Langkah 4: Gunakan kebijakan kepercayaan khusus](#)
- [Langkah 5: Terapkan kebijakan izin Anda](#)
- [Langkah 6: Masukkan nama peran](#)

Untuk mengatur Amazon Kinesis untuk notifikasi integrasi terkelola, ikuti langkah-langkah berikut:

Langkah 1: Buat aliran data Amazon Kinesis

Amazon Kinesis Data Stream dapat menelan sejumlah besar data secara real time, menyimpan data dengan tahan lama, dan membuat data tersedia untuk dikonsumsi oleh aplikasi.

Untuk membuat aliran data Amazon Kinesis

- Untuk membuat aliran data Kinesis, ikuti langkah-langkah yang diuraikan dalam [Membuat dan mengelola aliran data Kinesis](#).

Langkah 2: Buat kebijakan izin

Buat kebijakan izin yang memungkinkan integrasi terkelola mengakses aliran data Kinesis Anda.

Untuk membuat kebijakan izin

- Untuk membuat kebijakan izin, salin kebijakan di bawah ini dan ikuti langkah-langkah yang diuraikan dalam [Membuat kebijakan menggunakan editor JSON](#)

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": "kinesis:PutRecord",
      "Resource": "*",
      "Effect": "Allow"
    }
  ]
}
```

Langkah 3: Arahkan ke dasbor IAM dan pilih Peran

Buka dasbor IAM dan klik Peran.

Untuk menavigasi ke dasbor IAM

- Buka dasbor IAM dan klik Peran.

Untuk informasi selengkapnya, lihat [Pembuatan peran IAM](#) di Panduan AWS Identity and Access Management Pengguna.

Langkah 4: Gunakan kebijakan kepercayaan khusus

Anda dapat menggunakan kebijakan kepercayaan khusus untuk memberikan akses integrasi terkelola ke aliran data Kinesis.

Untuk menggunakan kebijakan kepercayaan khusus

- Buat peran baru dan pilih Kebijakan kepercayaan khusus. Klik Berikutnya.

Kebijakan berikut memungkinkan integrasi terkelola untuk mengambil peran, dan Condition pernyataan tersebut membantu mencegah masalah wakil yang membingungkan.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "iotmanagedintegrations.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "123456789012"
        },
        "ArnLike": {
          "aws:SourceArn": "arn:aws:iotmanagedintegrations:ca-central-1:123456789012:*"
        }
      }
    }
  ]
}
```

Langkah 5: Terapkan kebijakan izin Anda

Tambahkan kebijakan izin yang Anda buat di langkah 2 ke peran.

Untuk menambahkan kebijakan izin

- Pada halaman Tambahkan izin, cari dan tambahkan kebijakan izin yang Anda buat di langkah 2. Klik Berikutnya.

Langkah 6: Masukkan nama peran

- Masukkan nama peran dan klik Buat peran.

Siapkan notifikasi integrasi terkelola

Langkah-langkah persiapan pemberitahuan

- [Langkah 1: Berikan izin pengguna untuk memanggil API CreateDestination](#)
- [Langkah 2: Panggil CreateDestination API](#)
- [Langkah 3: Panggil CreateNotificationConfiguration API](#)

Untuk mengatur notifikasi integrasi terkelola, ikuti langkah-langkah berikut:

Langkah 1: Berikan izin pengguna untuk memanggil API CreateDestination

- Berikan izin pengguna untuk memanggil API **CreateDestination**

Kebijakan berikut menentukan persyaratan bagi pengguna untuk memanggil [CreateDestination](#) API.

Lihat [Memberikan izin pengguna untuk meneruskan peran ke AWS layanan](#) di Panduan AWS Identity and Access Management Pengguna untuk mendapatkan izin peran sandi ke integrasi terkelola.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iam:PassRole",
      "Resource": "arn:aws:iam::123456789012:role/ROLE_CREATED_IN_PREVIOUS_STEP",
      "Condition": {
        "StringEquals": {
          "iam:PassedToService": "iotmanagedintegrations.amazonaws.com"
        }
      }
    }
  ]
}
```

```
    }
  },
  {
    "Effect": "Allow",
    "Action": "iotmanagedintegrations:CreateDestination",
    "Resource": "*"
  }
]
```

Langkah 2: Panggil CreateDestination API

- Panggil **CreateDestination** API

Setelah Anda membuat aliran data Amazon Kinesis dan peran akses streaming, panggil [CreateDestination](#) API untuk membuat tujuan notifikasi tempat notifikasi akan diarahkan. Untuk `DeliveryDestinationArn` parameternya, gunakan arn aliran data Amazon Kinesis baru Anda.

```
{
  "DeliveryDestinationArn": "Your Kinesis arn"
  "DeliveryDestinationType": "KINESIS"
  "Name": "DestinationName"
  "ClientToken": "string"
  "RoleArn": "arn:aws:iam::accountID:role/ROLE_CREATED_IN_PREVIOUS_STEP"
}
```

Note

`ClientToken` adalah token idempotensi. Jika Anda mencoba lagi permintaan yang berhasil diselesaikan pada awalnya menggunakan token dan parameter klien yang sama, maka upaya coba lagi akan berhasil tanpa melakukan tindakan lebih lanjut.

Langkah 3: Panggil CreateNotificationConfiguration API

- Panggil **CreateNotificationConfiguration** API

Terakhir, gunakan [CreateNotificationConfiguration](#) API untuk membuat konfigurasi notifikasi yang merutekan jenis peristiwa yang dipilih ke tujuan yang diwakili oleh aliran data Kinesis. Dalam `DestinationName` parameter, gunakan nama tujuan yang sama seperti ketika Anda awalnya memanggil `CreateDestination` API.

```
{
  "EventType": "DEVICE_EVENT"
  "DestinationName" // This name has to be identical to the name in
  createDestination API
  "ClientToken": "string"
}
```

Jenis acara dipantau dengan integrasi terkelola

Berikut ini adalah jenis acara yang dipantau dengan notifikasi integrasi terkelola:

- `DEVICE_COMMAND`
- Status perintah [SendManagedThingCommand](#) API. Nilai yang valid adalah `succeeded` maupun `failed`.

```
{
  "version": "0",
  "messageId": "6a7e8feb-b491-4cf7-a9f1-bf3703467718",
  "messageType": "DEVICE_COMMAND",
  "source": "aws.iotmanagedintegrations",
  "customerAccountId": "123456789012",
  "timestamp": "2017-12-22T18:43:48Z",
  "region": "ca-central-1",
  "resources": [
    "arn:aws:iotmanagedintegrations:ca-
central-1:123456789012:managed-thing/6a7e8feb-b491-4cf7-a9f1-bf3703467718"
  ],
  "payload": {
    "traceId": "1234567890abcdef0",
    "receivedAt": "2017-12-22T18:43:48Z",
    "executedAt": "2017-12-22T18:43:48Z",
    "result": "failed"
  }
}
```

- **DEVICE_COMMAND_REQUEST**

- Permintaan perintah dari Web Real-Time Communication (WebRTC).

Standar WebRTC memungkinkan komunikasi antara dua rekan. Rekan-rekan ini dapat mengirimkan video real-time, audio, dan data arbitrer. Integrasi terkelola mendukung WebRTC untuk mengaktifkan jenis streaming antara aplikasi seluler pelanggan dan perangkat pengguna akhir. [Untuk informasi selengkapnya tentang standar WebRTC, lihat WebRTC.](#)

```
{
    "version": "0",
    "messageId": "6a7e8feb-b491-4cf7-a9f1-bf3703467718",
    "messageType": "DEVICE_COMMAND_REQUEST",
    "source": "aws.iotmanagedintegrations",
    "customerAccountId": "123456789012",
    "timestamp": "2017-12-22T18:43:48Z",
    "region": "ca-central-1",
    "resources": [
        "arn:aws:iotmanagedintegrations:ca-central-1:123456789012:managed-thing/6a7e8feb-b491-4cf7-a9f1-bf3703467718"
    ],
    "payload": {
        "endpoints": [
            {
                "endpointId": "1",
                "capabilities": [
                    {
                        "id": "aws.DoorLock",
                        "name": "Door Lock",
                        "version": "1.0"
                    }
                ]
            }
        ]
    }
}
```

- **DEVICE_DISCOVERY_STATUS**

- Status penemuan perangkat.

```
{
    "version": "0",
    "messageId": "6a7e8feb-b491-4cf7-a9f1-bf3703467718",
    "messageType": "DEVICE_DISCOVERY_STATUS",
    "source": "aws.iotmanagedintegrations",
    "customerAccountId": "123456789012",
    "timestamp": "2017-12-22T18:43:48Z",
```

```

    "region": "ca-central-1",
    "resources": [
      "arn:aws:iotmanagedintegrations:ca-central-1:123456789012:managed-thing/6a7e8feb-b491-4cf7-a9f1-bf3703467718"
    ],
    "payload": {
      "deviceCount": 1,
      "deviceDiscoveryId": "123",
      "status": "SUCCEEDED"
    }
  }
}

```

- **DEVICE_EVENT**

- Pemberitahuan peristiwa perangkat yang terjadi.

```

{
  "version": "1.0",
  "messageId": "2ed545027bd347a2b855d28f94559940",
  "messageType": "DEVICE_EVENT",
  "source": "aws.iotmanagedintegrations",
  "customerAccountId": "123456789012",
  "timestamp": "1731630247280",
  "resources": [
    "/quit/1b15b39992f9460ba82c6c04595d1f4f"
  ],
  "payload": {
    "endpoints": [
      {
        "endpointId": "1",
        "capabilities": [
          {
            "id": "aws.DoorLock",
            "name": "Door Lock",
            "version": "1.0",
            "properties": [
              {
                "name": "ActuatorEnabled",
                "value": "true"
              }
            ]
          }
        ]
      }
    ]
  }
}

```

- **DEVICE_LIFE_CYCLE**

- Status siklus hidup perangkat.

```
{
  "version": "1.0.0",
  "messageId": "8d1e311a473f44f89d821531a0907b05",
  "messageType": "DEVICE_LIFE_CYCLE",
  "source": "aws.iotmanagedintegrations",
  "customerAccountId": "123456789012",
  "timestamp": "2024-11-14T19:55:57.568284645Z",
  "region": "ca-central-1",
  "resources": [
    "arn:aws:iotmanagedintegrations:ca-central-1:123456789012:managed-thing/
d5c280b423a042f3933eed09cf408657"
  ],
  "payload": {
    "deviceDetails": {
      "id": "d5c280b423a042f3933eed09cf408657",
      "arn": "arn:aws:iotmanagedintegrations:ca-central-1:123456789012:managed-
thing/d5c280b423a042f3933eed09cf408657",
      "createdAt": "2024-11-14T19:55:57.515841147Z",
      "updatedAt": "2024-11-14T19:55:57.515841559Z"
    },
    "status": "UNCLAIMED"
  }
}
```

- DEVICE_OTA
 - Pemberitahuan OTA perangkat.
- DEVICE_STATE
 - Pemberitahuan saat status perangkat telah diperbarui.

```
{
  "messageType": "DEVICE_STATE",
  "source": "aws.iotmanagedintegrations",
  "customerAccountId": "123456789012",
  "timestamp": "1731623291671",
  "resources": [
    "arn:aws:iotmanagedintegrations:ca-central-1:123456789012:managed-
thing/61889008880012345678"
  ],
  "payload": {
    "addedStates": {
      "endpoints": [{
```

```
    "endpointId": "nonEndpointId",
    "capabilities": [{
      "id": "aws.OnOff",
      "name": "On/Off",
      "version": "1.0",
      "properties": [{
        "name": "OnOff",
        "value": {
          "propertyValue": "\"onoff\"",
          "lastChangedAt": "2024-06-11T01:38:09.000414Z"
        }
      }
    ]
  ]
}
```

Cloud-to-Cloud Konektor (C2C)

cloud-to-cloudKonektor memungkinkan Anda membuat dan memfasilitasi komunikasi dua arah antara perangkat pihak ketiga dan. AWS

Topik

- [Apa itu konektor cloud-to-cloud \(C2C\)?](#)
- [Apa katalog konektor C2C?](#)
- [AWS Lambda berfungsi sebagai konektor C2C](#)
- [Alur kerja konektor integrasi terkelola](#)
- [Pedoman untuk menggunakan konektor C2C \(\) cloud-to-cloud](#)
- [Bangun konektor C2C \(Cloud-to-Cloud\)](#)
- [Gunakan konektor C2C \(Cloud-to-Cloud\)](#)

Apa itu konektor cloud-to-cloud (C2C)?

cloud-to-cloudKonektor adalah paket perangkat lunak pra-bangun yang secara aman menautkan AWS Cloud ke titik akhir penyedia cloud pihak ketiga. Menggunakan konektor C2C, penyedia solusi dapat memanfaatkan integrasi terkelola untuk AWS IoT Device Management untuk mengontrol perangkat yang terhubung ke cloud pihak ketiga.

Integrasi terkelola mencakup katalog konektor tempat AWS pelanggan dapat melihat dan memilih konektor yang ingin mereka integrasikan. Untuk informasi selengkapnya, lihat [Apa katalog konektor C2C?](#)

Integrasi terkelola mengharuskan setiap konektor diimplementasikan sebagai AWS Lambda fungsi.

Apa katalog konektor C2C?

Integrasi terkelola untuk katalog konektor AWS IoT Device Management adalah kumpulan konektor C2C yang memfasilitasi komunikasi dua arah antara integrasi terkelola untuk AWS IoT Device Management dan penyedia cloud pihak ketiga. Anda dapat melihat konektor di Konsol Manajemen AWS atau AWS CLI.

Untuk menggunakan konsol untuk melihat katalog konektor integrasi terkelola

1. Buka konsol [integrasi terkelola](#)
2. Di panel navigasi kiri, pilih Integrasi terkelola
3. Di panel navigasi kiri konsol integrasi terkelola, pilih Katalog.

AWS Lambda berfungsi sebagai konektor C2C

Setiap konektor C2C fungsi Lambda menerjemahkan dan mengangkut perintah dan peristiwa antara integrasi terkelola dan tindakan terkait pada platform pihak ketiga. Untuk informasi selengkapnya tentang Lambda, lihat [Apa itu](#). AWS Lambda

Misalnya, pengguna akhir memiliki bola lampu pintar yang diproduksi oleh OEM pihak ketiga. Dengan konektor C2C, pengguna akhir dapat mengeluarkan perintah untuk menyalakan atau mematikan lampu ini melalui platform integrasi terkelola. Perintah ini kemudian akan diteruskan ke fungsi Lambda yang dihosting di konektor, yang akan menerjemahkan permintaan menjadi panggilan API terhadap platform pihak ketiga untuk menghidupkan atau menonaktifkan perangkat.

Fungsi Lambda diperlukan saat Anda memanggil API. `CreateCloudConnector` Kode yang digunakan ke dalam fungsi Lambda harus mengimplementasikan semua antarmuka dan fungsionalitas seperti yang disebutkan dalam. [Bangun konektor C2C \(Cloud-to-Cloud\)](#)

Alur kerja konektor integrasi terkelola

Pengembang harus mendaftarkan konektor C2C dengan integrasi terkelola untuk AWS IoT Device Management Proses pendaftaran ini menciptakan sumber daya konektor logis yang dapat diakses pelanggan untuk menggunakan konektor.

Note

Konektor C2C adalah seperangkat metadata yang dibuat dalam integrasi terkelola untuk AWS IoT Device Management untuk menggambarkan konektor.

Diagram berikut menggambarkan peran konektor C2C saat mengirim perintah dari aplikasi seluler ke perangkat yang terhubung dengan cloud. Konektor C2C bertindak sebagai lapisan terjemahan antara integrasi terkelola untuk AWS IoT Device Management dan platform cloud pihak ketiga.

Pedoman untuk menggunakan konektor C2C () cloud-to-cloud

Konektor C2C apa pun yang Anda buat adalah konten Anda, dan konektor C2C apa pun yang dibuat oleh pelanggan lain yang Anda akses adalah konten pihak ketiga. AWS tidak membuat atau mengelola konektor C2C apa pun sebagai bagian dari integrasi terkelola.

Anda dapat membagikan konektor C2C Anda dengan pelanggan integrasi terkelola lainnya. Jika Anda melakukannya, Anda memberi wewenang AWS sebagai penyedia layanan Anda untuk mencantumkan konektor C2C tersebut dan informasi kontak terkait di AWS konsol dan Anda memahami bahwa AWS pelanggan lain dapat menghubungi Anda. Anda bertanggung jawab penuh untuk memberikan pelanggan akses ke konektor C2C Anda dan untuk persyaratan apa pun yang mengatur akses AWS pelanggan lain ke konektor C2C Anda.

Bangun konektor C2C (Cloud-to-Cloud)

Bagian berikut mencakup langkah-langkah untuk membangun konektor C2C (Cloud-to-Cloud) untuk integrasi terkelola untuk AWS IoT Device Management.

Topik

- [Prasyarat](#)
- [Persyaratan konektor C2C](#)
- [OAuth 2.0 persyaratan untuk penautan akun](#)
- [Menerapkan operasi antarmuka konektor C2C](#)
- [Panggil konektor C2C Anda](#)
- [Tambahkan izin ke Peran IAM Anda](#)
- [Uji konektor C2C Anda secara manual](#)

Prasyarat

Sebelum Anda membuat konektor C2C (Cloud-to-Cloud), Anda memerlukan yang berikut ini:

- Akun AWS Untuk meng-host konektor C2C Anda dan mendaftarkannya melalui integrasi terkelola. Untuk informasi selengkapnya, lihat [Membuat Akun AWS](#).
- Ketika Anda membangun konektor Anda, Anda memerlukan izin IAM tertentu. Untuk menggunakan

- Pastikan bahwa penyedia cloud pihak ketiga yang dimaksudkan untuk konektor, mendukung otorisasi OAuth 2.0. Untuk informasi selengkapnya, lihat [OAuth 2.0 persyaratan untuk penautan akun](#).

Selain itu, untuk menguji konektor, pengembang konektor harus memiliki yang berikut:

- ID klien dari cloud pihak ketiga untuk dikaitkan dengan konektor C2C Anda
- Rahasia klien dari cloud pihak ketiga untuk dikaitkan dengan konektor C2C Anda
- URL otorisasi OAuth 2.0
- URL token OAuth 2.0
- Kunci API apa pun yang diperlukan oleh API pihak ketiga Anda
- Kunci API apa pun yang diperlukan oleh pendaftaran API pihak ketiga Anda atau daftar yang diizinkan untuk URL OAuth panggilan balik yang dihosting oleh. AWS Beberapa pihak ketiga secara eksplisit mengizinkan daftar URL OAuth pengalihan, sementara yang lain memiliki alur kerja di mana pengguna dapat masuk dan mendaftarkan URL. OAuth Konsultasikan dengan pihak ketiga tertentu untuk memahami apa yang diperlukan untuk mengizinkan daftar titik akhir pengalihan integrasi OAuth terkelola

Izin yang diperlukan

Ketika Anda membangun konektor Anda, Anda memerlukan izin IAM tertentu. Selain `iotmanagedintegrations`: izin untuk tindakan, Anda memerlukan izin berikut:

- [CreateAccountAssociation](#), [CreateConnectorDestination](#), [GetAccountAssociation](#), dan [StartAccountAssociationRefresh](#), membutuhkan `secretsmanager:GetSecretValue`
- [CreateCloudConnector](#) membutuhkan `lambda:Invoke`

Untuk informasi selengkapnya tentang `iotmanagedintegrations`: izin dan tindakan, lihat [Tindakan yang ditentukan oleh integrasi AWS terkelola](#)

Persyaratan konektor C2C

[Konektor C2C](#) yang Anda kembangkan memfasilitasi komunikasi dua arah antara integrasi terkelola untuk AWS IoT Device Management dan cloud vendor pihak ketiga. Konektor harus mengimplementasikan antarmuka untuk integrasi terkelola untuk AWS IoT Device Management untuk melakukan tindakan atas nama pengguna akhir. Antarmuka ini menyediakan fungsionalitas untuk menemukan perangkat pengguna akhir, memulai perintah perangkat yang dikirim dari integrasi

terkelola untuk AWS IoT Device Management, dan mengidentifikasi pengguna berdasarkan token akses. Untuk mendukung pengoperasian perangkat, konektor harus mengelola terjemahan pesan permintaan dan respons antara integrasi terkelola untuk AWS IoT Device Management dan platform pihak ketiga terkait.

Berikut ini adalah persyaratan untuk konektor C2C:

- Server Otorisasi pihak ketiga harus sesuai dengan standar OAuth 2.0 serta konfigurasi yang tercantum dalam. [OAuth persyaratan konfigurasi](#)
- Konektor C2C akan diperlukan untuk menafsirkan pengidentifikasi dari AWS implementasi Model Data Materi dan harus memancarkan respons dan peristiwa yang sesuai dengan AWS implementasi Model Data Materi. Untuk informasi selengkapnya, lihat [AWS implementasi model data Matter](#)
- Konektor C2C harus dapat memanggil integrasi terkelola untuk AWS IoT Device Management dengan autentikasi. APIs SigV4 Untuk peristiwa asinkron yang dikirim dengan SendConnectorEvent API, Akun AWS kredensial yang sama yang digunakan untuk mendaftarkan konektor harus digunakan untuk menandatangani permintaan terkait. SendConnectorEvent
- Konektor harus mengimplementasikan [AWS.ActivateUser](#), [AWS.DiscoverDevices](#), [AWS.SendCommand](#), dan [AWS.DeactivateUser](#) operasi.
- Saat konektor C2C Anda menerima peristiwa pihak ketiga yang terkait dengan respons perintah perangkat atau penemuan perangkat, konektor harus meneruskannya ke integrasi terkelola dengan API. SendConnectorEvent Untuk informasi selengkapnya tentang peristiwa ini dan SendConnectorEvent API, lihat [SendConnectorEvent](#).

Note

SendConnectorEventAPI adalah bagian dari SDK integrasi terkelola dan digunakan, bukan pembuatan manual dan penandatanganan permintaan.

OAuth 2.0 persyaratan untuk penautan akun

Setiap konektor C2C bergantung pada server otorisasi OAuth 2.0 untuk mengautentikasi pengguna akhir. Melalui server ini, pengguna akhir menautkan akun pihak ketiga mereka dengan platform perangkat pelanggan. Penautan akun adalah langkah pertama yang diperlukan oleh pengguna

akhir untuk menggunakan perangkat yang didukung oleh konektor C2C Anda. Untuk informasi selengkapnya tentang peran yang berbeda dalam penautan akun dan OAuth 2.0, lihat [Peran penautan akun](#).

Meskipun konektor C2C Anda tidak perlu menerapkan logika bisnis tertentu untuk mendukung aliran otorisasi, server otorisasi OAuth2 .0 yang terkait dengan konektor C2C Anda harus memenuhi. [OAuth persyaratan konfigurasi](#)

Note

Integrasi terkelola AWS IoT Device Management hanya mendukung OAuth 2.0 dengan alur kode otorisasi. Lihat [RFC 6749 untuk informasi](#) lebih lanjut.

Penautan akun adalah proses yang memungkinkan integrasi terkelola dan konektor untuk mengakses perangkat pengguna akhir dengan menggunakan token akses. Token ini menyediakan integrasi terkelola untuk AWS IoT Device Management dengan izin pengguna akhir, sehingga konektor dapat berinteraksi dengan data pengguna akhir melalui panggilan API. Untuk informasi selengkapnya, lihat [Alur kerja penautan akun](#).

Kami menyarankan Anda untuk tidak mencatat token sensitif ini di log apa pun. Namun, jika disimpan dalam log, kami sarankan Anda menggunakan kebijakan perlindungan data CloudWatch Log untuk menutupi token di log. Untuk informasi selengkapnya, lihat [Membantu melindungi data log sensitif dengan masking](#).

Integrasi terkelola untuk AWS IoT Device Management tidak mendapatkan token akses secara langsung; ia melakukannya melalui Jenis Hibah Kode Otorisasi. Pertama, integrasi terkelola untuk AWS IoT Device Management harus mendapatkan kode otorisasi. Kemudian menukar kode dengan token akses dan token penyegaran. Token penyegaran digunakan untuk meminta token akses baru saat token akses lama kedaluwarsa. Jika token akses dan token penyegaran kedaluwarsa, Anda harus melakukan alur penautan akun lagi. Anda dapat melakukan ini dengan operasi `StartAccountAssociationRefresh` API.

Important

Token akses yang dikeluarkan harus dicakup per pengguna, tetapi tidak per klien. OAuth Token tidak boleh menyediakan akses ke semua perangkat dari semua pengguna di bawah klien.

Server otorisasi harus melakukan salah satu hal berikut:

- Keluarkan token akses yang berisi ID pengguna akhir (pemilik sumber daya) yang dapat diekstraksi, seperti token JWT.
- Kembalikan ID pengguna akhir untuk setiap token akses yang dikeluarkan.

OAuth persyaratan konfigurasi

Tabel berikut menggambarkan parameter yang diperlukan dari server OAuth otorisasi Anda untuk integrasi terkelola untuk AWS IoT Device Management untuk melakukan penautan akun:

OAuth Parameter Server

Bidang	Diperlukan	Komentar
<code>clientId</code>	Ya	Pengenal publik untuk aplikasi Anda. Ini digunakan untuk memulai alur otentikasi dan dapat dibagikan secara publik.
<code>clientSecret</code>	Ya	Kunci rahasia yang digunakan untuk mengotentikasi aplikasi dengan server otorisasi, terutama ketika menukar kode otorisasi untuk token akses. Itu harus dirahasiakan dan tidak dibagikan secara publik.
<code>authorizationType</code>	Ya	Jenis otorisasi yang didukung oleh konfigurasi otorisasi ini. Saat ini, "OAuth 2.0" adalah satu-satunya nilai yang didukung.
<code>authUrl</code>	Ya	URL otorisasi untuk penyedia cloud pihak ketiga.
<code>tokenUrl</code>	Ya	URL token untuk penyedia cloud pihak ketiga.

<code>tokenEndpointAuthenticationScheme</code>	Ya	Skema otentikasi baik "HTTP_BASIC" atau "REQUEST_BODY_CREDENTIALS". HTTP_BASIC memberi sinyal bahwa kredensi klien disertakan dalam header otorisasi, sedangkan tangga memberi sinyal mereka disertakan dalam badan permintaan.
--	----	---

OAuth Server yang Anda gunakan harus dikonfigurasi sehingga nilai string token akses harus dikodekan Base64 dengan set karakter UTF-8.

Peran penautan akun

Untuk membuat konektor C2C, Anda memerlukan server otorisasi OAuth 2.0 dan penautan akun. Untuk informasi selengkapnya, lihat [Alur kerja penautan akun](#).

OAuth 2.0 mendefinisikan empat peran berikut saat menerapkan penautan akun:

1. Server otorisasi
2. Pemilik sumber daya (Pengguna Akhir)
3. Server sumber daya
4. Klien

Berikut ini menentukan masing-masing OAuth peran ini:

Server Otorisasi

Server otorisasi adalah server yang mengidentifikasi dan mengotentikasi identitas pengguna akhir di cloud pihak ketiga. Token akses yang disediakan oleh server ini dapat menautkan akun platform pelanggan pengguna AWS akhir dan akun platform pihak ketiga mereka. Proses ini disebut sebagai penautan akun.

Server otorisasi mendukung penautan akun dengan memberikan yang berikut:

- Menampilkan halaman login bagi pengguna akhir untuk masuk ke sistem Anda. Ini biasanya disebut sebagai titik akhir otorisasi.
- Mengautentikasi pengguna akhir di sistem Anda.
- Menghasilkan kode otorisasi yang mengidentifikasi pengguna akhir.
- Meneruskan kode otorisasi ke integrasi terkelola untuk AWS IoT Device Management.
- Menerima kode otorisasi dari integrasi terkelola untuk AWS IoT Device Management dan mengembalikan token akses yang dapat digunakan integrasi terkelola untuk AWS IoT Device Management untuk mengakses data pengguna akhir di sistem Anda. Ini biasanya diselesaikan melalui URI terpisah, yang disebut token URI atau endpoint.

Important

[Server otorisasi harus mendukung alur Kode Otorisasi OAuth 2.0 untuk digunakan dengan integrasi terkelola untuk AWS IoT Device Management Connector. integrasi terkelola untuk AWS IoT Device Management juga mendukung alur kode otorisasi dengan Proof Key for Code Exchange \(PKCE\).](#)

Server otorisasi harus:

- Keluarkan token akses yang berisi ID pengguna akhir atau pemilik sumber daya yang dapat diekstraksi, misalnya token JWT
- Dapat mengembalikan ID pengguna akhir untuk setiap token akses yang dikeluarkan. Jika tidak, konektor Anda tidak akan dapat mendukung `AWS.ActivateUser` operasi yang diperlukan. Ini akan mencegah penggunaan konektor dengan integrasi terkelola.

Jika pengembang konektor atau pemilik tidak memelihara server otorisasi mereka sendiri, server otorisasi yang digunakan harus memberikan otorisasi untuk sumber daya yang dikelola oleh platform pihak ketiga pengembang konektor. Ini berarti bahwa setiap token yang diterima oleh integrasi terkelola dari server otorisasi harus memberikan batasan keamanan yang berarti pada perangkat (sumber daya). Misalnya, token pengguna akhir tidak mengizinkan perintah di perangkat pengguna akhir lain; izin yang diberikan oleh token dipetakan ke sumber daya dalam platform. Perhatikan contoh Lights Incorporated. Ketika pengguna akhir memulai alur penautan akun dengan konektornya, mereka diarahkan ke halaman login Lights Incorporated yang berada di depan server otorisasi mereka. Setelah mereka masuk dan memberikan izin kepada klien, mereka memberikan token yang memberikan akses konektor ke sumber daya dalam akun Lights Incorporated mereka.

Pemilik sumber daya (Pengguna Akhir)

Sebagai pemilik sumber daya, Anda mengizinkan integrasi terkelola untuk akses pelanggan AWS IoT Device Management ke sumber daya yang terkait dengan akun Anda dengan melakukan penautan akun. Misalnya, pertimbangkan bohlam pintar yang telah dimasukkan pengguna akhir ke aplikasi seluler Lights Incorporated. Pemilik sumber daya mengacu pada akun pengguna akhir yang telah membeli dan melakukan onboard perangkat. Dalam contoh kita, pemilik sumber daya dimodelkan sebagai akun Lights Incorporated OAuth2 .0. Sebagai pemilik sumber daya, akun ini memberikan izin untuk mengeluarkan perintah dan mengelola perangkat.

Server sumber daya

Ini adalah server yang menampung sumber daya yang dilindungi yang memerlukan otorisasi untuk mengakses (data perangkat). AWS Pelanggan perlu mengakses sumber daya yang dilindungi atas nama pengguna akhir, dan mereka melakukannya melalui integrasi terkelola untuk konektor AWS IoT Device Management setelah penautan akun. Mempertimbangkan bohlam pintar dari sebelumnya sebagai contoh, server sumber daya adalah layanan berbasis cloud yang dimiliki oleh Lights Incorporated yang mengelola bohlam setelah di-onboard. Melalui server sumber daya, pemilik sumber daya dapat mengeluarkan perintah ke bohlam pintar, seperti menyalakan dan mematikannya. Sumber daya yang dilindungi hanya memberikan izin ke akun pengguna akhir dan lainnya yang mungkin telah accounts/entities mereka berikan izin.

Klien

Dalam konteks ini, klien adalah konektor C2C Anda. Klien didefinisikan sebagai aplikasi yang diberikan akses ke sumber daya dalam server sumber daya atas nama pengguna akhir. Proses penautan akun mewakili konektor, klien, yang meminta akses ke sumber daya pengguna akhir dalam cloud pihak ketiga.

Meskipun konektornya adalah OAuth klien, integrasi terkelola untuk AWS IoT Device Management melakukan operasi atas nama konektor. Misalnya, integrasi terkelola untuk AWS IoT Device Management membuat permintaan ke server otorisasi untuk mendapatkan token akses. Konektor masih dianggap klien karena merupakan satu-satunya komponen yang pernah mengakses sumber daya yang dilindungi (data perangkat) di server sumber daya.

Pertimbangkan bohlam pintar yang telah di-onboard oleh pengguna akhir. Setelah penautan akun selesai antara platform pelanggan dan server otorisasi Lights Incorporated, konektor itu sendiri akan berkomunikasi dengan server sumber daya untuk mengambil informasi tentang bohlam pintar pengguna akhir. Konektor kemudian dapat menerima perintah dari pengguna akhir. Ini

termasuk menyalakan atau mematikan lampu atas nama mereka melalui server sumber daya Lights Incorporated. Dengan demikian, kami menunjuk konektor sebagai klien.

Alur kerja penautan akun

Agar integrasi terkelola pelanggan untuk platform AWS IoT Device Management dapat berinteraksi dengan perangkat pengguna akhir di platform pihak ketiga Anda melalui konektor C2C Anda, ia memperoleh token akses melalui alur kerja berikut:

1. Saat pengguna memulai orientasi perangkat pihak ketiga melalui aplikasi pelanggan, integrasi terkelola untuk AWS IoT Device Management mengembalikan URI Otorisasi serta. AssociationId
2. Aplikasi front-end menyimpan AssociationId dan mengarahkan pengguna akhir ke halaman login platform pihak ketiga.
 - Pengguna akhir masuk. Pengguna akhir memberikan klien akses ke data perangkat mereka.
3. Platform pihak ketiga membuat kode otorisasi. Pengguna akhir diarahkan ke integrasi terkelola untuk URI callback platform AWS IoT Device Management termasuk kode yang dilampirkan pada permintaan pengalihan.
4. Integrasi terkelola menukar kode ini dengan URI token platform pihak ketiga.
5. URI token memvalidasi kode otorisasi dan mengembalikan token OAuth2 akses.0 dan token penyegaran, yang terkait dengan pengguna akhir.
6. Integrasi terkelola memanggil konektor C2C dengan `AWS.ActivateUser` operasi untuk menyelesaikan alur Penautan Akun dan dapatkan. UserId
7. Integrasi terkelola mengembalikan OAuth RedirectUrl (dari konfigurasi Kebijakan Konektor) dari halaman otentikasi yang berhasil ke aplikasi pelanggan.

Note

Jika terjadi kegagalan, integrasi terkelola untuk AWS IoT Device Management menambahkan kesalahan dan parameter kueri `error_description` ke URL yang memberikan detail kesalahan ke aplikasi pelanggan.

8. Aplikasi pelanggan mengarahkan pengguna akhir ke file. OAuth RedirectUrl Pada titik ini front-end aplikasi mengetahui asosiasi AssociationId dari langkah pertama.

Semua permintaan berikutnya yang dibuat dari integrasi terkelola untuk AWS IoT Device Management melalui konektor C2C ke platform cloud pihak ketiga, seperti perintah untuk menemukan perangkat dan mengirim perintah, akan menyertakan OAuth2 token akses.0.

Diagram berikut menunjukkan hubungan antara komponen kunci dari penautan akun:

Menerapkan operasi antarmuka konektor C2C

Integrasi terkelola untuk AWS IoT Device Management mendefinisikan empat operasi yang AWS Lambda harus Anda tangani agar memenuhi syarat sebagai konektor. Konektor C2C Anda harus mengimplementasikan setiap operasi berikut:

1. [AWS.ActivateUser](#)- Integrasi terkelola untuk AWS IoT Device Management layanan memanggil API ini untuk mengambil pengenalan pengguna unik secara global yang terkait dengan token.0 yang disediakan OAuth2. Operasi ini secara opsional dapat digunakan untuk melakukan persyaratan tambahan apa pun untuk proses penautan akun.
2. [AWS.DiscoverDevices](#)- Integrasi terkelola untuk layanan AWS IoT Device Management memanggil API ini ke konektor Anda untuk menemukan perangkat pengguna
3. [AWS.SendCommand](#)- Integrasi terkelola untuk layanan AWS IoT Device Management memanggil API ini ke konektor Anda untuk mengirim perintah untuk perangkat pengguna
4. [AWS.DeactivateUser](#)- Integrasi terkelola untuk layanan AWS IoT Device Management memanggil API ini ke konektor Anda untuk menonaktifkan token akses pengguna untuk menghapus tautan di server otorisasi Anda.

Integrasi terkelola untuk AWS IoT Device Management selalu memanggil fungsi Lambda dengan muatan string JSON melalui tindakan. AWS Lambda `invokeFunction` Operasi permintaan harus menyertakan `operationName` bidang di setiap payload permintaan. Untuk selengkapnya, lihat [Memanggil di Referensi AWS Lambda API](#).

Setiap batas waktu pemanggilan diatur ke dua detik, dan jika pemanggilan gagal, itu akan dicoba ulang lima kali.

Lambda yang Anda terapkan untuk konektor Anda akan mengurai `operationName` dari payload permintaan, dan mengimplementasikan fungsionalitas yang sesuai untuk dipetakan ke cloud pihak ketiga:

```

public ConnectorResponse handleRequest(final ConnectorRequest request)
    throws OperationFailedException {
    Operation operation;
    try {
        operation = Operation.valueOf(request.payload().operationName());
    } catch (IllegalArgumentException ex) {
        throw new ValidationException(
            "Unknown operation '%s'".formatted(request.payload().operationName()),
            ex
        );
    }

    return switch (operation) {
        case ActivateUser -> activateUserManager.activateUser(request);
        case DiscoverDevices -> deviceDiscoveryManager.listDevices(request);
        case SendCommand -> sendCommandManager.sendCommand(request);
        case DeactivateUser -> deactivateUser.deactivateUser(request);
    };
}

```

Note

Pengembang konektor harus

mengimplementasikan `activateUserManager.activateUser(request)`, `deviceDiscoveryManager.listDevices(request)`, dan `deactivateUser.deactivateUser` operasi yang tercantum dalam contoh sebelumnya.

Contoh berikut merinci permintaan konektor generik dari integrasi terkelola, di mana bidang umum untuk setiap antarmuka yang diperlukan hadir. Dari contoh, Anda dapat melihat ada header permintaan dan payload permintaan. Header permintaan umum di setiap antarmuka operasi.

```

{
  "header": {
    "auth": {
      "token": "ashriu32yr97feqy7afsaf",
      "type": "OAuth2.0"
    }
  },
  "payload": {
    "operationName": "AWS.SendCommand",

```

```

"operationVersion": "1.0",
"connectorId": "exampleId",
...
}
}

```

Header permintaan default

Bidang header default adalah sebagai berikut.

```

{
  "header": {
    "auth": {
      "token": string, // end user's Access Token
      "type": ENUM ["OAuth2.0"],
    }
  }
}

```

API apa pun yang dihosting oleh konektor harus memproses parameter header berikut:

Header dan bidang default

Bidang	Dibutuhkan/Opsional	Deskripsi
header:auth	Ya	Informasi otorisasi yang diberikan oleh pembuat konektor C2C selama pendaftaran konektor mereka.
header:auth:token	Ya	Token otorisasi pengguna yang dihasilkan oleh penyedia cloud pihak ketiga dan ditautkan keconnector AssociationID .
header:auth:type	Ya	Jenis otorisasi yang dibutuhkan.

Note

Semua permintaan ke konektor Anda akan memiliki token akses pengguna akhir yang terpasang. Anda dapat berasumsi bahwa penautan akun antara pengguna akhir dan integrasi terkelola pelanggan telah terjadi.

Meminta muatan

Selain header umum, setiap permintaan akan memiliki muatan. Meskipun payload ini akan memiliki bidang unik untuk setiap jenis operasi, setiap payload memiliki satu set bidang default yang akan selalu ada.

Minta bidang payload:

- `operationName`: Pengoperasian permintaan yang diberikan, sama dengan salah satu nilai berikut: `AWS.ActivateUser`, `AWS.SendCommand`, `AWS.DiscoverDevices`, `AWS.DeactivateUser`.
- `operationVersion`: Setiap operasi diberi versi untuk memungkinkan evolusinya dari waktu ke waktu dan memberikan definisi antarmuka yang stabil untuk konektor pihak ketiga. Integrasi terkelola melewati bidang versi dalam muatan semua permintaan.
- `connectorId`: ID konektor di mana permintaan telah dikirim ke.

Header respons default

Setiap operasi akan merespons dengan integrasi terkelola ACK untuk AWS IoT Device Management yang mengonfirmasi konektor C2C Anda telah menerima permintaan dan mulai memprosesnya.

Berikut ini adalah contoh umum dari tanggapan tersebut:

```
{
  "header": {
    "responseCode": 200
  },
  "payload": {
    "responseMessage": "Example response!"
  }
}
```

Setiap respon operasi harus memiliki header umum berikut:

```
{
  "header": {
    "responseCode": Integer
  }
}
```

Tabel berikut mencantumkan header respon default:

Header dan bidang respons default

Bidang	Dibutuhkan/Opsional	Komentar
header:responseCode	Ya	ENUM nilai yang menunjukkan status eksekusi permintaan.

Sepanjang berbagai Antarmuka Konektor dan skema API yang dijelaskan dalam dokumen ini ada bidang `responseMessage` atau `message`. Ini adalah bidang opsional yang digunakan untuk konektor C2C Lambda untuk merespons dengan konteks apa pun mengenai permintaan dan pelaksanaannya. Lebih disukai, kesalahan apa pun yang menghasilkan kode status selain 200 harus menyertakan nilai pesan yang menjelaskan kesalahan.

Menanggapi permintaan operasi konektor C2C dengan API `SendConnectorEvent`

Integrasi terkelola untuk AWS IoT Device Management mengharapkan konektor Anda berperilaku asinkron untuk setiap operasi dan operasi. `AWS.DiscoverDevices` Ini berarti bahwa respon awal untuk operasi ini, cukup “mengakui” bahwa konektor C2C Anda telah menerima permintaan.

Dengan menggunakan `SendConnectorEvent` API, konektor Anda diharapkan mengirim jenis peristiwa dari daftar di bawah ini ke `AWS.DiscoverDevices` dan `AWS.SendCommand` operasi, serta peristiwa perangkat proaktif (seperti lampu yang dinyalakan dan dimatikan secara manual). Untuk membaca penjelasan rinci tentang jenis acara ini dan kasus penggunaannya, lihat [Menerapkan AWS. DiscoverDevices operasi](#), [Menerapkan AWS. SendCommand operasi](#), dan [Mengirim peristiwa perangkat dengan SendConnectorEvent API](#).

Misalnya, jika konektor C2C Anda menerima `DiscoverDevices` permintaan, integrasi terkelola untuk AWS IoT Device Management mengharapkannya merespons secara serempak dengan format respons yang ditentukan di atas. Kemudian, Anda harus memanggil `SendConnectorEvent` API dengan struktur permintaan yang ditentukan dalam [Menerapkan AWS. DiscoverDevices operasi](#),

untuk acara `DEVICE_DISCOVERY`. Panggilan API aktif dapat dilakukan `SendConnectorEvent` di mana saja Anda memiliki akses ke kredensial Lambda konektor C2C Anda. Akun AWS Alur penemuan perangkat tidak berhasil sampai integrasi terkelola untuk AWS IoT Device Management menerima acara ini.

Note

Atau, panggilan `SendConnectorEvent` API dapat terjadi sebelum respons pemanggilan Lambda konektor C2C jika perlu. Namun, aliran ini bertentangan dengan model asinkron untuk pengembangan perangkat lunak.

- `SendConnectorEvent`- Konektor Anda memanggil integrasi terkelola ini untuk AWS IoT Device Management API untuk mengirim peristiwa perangkat ke integrasi terkelola untuk AWS IoT Device Management. Hanya 3 jenis acara yang diterima oleh integrasi terkelola:
 - “`DEVICE_DISCOVERY`” — Operasi acara ini akan digunakan untuk mengirim daftar perangkat yang ditemukan dalam cloud pihak ketiga untuk token akses tertentu.
 - “`DEVICE_COMMAND_RESPONSE`” — Operasi peristiwa ini akan digunakan untuk mengirim peristiwa perangkat tertentu sebagai hasil dari eksekusi perintah.
 - “`DEVICE_EVENT`” - Operasi peristiwa ini harus digunakan untuk setiap peristiwa yang berasal dari perangkat yang bukan merupakan hasil langsung dari perintah berbasis pengguna. Ini dapat berfungsi sebagai jenis acara umum untuk secara proaktif melaporkan perubahan atau pemberitahuan status perangkat.

Menerapkan AWS. `ActivateUser` operasi

`AWS. ActivateUser` Operasi ini diperlukan untuk integrasi terkelola AWS IoT Device Management untuk mengambil pengenal pengguna dari token .0 pengguna akhir. OAuth2 Integrasi terkelola untuk AWS IoT Device Management akan meneruskan OAuth token dalam header permintaan, dan mengharapkan konektor Anda menyertakan pengenal pengguna unik secara global dalam muatan respons. Operasi ini terjadi setelah alur penautan akun berhasil.

Daftar berikut menguraikan persyaratan konektor Anda untuk memfasilitasi alur `AWS. Activate` pengguna yang sukses.

- Konektor C2C Lambda Anda dapat memproses pesan permintaan `AWS. ActivateUser` operasi dari integrasi terkelola untuk AWS IoT Device Management.

- Konektor C2C Lambda Anda dapat menentukan pengenal pengguna unik dari token.0 yang disediakan. OAuth2 Biasanya, itu dapat diekstraksi baik dari token itu sendiri, jika itu adalah token JWT, atau diminta dari server Otorisasi oleh token.

AWS.ActivateUser alur kerja

1. Integrasi terkelola untuk AWS IoT Device Management memanggil Lambda konektor C2C Anda dengan muatan berikut:

```
{
  "header": {
    "auth": {
      "token": "ashriu32yr97feqy7afsaf",
      "type": "OAuth2.0"
    }
  },
  "payload": {
    "operationName": "AWS.ActivateUser",
    "operationVersion": "1.0.0",
    "connectorId": "Your-Connector-ID",
  }
}
```

2. Konektor C2C menentukan ID pengguna, baik dari token atau dengan menanyakan server sumber daya pihak ketiga Anda, untuk disertakan dalam respons. AWS.ActivateUser
3. Konektor C2C merespons pemanggilan operasi AWS.ActivateUser Lambda, termasuk payload default serta pengenal pengguna yang sesuai di dalam bidang. `userId`

```
{
  "header": {
    "responseCode": 200
  },
  "payload": {
    "responseMessage": "Successfully activated user with connector-id `Your-Connector-Id.",
    "userId": "123456"
  }
}
```

Menerapkan AWS. DiscoverDevices operasi

Penemuan perangkat menyelaraskan daftar perangkat fisik yang dimiliki oleh pengguna akhir dengan representasi digital dari perangkat pengguna akhir yang dikelola dalam integrasi terkelola untuk AWS IoT Device Management. Ini dilakukan oleh AWS pelanggan pada perangkat yang dimiliki oleh pengguna akhir hanya setelah penautan akun selesai antara pengguna dan integrasi terkelola untuk AWS IoT Device Management. Penemuan perangkat adalah proses asinkron di mana integrasi terkelola untuk AWS IoT Device Management memanggil konektor untuk memulai permintaan penemuan perangkat. Konektor C2C mengembalikan daftar perangkat pengguna akhir yang ditemukan secara asinkron dengan pengidentifikasi referensi (disebut sebagai `deviceDiscoveryId`) yang dihasilkan oleh integrasi terkelola.

Diagram berikut menggambarkan alur kerja penemuan perangkat antara pengguna akhir dan integrasi terkelola untuk AWS IoT Device Management:

AWS. DiscoverDevices alur kerja

1. Pelanggan memulai proses penemuan perangkat atas nama pengguna akhir.
2. Integrasi terkelola untuk AWS IoT Device Management menghasilkan pengenalan referensi yang dipanggil `deviceDiscoveryId` untuk permintaan penemuan perangkat yang dihasilkan oleh Pelanggan. AWS
3. Integrasi terkelola untuk AWS IoT Device Management mengirimkan permintaan penemuan perangkat ke konektor C2C menggunakan antarmuka `AWS.DiscoverDevices` operasi, termasuk pengguna OAuth `accessToken` akhir yang valid dan juga `deviceDiscoveryId`
4. Konektor Anda menyimpan `deviceDiscoveryId` untuk dimasukkan dalam `DEVICE_DISCOVERY` acara tersebut. Acara ini juga akan berisi daftar perangkat pengguna akhir yang ditemukan, dan harus dikirim ke integrasi terkelola untuk AWS IoT Device Management dengan `SendConnectorEvent` API sebagai `DEVICE_DISCOVERY` acara.
5. Konektor C2C Anda harus memanggil server sumber daya untuk mengambil semua perangkat yang dimiliki oleh pengguna akhir.
6. Konektor C2C Anda Lambda merespons pemanggilan Lambda `invokeFunction()` dengan respons ACK kembali ke integrasi terkelola untuk AWS IoT Device Management, yang bertindak sebagai respons awal untuk operasi. `AWS.DiscoverDevices` Integrasi terkelola memberi tahu pelanggan dengan ACK ke proses penemuan perangkat yang dimulai.
7. Server sumber daya Anda mengirim Anda daftar perangkat yang dimiliki dan dioperasikan oleh pengguna akhir.

8. Konektor Anda mengonversi setiap perangkat pengguna akhir menjadi integrasi terkelola untuk format perangkat yang diperlukan AWS IoT Device Management, termasuk `ConnectorDeviceId`, `ConnectorDeviceName` dan laporan Kemampuan untuk setiap perangkat.
9. Konektor C2C juga menyediakan `UserId` pemilik perangkat yang ditemukan. Ini dapat diambil dari server sumber daya Anda baik sebagai bagian dari daftar perangkat atau dalam panggilan terpisah tergantung pada implementasi server sumber daya Anda.
10. Selanjutnya, konektor C2C Anda akan memanggil integrasi terkelola untuk AWS IoT Device Management `APISendConnectorEvent`, melalui SigV4 Akun AWS menggunakan kredensial dan dengan parameter operasi ditetapkan sebagai "DEVICE_DISCOVERY". Setiap perangkat dalam daftar perangkat yang dikirim ke integrasi terkelola untuk AWS IoT Device Management akan diwakili oleh parameter khusus perangkat seperti `connectorDeviceId`, dan file `connectorDeviceName capabilityReport`
 - Berdasarkan respons server sumber daya Anda, Anda perlu memberi tahu integrasi terkelola untuk AWS IoT Device Management.

Misalnya, jika server sumber daya Anda memiliki respons paginasi ke daftar perangkat yang ditemukan untuk pengguna akhir, maka untuk setiap jajak pendapat Anda dapat mengirim peristiwa `DEVICE_DISCOVERY` operasi individual, dengan `statusCode` parameter. 3xx
Jika penemuan perangkat Anda masih dalam proses, ulangi langkah 5, 6, dan 7.

11. Integrasi terkelola untuk AWS IoT Device Management mengirimkan pemberitahuan kepada pelanggan tentang menemukan perangkat pengguna akhir.
12. Jika konektor C2C Anda mengirimkan peristiwa `DEVICE_DISCOVERY` operasi dengan `statusCode` parameter diperbarui dengan nilai 200, maka integrasi terkelola akan memberi tahu pelanggan tentang penyelesaian alur kerja penemuan perangkat.

Important

Langkah 7 hingga 11 dapat terjadi sebelum langkah 6, jika diinginkan. Misalnya, jika platform pihak ketiga Anda memiliki API untuk mencantumkan perangkat pengguna akhir, peristiwa `DEVICE_DISCOVERY` dapat dikirim `SendConnectorEvent` sebelum konektor C2C Lambda merespons dengan ACK biasa.

Persyaratan konektor C2C untuk penemuan perangkat

Daftar berikut menguraikan persyaratan untuk konektor C2C Anda untuk memfasilitasi penemuan perangkat yang sukses.

- Konektor C2C Lambda dapat memproses pesan permintaan penemuan perangkat dari integrasi terkelola untuk AWS IoT Device Management dan menangani operasi. `AWS.DiscoverDevices`
- Konektor C2C Anda dapat memanggil integrasi terkelola untuk AWS IoT Device APIs Management melalui SigV4 menggunakan kredensial yang digunakan untuk mendaftarkan konektor. Akun AWS

Proses penemuan perangkat

Langkah-langkah berikut menguraikan proses penemuan perangkat dengan konektor C2C dan integrasi terkelola untuk AWS IoT Device Management.

Proses penemuan perangkat

1. Integrasi terkelola memicu penemuan perangkat:

- Kirim permintaan POST ke `DiscoverDevices` dengan muatan JSON berikut:

```
/DiscoverDevices
{
  "header": {
    "auth": {
      "token": "ashriu32yr97feqy7afsaf",
      "type": "OAuth2.0"
    }
  },
  "payload": {
    "operationName": "AWS.DiscoverDevices",
    "operationVersion": "1.0",
    "connectorId": "Your-Connector-Id",
    "deviceDiscoveryId": "12345678"
  }
}
```

2. Konektor mengakui penemuan:

- Konektor mengirimkan pengakuan dengan respons JSON berikut:

```
{
  "header": {
    "responseCode":200
  },
  "payload": {
    "responseMessage": "Discovering devices for discovery-job-id
'12345678' with connector-id `Your-Connector-Id`"
  }
}
```

3. Konektor mengirimkan Acara Penemuan Perangkat:

- Kirim permintaan POST ke `/connector-event/{your_connector_id}` dengan muatan JSON berikut:

```
AWS API - /SendConnectorEvent
URI - POST /connector-event/{your_connector_id}
{
  "UserId": "6109342",
  "Operation": "DEVICE_DISCOVERY",
  "OperationVersion": "1.0",
  "StatusCode": 200,
  "DeviceDiscoveryId": "12345678",
  "ConnectorId": "Your_connector_Id",
  "Message": "Device discovery for discovery-job-id '12345678' successful",
  "Devices": [
    {
      "ConnectorDeviceId": "Your_Device_Id_1",
      "ConnectorDeviceName": "Your-Device-Name",
      "CapabilityReport": {
        "nodeId":"1",
        "version":"1.0.0",
        "endpoints":[{"
          "id":"1",
          "deviceTypes":["Camera"],
          "clusters":[{"
            "id":"0x0006",
            "revision":1,
            "attributes":[{"
              "id":"0x0000",
            }],
          "commands":["0x00","0x01"],
```

```
        "events":["0x00"]
      ]
    }
  }
}
```

Membangun sebuah CapabilityReport untuk acara DISCOVER_DEVICES

Seperti yang terlihat dalam struktur peristiwa yang didefinisikan di atas, setiap perangkat yang dilaporkan dalam peristiwa DISCOVER_DEVICES, yang berfungsi sebagai respons terhadap `AWS.DiscoverDevices` operasi, akan memerlukan a `CapabilityReport` untuk menggambarkan kemampuan perangkat yang sesuai. A `CapabilityReport` memberi tahu integrasi terkelola untuk kemampuan perangkat AWS IoT Device Management dalam format yang sesuai dengan Matter. Bidang berikut harus disediakan di `CapabilityReport`:

- `nodeId`, String: Identifier untuk node perangkat yang berisi berikut endpoints
- `version`, String: Versi node perangkat ini, ditetapkan oleh pengembang konektor
- `endpoints`, Daftar<Cluster>: Daftar AWS implementasi Model Data Materi yang didukung oleh titik akhir perangkat ini.
 - `id`, String: Pengidentifikasi titik akhir yang ditetapkan oleh pengembang konektor
 - `deviceTypes`, Daftar<String>: Daftar jenis perangkat yang ditangkap titik akhir ini, yaitu "Kamera".
 - `clusters`, Daftar<Cluster>: Daftar AWS implementasi Model Data Materi yang didukung oleh titik akhir ini.
 - `id`, String: Pengidentifikasi cluster seperti yang didefinisikan oleh standar Matter.
 - `revision`, Integer: Nomor revisi cluster sebagaimana didefinisikan oleh standar Matter.
 - `attributes`, Peta<String, Object>: Peta pengidentifikasi atribut dan nilai status perangkat yang sesuai saat ini, dengan pengidentifikasi dan nilai valid yang ditentukan oleh standar materi.
 - `id`, String: ID Atribut sebagaimana didefinisikan oleh AWS implementasi Model Data Materi.
 - `value`, Object: Nilai saat ini dari atribut yang ditentukan oleh ID atribut. Jenis 'nilai' dapat berubah tergantung pada atribut. `value` bidang ini opsional untuk setiap atribut dan hanya boleh disertakan jika lambda konektor Anda dapat menentukan status saat ini selama penemuan.

- `commands`, Daftar<String>: Daftar perintah IDs mendukung cluster ini seperti yang didefinisikan oleh standar Matter.
- `events`, Daftar<String>: Daftar acara IDs mendukung cluster ini sebagaimana didefinisikan oleh standar Matter.

Untuk daftar kemampuan yang didukung saat ini dan [AWS implementasi yang sesuai dari Model Data Matter](#), lihat rilis terbaru dari dokumentasi Model Data.

Menerapkan AWS. SendCommand operasi

`AWS . SendCommandOperasi` ini memungkinkan integrasi terkelola untuk AWS IoT Device Management untuk mengirim perintah yang dimulai oleh pengguna akhir melalui pelanggan ke server AWS sumber daya Anda. Server sumber daya Anda dapat mendukung beberapa jenis perangkat, di mana setiap jenis memiliki model responsnya sendiri. Eksekusi perintah adalah proses asinkron di mana integrasi terkelola untuk AWS IoT Device Management mengirimkan permintaan untuk eksekusi perintah dengan `TraceID`, yang akan disertakan konektor Anda dalam respons perintah yang dikirim kembali ke integrasi terkelola melalui `API. SendConnectorEvent` integrasi terkelola untuk AWS IoT Device Management mengharapkan server sumber daya mengembalikan respons yang mengakui bahwa perintah telah diterima, tetapi tidak selalu menunjukkan perintah itu dieksekusi.

Diagram berikut menggambarkan alur eksekusi perintah dengan contoh di mana pengguna akhir mencoba menyalakan lampu rumah mereka:

Alur kerja eksekusi perintah perangkat

1. Pengguna akhir mengirimkan perintah untuk menyalakan lampu menggunakan aplikasi AWS pelanggan.
2. Pelanggan menyampaikan informasi perintah ke integrasi terkelola untuk AWS IoT Device Management dengan informasi perangkat pengguna akhir.
3. Integrasi terkelola menghasilkan `traceID` yang akan digunakan konektor Anda saat mengirim respons perintah kembali ke layanan.
4. integrasi terkelola untuk AWS IoT Device Management mengirimkan permintaan perintah ke konektor Anda, menggunakan antarmuka operasi `AWS . SendCommand`.

- Payload yang ditentukan oleh antarmuka ini terdiri dari pengenalan perangkat, perintah perangkat yang dirumuskan sebagai Matterendpoints/clusters/commands, token akses pengguna akhir, dan parameter lain yang diperlukan.
5. Konektor Anda `traceId` menyimpan yang akan disertakan dalam respons perintah.
 - Konektor Anda menerjemahkan permintaan perintah integrasi terkelola ke dalam format server sumber daya yang sesuai.
 6. Konektor Anda mendapatkan `UserId` dari token akses pengguna akhir yang disediakan dan mengaitkannya dengan perintah.
 - a. `UserId` dapat diambil dari server sumber daya Anda menggunakan panggilan terpisah atau diekstrak dari token akses jika terjadi JWT dan token serupa.
 - b. Implementasi tergantung pada server sumber daya dan detail token akses Anda.
 7. Konektor Anda memanggil server sumber daya ke lampu pengguna akhir “Aktifkan”.
 8. Server sumber daya berinteraksi dengan perangkat.
 - a. Konektor menyampaikan ke integrasi terkelola untuk AWS IoT Device Management yang server sumber daya telah mengirimkan perintah, merespons dengan ACK sebagai respons perintah awal yang sinkron.
 - b. Integrasi terkelola kemudian menyampaikannya kembali ke aplikasi pelanggan.
 9. Setelah perangkat menyalakan lampu, peristiwa perangkat tersebut ditangkap oleh server sumber daya Anda.
 10. Server sumber daya Anda mengirimkan peristiwa perangkat ke konektor.
 11. Konektor Anda mengubah peristiwa perangkat yang dihasilkan oleh server sumber daya menjadi integrasi terkelola tipe operasi peristiwa `DEVICE_COMMAND_RESPONSE`.
 12. Konektor Anda memanggil `SendConnectorEvent` API dengan operasi sebagai “`DEVICE_COMMAND_RESPONSE`”.
 - Ini melampirkan integrasi yang `traceId` disediakan oleh terkelola untuk AWS IoT Device Management dalam permintaan awal.
 13. Integrasi terkelola memberi tahu pelanggan tentang perubahan status perangkat pengguna akhir.
 14. Pelanggan memberi tahu pengguna akhir bahwa lampu perangkat telah menyala.

Note

Konfigurasi server sumber daya Anda menentukan logika untuk menangani permintaan perintah perangkat yang gagal dan pesan respons. Ini termasuk percobaan ulang pesan menggunakan `referenceId` yang sama untuk perintah.

Persyaratan konektor C2C untuk eksekusi perintah perangkat

Daftar berikut menguraikan persyaratan untuk konektor C2C Anda untuk memfasilitasi eksekusi perintah perangkat yang berhasil.

- Konektor C2C Lambda dapat memproses pesan permintaan `AWS.SendCommand` operasi dari integrasi terkelola untuk AWS IoT Device Management.
- Konektor C2C Anda harus melacak perintah yang dikirim ke server sumber daya Anda dan memetakannya dengan `traceID` yang sesuai.
- Anda dapat memanggil integrasi terkelola untuk AWS IoT Device Management API layanan melalui SigV4 AWS menggunakan Akun AWS kredensial yang digunakan untuk mendaftarkan konektor C2C.

1. Integrasi terkelola mengirimkan perintah ke konektor (Lihat langkah 4 pada diagram sebelumnya).

```
/Send-Command
{
  "header": {
    "auth": {
      "token": "ashriu32yr97feqy7afsaf",
      "type": "OAuth2.0"
    }
  },
  "payload": {
    "operationName": "AWS.SendCommand",
    "operationVersion": "1.0",
    "connectorId": "Your-Connector-Id",
    "connectorDeviceId": "Your_Device_Id",
    "traceId": "traceId-3241u78123419",
    "endpoints": [{
```

```

        "id": "1",
        "clusters": [{
            "id": "0x0202",
            "commands": [{
                "0xff01":
                    {
                        "0x0000": "3"
                    }
            }]
        }]
    }
}

```

- Perintah ACK konektor C2C (Lihat langkah 7 pada diagram sebelumnya di mana konektor mengirimkan ACK ke integrasi terkelola untuk AWS IoT Device Management Service).

- ```

{
 "header":{
 "responseCode":200
 },
 "payload":{
 "responseMessage": "Successfully received send-command request for
connector 'Your-Connector-Id' and connector-device-id 'Your_Device_Id'"
 }
}

```

- Konektor mengirimkan peristiwa Respons Perintah Perangkat (Lihat langkah 11 pada diagram sebelumnya).

- ```

AWS-API: /SendConnectorEvent
URI: POST /connector-event/{Your-Connector-Id}

{
  "UserId": "End-User-Id",
  "Operation": "DEVICE_COMMAND_RESPONSE",
  "OperationVersion": "1.0",
  "StatusCode": 200,
  "Message": "Example message",
  "ConnectorDeviceId": "Your_Device_Id",
  "TraceId": "traceId-3241u78123419",
  "MatterEndpoint": {
    "id": "1",

```

```
    "clusters": [{
      "id": "0x0202",
      "attributes": [
        {
          "0x0000": "3"
        }
      ],
      "commands": [
        "0xff01":
        {
          "0x0000": "3"
        }
      ]
    }
  ]
}
```

Note

Perubahan status perangkat sebagai akibat dari eksekusi perintah tidak akan tercermin dalam integrasi terkelola untuk AWS IoT Device Management hingga peristiwa `DEVICE_COMMAND_RESPONSE` yang sesuai telah diterima melalui API. `SendConnectorEvent` ini berarti bahwa hingga integrasi terkelola menerima peristiwa dari langkah 3 sebelumnya, terlepas dari apakah respons pemanggilan konektor Anda menunjukkan keberhasilan atau tidak, status perangkat tidak akan diperbarui.

Menafsirkan 'titik akhir' materi yang disertakan dalam AWS. `SendCommand` permintaan

Integrasi terkelola akan menggunakan kemampuan perangkat yang dilaporkan selama penemuan perangkat untuk menentukan perintah apa yang dapat diterima perangkat. Setiap kemampuan perangkat dimodelkan melalui AWS implementasi Model Data Matter; dengan demikian, semua perintah yang masuk akan diturunkan dari bidang `commands` dalam cluster tertentu. Adalah tanggung jawab konektor Anda untuk mengurai bidang `titik akhir`, menentukan perintah Matter yang sesuai, dan menerjemahkannya sedemikian rupa sehingga perintah yang benar mencapai perangkat. Biasanya, ini berarti menerjemahkan model data Matter ke dalam permintaan API terkait.

Setelah perintah dijalankan, konektor Anda kemudian menentukan `atribut` mana yang ditentukan oleh AWS implementasi Model Data Materi yang telah berubah sebagai hasilnya. Perubahan ini

kemudian dilaporkan ke integrasi terkelola untuk AWS IoT Device Management melalui peristiwa API `DEVICE_COMMAND_RESPONSE` yang dikirim bersama API. `SendConnectorEvent`


Pertimbangkan bidang `endpoints` yang disertakan dalam contoh payload berikut:
`AWS.SendCommand`

```
"endpoints": [{
  "id": "1",
  "clusters": [{
    "id": "0x0202",
    "commands": [{
      "0xff01":
        {
          "0x0000": "3"
        }
    ]
  }]
}]
```

Dari objek ini, konektor dapat menentukan yang berikut:

1. Mengatur informasi endpoint dan cluster:

- a. Atur titik akhir `id` ke "1".

 Note

Jika perangkat mendefinisikan beberapa titik akhir sehingga satu cluster (seperti On/Off) can control multiple capabilities (i.e. turn a light on/off as well as turning a strobe on/off), `id` ini digunakan untuk merutekan perintah ke kemampuan yang benar.

- b. Atur cluster `id` ke "0x0202" (cluster Kontrol Kipas).

2. Mengatur informasi perintah:

- a. Atur pengidentifikasi perintah ke "0xff01" (Perbarui perintah Status yang ditentukan oleh AWS)
- b. Perbarui pengidentifikasi atribut yang disertakan dengan nilai yang disediakan dalam permintaan.

3. Perbarui atribut:

- a. Setel pengenalan atribut ke "0x0000" (FanMode atribut dari Fan Control Cluster).
- b. Atur nilai atribut ke "3" (Kecepatan kipas tinggi).

Integrasi terkelola telah mendefinisikan dua jenis perintah "kustom" yang tidak ditentukan secara ketat oleh AWS implementasi Model Data Materi: Perintah ReadState dan UpdateState . Untuk mendapatkan dan menyetel atribut cluster yang ditentukan Matter, integrasi terkelola akan mengirimkan AWS . SendCommand permintaan konektor Anda dengan perintah yang IDs berkaitan dengan UpdateState (id: 0xff01) atau ReadState (id: 0xff02), dengan parameter atribut yang sesuai yang harus diperbarui atau dibaca. Perintah ini dapat dipanggil untuk tipe perangkat APAPUN untuk atribut yang disetel sebagai bisa berubah (dapat diperbarui) atau dapat diambil (dapat dibaca) dari implementasi Model Data Matter yang sesuai AWS .

Mengirim peristiwa perangkat dengan SendConnectorEvent API

Ikhtisar peristiwa yang dimulai perangkat

Meskipun SendConnectorEvent API digunakan untuk merespons AWS . SendCommand dan AWS . DiscoverDevices operasi secara asinkron, API juga digunakan untuk memberi tahu integrasi terkelola dari setiap peristiwa yang dimulai perangkat. Peristiwa yang dimulai perangkat dapat didefinisikan sebagai peristiwa apa pun yang dihasilkan oleh perangkat tanpa perintah yang dimulai pengguna. Peristiwa perangkat ini mungkin termasuk, tetapi tidak terbatas pada, perubahan status perangkat, deteksi gerakan, tingkat baterai, dan banyak lagi. Anda dapat mengirim peristiwa ini kembali ke integrasi terkelola menggunakan SendConnectorEvent API dengan operasi DEVICE_EVENT.

Bagian berikut menggunakan kamera pintar yang dipasang di rumah sebagai contoh untuk menjelaskan lebih lanjut alur kerja dari peristiwa ini:

Alur kerja acara perangkat

1. Kamera Anda mendeteksi gerakan yang menghasilkan peristiwa yang dikirim ke server sumber daya Anda.
2. Server sumber daya Anda memproses acara dan mengirimkannya ke konektor C2C Anda.
3. Konektor Anda menerjemahkan peristiwa ini ke integrasi terkelola untuk antarmuka AWS IoT Device Management. DEVICE_EVENT

4. Konektor C2C Anda mengirimkan peristiwa perangkat ini ke integrasi terkelola menggunakan `SendConnectorEvent` API dengan Operasi disetel ke "DEVICE_EVENT".
5. Integrasi terkelola mengidentifikasi pelanggan yang relevan, dan menyampaikan acara ini kembali ke pelanggan.
6. Pelanggan menerima acara ini dan menampilkannya kepada pengguna melalui pengenalan pengguna.

Untuk informasi selengkapnya tentang operasi `SendConnectorEvent` API, lihat `SendConnectorEvent` di integrasi terkelola untuk Panduan Referensi API AWS IoT Device Management.

Persyaratan acara yang dimulai perangkat

Berikut ini adalah beberapa persyaratan untuk acara yang dimulai perangkat.

- Sumber daya konektor C2C Anda harus dapat menerima peristiwa perangkat asinkron dari server sumber daya Anda
- Sumber daya konektor C2C Anda harus dapat memanggil integrasi terkelola untuk API layanan AWS IoT Device Management melalui SigV4 AWS menggunakan kredensial yang digunakan untuk mendaftarkan konektor C2C. Akun AWS

Contoh berikut menunjukkan konektor yang mengirimkan peristiwa yang berasal dari perangkat melalui API: `SendConnectorEvent`

```
AWS-API: /SendConnectorEvent
URI: POST /connector-event/{Your-Connector-Id}

{
  "UserId": "Your-End-User-ID",
  "Operation": "DEVICE_EVENT",
  "OperationVersion": "1.0",
  "StatusCode": 200,
  "Message": None,
  "ConnectorDeviceId": "Your_Device_Id",
  "MatterEndpoint": {
    "id": "1",
    "clusters": [{
      "id": "0x0202",
      "attributes": [
```

```
    {
      "0x0000": "3"
    }
  ]
}]
}
```

Dari contoh berikut, kita melihat yang berikut:

- Ini berasal dari titik akhir perangkat dengan ID sama dengan 1.
- Kemampuan perangkat yang terkait dengan acara ini memiliki ID cluster 0x0202, yang berkaitan dengan cluster materi Kontrol Kipas.
- Atribut yang telah berubah memiliki ID 0x000, yang berkaitan dengan Fan Mode Enum dalam cluster. Ini telah diperbarui ke nilai 3, berkaitan dengan nilai Tinggi.
- Karena `connectorId` merupakan parameter yang dikembalikan oleh layanan cloud saat pembuatan, Konektor harus melakukan kueri menggunakan `GetCloudConnector` dan memfilter berdasarkan `lambdaARN`. Lambda sendiri ARN ditanyakan menggunakan `Lambda.get_function_url_config` API. Hal ini memungkinkan `CloudConnectorId` untuk diakses secara dinamis di lambda, dan tidak dikonfigurasi secara statis seperti sebelumnya.

Menerapkan AWS. DeactivateUser operasi

Ikhtisar penonaktifan pengguna

Penonaktifan token akses pengguna yang disediakan diperlukan ketika pelanggan menghapus akun AWS pelanggan mereka; atau ketika pengguna akhir ingin memutuskan tautan akun mereka di sistem dari sistem AWS pelanggan. Dalam kedua kasus penggunaan integrasi terkelola perlu memfasilitasi alur kerja ini menggunakan konektor C2C.

Gambar di bawah ini menggambarkan delinking akun pengguna akhir dari sistem

Alur kerja penonaktifan pengguna

1. Pengguna memulai proses delinking antara akun AWS pelanggan dan server otorisasi pihak ketiga yang terkait dengan konektor C2C.
2. Pelanggan memulai penghapusan asosiasi pengguna melalui integrasi terkelola untuk AWS IoT Device Management.

3. Integrasi terkelola memulai proses penonaktifan melalui permintaan ke konektor Anda menggunakan antarmuka Operasi. `AWS.DeactivateUser`
 - Token akses/user disertakan dalam header permintaan.
4. Konektor C2C Anda menerima permintaan dan memanggil server otorisasi Anda untuk mencabut token dan akses apa pun yang disediakan.
 - Misalnya, peristiwa dari akun pengguna yang tidak ditautkan seharusnya tidak lagi dikirim ke integrasi terkelola setelah dilakukan. `AWS.DeactivateUser`
5. Server otorisasi Anda mencabut akses dan mengirimkan respons kembali ke konektor C2C Anda.
6. Konektor C2C Anda mengirimkan integrasi terkelola untuk AWS IoT Device Management ACK yang token akses pengguna telah dicabut.
7. Integrasi terkelola menghapus semua sumber daya yang dimiliki oleh pengguna akhir yang terkait dengan server sumber daya Anda.
8. Integrasi terkelola mengirimkan ACK ke pelanggan, yang menyatakan semua asosiasi yang berkaitan dengan sistem Anda dihapus.
9. Pelanggan memberi tahu pengguna akhir bahwa akun mereka telah dibatalkan tautannya dari platform Anda.

AWS. DeactivateUser persyaratan

- Fungsi Lambda konektor C2C menerima pesan permintaan dari integrasi terkelola untuk menangani operasi. `AWS.DeactivateUser`
- Konektor C2C harus mencabut token OAuth2.0 yang disediakan dan token penyegaran yang sesuai dari pengguna dalam server otorisasi Anda.

Berikut ini adalah contoh `AWS.DeactivateUser` permintaan yang akan diterima konektor Anda:

```
{
  "header": {
    "auth": {
      "token": "ashriu32yr97feqy7afsaf",
      "type": "OAuth2.0"
    }
  }
}
```

```
  },
  "payload":{
    "operationName": "AWS.DeactivateUser"
    "operationVersion": "1.0"
    "connectorId": "Your-connector-Id"
  }
}
```

Panggil konektor C2C Anda

AWS Lambda memungkinkan kebijakan berbasis sumber daya untuk mengotorisasi siapa yang dapat memanggil Lambda. Karena integrasi terkelola untuk AWS IoT Device Management Layanan AWS adalah, Anda harus mengizinkan integrasi terkelola untuk memanggil konektor C2C Lambda melalui kebijakan sumber daya.

Lampirkan kebijakan sumber daya dengan setidaknya izin minimal berikut ke Lambda konektor C2C Anda. Ini menyediakan integrasi terkelola dengan hak istimewa pemanggilan fungsi Lambda. Kebijakan ini mencakup Condition kunci untuk membantu Anda membatasi kegunaan Anda hanya `connectorId` untuk pengguna yang dituju.

JSON

```
{
  "Version":"2012-10-17",
  "Id": "default",
  "Statement": [
    {
      "Sid": "Your-Desired-Policy-ID",
      "Effect": "Allow",
      "Principal": {
        "Service": "iotmanagedintegrations.amazonaws.com"
      },
      "Action": "lambda:InvokeFunction",
      "Resource": "arn:aws:lambda:ca-central-1:444455556666:function:connector-  
lambda-name",
      "Condition": {
        "StringEquals": {
          "aws:SourceArn": "arn:aws:iotmanagedintegrations:ca-  
central-1:444455556666:account-association/account-association-id"
        }
      }
    }
  ]
}
```

```
    }  
  ]  
}
```

Tambahkan izin ke Peran IAM Anda

Semua integrasi terkelola APIs memerlukan otentikasi AWS SiGv4 untuk dipanggil. SiGv4 menandatangani protokol untuk mengautentikasi permintaan AWS API menggunakan kredensial Anda. Akun AWS Peran IAM yang Anda gunakan untuk memanggil integrasi terkelola APIs harus memiliki izin berikut agar dapat berhasil memanggil: APIs

```
"Version": "2012-10-17",  
"Statement": [  
  {  
    "Sid": "Statement1",  
    "Effect": "Allow",  
    "Action": [  
      "iotmanagedintegrations:Your-Required-Actions"  
    ],  
    "Resource": [  
      "Your-Resource"  
    ]  
  }  
]
```

Untuk informasi tambahan tentang menambahkan izin ini, hubungi Dukungan.

Sumber daya tambahan

Untuk mendaftarkan konektor C2C Anda, Anda memerlukan yang berikut:

- Lambda ARN menunjuk konektor yang ingin Anda daftarkan.

Uji konektor C2C Anda secara manual

Untuk menguji konektor C2C Anda secara manual end-to-end, Anda harus mensimulasikan pelanggan dan pengguna akhir.

Anda akan membutuhkan sumber daya berikut:

- AWS Lambda ARN yang menunjuk konektor yang ingin Anda uji.
- Menguji akun pengguna OAuth 2.0 dari platform cloud Anda.
- Konektor yang terdaftar dengan integrasi terkelola untuk AWS IoT Device Management. Untuk informasi selengkapnya, lihat [Gunakan konektor C2C \(Cloud-to-Cloud\)](#).

Gunakan konektor C2C (Cloud-to-Cloud)

Konektor C2C mengelola terjemahan pesan permintaan dan respons, dan memungkinkan komunikasi antara integrasi terkelola dan cloud vendor pihak ketiga. Ini memfasilitasi kontrol terpadu di berbagai jenis perangkat, platform, dan protokol yang memungkinkan perangkat pihak ketiga untuk di-onboard dan dikelola.

Prosedur berikut mencantumkan langkah-langkah untuk menggunakan konektor C2C.

Langkah-langkah untuk menggunakan konektor C2C:

1. CreateCloudConnector

Konfigurasi konektor untuk mengaktifkan komunikasi dua arah antara integrasi terkelola Anda dan cloud vendor pihak ketiga.

Saat mengatur konektor, berikan detail berikut:

- Nama: Pilih nama deskriptif untuk konektor.
- Deskripsi: Berikan ringkasan singkat tentang tujuan dan kemampuan konektor.
- AWS Lambda ARN: Tentukan Nama Sumber Daya Amazon (ARN) dari AWS Lambda fungsi yang akan memberi daya pada konektor.

Buat dan terapkan AWS Lambda fungsi yang berkomunikasi dengan vendor pihak ketiga APIs untuk membuat konektor. Selanjutnya, panggil [CreateCloudConnector](#) API dalam integrasi terkelola, dan berikan AWS Lambda fungsi ARN untuk pendaftaran. Pastikan bahwa AWS Lambda fungsi tersebut diterapkan di AWS akun yang sama di mana Anda memasang konektor dalam integrasi terkelola. Anda akan diberi ID Konektor unik untuk mengidentifikasi integrasi.

Contoh Permintaan dan Respons CreateCloudConnector API:

Request:

```
{
  "Name": "CreateCloudConnector",
  "Description": "Testing for C2C",
  "EndpointType": "LAMBDA",
  "EndpointConfig": {
    "lambda": {
      "arn": "arn:aws:lambda:us-east-1:xxxxxx:function:TestingConnector"
    }
  },
  "ClientToken": "abc"
}
```

Response:

```
{
  "Id": "string"
}
```

Aliran penciptaan:

Note

Gunakan [GetCloudConnector](#), [UpdateCloudConnector](#), [DeleteCloudConnector](#), dan sesuai [ListCloudConnectors](#) APIs kebutuhan untuk prosedur ini.

2. CreateConnectorDestination

Konfigurasi Destinasinya untuk menyediakan pengaturan dan kredensial otentikasi yang diperlukan konektor untuk membuat koneksi aman dengan cloud vendor pihak ketiga. Gunakan Destinasinya untuk mendaftarkan kredensial autentikasi pihak ketiga Anda dengan integrasi terkelola, seperti detail otorisasi OAuth 2.0 termasuk URL otorisasi, skema otentikasi, dan lokasi kredensial di dalamnya. AWS Secrets Manager

Prasyarat

Sebelum membuat ConnectorDestination, Anda harus:

- Panggil [CreateCloudConnector](#) API untuk membuat konektor. ID yang dikembalikan fungsi digunakan dalam panggilan [CreateConnectorDestination](#) API API.
- Ambil tokenUrl untuk platform 3P konektor. (Anda dapat menukar AuthCode dengan AccessToken).
- Ambil AuthURL untuk platform 3P konektor. (Pengguna akhir dapat mengautentikasi menggunakan nama pengguna dan kata sandi mereka).
- Gunakan clientId dan clientSecret (dari platform 3P) di manajer rahasia akun Anda.

Contoh Permintaan dan Respons CreateConnectorDestination API:

Request:

```
{
  "Name": "CreateConnectorDestination",
  "Description": "CreateConnectorDestination",
  "AuthType": "OAUTH",
  "AuthConfig": {
    "oAuth": {
      "authUrl": "https://xxxx.com/oauth2/authorize",
      "tokenUrl": "https://xxxx/oauth2/token",
      "scope": "testScope",
      "tokenEndpointAuthenticationScheme": "HTTP_BASIC",
      "oAuthCompleteRedirectUrl": "about:blank",
      "proactiveRefreshTokenRenewal": {
        "enabled": false,
        "DaysBeforeRenewal": 30
      }
    }
  },
  "CloudConnectorId": "<connectorId>", // The connectorID instance from response
  of Step 1.
  "SecretsManager": {
    "arn": "arn:aws:secretsmanager:*****:secret:*****",
    "versionId": "*****"
  },
  "ClientToken": "****"
}
```

Response:

```
{
  "Id": "string"
}
```

Alur pembuatan tujuan cloud:

Note

Gunakan [GetCloudConnector](#), [UpdateCloudConnectorDeleteCloudConnector](#), dan sesuai [ListCloudConnectors](#) APIs kebutuhan untuk prosedur ini.

3. CreateAccountAssociation

Asosiasi mewakili hubungan antara akun cloud pihak ketiga pengguna akhir dan tujuan konektor. Setelah membuat Asosiasi dan menautkan pengguna akhir ke integrasi terkelola, perangkat mereka dapat diakses melalui ID Asosiasi yang unik. Integrasi ini memungkinkan tiga fungsi utama: menemukan perangkat, mengirim perintah, dan menerima acara.

Prasyarat

Sebelum membuat, AccountAssociationAnda harus menyelesaikan yang berikut:

- Panggil [CreateConnectorDestination](#) API untuk membuat tujuan. ID yang dikembalikan fungsi digunakan dalam panggilan [CreateAccountAssociation](#) API.
- Memanggil [CreateAccountAssociation](#) API.

Contoh Permintaan dan Respons CreateAccountAssociation API:

Request:

```
{
  "Name": "CreateAccountAssociation",
  "Description": "CreateAccountAssociation",
  "ConnectorDestinationId": "<destinationId>", //The destinationID from
  destination creation.
  "ClientToken": "****"
}
```

Response:

```
{
  "Id": "string"
}
```

Note

Gunakan [GetCloudConnector](#), [UpdateCloudConnectorDeleteCloudConnector](#), dan sesuai [ListCloudConnectors](#) APIs kebutuhan untuk prosedur ini.

An AccountAssociation memiliki status yang ditanyakan dari [GetAccountAssociation](#) dan [ListAccountAssociations](#) APIs. Ini APIs menunjukkan keadaan Asosiasi.

[StartAccountAssociationRefresh](#) API memungkinkan penyegaran AccountAssociation status saat token penyegarannya kedaluwarsa.

4. Penemuan perangkat

Setiap hal yang dikelola ditautkan ke detail khusus perangkat, seperti nomor seri dan model data. Model data menjelaskan fungsionalitas perangkat, menunjukkan apakah itu bola lampu, sakelar, termostat, atau jenis perangkat lain. Untuk menemukan perangkat 3P dan membuat ManagedThing untuk perangkat 3P, Anda harus mengikuti langkah-langkah di bawah ini secara berurutan.

- a. Panggil [StartDeviceDiscovery](#) API untuk memulai proses penemuan perangkat.

Contoh Permintaan dan Respons StartDeviceDiscovery API:

Request:

```
{
  "DiscoveryType": "CLOUD",
  "AccountAssociationId": "*****",
  "ClientToken": "abc"
}
```

Response:

```
{
  "Id": "string",
  "StartedAt": number
}
```

```
}
```

- b. Memanggil [GetDeviceDiscovery](#) API untuk memeriksa status proses penemuan.
- c. Panggil [ListDiscoveredDevices](#) API untuk membuat daftar perangkat yang ditemukan.

Contoh Permintaan dan Respons ListDiscoveredDevices API:

Request:

```
//Empty body
```

Response:

```
{
  "Items": [
    {
      "Brand": "string",
      "ConnectorDeviceId": "string",
      "ConnectorDeviceName": "string",
      "DeviceTypes": [ "string" ],
      "DiscoveredAt": number,
      "ManagedThingId": "string",
      "Model": "string",
      "Modification": "string"
    }
  ],
  "NextToken": "string"
}
```

- d. Memanggil [CreateManagedThing](#) API untuk memilih perangkat dari daftar penemuan yang akan diimpor ke integrasi terkelola.

Contoh Permintaan dan Respons CreateManagedThing API:

Request:

```
{
  "Role": "DEVICE",
  "AuthenticationMaterial": "CLOUD:XXXX:<connectorDeviceId1>",
  "AuthenticationMaterialType": "DISCOVERED_DEVICE",
  "Name": "sample-device-name"
  "ClientToken": "xxx"
}
```

Response:

```
{
  "Arn": "string", // This is the ARN of the managedThing
  "CreatedAt": number,
  "Id": "string"
}
```

- e. Panggil [GetManagedThing](#) API untuk melihat ini yang baru dibuat managedThing. Statusnya akan UNASSOCIATED.
- f. Panggil [RegisterAccountAssociation](#) API untuk mengaitkan ini managedThing dengan yang spesifik accountAssociation. Di akhir [RegisterAccountAssociation](#) API yang berhasil, managedThing perubahan ASSOCIATED status.

Contoh Permintaan dan Respons RegisterAccountAssociation API:

Request:

```
{
  "AccountAssociationId": "string",
  "DeviceDiscoveryId": "string",
  "ManagedThingId": "string"
}
```

Response:

```
{
  "AccountAssociationId": "string",
  "DeviceDiscoveryId": "string",
  "ManagedThingId": "string"
}
```

5. Kirim perintah ke perangkat 3P

Untuk mengontrol perangkat yang baru di-onboard, gunakan [SendManagedThingCommand](#) API, dengan ID Asosiasi yang dibuat sebelumnya dan tindakan kontrol berdasarkan kemampuan yang didukung oleh perangkat. Konektor menggunakan kredensial tersimpan dari proses penautan akun, untuk mengautentikasi dengan cloud pihak ketiga dan memanggil panggilan API yang relevan untuk operasi tersebut.

Contoh Permintaan dan Respons SendManagedThingCommand API:

Request:

```
{
  "AccountAssociationId": "string",
  "ConnectorAssociationId": "string",
  "Endpoints": [
    {
      "capabilities": [
        {
          "actions": [
            {
              "actionTraceId": "string",
              "name": "string",
              "parameters": JSON value,
              "ref": "string"
            }
          ],
          "id": "string",
          "name": "string",
          "version": "string"
        }
      ],
      "endpointId": "string"
    }
  ]
}
```

Response:

```
{
  "TraceId": "string"
}
```

Kirim perintah ke aliran perangkat 3P:

6. Konektor mengirimkan acara ke integrasi terkelola

[SendConnectorEvent](#) API menangkap empat jenis peristiwa dari konektor ke integrasi terkelola, yang diwakili oleh nilai enum berikut untuk parameter Jenis Operasi:

- **DEVICE_COMMAND_RESPONSE**: Respons asinkron yang dikirim konektor sebagai respons terhadap perintah.
- **DEVICE_DISCOVERY**: Menanggapi proses penemuan perangkat, konektor mengirimkan daftar perangkat yang ditemukan ke integrasi terkelola, ia menggunakan API [SendConnectorEvent](#)
- **DEVICE_EVENT**: Mengirim peristiwa perangkat yang diterima.
- **DEVICE_COMMAND_REQUEST**: Permintaan perintah dimulai dari perangkat. Misalnya, alur kerja WebRTC.

Konektor juga dapat meneruskan peristiwa perangkat menggunakan [SendConnectorEvent](#) API, dengan `userId` parameter opsional.

- Untuk acara perangkat dengan `userId`:

Contoh Permintaan dan Respons `SendConnectorEvent` API:

Request:

```
{
  "UserId": "*****",
  "Operation": "DEVICE_EVENT",
  "OperationVersion": "1.0",
  "StatusCode": 200,
  "ConnectorId": "*****",
  "ConnectorDeviceId": "****",
  "TraceId": "****",
  "MatterEndpoint": {
    "id": "***",
    "clusters": [{
      .....
    }]
  }
}
```

Response:

```
{
  "ConnectorId": "string"
```

```
}
```

- Untuk acara perangkat tanpa `pauseId`:

Contoh Permintaan dan Respons `SendConnectorEvent` API:

Request:

```
{
  "Operation": "DEVICE_EVENT",
  "OperationVersion": "1.0",
  "StatusCode": 200,
  "ConnectorId": "*****",
  "ConnectorDeviceId": "*****",
  "TraceId": "*****",
  "MatterEndpoint": {
    "id": "***",
    "clusters": [{
      ....
    }]
  }
}
```

Response:

```
{
  "ConnectorId": "string"
}
```

Untuk menghapus tautan antara asosiasi tertentu `managedThing` dan akun, gunakan mekanisme `deregister`:

Contoh Permintaan dan Respons `DeregisterAccountAssociation` API:

Request:

```
{
  "AccountAssociationId": "*****",
  "ManagedThingId": "*****"
}
```

Response:

```
HTTP/1.1 200 // Empty body
```

Kirim alur acara:

7. Perbarui status konektor ke “Terdaftar” agar terlihat oleh pelanggan integrasi terkelola lainnya

Secara default, konektor bersifat pribadi dan hanya terlihat oleh AWS akun yang membuatnya. Anda dapat memilih untuk membuat konektor terlihat oleh pelanggan integrasi terkelola lainnya.

Untuk membagikan konektor Anda dengan pengguna lain, gunakan opsi Make visible Konsol Manajemen AWS di halaman detail konektor untuk mengirimkan ID konektor Anda AWS untuk ditinjau. Setelah disetujui, konektor tersedia untuk semua pengguna integrasi terkelola dalam hal yang sama Wilayah AWS. Selain itu, Anda dapat membatasi akses ke AWS akun tertentu IDs dengan memodifikasi kebijakan akses pada fungsi terkait AWS Lambda konektor. Untuk memastikan konektor Anda dapat digunakan oleh pelanggan lain, kelola izin akses IAM pada fungsi Lambda Anda dari AWS akun lain ke konektor yang terlihat.

Tinjau Layanan AWS persyaratan dan kebijakan organisasi Anda yang mengatur berbagi konektor dan izin akses sebelum membuat konektor terlihat oleh pelanggan integrasi terkelola lainnya.

Integrasi terkelola Hub SDK

Gunakan topik di bagian ini untuk mempelajari cara onboard dan mengontrol perangkat hub IoT menggunakan integrasi terkelola Hub SDK. Untuk informasi selengkapnya tentang integrasi terkelola Akhiri SDK perangkat, lihat. [Integrasi terkelola Akhiri perangkat SDK](#)

Arsitektur SDK Hub

Orientasi perangkat

Tinjau bagaimana komponen SDK Hub mendukung orientasi perangkat sebelum Anda mulai bekerja dengan integrasi terkelola. Bagian ini mencakup komponen arsitektur penting yang Anda butuhkan untuk orientasi perangkat, termasuk bagaimana penyedia inti dan plugin khusus protokol bekerja sama untuk menangani otentikasi perangkat, komunikasi, dan pengaturan pengguna.

Komponen SDK Hub untuk onboarding perangkat

Komponen SDK

- [Penyedia inti](#)
- [Plugin penyedia khusus protokol](#)
- [Middleware khusus protokol](#)

Penyedia inti

Penyedia inti adalah komponen utama yang mengatur orientasi perangkat dalam penerapan hub IoT Anda. Ini mengoordinasikan semua komunikasi antara integrasi terkelola dan plugin penyedia khusus protokol Anda, memastikan orientasi perangkat yang aman dan andal. Saat Anda menggunakan perangkat, penyedia inti menangani alur autentikasi, mengelola pesan MQTT, dan memproses permintaan perangkat melalui fungsi-fungsi ini:

Koneksi MQTT

Membuat koneksi dengan broker MQTT untuk penerbitan dan berlangganan topik cloud.

Antrian pesan dan handler

Memproses masuk menambah dan menghapus permintaan perangkat secara berurutan.

Antarmuka plugin protokol

Bekerja dengan plugin penyedia khusus protokol untuk orientasi perangkat dengan mengelola otentikasi dan mode penggabungan radio.

Klien Hub SDK APIs

Menerima dan meneruskan laporan kemampuan perangkat dari plugin CDMB khusus protokol ke integrasi terkelola.

Plugin penyedia khusus protokol

Plugin penyedia khusus protokol adalah pustaka yang mengelola orientasi perangkat untuk protokol komunikasi yang berbeda. Setiap plugin menerjemahkan perintah dari penyedia inti menjadi tindakan khusus protokol untuk perangkat IoT Anda. Plugin ini melakukan:

- Inisialisasi middleware khusus protokol
- Konfigurasi mode penggabungan radio berdasarkan permintaan penyedia inti
- Penghapusan perangkat melalui panggilan API middleware

Middleware khusus protokol

Middleware khusus protokol bertindak sebagai lapisan terjemahan antara protokol perangkat Anda dan integrasi terkelola. Komponen ini memproses komunikasi di kedua arah—menerima perintah dari plugin penyedia dan mengirimkannya ke tumpukan protokol, sementara juga mengumpulkan respons dari perangkat dan merutekan mereka kembali melalui sistem.

Alur orientasi perangkat

Tinjau urutan operasi yang terjadi saat Anda melakukan onboard perangkat menggunakan Hub SDK. Bagian ini menampilkan bagaimana komponen berinteraksi selama proses orientasi dan menguraikan metode orientasi yang didukung.

Arus orientasi

- [Pengaturan sederhana \(SS\)](#)
- [Pengaturan tanpa sentuhan \(ZTS\)](#)
- [Pengaturan yang dipandu pengguna \(UGS\)](#)

Pengaturan sederhana (SS)

Pengguna akhir memberi daya pada perangkat IoT dan memindai kode QR-nya menggunakan aplikasi produsen perangkat. Perangkat kemudian terdaftar ke cloud integrasi terkelola dan terhubung ke hub IoT.

Pengaturan tanpa sentuhan (ZTS)

Zero-touch setup (ZTS) merampingkan orientasi perangkat dengan melakukan pra-asosiasi perangkat di hulu dalam rantai pasokan. Misalnya, alih-alih pengguna akhir memindai kode QR perangkat, langkah ini diselesaikan lebih awal untuk menautkan perangkat ke akun pelanggan. Misalnya, langkah ini dapat diselesaikan di pusat pemenuhan.

Ketika pengguna akhir menerima dan menyalakan perangkat, ia secara otomatis mendaftarkan di cloud integrasi terkelola dan terhubung ke hub IoT tanpa memerlukan tindakan pengaturan tambahan apa pun.

Pengaturan yang dipandu pengguna (UGS)

Pengguna akhir memberi daya pada perangkat dan mengikuti langkah-langkah interaktif untuk memasukkannya ke integrasi terkelola. Ini mungkin termasuk menekan tombol pada hub IoT, menggunakan aplikasi produsen perangkat, atau menekan tombol pada hub dan perangkat. Anda dapat menggunakan metode ini jika Setup sederhana gagal.

Kontrol perangkat

Integrasi terkelola menangani pendaftaran perangkat, eksekusi perintah, dan kontrol. Anda dapat membangun pengalaman pengguna akhir tanpa mengetahui protokol khusus perangkat menggunakan vendor dan manajemen perangkat agnostiknya.

Dengan kontrol perangkat, Anda dapat melihat dan memodifikasi status perangkat, seperti kecerahan bola lampu atau posisi pintu. Fitur ini memancarkan peristiwa untuk perubahan status, yang dapat Anda gunakan untuk analitik, aturan, dan pemantauan.

Fitur utama

Ubah atau baca status perangkat

Melihat dan mengubah atribut perangkat berdasarkan jenis perangkat. Anda dapat mengakses:

- Status perangkat: Nilai atribut perangkat saat ini
- Status konektivitas: Status jangkauan perangkat
- Status Kesehatan: Nilai sistem seperti level baterai dan kekuatan sinyal (RSSI)

Pemberitahuan perubahan status

Terima peristiwa saat atribut perangkat atau status konektivitas berubah, seperti penyesuaian kecerahan bola lampu atau perubahan status kunci pintu.

Mode offline

Perangkat berkomunikasi dengan perangkat lain di hub IoT yang sama bahkan tanpa konektivitas internet. Status perangkat disinkronkan dengan cloud saat konektivitas dilanjutkan.

Sinkronisasi negara

Lacak perubahan status dari berbagai sumber, aplikasi produsen perangkat, dan penyesuaian perangkat manual.

Tinjau komponen dan proses Hub SDK yang Anda perlukan untuk mengontrol perangkat melalui integrasi terkelola. Topik ini menjelaskan bagaimana Edge Agent, Common Data Model Bridge (CDMB), dan plugin khusus protokol bekerja sama untuk menangani perintah perangkat, mengelola status perangkat, dan memproses respons di berbagai protokol.

Aliran kontrol perangkat

Diagram berikut menunjukkan aliran kontrol end-to-end perangkat dengan menjelaskan bagaimana pengguna akhir menyalakan steker pintar Zigbee.

Komponen SDK Hub untuk kontrol perangkat

Arsitektur Hub SDK menggunakan komponen berikut untuk memproses dan merutekan perintah kontrol perangkat dalam implementasi IoT Anda. Setiap komponen memainkan peran khusus dalam menerjemahkan perintah cloud ke dalam tindakan perangkat, mengelola status perangkat, dan menangani respons. Bagian berikut merinci cara komponen ini bekerja sama dalam penerapan Anda:

Hub SDK terdiri dari komponen-komponen berikut, dan memfasilitasi orientasi dan kontrol perangkat pada hub IoT.

Komponen utama:

Agen Edge

Bertindak sebagai gateway antara hub IoT dan integrasi terkelola.

Jembatan model data umum (CDMB)

Menerjemahkan antara model AWS data dan model data protokol lokal seperti Z-Wave dan Zigbee. Ini termasuk CDMB inti dan plugin CDMB khusus protokol.

Penyedia

Menangani penemuan dan orientasi perangkat. Ini mencakup penyedia inti dan plugin penyedia khusus protokol untuk tugas orientasi khusus protokol.

Komponen sekunder

Orientasi hub

Menyediakan hub dengan sertifikat klien dan kunci untuk komunikasi cloud yang aman.

Proksi MQTT

Menyediakan koneksi MQTT ke cloud integrasi terkelola.

Pencatat

Menulis log secara lokal atau ke cloud integrasi terkelola.

Instal dan validasi integrasi terkelola Hub SDK

Pilih di antara metode penerapan berikut untuk menginstal integrasi terkelola Hub SDK di perangkat Anda—AWS IoT Greengrass untuk penerapan otomatis atau penginstalan skrip manual. Bagian ini menjelaskan langkah-langkah penyiapan dan validasi untuk kedua pendekatan.

Metode deployment

- [Instal Hub SDK dengan AWS IoT Greengrass](#)
- [Menerapkan Hub SDK dengan skrip](#)
- [Menyebarkan Hub SDK dengan systemd](#)

Instal Hub SDK dengan AWS IoT Greengrass

Menerapkan integrasi terkelola komponen Hub SDK untuk perangkat Anda menggunakan AWS IoT Greengrass (Versi Java).

Note

Anda harus sudah mengatur dan memiliki pemahaman tentang AWS IoT Greengrass. Untuk informasi selengkapnya, lihat [Apa yang ada AWS IoT Greengrass](#) di dokumentasi panduan AWS IoT Greengrass pengembang.

AWS IoT Greengrass Pengguna harus memiliki izin untuk memodifikasi direktori berikut:

- /dev/aipc
- /data/aws/iotmi/config
- /data/ace/kvstorage

Topik

- [Menyebarkan komponen secara lokal](#)
- [Penyebaran cloud](#)
- [Verifikasi penyediaan hub](#)
- [Verifikasi operasi CDMB](#)
- [Verifikasi operasi penyedia LPW](#)

Menyebarkan komponen secara lokal

Gunakan [CreateDeployment](#) AWS IoT Greengrass API di perangkat Anda untuk menerapkan komponen Hub SDK. Nomor versi tidak statis dan dapat bervariasi berdasarkan versi yang Anda gunakan saat itu. Gunakan format berikut untuk **version**: com.amazon.io.TManagedIntegrationsDeviceAceCommon=**version**.0.

```
/greengrass/v2/bin/greengrass-cli deployment create \  
--recipeDir recipes \  
--artifactDir artifacts \  
-m "com.amazon.IoTManagedIntegrationsDevice.AceCommon=version" \  

```

```
-m "com.amazon.IoTManagedIntegrationsDevice.HubOnboarding=version" \  
-m "com.amazon.IoTManagedIntegrationsDevice.AceZigbee=version" \  
-m "com.amazon.IoTManagedIntegrationsDevice.LPW-Provisioner=version" \  
-m "com.amazon.IoTManagedIntegrationsDevice.Agent=version" \  
-m "com.amazon.IoTManagedIntegrationsDevice.MQTTProxy=version" \  
-m "com.amazon.IoTManagedIntegrationsDevice.CDMB=version" \  
-m "com.amazon.IoTManagedIntegrationsDevice.AceZwave=version"
```

Penyebaran cloud

Ikuti petunjuk dalam [panduan AWS IoT Greengrass pengembang](#) untuk melakukan langkah-langkah berikut:

1. Unggah artefak ke Amazon S3.
2. Perbarui resep untuk menyertakan lokasi artefak Amazon S3.
3. Buat penyebaran cloud ke perangkat untuk komponen baru.

Verifikasi penyediaan hub

Konfirmasikan penyediaan yang berhasil dengan memeriksa file konfigurasi Anda. Buka `/data/aws/iotmi/config/iotmi_config.json` file dan verifikasi status diatur ke `PROVISIONED`.

Verifikasi operasi CDMB

Periksa file log untuk pesan startup CDMB dan inisialisasi yang berhasil. *logs file* Lokasi dapat bervariasi tergantung di mana AWS IoT Greengrass dipasang.

```
tail -f -n 100 /greengrass/v2/logs/com.amazon.IoTManagedIntegrationsDevice.CDMB.log
```

Contoh

```
[2024-09-06 02:31:54.413758906][IoTManagedIntegrationsDevice_CDMB][info] Successfully  
subscribed to topic: south/bf|gi_044F8821D0193608C8D5BF80858E20A56E3A8490/control  
[2024-09-06 02:31:54.513956059][IoTManagedIntegrationsDevice_CDMB][info] Successfully  
subscribed to topic: south/bf|gi_044F8821D0193608C8D5BF80858E20A56E3A8490/setup
```

Verifikasi operasi penyedia LPW

Periksa file log untuk pesan startup LPW-Provisioner dan inisialisasi yang berhasil. *logs file* Lokasi dapat bervariasi tergantung di mana AWS IoT Greengrass dipasang.

```
tail -f -n 100 /greengrass/v2/logs/com.amazon.IoTManagedIntegrationsDevice.LPW-
Provisioner.log
```

Contoh

```
[2024-09-06 02:33:22.068898877][LPWProvisionerCore][info] Successfully subscribed to
topic: south/bF|gi_044F8821D0193608C8D5BF80858E20A56E3A8490/setup
```

Menerapkan Hub SDK dengan skrip

Menerapkan integrasi terkelola komponen Hub SDK secara manual menggunakan skrip instalasi, lalu validasi penerapan. Bagian ini menjelaskan langkah-langkah eksekusi skrip dan proses verifikasi.

Topik

- [Persiapkan lingkungan Anda](#)
- [Jalankan skrip Hub SDK](#)
- [Verifikasi penyediaan hub](#)
- [Verifikasi operasi agen](#)
- [Verifikasi operasi penyedia LPW](#)

Persiapkan lingkungan Anda

Selesaikan langkah-langkah ini sebelum menjalankan skrip instalasi SDK:

1. Buat folder bernama `middleware` di dalam `artifacts` folder.
2. Salin file `middleware hub` Anda ke folder `middleware`
3. Jalankan perintah inisialisasi sebelum memulai SDK.

Important

Ulangi perintah inisialisasi setelah setiap hub reboot.

```
#Get the current user
_user=$(whoami)
```

```
#Get the current group
_grp=$(id -gn)

#Display the user and group
echo "Current User: $_user"
echo "Current Group: $_grp"

sudo mkdir -p /dev/aipc/
sudo chown -R $_user:$_grp /dev/aipc
sudo mkdir -p /data/ace/kvstorage
sudo chown -R $_user:$_grp /data/ace/kvstorage
```

Jalankan skrip Hub SDK

Arahkan ke direktori artefak dan jalankan `start_iotmi_sdk.sh` skrip. Skrip ini meluncurkan komponen SDK hub dalam urutan yang benar. Tinjau log contoh berikut untuk memverifikasi startup yang berhasil:

Note

Log untuk semua komponen yang berjalan dapat ditemukan di dalam `artifacts/logs` folder.

```
hub@hub-293ea release_Oct_17$ ./start_iotmi_sdk.sh
-----Stopping SDK running processes---
DeviceAgent: no process found
-----Starting SDK-----
-----Creating logs directory-----
Logs directory created.
-----Verifying Middleware paths-----
All middleware libraries exist
-----Verifying Middleware pre reqs---
AIPC and KVstroage directories exist
-----Starting HubOnboarding-----
-----Starting MQTT Proxy-----
-----Starting Event Manager-----
-----Starting Zigbee Service-----
-----Starting Zwave Service-----
/data/release_Oct_17/middleware/AceZwave/bin /data/release_Oct_17
/data/release_Oct_17
```

```

-----Starting CDMB-----
-----Starting Agent-----
-----Starting Provisioner-----
-----Checking SDK status-----
hub          6199  1.7  0.7 1004952 15568 pts/2    Sl+  21:41   0:00 ./iotmi_mqtt_proxy -
C /data/aws/iotmi/config/iotmi_config.json
Process 'iotmi_mqtt_proxy' is running.
hub          6225  0.0  0.1 301576  2056 pts/2    Sl+  21:41   0:00 ./middleware/
AceCommon/bin/ace_eventmgr
Process 'ace_eventmgr' is running.
hub          6234  104  0.2 238560  5036 pts/2    Sl+  21:41   0:38 ./middleware/
AceZigbee/bin/ace_zigbee_service
Process 'ace_zigbee_service' is running.
hub          6242  0.4  0.7 1569372 14236 pts/2    Sl+  21:41   0:00 ./zwave_svc
Process 'zwave_svc' is running.
hub          6275  0.0  0.2 1212744  5380 pts/2    Sl+  21:41   0:00 ./DeviceCdmdb
Process 'DeviceCdmdb' is running.
hub          6308  0.6  0.9 1076108 18204 pts/2    Sl+  21:41   0:00 ./
IoTManagedIntegrationsDeviceAgent
Process 'DeviceAgent' is running.
hub          6343  0.7  0.7 1388132 13812 pts/2    Sl+  21:42   0:00 ./
iotmi_lpw_provisioner
Process 'iotmi_lpw_provisioner' is running.
-----Successfully Started SDK-----

```

Verifikasi penyediaan hub

Periksa apakah `iot_provisioning_state` bidang di `/data/aws/iotmi/config/iotmi_config.json` diatur ke `PROVISIONED`.

Verifikasi operasi agen

Periksa file log untuk pesan startup agen dan inialisasi yang berhasil.

```
tail -f -n 100 logs/agent_logs.txt
```

Contoh

```
[2024-09-06 02:31:54.413758906][Device_Agent][info] Successfully subscribed to topic:
south/bF|gi_044F8821D0193608C8D5BF80858E20A56E3A8490/control
[2024-09-06 02:31:54.513956059][Device_Agent][info] Successfully subscribed to topic:
south/bF|gi_044F8821D0193608C8D5BF80858E20A56E3A8490/setup
```

Note

Periksa apakah `iotmi.db` database ada di `artifacts` direktori Anda.

Verifikasi operasi penyedia LPW

Periksa file log untuk pesan LPW-Provisioner startup dan inisialisasi yang berhasil.

```
tail -f -n 100 logs/provisioner_logs.txt
```

Kode berikut menunjukkan contoh.

```
[2024-09-06 02:33:22.068898877][LPWProvisionerCore][info] Successfully subscribed to  
topic: south/bF|gi_044F8821D0193608C8D5BF80858E20A56E3A8490/setup
```

Menyebarkan Hub SDK dengan systemd

Important

Ikuti `readme.md` di `hubSystemdSetup` direktori `file.tgz` rilis.tgz untuk pembaruan terbaru.

Bagian ini menjelaskan skrip dan proses untuk menerapkan dan mengonfigurasi layanan pada perangkat hub berbasis Linux.

Gambaran Umum

Proses penyebaran terdiri dari dua skrip utama:

- `copy_to_hub.sh`: Berjalan pada mesin host untuk menyalin file yang diperlukan ke hub
- `setup_hub.sh`: Berjalan di hub untuk mengonfigurasi lingkungan dan menyebarkan layanan

Selain itu, `systemd/deploy_iotshd_services_on_hub.sh` menangani urutan bootstrap proses dan manajemen izin proses, dan secara otomatis dipicu oleh `setup_hub.sh`.

Prasyarat

Prasyarat yang terdaftar diperlukan untuk penerapan yang berhasil.

- layanan systemd tersedia di hub
- Akses SSH ke perangkat hub
- Hak istimewa Sudo pada perangkat hub
- scp utilitas diinstal pada mesin host
- sed utilitas diinstal pada mesin host
- unzip utilitas diinstal pada mesin host

Struktur file

Struktur file dirancang untuk memfasilitasi organisasi dan manajemen berbagai komponennya, memungkinkan akses dan navigasi konten yang efisien.

```
hubSystemdSetup/  
### README.md  
### copy_to_hub.sh  
### setup_hub.sh  
### iotshd_config.json # Sample configuration file  
### local_certs/ # Directory for DHA certificates  
### systemd/  
    ### *.service.template # Systemd service templates  
    ### deploy_iotshd_services_on_hub.sh
```

Dalam file tgz rilis SDK, struktur file keseluruhan adalah:

```
IoT-managed-integrations-Hub-SDK-aarch64-v1.0.0.tgz  
###package/  
    ###greengrass/  
        ###artifacts/  
        ###recipes/  
    ###hubSystemdSetup/  
        ### REAME.md  
        ### copy_to_hub.sh  
        ### setup_hub.sh  
        ### iotshd_config.json # Sample configuration file  
        ### local_certs/ # Directory for DHA certificates  
        ### systemd/  
            ### *.service.template # Systemd service templates  
            ### deploy_iotshd_services_on_hub.sh
```

Pengaturan awal

Ekstrak paket SDK

```
tar -xzf managed-integrations-Hub-SDK-vVersion-linux-aarch64-timestamp.tgz
```

Arahkan ke direktori yang diekstrak dan siapkan paketnya:

```
# Create package.zip containing required artifacts
zip -r package.zip package/greengrass/artifacts
# Move package.zip to the hubSystemdSetup directory
mv package.zip ../hubSystemdSetup/
```

Tambahkan file konfigurasi perangkat

Ikuti dua langkah yang tercantum untuk membuat file konfigurasi perangkat, dan salin ke hub.

1. [Tambahkan file konfigurasi perangkat](#) untuk membuat file konfigurasi perangkat yang diperlukan. SDK menggunakan file ini untuk fungsinya.
2. [Salin file konfigurasi](#) untuk menyalin file konfigurasi yang dibuat ke hub.

Salin file ke hub

Jalankan skrip penerapan dari mesin host Anda:

```
chmod +x copy_to_hub.sh
./copy_to_hub.sh hub_ip_address package_file
```

Example Contoh

```
./copy_to_hub.sh 192.168.50.223 ~/Downloads/EAR3-package.zip
```

Salinan ini:

- File paket (berganti nama menjadi package.zip di hub)
- File konfigurasi
- Sertifikat
- File layanan Systemd

Siapkan hub

Setelah file disalin, SSH ke hub dan jalankan skrip pengaturan:

```
ssh hub@hub_ip
chmod +x setup_hub.sh
sudo ./setup_hub.sh
```

Konfigurasi pengguna dan grup

Secara default, kami menggunakan hub pengguna dan hub grup untuk komponen SDK. Ada beberapa cara untuk mengkonfigurasinya:

- Gunakan pengguna/grup khusus:

```
sudo ./setup_hub.sh --user=USERNAME --group=GROUPNAME
```

- Buat secara manual sebelum menjalankan skrip pengaturan:

```
sudo groupadd -f GROUPNAME
sudo useradd -r -g GROUPNAME USERNAME
```

- Tambahkan perintah di `setup_hub.sh`.

Kelola layanan

Untuk memulai ulang semua layanan, jalankan skrip berikut dari hub:

```
sudo /usr/local/bin/deploy_iotshd_services_on_hub.sh
```

Skrip pengaturan akan membuat direktori yang diperlukan, mengatur izin yang sesuai, dan menyebarkan layanan secara otomatis. Jika Anda tidak menggunakan SSH/SCP, Anda harus memodifikasi `copy_to_hub.sh` untuk metode penerapan spesifik Anda. Pastikan semua file sertifikat dan konfigurasi diatur dengan benar sebelum penerapan.

Onboard hub Anda ke integrasi terkelola

Siapkan perangkat hub Anda untuk berkomunikasi dengan integrasi terkelola dengan mengonfigurasi struktur direktori, sertifikat, dan file konfigurasi perangkat yang diperlukan. Bagian ini menjelaskan

cara komponen subsistem orientasi hub bekerja sama, tempat menyimpan sertifikat dan file konfigurasi, cara membuat dan memodifikasi file konfigurasi perangkat, dan langkah-langkah untuk menyelesaikan proses penyediaan hub.

Subsistem orientasi hub

Subsistem orientasi hub menggunakan komponen inti ini untuk mengelola penyediaan dan konfigurasi perangkat:

Komponen orientasi hub

Mengelola proses orientasi hub dengan mengoordinasikan status hub, pendekatan penyediaan, dan materi otentikasi.

File konfigurasi perangkat

Menyimpan data konfigurasi hub penting di perangkat, termasuk:

- Status penyediaan perangkat (disediakan atau tidak disediakan)
- Sertifikat dan lokasi utama
- Informasi otentikasi Proses SDK lainnya, seperti proxy MQTT, mereferensikan file ini untuk menentukan status hub dan pengaturan koneksi.

Antarmuka handler sertifikat

Menyediakan antarmuka utilitas untuk membaca dan menulis sertifikat dan kunci perangkat. Anda dapat menerapkan antarmuka ini untuk bekerja dengan:

- Penyimpanan sistem file
- Modul keamanan perangkat keras (HSM)
- Modul platform tepercaya (TPM)
- Solusi penyimpanan aman khusus

Komponen proxy MQTT

Mengelola device-to-cloud komunikasi menggunakan:

- Sertifikat dan kunci klien yang disediakan
- Informasi status perangkat dari file konfigurasi
- Koneksi MQTT ke integrasi terkelola

Diagram berikut menjelaskan arsitektur subsistem orientasi hub dan komponennya. Jika Anda tidak menggunakan AWS IoT Greengrass, Anda dapat mengabaikan komponen diagram itu.

Pengaturan orientasi hub

Selesaikan langkah-langkah pengaturan ini untuk setiap perangkat hub sebelum Anda memulai proses orientasi penyediaan armada. Bagian ini menjelaskan cara membuat hal-hal yang dikelola, mengatur struktur direktori, dan mengonfigurasi sertifikat yang diperlukan.

Langkah-langkah persiapan

- [Langkah 1: Daftarkan titik akhir kustom](#)
- [Langkah 2: Buat profil penyediaan](#)
- [Langkah 3: Buat hal yang dikelola \(penyediaan armada\)](#)
- [Langkah 4: Buat struktur direktori](#)
- [Langkah 5: Tambahkan bahan otentikasi ke perangkat hub](#)
- [Langkah 6: Buat file konfigurasi perangkat](#)
- [Langkah 7: Salin file konfigurasi ke hub Anda](#)

Langkah 1: Daftarkan titik akhir kustom

Buat titik akhir komunikasi khusus yang digunakan perangkat Anda untuk bertukar data dengan integrasi terkelola. Titik akhir ini menetapkan titik koneksi aman untuk semua device-to-cloud pesan, termasuk perintah perangkat, pembaruan status, dan pemberitahuan.

Untuk mendaftarkan titik akhir

- Gunakan [RegisterCustomEndpointAPI](#) untuk membuat titik akhir untuk komunikasi device-to-managed integrasi.

RegisterCustomEndpointMinta contoh

```
aws iot-managed-integrations register-custom-endpoint
```

Tanggapan:

```
{
```

```
[ACCOUNT-PREFIX]-ats.iot.AWS-REGION.amazonaws.com  
}
```

Note

Simpan alamat titik akhir. Anda akan membutuhkannya untuk komunikasi perangkat masa depan.

Untuk mengembalikan informasi titik akhir, gunakan `GetCustomEndpoint` API.

Untuk informasi selengkapnya, lihat [RegisterCustomEndpoint](#) API dan [GetCustomEndpoint](#) API di Panduan Referensi API integrasi terkelola.

Langkah 2: Buat profil penyediaan

Profil penyediaan berisi kredensi keamanan dan setelan konfigurasi yang dibutuhkan perangkat Anda untuk terhubung ke integrasi terkelola.

Untuk membuat profil penyediaan armada

- Panggil [CreateProvisioningProfile](#) API untuk menghasilkan yang berikut:
 - Template penyediaan yang mendefinisikan setelan koneksi perangkat
 - Sertifikat klaim dan kunci pribadi untuk otentikasi perangkat

Important

Simpan sertifikat klaim, kunci pribadi, dan ID templat dengan aman. Anda akan memerlukan kredensial ini ke perangkat onboard untuk integrasi terkelola. Jika Anda kehilangan kredensial ini, Anda harus membuat profil penyediaan baru.

CreateProvisioningProfile contoh permintaan

```
aws iot-managed-integrations create-provisioning-profile \  
  --provisioning-type FLEET_PROVISIONING \  
  --name PROFILE_NAME
```

Tanggapan:


```
{
  "Arn": "arn:aws:iotmanagedintegrations:AWS-REGION:ACCOUNT-ID:provisioning-
profile/PROFILE-ID",
  "ClaimCertificate":
  "-----BEGIN CERTIFICATE-----
MIICiTCCAfICCD6m7.....w3rrszlaEXAMPLE=
-----END CERTIFICATE-----",
  "ClaimCertificatePrivateKey":
  "-----BEGIN RSA PRIVATE KEY-----
MIICiTCCAfICQ...3rrszlaEXAMPLE=
-----END RSA PRIVATE KEY-----",
  "Id": "PROFILE-ID",
  "PROFILE-NAME",
  "ProvisioningType": "FLEET_PROVISIONING"
}
```

Langkah 3: Buat hal yang dikelola (penyediaan armada)

Gunakan `CreateManagedThing` API untuk membuat hal terkelola untuk perangkat hub Anda. Setiap hub membutuhkan hal yang dikelola sendiri dengan materi otentikasi yang unik. Untuk informasi selengkapnya, lihat [CreateManagedThing](#) API dalam Referensi API integrasi terkelola.

Saat Anda membuat hal yang dikelola, tentukan parameter ini:

- `Role`: Setel nilai ini ke `CONTROLLER` untuk hub yang tidak mendukung perintah dan kontrol, jika tidak diatur ke `DEVICE`.
- `AuthenticationMaterialType`: Tetapkan nilai ini ke `WIFI_SETUP_QR_BAR_CODE`.
- `AuthenticationMaterial`: Sertakan bidang-bidang berikut. Anda dapat menggunakan salah satu UPC atau EAN tetapi tidak keduanya.
 - `SN`: Nomor seri unik untuk perangkat ini
 - `UPC`: Kode produk universal untuk perangkat ini
 - `EAN`: Nomor artikel internasional untuk perangkat ini

 Important

Setiap perangkat harus memiliki nomor seri unik (SN) dalam materi otentikasi.

CreateManagedThingContoh permintaan:

```
{
  "Role": "CONTROLLER",
  "AuthenticationMaterialType": "WIFI_SETUP_QR_BAR_CODE",
  "AuthenticationMaterial": "SN:123456789524;UPC:829576019524"
}
```

Untuk informasi selengkapnya, lihat [CreateManagedThing](#) Referensi API integrasi terkelola.

(Opsional) Dapatkan hal yang dikelola

Hal `ProvisioningStatus` yang Anda kelola harus `PRE_ASSOCIATED` sebelum Anda dapat melanjutkan. Untuk informasi selengkapnya `ProvisioningStatus`, lihat [Penyediaan Perangkat](#). Gunakan `GetManagedThing` API untuk memverifikasi bahwa hal terkelola Anda ada dan siap untuk disediakan. Untuk informasi selengkapnya, lihat [GetManagedThing](#) Referensi API integrasi terkelola.

Langkah 4: Buat struktur direktori

Buat direktori untuk file konfigurasi dan sertifikat Anda. Secara default, proses orientasi hub menggunakan file `/data/aws/iotmi/config/iotmi_config.json`

Anda dapat menentukan jalur kustom untuk sertifikat dan kunci pribadi dalam file konfigurasi. Panduan ini menggunakan jalur default `/data/aws/iotmi/certs`.

```
mkdir -p /data/aws/iotmi/config
mkdir -p /data/aws/iotmi/certs
```

```
/data/
  aws/
    iotmi/
      config/
      certs/
```

Langkah 5: Tambahkan bahan otentikasi ke perangkat hub

Salin sertifikat dan kunci ke perangkat hub Anda, lalu buat file konfigurasi khusus perangkat. File-file ini membangun komunikasi yang aman antara hub Anda dan integrasi terkelola selama proses penyediaan.

Untuk menyalin sertifikat klaim dan kunci

- Salin file autentikasi ini dari respons `CreateProvisioningProfile` API Anda ke perangkat hub Anda:
 - `claim_cert.pem`: Sertifikat klaim (umum untuk semua perangkat)
 - `claim_pk.key`: Kunci pribadi untuk sertifikat klaim

Tempatkan kedua file di `/data/aws/iotmi/certs` direktori.

Important

Saat menyimpan sertifikat dan kunci pribadi dalam format PEM, pastikan pemformatan yang tepat dengan menangani karakter baris baru dengan benar. Untuk file yang dikodekan PEM, karakter baris baru (`\n`) harus diganti dengan pemisah baris yang sebenarnya, karena hanya menyimpan baris baru yang lolos tidak akan diambil dengan benar nanti.

Note

Jika Anda menggunakan penyimpanan aman, simpan kredensial ini di lokasi penyimpanan aman Anda alih-alih sistem file. Untuk informasi selengkapnya, lihat [Buat handler sertifikat khusus untuk penyimpanan aman](#).

Langkah 6: Buat file konfigurasi perangkat

Buat file konfigurasi yang berisi pengenalan perangkat unik, lokasi sertifikat, dan pengaturan penyediaan. SDK menggunakan file ini selama orientasi hub untuk mengautentikasi perangkat Anda, mengelola status penyediaan, dan menyimpan setelan koneksi.

Note

Setiap perangkat hub memerlukan file konfigurasinya sendiri dengan nilai khusus perangkat yang unik.

Gunakan prosedur berikut untuk membuat atau memodifikasi file konfigurasi Anda, dan salin ke hub.

- Membuat atau memodifikasi file konfigurasi (penyediaan armada).

Konfigurasi bidang wajib ini dalam file konfigurasi perangkat:

- Jalur sertifikat

1. `iot_claim_cert_path`: Lokasi sertifikat klaim Anda (`claim_cert.pem`)
2. `iot_claim_pk_path`: Lokasi kunci pribadi Anda (`claim_pk.key`)
3. Gunakan `SECURE_STORAGE` untuk kedua bidang saat menerapkan Secure Storage Cert Handler

- Pengaturan koneksi

1. `fp_template_name`: ProvisioningProfile Nama dari sebelumnya.
2. `endpoint_url`: URL titik akhir integrasi terkelola Anda dari respons RegisterCustomEndpoint API (sama untuk semua perangkat di Wilayah).

- Pengidentifikasi perangkat

1. SN: Nomor seri perangkat yang cocok dengan panggilan CreateManagedThing API Anda (unik per perangkat)
2. UPC Kode produk universal dari panggilan CreateManagedThing API Anda (sama untuk semua perangkat produk ini)

```
{
  "ro": {
    "iot_provisioning_method": "FLEET_PROVISIONING",
    "iot_claim_cert_path": "<SPECIFY_THIS_FIELD>",
    "iot_claim_pk_path": "<SPECIFY_THIS_FIELD>",
    "fp_template_name": "<SPECIFY_THIS_FIELD>",
    "endpoint_url": "<SPECIFY_THIS_FIELD>",
    "SN": "<SPECIFY_THIS_FIELD>",
    "UPC": "<SPECIFY_THIS_FIELD>"
  },
  "rw": {
    "iot_provisioning_state": "NOT_PROVISIONED"
  }
}
```

Isi file konfigurasi

Tinjau isi `iotmi_config.json` file.

Daftar Isi

Key	Nilai	Ditambahkan oleh pelanggan?	Catatan
<code>iot_provisioning_method</code>	FLEET_PROVISIONING	Ya	Tentukan metode penyediaan yang ingin Anda gunakan.
<code>iot_claim_cert_path</code>	Jalur file yang Anda tentukan atau <code>SECURE_STORAGE</code> . Sebagai contoh, <code>/data/aws/iotmi/certs/claim_cert.pem</code> .	Ya	Tentukan jalur file yang ingin Anda gunakan atau <code>SECURE_STORAGE</code> .
<code>iot_claim_pk_path</code>	Jalur file yang Anda tentukan atau <code>SECURE_STORAGE</code> . Sebagai contoh, <code>/data/aws/iotmi/certs/claim_pk.pem</code> .	Ya	Tentukan jalur file yang ingin Anda gunakan atau <code>SECURE_STORAGE</code> .
<code>fp_template_name</code>	Nama template penyediaan armada harus sama dengan nama <code>ProvisioningProfile</code> yang digunakan sebelumnya.	Ya	Sama dengan nama <code>ProvisioningProfile</code> yang digunakan sebelumnya
<code>endpoint_url</code>	URL endpoint untuk integrasi terkelola.	Ya	Perangkat Anda menggunakan URL ini untuk terhubung ke cloud integrasi terkelola. Untuk mendapatkan informasi ini,

Key	Nilai	Ditambahkan oleh pelanggan ?	Catatan
			gunakan RegisterCustomEndpointAPI .
SN	Nomor seri perangkat. Misalnya, AIDACKCEV SQ6C2EXAMPLE .	Ya	Anda harus memberikan informasi unik ini untuk setiap perangkat.
UPC	Kode produk universal perangkat. Misalnya, 841667145075 .	Ya	Anda harus memberikan informasi ini untuk perangkat.
managed_t hing_id	ID dari hal yang dikelola.	Tidak	Informasi ini ditambahkan kemudian oleh proses orientasi setelah penyediaan hub.
iot_provi sioning_s tate	Status penyediaan.	Ya	Status penyediaan harus ditetapkan sebagai. NOT_PROVISIONED
iot_perma nent_cert _path	Jalur sertifikat IoT. Misalnya, /data/aws/iotmi/iot_cert.pem .	Tidak	Informasi ini ditambahkan kemudian oleh proses orientasi setelah penyediaan hub.
iot_perma nent_pk_path	Jalur file kunci pribadi IoT. Misalnya, /data/aws/iotmi/iot_pk.pem .	Tidak	Informasi ini ditambahkan kemudian oleh proses orientasi setelah penyediaan hub.

Key	Nilai	Ditambahkan oleh pelanggan ?	Catatan
<code>client_id</code>	ID klien yang akan digunakan untuk koneksi MQTT.	Tidak	Informasi ini ditambahkan kemudian oleh proses orientasi setelah penyediaan hub, agar komponen lain dapat dikonsumsi.
<code>mqtt_keep_alive_interval</code>	Rentang 30-1200, dan unit dalam hitungan detik. Nilai defaultnya adalah 300.	Ya	Gunakan ini untuk mengatur interval keep-alive untuk koneksi MQTT.
<code>event_manager_upper_bound</code>	Nilai default adalah 500.	Tidak	Informasi ini ditambahkan kemudian oleh proses orientasi setelah penyediaan hub, agar komponen lain dapat dikonsumsi.

Langkah 7: Salin file konfigurasi ke hub Anda

Salin file konfigurasi Anda ke `/data/aws/iotmi/config` atau jalur direktori kustom Anda. Anda akan memberikan jalur ini ke HubOnboarding biner selama proses orientasi.

Untuk penyediaan armada

```

/data/
  aws/
    iotmi/
      config/
        iotmi_config.json
      certs/
        claim_cert.pem
        claim_pk.key

```

Perangkat onboard dan mengoperasikannya di hub

Siapkan perangkat Anda untuk onboard ke hub integrasi terkelola dengan membuat hal terkelola dan menghubungkannya ke hub Anda. Perangkat dapat di-onboard ke hub melalui pengaturan sederhana atau pengaturan yang dipandu pengguna.

Topik

- [Pengaturan sederhana untuk onboard dan mengoperasikan perangkat](#)
- [Penyiapan yang dipandu pengguna untuk onboard dan mengoperasikan perangkat](#)

Pengaturan sederhana untuk onboard dan mengoperasikan perangkat

Siapkan perangkat Anda untuk onboard ke hub integrasi terkelola dengan membuat hal terkelola dan menghubungkannya ke hub Anda. Bagian ini menjelaskan langkah-langkah untuk menyelesaikan proses orientasi perangkat menggunakan pengaturan sederhana.

Prasyarat

Selesaikan langkah-langkah ini sebelum mencoba untuk onboard perangkat:

- Mengonboard perangkat hub ke hub integrasi terkelola.
- Instal versi terbaru AWS CLI dari Referensi [AWS CLI Perintah Integrasi Terkelola](#)
- Berlangganan pemberitahuan acara [DEVICE_LIFE_CYCLE](#).

Langkah-langkah penyiapan

- [Langkah 1: Buat loker kredensial](#)
- [Langkah 2: Tambahkan loker kredensial ke hub Anda](#)
- [Langkah 3: Buat hal yang dikelola dengan kredensial.](#)
- [Langkah 4: Colokkan perangkat dan periksa statusnya.](#)
- [Langkah 5: Dapatkan Kemampuan Perangkat](#)
- [Langkah 6: Kirim perintah ke hal yang dikelola](#)
- [Langkah 7: Hapus hal yang dikelola dari hub Anda](#)

Langkah 1: Buat loker kredenal

Buat loker kredenal untuk perangkat Anda.

Untuk membuat loker kredenal

- Gunakan perintah `create-credential-locker`. Menjalankan perintah ini akan memicu pembuatan semua sumber daya manufaktur termasuk key pair pengaturan Wi-Fi dan sertifikat perangkat.

`create-credential-locker` contoh

```
aws iot-managed-integrations create-credential-locker \  
  --name "DEVICE_NAME"
```

Tanggapan:

```
{  
  "Id": "LOCKER_ID"  
  "Arn": "arn:aws:iotmanagedintegrations:AWS_REGION:AWS_ACCOUNT_ID:credential-  
locker/LOCKER_ID"  
  "CreatedAt": "2025-06-09T13:58:52.977000+08:00"  
}
```

Untuk informasi selengkapnya, lihat [create-credential-locker](#) perintah dalam Integrasi terkelola AWS CLI Command Reference.

Langkah 2: Tambahkan loker kredenal ke hub Anda

Tambahkan loker kredenal ke hub Anda.

Untuk menambahkan loker kredenal ke hub

- Gunakan perintah berikut untuk menambahkan loker kredenal ke hub Anda.

```
aws iotmi --region AWS_REGION --endpoint AWS_ENDPOINT update-managed-thing \  
  --identifier "HUB_MANAGED_THING_ID" --credential-locker-id "LOCKER_ID"
```

Langkah 3: Buat hal yang dikelola dengan kredensi.

Buat hal yang dikelola dengan kredensial untuk perangkat Anda. Setiap perangkat membutuhkan hal yang dikelola sendiri.

Untuk membuat hal yang dikelola

- Gunakan `create-managed-thing` perintah untuk membuat hal yang dikelola untuk perangkat Anda.

`create-managed-thing` contoh

```
#ZWAVE:
aws iot-managed-integrations create-managed-thing --role DEVICE \
--authentication-material '900137947003133...' \ #auth material from zwave qr code
--authentication-material-type ZWAVE_QR_BAR_CODE \
--credential-locker-id ${locker_id}

#ZIGBEE:
aws iot-managed-integrations create-managed-thing --role DEVICE \
--authentication-material 'Z:286...$I:A4DC00.' \ #auth material from zigbee qr code
--authentication-material-type ZIGBEE_QR_BAR_CODE \
--credential-locker-id ${locker_id}
```

Note

Ada perintah terpisah untuk perangkat Z-wave dan Zigbee.

Tanggapan:

```
{
  "Id": "DEVICE_MANAGED_THING_ID"
  "Arn": "arn:aws:iotmanagedintegrations:AWS_REGION:AWS_ACCOUNT_ID:managed-
thing/DEVICE_MANAGED_THING_ID"
  "CreatedAt": "2025-06-09T13:58:52.977000+08:00"
}
```

Untuk informasi selengkapnya, lihat [create-managed-thing](#) perintah dalam Integrasi terkelola AWS CLI Command Reference.

Langkah 4: Colokkan perangkat dan periksa statusnya.

Colokkan perangkat dan periksa statusnya.

- Gunakan `get-managed-thing` perintah untuk memeriksa status perangkat Anda. Hal `ProvisioningStatus` yang dikelola Anda harus **DIAKTIFKAN**. Untuk informasi selengkapnya `ProvisioningStatus`, lihat [Penyediaan Perangkat](#).

`get-managed-thing`contoh

```
#KINESIS NOTIFICATION:
{
  "version": "1.0.0",
  "messageId": "4ac684bb7f4c41adbb2eccc1e7991xxx",
  "messageType": "DEVICE_LIFE_CYCLE",
  "source": "aws.iotmanagedintegrations",
  "customerAccountId": "12345678901",
  "timestamp": "2025-06-10T05:30:59.852659650Z",
  "region": "us-east-1",
  "resources": ["XXX"],
  "payload": {
    "deviceDetails": {
      "id": "1e84f61fa79a41219534b6fd57052XXX",
      "arn": "XXX",
      "createdAt": "2025-06-09T06:24:34.336120179Z",
      "updatedAt": "2025-06-10T05:30:59.784157019Z"
    },
    "status": "ACTIVATED"
  }
}
aws iot-managed-integrations get-managed-thing \
--identifier :"DEVICE_MANAGED_THING_ID"
```

Tanggapan:

```
{
  "Id": "DEVICE_MANAGED_THING_ID"
  "Arn": "arn:aws:iotmanagedintegrations:AWS_REGION:AWS_ACCOUNT_ID:managed-thing/MANAGED_THING_ID"
  "CreatedAt": "2025-06-09T13:58:52.977000+08:00"
}
```

Untuk informasi selengkapnya, lihat [get-managed-thing](#) perintah dalam Integrasi terkelola AWS CLI Command Reference.

Langkah 5: Dapatkan Kemampuan Perangkat

Gunakan `get-managed-thing-capabilities` perintah untuk mendapatkan ID titik akhir Anda dan melihat daftar tindakan yang mungkin untuk perangkat Anda.

Untuk mendapatkan kemampuan perangkat

- Gunakan `get-managed-thing-capabilities` perintah dan catat ID titik akhir.

`get-managed-thing-capabilities` contoh

```
aws iotmi get-managed-thing-capabilities \  
--identifier "DEVICE_MANAGED_THING_ID"
```

Tanggapan:

```
{  
  "ManagedThingId": "1e84f61fa79a41219534b6fd57052cbc",  
  "CapabilityReport": {  
    "version": "1.0.0",  
    "nodeId": "zw.FCB10009+06",  
    "endpoints": [  
      {  
        "id": "ENDPOINT_ID"  
        "deviceTypes": [  
          "On/Off Switch"  
        ],  
        "capabilities": [  
          {  
            "id": "matter.OnOff@1.4",  
            "name": "On/Off",  
            "version": "6",  
            "properties": [  
              "OnOff"  
            ],  
            "actions": [  
              "Off",  
              "On"  
            ]  
          }  
        ]  
      }  
    ]  
  }  
}
```

```

        ],
        "events": []
      }
      ...
    }
  }
}

```

Untuk informasi selengkapnya, lihat [get-managed-thing-capabilities](#) perintah dalam Integrasi terkelola AWS CLI Command Reference.

Langkah 6: Kirim perintah ke hal yang dikelola

Gunakan `send-managed-thing-command` perintah untuk mengirim perintah toggle action ke hal yang Anda kelola.

Untuk mengirim perintah ke hal yang Anda kelola

- Gunakan `send-managed-thing-command` perintah untuk mengirim perintah ke hal yang Anda kelola.

`send-managed-thing-command` contoh

```

json=$(jq -cr '.|@json') <<EOF
[
  {
    "endpointId": "1",
    "capabilities": [
      {
        "id": "matter.OnOff@1.4",
        "name": "On/Off",
        "version": "1",
        "actions": [
          {
            "name": "Toggle",
            "parameters": {}
          }
        ]
      }
    ]
  }
]
EOF
aws iot-managed-integrations send-managed-thing-command \

```

```
--managed-thing-id "DEVICE_MANAGED_THING_ID" --endpoints "ENDPOINT_ID"
```

Note

Contoh ini menggunakan jq cli tetapi Anda juga dapat meneruskan seluruh string `endpointId`

Tanggapan:

```
{  
  "TraceId": "TRACE_ID"  
}
```

Untuk informasi selengkapnya, lihat [send-managed-thing-command](#) perintah dalam Integrasi terkelola AWS CLI Command Reference.

Langkah 7: Hapus hal yang dikelola dari hub Anda

Bersihkan hub Anda dengan menghapus hal yang dikelola.

Untuk menghapus hal yang dikelola

- Gunakan `delete-managed-thing` perintah untuk menghapus hal yang dikelola dari hub perangkat Anda.

`delete-managed-thing` contoh

```
aws iot-managed-integrations delete-managed-thing \  
--identifier "DEVICE_MANAGED_THING_ID"
```

Untuk informasi selengkapnya, lihat [delete-managed-thing](#) perintah dalam Integrasi terkelola AWS CLI Command Reference.

Note

Jika perangkat macet dalam DELETE_IN_PROGRESS keadaan, tambahkan `--force` bendera ke file. `delete-managed-thing` command

Note

Untuk perangkat gelombang-Z, Anda harus memasukkan perangkat ke mode berpasangan setelah menjalankan perintah.

Penyiapan yang dipandu pengguna untuk onboard dan mengoperasikan perangkat

Siapkan perangkat Anda untuk onboard ke hub integrasi terkelola Anda dengan membuat hal terkelola dan menghubungkannya ke hub Anda. Bagian ini menjelaskan langkah-langkah untuk menyelesaikan proses orientasi perangkat menggunakan pengaturan yang dipandu pengguna.

Prasyarat

Selesaikan langkah-langkah ini sebelum mencoba untuk onboard perangkat:

- Mengonboard perangkat hub ke hub integrasi terkelola.
- Instal versi terbaru AWS CLI dari Referensi [AWS CLI Perintah Integrasi Terkelola](#)
- Berlangganan pemberitahuan acara [DEVICE_DISCOVERY-STATUS](#).

Langkah-langkah penyiapan yang dipandu pengguna

- [Prasyarat: Aktifkan mode pemasangan pada perangkat Z Wave Anda](#)
- [Langkah 1: Mulai penemuan perangkat](#)
- [Langkah 2: Kueri ID pekerjaan penemuan](#)
- [Langkah 3: Buat hal yang dikelola untuk perangkat Anda](#)
- [Langkah 4: Kueri hal yang dikelola](#)
- [Langkah 5: Dapatkan kemampuan hal yang dikelola](#)
- [Langkah 6: Kirim perintah ke hal yang dikelola](#)

- [Langkah 7: Periksa status hal yang dikelola](#)
- [Langkah 8: Hapus hal yang dikelola dari hub Anda](#)

Prasyarat: Aktifkan mode pemasangan pada perangkat Z Wave Anda

Aktifkan mode pemasangan pada perangkat gelombang-Z. Mode pemasangan dapat bervariasi untuk setiap perangkat Z-Wave, jadi lihat instruksi perangkat untuk mengatur mode pemasangan dengan benar. Biasanya tombol yang harus ditekan pengguna.

Langkah 1: Mulai penemuan perangkat

Mulai penemuan perangkat untuk hub Anda untuk mendapatkan ID pekerjaan penemuan yang digunakan untuk onboard perangkat Anda.

Untuk memulai penemuan perangkat

- Gunakan [start-device-discovery](#) perintah untuk mendapatkan ID pekerjaan penemuan.

start-device-discovery contoh

```
#For Zigbee
aws iot-managed-integrations start-device-discovery --discovery-type ZIGBEE \
--controller-identifier HUB_MANAGED_THING_ID

#For Zwave
aws iot-managed-integrations start-device-discovery --discovery-type ZWAVE \
--controller-identifier HUB_MANAGED_THING \
--authentication-material-type ZWAVE_INSTALL_CODE \
--authentication-material 13333

#For Cloud
aws iot-managed-integrations start-device-discovery --discovery-type CLOUD \
--account-association-id C2C_ASSOCIATION_ID \

#For Custom
aws iot-managed-thing start-device-discovery --discovery-type CUSTOM \
--controller-identifier HUB_MANAGED_THING_ID \
--custom-protocol-detail NAME : NON_EMPTY_STRING \
```

Tanggapan:

```
{
  "Id": DISCOVERY_JOB_ID,
  "StartedAt": "2025-06-03T14:43:12.726000-07:00"
}
```

Note

Ada perintah terpisah untuk perangkat Z-wave dan Zigbee.

Untuk informasi selengkapnya, lihat [start-device-discovery](#) API di Referensi AWS CLI Perintah integrasi terkelola.

Langkah 2: Kueri ID pekerjaan penemuan

Gunakan `list-discovered-devices` perintah untuk mendapatkan materi otentikasi perangkat Anda.

Untuk menanyakan ID pekerjaan penemuan Anda

- Gunakan ID pekerjaan penemuan dengan `list-discovered-devices` perintah untuk mendapatkan materi otentikasi perangkat Anda.

```
aws iot-managed-integrations list-discovered-devices --identifier DISCOVERY_JOB_ID
```

Tanggapan:

```
"Items": [
  {
    "DeviceTypes": [],
    "DiscoveredAt": "2025-06-03T14:43:37.619000-07:00",
    "AuthenticationMaterial": AUTHENTICATION_MATERIAL
  }
]
```

Langkah 3: Buat hal yang dikelola untuk perangkat Anda

Gunakan `create-managed-thing` perintah untuk membuat hal yang dikelola untuk perangkat Anda. Setiap perangkat membutuhkan hal yang dikelola sendiri.

Untuk membuat hal yang dikelola

- Gunakan `create-managed-thing` perintah untuk membuat hal yang dikelola untuk perangkat Anda.

`create-managed-thing` contoh

```
aws iot-managed-integrations create-managed-thing \  
  --role DEVICE --authentication-material-type DISCOVERED_DEVICE \  
  --authentication-material "AUTHENTICATION_MATERIAL"
```

Tanggapan:

```
{  
  "Id": "DEVICE_MANAGED_THING_ID"  
  "Arn": "arn:aws:iotmanagedintegrations:AWS_REGION:AWS_ACCOUNT_ID:managed-  
thing/DEVICE_MANAGED_THING_ID"  
  "CreatedAt": "2025-06-09T13:58:52.977000+08:00"  
}
```

Untuk informasi selengkapnya, lihat [create-managed-thing](#) perintah dalam Integrasi terkelola AWS CLI Command Reference.

Langkah 4: Kueri hal yang dikelola

Anda dapat memeriksa apakah hal yang dikelola diaktifkan dengan menggunakan `get-managed-thing` perintah.

Untuk menanyakan hal yang dikelola

- Gunakan `get-managed-thing` perintah untuk memeriksa apakah status penyediaan hal yang dikelola disetel ke `ACTIVATED` Untuk informasi selengkapnya tentang status penyediaan, lihat Penyediaan [Perangkat](#).

`get-managed-thing` contoh

```
aws iot-managed-integrations get-managed-thing \  
  --identifier "DEVICE_MANAGED_THING_ID"
```

Tanggapan:

```
{  
  "Id": "DEVICE_MANAGED_THING_ID",  
  "Arn": "arn:aws:iotmanagedintegrations:AWS_REGION:AWS_ACCOUNT_ID:managed-  
thing/DEVICE_MANAGED_THING_ID",  
  "Role": "DEVICE",  
  "ProvisioningStatus": "ACTIVATED",  
  "MacAddress": "MAC_ADDRESS",  
  "ParentControllerId": "PARENT_CONTROLLER_ID",  
  "CreatedAt": "2025-06-03T14:46:35.149000-07:00",  
  "UpdatedAt": "2025-06-03T14:46:37.500000-07:00",  
  "Tags": {}  
}
```

Untuk informasi selengkapnya, lihat [get-managed-thing](#) perintah dalam Integrasi terkelola AWS CLI Command Reference.

Langkah 5: Dapatkan kemampuan hal yang dikelola

Anda dapat melihat daftar tindakan yang tersedia dari hal terkelola dengan menggunakan `get-managed-thing-capabilities`.

Untuk mendapatkan kemampuan perangkat

- Gunakan `get-managed-thing-capabilities` perintah untuk mendapatkan ID endpoint. Perhatikan juga daftar tindakan yang mungkin.

`get-managed-thing-capabilities` contoh

```
aws iot-managed-integrations get-managed-thing-capabilities \  
  --identifier "DEVICE_MANAGED_THING_ID"
```

Tanggapan:

```
{
```

```
"ManagedThingId": "DEVICE_MANAGED_THING_ID",
"CapabilityReport": {
  "version": "1.0.0",
  "nodeId": "zb.539D+4A1D",
  "endpoints": [
    {
      "id": "1",
      "deviceTypes": [
        "Unknown Device"
      ],
      "capabilities": [
        {
          "id": "matter.OnOff@1.4",
          "name": "On/Off",
          "version": "6",
          "properties": [
            "OnOff",
            "OnOff",
            "OnTime",
            "OffWaitTime"
          ],
          "actions": [
            "Off",
            "On",
            "Toggle",
            "OffWithEffect",
            "OnWithRecallGlobalScene",
            "OnWithTimedOff"
          ]
        },
        ...
      ]
    }
  ]
}
```

Untuk informasi selengkapnya, lihat [get-managed-thing-capabilities](#) perintah dalam Integrasi terkelola AWS CLI Command Reference.

Langkah 6: Kirim perintah ke hal yang dikelola

Anda dapat menggunakan `send-managed-thing-command` perintah untuk mengirim perintah toggle action ke hal yang Anda kelola.

Kirim perintah ke hal yang dikelola menggunakan tindakan sakelar.

- Gunakan `send-managed-thing-command` perintah untuk mengirim perintah toggle action.

send-managed-thing-commandcontoh

```
json=$(jq -cr '.|@json') <<EOF
[
  {
    "endpointId": "1",
    "capabilities": [
      {
        "id": "matter.OnOff@1.4",
        "name": "On/Off",
        "version": "1",
        "actions": [
          {
            "name": "Toggle",
            "parameters": {}
          }
        ]
      }
    ]
  }
]
EOF
aws iot-managed-integrations send-managed-thing-command \
--managed-thing-id ${device_managed_thing_id} --endpoints ENDPOINT_ID
```

Note

Contoh ini menggunakan jq cli tetapi Anda juga dapat meneruskan seluruh string `endpointId`

Tanggapan:

```
{
  "TraceId": TRACE_ID
}
```

Untuk informasi selengkapnya, lihat [send-managed-thing-command](#) perintah dalam Integrasi terkelola AWS CLI Command Reference.

Langkah 7: Periksa status hal yang dikelola

Periksa status hal yang dikelola untuk memvalidasi tindakan sakelar berhasil.

Untuk memeriksa status perangkat yang dikelola

- Gunakan `get-managed-thing-state` perintah untuk memvalidasi tindakan toggle berhasil.

`get-managed-thing-state` contoh

```
aws iot-managed-integrations get-managed-thing-state --managed-thing-id DEVICE_MANAGED_THING_ID
```

Tanggapan:

```
{
  "Endpoints": [
    {
      "endpointId": "1",
      "capabilities": [
        {
          "id": "matter.OnOff@1.4",
          "name": "On/Off",
          "version": "1.4",
          "properties": [
            {
              "name": "OnOff",
              "value": {
                "propertyValue": true,
                "lastChangedAt": "2025-06-03T21:50:39.886Z"
              }
            }
          ]
        }
      ]
    }
  ]
}
```

```
}
```

Untuk informasi selengkapnya, lihat [get-managed-thing-state](#) perintah dalam Integrasi terkelola AWS CLI Command Reference.

Langkah 8: Hapus hal yang dikelola dari hub Anda

Bersihkan hub Anda dengan menghapus hal yang dikelola.

Untuk menghapus hal yang dikelola

- Gunakan [delete-managed-thing](#) perintah untuk menghapus hal yang dikelola.

delete-managed-thing contoh

```
aws iot-managed-integrations delete-managed-thing \  
  --identifier MANAGED_THING_ID
```

Untuk informasi selengkapnya, lihat [delete-managed-thing](#) perintah dalam Integrasi terkelola AWS CLI Command Reference.

Note

Jika perangkat macet dalam DELETE_IN_PROGRESS keadaan, tambahkan `--force` bendera ke `delete-managed-thing` perintah.

Note

Untuk perangkat gelombang-Z, Anda harus memasukkan perangkat ke mode berpasangan setelah menjalankan perintah.

Buat handler sertifikat khusus untuk penyimpanan aman

Manajemen sertifikat perangkat sangat penting saat melakukan onboarding hub integrasi terkelola. Meskipun sertifikat disimpan dalam sistem file secara default, Anda dapat membuat penanganan sertifikat khusus untuk keamanan yang ditingkatkan dan manajemen kredensi yang fleksibel.

Integrasi terkelola SDK perangkat akhir menyediakan penanganan sertifikat untuk mengamankan antarmuka penyimpanan yang dapat Anda terapkan sebagai pustaka objek bersama (.so). Bangun implementasi penyimpanan aman Anda untuk membaca dan menulis sertifikat, lalu tautkan file pustaka ke HubOnboarding proses saat runtime.

Definisi dan komponen API

Tinjau `secure_storage_cert_handler_interface.hpp` file berikut untuk memahami komponen dan persyaratan API untuk implementasi Anda

Topik

- [Definisi API](#)
- [Komponen kunci](#)

Definisi API

Isi dari `secure_storage_cert_handler_interface.hpp`

```
/*
 * Copyright 2024 Amazon.com, Inc. or its affiliates. All rights reserved.
 *
 * AMAZON PROPRIETARY/CONFIDENTIAL
 *
 * You may not use this file except in compliance with the terms and
 * conditions set forth in the accompanying LICENSE.txt file.
 *
 * THESE MATERIALS ARE PROVIDED ON AN "AS IS" BASIS. AMAZON SPECIFICALLY
 * DISCLAIMS, WITH RESPECT TO THESE MATERIALS, ALL WARRANTIES, EXPRESS,
 * IMPLIED, OR STATUTORY, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT.
 */
#ifndef SECURE_STORAGE_CERT_HANDLER_INTERFACE_HPP
#define SECURE_STORAGE_CERT_HANDLER_INTERFACE_HPP

#include <iostream>
#include <memory>

namespace IoTManagedIntegrationsDevice {
namespace CertHandler {
/**
```

```
* @enum CERT_TYPE_T
* @brief enumeration defining certificate types.
*/
typedef enum { CLAIM = 0, DHA = 1, PERMANENT = 2 } CERT_TYPE_T;
class SecureStorageCertHandlerInterface {
public:
    /**
     * @brief Read certificate and private key value of a particular certificate
     * type from secure storage.
     */
    virtual bool read_cert_and_private_key(const CERT_TYPE_T cert_type,
                                           std::string &cert_value,
                                           std::string &private_key_value) = 0;

    /**
     * @brief Write permanent certificate and private key value to secure storage.
     */
    virtual bool write_permanent_cert_and_private_key(
        std::string_view cert_value, std::string_view private_key_value) = 0;
};
    std::shared_ptr<SecureStorageCertHandlerInterface>
createSecureStorageCertHandler();
} //namespace CertHandler
} //namespace IoTManagedIntegrationsDevice

#endif //SECURE_STORAGE_CERT_HANDLER_INTERFACE_HPP
```

Komponen kunci

- CERT_TYPE_T - berbagai jenis sertifikat di hub.
 - KLAIM - sertifikat klaim yang awalnya ada di hub, akan ditukar dengan sertifikat permanen.
 - DHA - tidak digunakan untuk saat ini.
 - PERMANEN - sertifikat permanen untuk terhubung dengan titik akhir integrasi terkelola.
- read_cert_and_private_key - (FUNGSI UNTUK DIIMPLEMENTASIKAN) Membaca sertifikat dan nilai kunci ke input referensi. Fungsi ini harus dapat membaca sertifikat KLAIM dan PERMANEN, dan dibedakan dengan jenis sertifikat yang disebutkan di atas.
- write_permanent_cert_and_private_key - (FUNGSI UNTUK DIIMPLEMENTASIKAN) menulis sertifikat permanen dan nilai kunci ke lokasi yang diinginkan.

Contoh membangun

Pisahkan header implementasi internal Anda dari antarmuka publik (`secure_storage_cert_handler_interface.hpp`) untuk mempertahankan struktur proyek yang bersih. Dengan pemisahan ini, Anda dapat mengelola komponen publik dan pribadi sambil membangun penangan sertifikat Anda.

Note

Menyatakan `secure_storage_cert_handler_interface.hpp` sebagai publik.

Topik

- [Struktur proyek](#)
- [Mewarisi antarmuka](#)
- [Implementasi](#)
- [CMakeList.txt](#)

Struktur proyek

Mewarisi antarmuka

Buat kelas konkret yang mewarisi antarmuka. Sembunyikan file header ini dan file lainnya di bawah direktori terpisah sehingga header pribadi dan publik dapat dibedakan dengan mudah saat membangun.

```
#ifndef IOTMANAGEDINTEGRATIONSDEVICE_SDK_STUB_SECURE_STORAGE_CERT_HANDLER_HPP
#define IOTMANAGEDINTEGRATIONSDEVICE_SDK_STUB_SECURE_STORAGE_CERT_HANDLER_HPP

#include "secure_storage_cert_handler_interface.hpp"

namespace IoTManagedIntegrationsDevice::CertHandler {
    class StubSecureStorageCertHandler : public SecureStorageCertHandlerInterface {
    public:
        StubSecureStorageCertHandler() = default;

        bool read_cert_and_private_key(const CERT_TYPE_T cert_type,
                                      std::string &cert_value,
```

```

        std::string &private_key_value) override;

    bool write_permanent_cert_and_private_key(
        std::string_view cert_value, std::string_view private_key_value) override;
    /*
     * any other resource for function you might need
     */

};
}
#endif //IOTMANAGEDINTEGRATIONSDEVICE_SDK_STUB_SECURE_STORAGE_CERT_HANDLER_HPP

```

Implementasi

Menerapkan kelas penyimpanan yang didefinisikan di atas,src/
stub_secure_storage_cert_handler.cpp.

```

/*
 * Copyright 2024 Amazon.com, Inc. or its affiliates. All rights reserved.
 *
 * AMAZON PROPRIETARY/CONFIDENTIAL
 *
 * You may not use this file except in compliance with the terms and
 * conditions set forth in the accompanying LICENSE.txt file.
 *
 * THESE MATERIALS ARE PROVIDED ON AN "AS IS" BASIS. AMAZON SPECIFICALLY
 * DISCLAIMS, WITH RESPECT TO THESE MATERIALS, ALL WARRANTIES, EXPRESS,
 * IMPLIED, OR STATUTORY, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT.
 */

#include "stub_secure_storage_cert_handler.hpp"

using namespace IoTManagedIntegrationsDevice::CertHandler;

bool StubSecureStorageCertHandler::write_permanent_cert_and_private_key(
    std::string_view cert_value, std::string_view private_key_value) {
    // TODO: implement write function
    return true;
}

```

```

bool StubSecureStorageCertHandler::read_cert_and_private_key(const CERT_TYPE_T
cert_type,
                                                    std::string &cert_value,
                                                    std::string
&private_key_value) {
    std::cout<<"Using Stub Secure Storage Cert Handler, returning dummy values";
    cert_value = "StubCertVal";
    private_key_value = "StubKeyVal";
    // TODO: implement read function
    return true;
}

```

Menerapkan fungsi pabrik yang didefinisikan dalam antarmuka,src/
secure_storage_cert_handler.cpp.

```

#include "stub_secure_storage_cert_handler.hpp"

std::shared_ptr<IoTManagedIntegrationsDevice::CertHandler::SecureStorageCertHandlerInterface>
IoTManagedIntegrationsDevice::CertHandler::createSecureStorageCertHandler() {
    // TODO: replace with your implementation
    return
std::make_shared<IoTManagedIntegrationsDevice::CertHandler::StubSecureStorageCertHandler>();
}

```

CMakeList.txt

```

#project name must stay the same
project(SecureStorageCertHandler)

# Public Header files. The interface definition must be in top level with exactly
the same name
#ie. Not in anotherDir/secure_storage_cert_handler_interface.hpp
set(PUBLIC_HEADERS
    ${PROJECT_SOURCE_DIR}/include
)

```

```
# private implementation headers.
set(PRIVATE_HEADERS
    ${PROJECT_SOURCE_DIR}/internal/stub
)

#set all sources
set(SOURCES
    ${PROJECT_SOURCE_DIR}/src/secure_storage_cert_handler.cpp
    ${PROJECT_SOURCE_DIR}/src/stub_secure_storage_cert_handler.cpp
)

# Create the shared library
add_library(${PROJECT_NAME} SHARED ${SOURCES})
target_include_directories(
    ${PROJECT_NAME}
    PUBLIC
        ${PUBLIC_HEADERS}
    PRIVATE
        ${PRIVATE_HEADERS}
)

# Set the library output location. Location can be customized but version must
stay the same
set_target_properties(${PROJECT_NAME} PROPERTIES
    LIBRARY_OUTPUT_DIRECTORY ${CMAKE_BINARY_DIR}/../lib
    VERSION 1.0
    SOVERSION 1
)

# Install rules
install(TARGETS ${PROJECT_NAME}
    LIBRARY DESTINATION lib
    ARCHIVE DESTINATION lib
)

install(FILES ${HEADERS}
    DESTINATION include/SecureStorageCertHandler
)
```

Penggunaan

Setelah kompilasi, Anda akan memiliki file pustaka objek `libSecureStorageCertHandler.so` bersama dan tautan simbolik yang terkait. Salin file pustaka dan tautan simbolik ke lokasi perpustakaan yang diharapkan oleh biner. `HubOnboarding`

Topik

- [Pertimbangan utama](#)
- [Gunakan penyimpanan yang aman](#)

Pertimbangan utama

- Verifikasi bahwa akun pengguna Anda telah membaca dan menulis izin untuk `HubOnboarding` biner dan `libSecureStorageCertHandler.so` pustaka.
- Simpan `secure_storage_cert_handler_interface.hpp` sebagai satu-satunya file header publik Anda. Semua file header lainnya harus tetap dalam implementasi pribadi Anda.
- Verifikasi nama pustaka objek bersama Anda. Saat Anda membangun `libSecureStorageCertHandler.so`, `HubOnboarding` mungkin memerlukan versi tertentu dalam nama file, seperti `libSecureStorageCertHandler.so.1.0`. Gunakan `ldd` perintah untuk memeriksa dependensi perpustakaan dan membuat tautan simbolik sesuai kebutuhan.
- Jika implementasi pustaka bersama Anda memiliki dependensi eksternal, simpan di direktori yang `HubOnboarding` dapat diakses, seperti `/usr/lib` or the `iotmi_common` direktori.

Gunakan penyimpanan yang aman

Perbarui `iotmi_config.json` file Anda dengan mengatur keduanya `iot_claim_cert_path` dan `iot_claim_pk_path` ke **SECURE_STORAGE**.

```
{
  "ro": {
    "iot_provisioning_method": "FLEET_PROVISIONING",
    "iot_claim_cert_path": "SECURE_STORAGE",
    "iot_claim_pk_path": "SECURE_STORAGE",
    "fp_template_name": "device-integration-example",
    "iot_endpoint_url": "[ACCOUNT-PREFIX]-ats.iot.AWS-REGION.amazonaws.com",
    "SN": "1234567890",
```

```
"UPC": "1234567890"  
},  
"rw": {  
  "iot_provisioning_state": "NOT_PROVISIONED"  
}  
}
```

Plugin protokol kustom

Anda dapat menggunakan plugin protokol khusus untuk mengintegrasikan protokol IoT milik Anda ke dalam integrasi terkelola untuk ekosistem. AWS IoT Device Management Melalui antarmuka SDK yang terdefinisi dengan baik, Anda dapat melakukan onboard perangkat, menentukan kemampuan, dan menangani alur kontrol waktu nyata sambil mempertahankan kompatibilitas penuh dengan integrasi terkelola dan komponen SDK hub.

Daftar berikut membahas fitur utama dari plugin protokol kustom.

Kustomisasi model data

Tentukan skema model AWS data Anda sendiri dan unggah ke integrasi terkelola selama alur penyediaan. Anda dapat menggunakan skema ini nanti di alur kerja Anda.

Implementasi plugin yang fleksibel

- Bangun komponen plugin Anda sendiri menggunakan [Klien Hub SDK](#).
- Menerapkan plugin terpisah untuk fungsi yang berbeda, seperti penyediaan dan kontrol, atau buat klien terpadu untuk keduanya.
- Pertahankan batas yang jelas antara aset integrasi terkelola dan aset Anda sendiri seperti tumpukan middleware, untuk implementasi logika kode yang dipisahkan dan ramah pengembangan.

Kompatibilitas mundur

Untuk pelanggan yang sudah ada, gunakan protokol kustom baru Anda dengan lancar sambil juga menjaga jenis radio yang ada berfungsi apa adanya.

Diagram berikut menggambarkan arsitektur plugin protokol kustom.

Klien Hub SDK

Pustaka Klien SDK Hub berfungsi sebagai antarmuka antara integrasi terkelola Hub SDK dan tumpukan protokol Anda sendiri yang berjalan di hub yang sama. Ini mengekspos satu set publik APIs untuk memfasilitasi interaksi tumpukan protokol Anda dengan komponen Device Hub SDK. Kasus penggunaan termasuk kontrol plugin khusus, penyedia plugin khusus, dan pengontrol lokal.

Topik

- [Dapatkan integrasi terkelola Hub SDK](#)
- [Tentang toolkit Hub SDK](#)
- [Buat aplikasi kustom Anda dengan klien Hub SDK](#)
- [Menjalankan aplikasi kustom Anda](#)
- [Referensi API klien SDK Hub](#)
- [Jenis Data](#)

Dapatkan integrasi terkelola Hub SDK

Klien Hub SDK dilengkapi dengan SDK integrasi terkelola. Hubungi kami dari [konsol integrasi terkelola](#) untuk mengakses SDK hub.

Tentang toolkit Hub SDK

Setelah mengunduh, Anda akan melihat IotMI-DeviceSDK-Toolkit folder, yang berisi semua file header publik dan .so file yang dapat Anda konsumsi dalam aplikasi Anda. Tim integrasi terkelola juga memberikan contoh main.cpp untuk tujuan demo, bersama dengan biner aplikasi demo di bawah bin/ yang dapat Anda jalankan secara langsung. Anda dapat secara opsional menggunakan ini sebagai titik awal untuk aplikasi Anda.

Buat aplikasi kustom Anda dengan klien Hub SDK

Gunakan langkah-langkah berikut untuk membuat aplikasi kustom Anda.

1. Sertakan file header (.h) dan file objek bersama (.so) dalam aplikasi Anda.

Anda harus menyertakan file header publik (.h) dan file objek bersama (.so) dalam aplikasi Anda. Untuk file.so, Anda dapat menempatkannya di folder lib. Tata letak akhir akan mirip dengan ini:

```
### include
#   ### iotmi_device_sdk_client
#   #   ### iotmi_device_sdk_client_common_types.h
#   ### iotmi_device_sdk_client.h
#   ### iotshd_status.h
### lib
#   ### libiotmi_devicesdk_client_module.so
#   ### libiotmi_log_c.so
```

2. Buat klien Hub SDK di aplikasi utama Anda.

- a. Di aplikasi utama Anda, Anda harus terlebih dahulu menginisialisasi klien Hub SDK sebelum dapat digunakan untuk memproses permintaan. Anda dapat dengan mudah membangun klien dengan `clientId`.
- b. Setelah Anda memiliki klien, Anda dapat menghubungkannya ke integrasi terkelola Device SDK.

Berikut ini adalah contoh pembuatan klien Hub SDK dan cara menghubungkannya.

```
#include <cstdlib>
#include <string>
#include "iotshd_status.h"
#include "iotmi_device_sdk_client.h"

auto client = std::make_unique<DeviceSDKClient>(your_own_clientId);
iotmi_statusCode_t status = client->connect();
```

Note

`your_own_clientId` harus sama dengan yang Anda tentukan [start-device-discovery](#) dalam pengaturan yang dipandu pengguna, atau untuk [create-managed-thing](#) dalam alur penyediaan pengaturan sederhana.

3. Publikasikan dan berlangganan dengan melakukan langkah-langkah berikut.

- a. Setelah koneksi dibuat, Anda sekarang dapat berlangganan tugas masuk dari integrasi terkelola Hub SDK. Tugas yang masuk dapat berupa tugas kontrol atau tugas penyediaan. Anda juga perlu menentukan fungsi callback Anda sendiri pada tugas yang diterima, dan konteks kustom Anda sendiri untuk tujuan penelusuran Anda sendiri.

```
// subscribe to provisioning tasks
iotmi_statusCode_t status = client->iotmi_provision_subscribe_to_tasks(
    example_subscriber_callback, custom_context);

// subscribe to control tasks
iotmi_statusCode_t status = client->iotmi_control_subscribe_to_tasks(
    example_subscriber_callback, custom_context);
```

- b. Setelah koneksi dibuat, Anda sekarang dapat mempublikasikan permintaan dari aplikasi Anda ke integrasi terkelola Hub SDK. Anda dapat menentukan jenis pesan tugas Anda sendiri, dengan muatan yang berbeda untuk tujuan bisnis yang berbeda. Permintaan dapat mencakup permintaan kontrol dan permintaan ketentuan, mirip dengan alur berlangganan. Terakhir, Anda dapat menetapkan alamat `rspPayload` untuk mendapatkan respons dari integrasi terkelola Hub SDK dengan cara yang disinkronkan.

```
// publish control request
iotmi_client_request_t api_payload = {
    .messageType = C2MIMessageType::C2MI_CONTROL_EVENT,
    .reqPayload = (uint8_t *)"define_your_req_payload",
    .rspPayload = (uint8_t *)calloc(1000, sizeof(uint8_t))
};

status = client->iotmi_control_publish_request(&api_payload);

// publish provision request
iotmi_client_request_t api_payload = {
    .messageType = C2MIMessageType::C2MI_DEVICE_ONBOARDED,
    .reqPayload = (uint8_t *)"define_your_req_payload",
    .rspPayload = (uint8_t *)calloc(1000, sizeof(uint8_t))
};

status = client->iotmi_provision_publish_request(&api_payload);
```

4. Bangun aplikasi Anda sendiri `CMakeLists.txt` dan bangun aplikasi Anda dari sana. Output akhir bisa berupa biner yang dapat dieksekusi seperti `MyFirstApplication`

Menjalankan aplikasi kustom Anda

Sebelum menjalankan aplikasi kustom, selesaikan langkah-langkah berikut untuk menyiapkan hub dan memulai integrasi terkelola Hub SDK:

- Ikuti instruksi orientasi di [Onboard hub Anda ke integrasi terkelola](#).
- Selesaikan proses instalasi yang didokumentasikan di [Instal dan validasi integrasi terkelola Hub SDK](#).

Setelah prasyarat terpenuhi, Anda dapat menjalankan aplikasi kustom Anda. Misalnya:

```
./MyFirstApplication
```

Important

Anda harus memperbarui skrip awal yang tercantum secara manual [Menerapkan Hub SDK dengan skrip](#) dengan skrip untuk memulai aplikasi Anda sendiri. Urutannya penting, jangan ubah urutannya.

Perbarui yang berikut ini. Perubahan

```
./IotMI-DeviceSDK-Toolkit/bin/DeviceSDKClientDemo >> $LOGS_DIR/  
logDeviceSDKClientDemo_logs.txt &
```

kepada

```
./MyFirstApplication >> $LOGS_DIR/MyFirstApplication_logs.txtt &
```

Referensi API klien SDK Hub

Klien Hub SDK (SDKClient Kelas perangkat) menyediakan antarmuka untuk aplikasi kustom Anda untuk berinteraksi dengan integrasi terkelola Device SDK. Dengan klien ini, Anda dapat melakukan hal berikut:

- Berlangganan tugas terkait penyediaan dan kontrol dari komponen integrasi terkelola.
- Publikasikan permintaan terkait penyediaan dan kontrol terkait ke komponen integrasi terkelola.

Untuk informasi tentang integrasi terkelola AWS IoT Device Management APIs, lihat [Apa itu AWS Lambda](#).

Topik

- [Inisialisasi klien](#)
- [Langganan tugas penyediaan](#)
- [Publikasi tugas penyediaan](#)
- [Kontrol langganan tugas](#)
- [Kontrol publikasi tugas](#)
- [Fitur logging](#)
- [Lainnya APIs](#)

Inisialisasi klien

Untuk mulai menggunakan `DeviceSDKClient`, inisialisasi dengan ID klien.

```
iotmi_statusCode_t DeviceSDKClient(const std::string& clientId)
```

Ini menciptakan `DeviceSDKClient` instance baru dengan yang ditentukan `clientId`. `clientId` harus cocok dengan yang Anda daftarkan dengan integrasi terkelola.

Parameter

`clientId(string)` - ID klien untuk contoh ini.

```
connect()
```

Menghubungkan `DeviceSDKClient` instance ke integrasi terkelola.

Pengembalian

- `IOTMI_STATUS_OK`- Koneksi berhasil.
- `IOTMI_STATUS_CUSTOM_PLUGIN_CONNECTION_ERROR`- Terjadi kesalahan saat menghubungkan ke integrasi terkelola.

Langganan tugas penyediaan

Gunakan metode ini untuk berlangganan tugas terkait penyediaan dari komponen integrasi terkelola.

```
iotmi_statusCode_t  
iotmi_provision_subscribe_to_tasks(DeviceSDKClient_SubscriberCallback callback, char*  
context)
```

Berlangganan tugas terkait penyediaan, seperti orientasi perangkat dan deprovisioning, dari komponen integrasi terkelola.

Parameter

- `callback(Device SDKClient _SubscriberCallback)` - Fungsi callback yang dijalankan saat tugas diterima.
- `context(char*)` - Konteks khusus diteruskan ke fungsi callback.

Pengembalian

- `IOTMI_STATUS_OK`- Langganan berhasil.
- `IOTMI_STATUS_CUSTOM_PLUGIN_CLIENT_NOT_CONNECTED`- SDKClient Instans Perangkat tidak terhubung ke integrasi terkelola.
- `IOTMI_STATUS_CUSTOM_PLUGIN_SUBSCRIBE_ERROR`- Terjadi kesalahan saat berlangganan tugas.

Publikasi tugas penyediaan

Gunakan metode ini untuk mempublikasikan permintaan terkait penyediaan ke komponen integrasi terkelola.

```
iotmi_statusCode_t iotmi_provision_publish_request(DataModel::iotmi_client_request_t  
request)
```

Menerbitkan permintaan terkait penyediaan ke komponen integrasi terkelola. Misalnya, peristiwa onboard perangkat atau status deprovisioning

Parameter

`request(DataModel: :iotmi_client_request_t)` - Sebuah pointer ke struktur permintaan yang berisi rincian.

Pengembalian

- `IOTMI_STATUS_OK`- Permintaan berhasil diterbitkan.

- `IOTMI_STATUS_CUSTOM_PLUGIN_CLIENT_NOT_CONNECTED`- DeviceSDKClient Instance tidak terhubung ke integrasi terkelola.
- `IOTMI_STATUS_INVALID_PARAMETER`- Satu atau lebih parameter dalam permintaan tidak valid.
- `IOTMI_STATUS_INVALID_JSON_OBJECT`- Payload permintaan bukan objek JSON yang valid.
- `IOTMI_STATUS_NO_MEMORY`- Terjadi kesalahan alokasi memori.

Kontrol langganan tugas

Gunakan metode ini untuk berlangganan tugas terkait kontrol dari komponen integrasi terkelola.

```
iotmi_statusCode_t iotmi_control_subscribe_to_tasks(DeviceSDKClient_SubscriberCallback  
callback, char context)
```

Berlangganan tugas terkait kontrol (misalnya, permintaan kontrol perangkat) dari komponen integrasi terkelola.

Parameter

- `callback(Device SDKClient _SubscriberCallback)` - Fungsi callback yang dijalankan saat tugas diterima.
- `context(char)` - Konteks khusus diteruskan ke fungsi callback.

Pengembalian

- `IOTMI_STATUS_OK`- Langganan berhasil.
- `IOTMI_STATUS_CUSTOM_PLUGIN_CLIENT_NOT_CONNECTED`- DeviceSDKClient Instance tidak terhubung ke integrasi terkelola.
- `IOTMI_STATUS_CUSTOM_PLUGIN_SUBSCRIBE_ERROR`- Terjadi kesalahan saat berlangganan tugas.

Kontrol publikasi tugas

Gunakan metode ini untuk mempublikasikan permintaan terkait kontrol ke komponen integrasi terkelola.

```
iotmi_statusCode_t iotmi_control_publish_request(DataModel::iotmi_client_request_t  
request)
```

Menerbitkan permintaan terkait kontrol ke komponen integrasi terkelola. Misalnya, peristiwa yang tidak diminta, permintaan perintah, atau kueri status perangkat.

Parameter

`request(DataModel: :iotmi_client_request_t)` - Sebuah pointer ke struktur permintaan yang berisi rincian.

Pengembalian

- `IOTMI_STATUS_OK`- Permintaan berhasil diterbitkan.
- `IOTMI_STATUS_CUSTOM_PLUGIN_CLIENT_NOT_CONNECTED`- SDKClient Instans Perangkat tidak terhubung ke integrasi terkelola.
- `IOTMI_STATUS_INVALID_PARAMETER`- Satu atau lebih parameter dalam permintaan tidak valid.
- `IOTMI_STATUS_INVALID_JSON_OBJECT`- Payload permintaan bukan objek JSON yang valid.
- `IOTMI_STATUS_NO_MEMORY`- Terjadi kesalahan alokasi memori.

Fitur logging

Gunakan metode ini untuk mengimplementasikan fitur logging yang disediakan integrasi terkelola.

Inisialisasi logger

```
void iotmi_devicesdk_log_init(const char* logger_name)
```

Anda harus menginisialisasi logger sebelum menggunakan fungsi logging apa pun.

Parameter

`logger_name`- Nama logger yang Anda tentukan. Nilai default adalah: `MyApplication`

Pencatatan makro

```
LOGGER_LOGD(...)
```

Gunakan makro ini dalam aplikasi Anda untuk pencatatan tingkat `DEBUG`.

```
LOGGER_LOGI(...)
```

Gunakan makro ini di aplikasi Anda untuk pencatatan tingkat `INFO`.

LOGGER_LOGW(. . .)

Gunakan makro ini di aplikasi Anda untuk pencatatan level WARN.

LOGGER_LOGE(. . .)

Gunakan makro ini di aplikasi Anda untuk pencatatan tingkat ERROR.

Note

Untuk informasi selengkapnya tentang fitur logging, lihat [Dokumentasi logging Hub](#). Plugin protokol khusus sepenuhnya mendukung semua fitur logging yang ditawarkan integrasi terkelola.

Lainnya APIs

```
std::string get_client_id()
```

Mengembalikan ID klien yang terkait dengan SDKClient instance Perangkat.

Pengembalian

ID klien.

Jenis Data

Bagian ini mendefinisikan tipe data yang digunakan untuk plugin protokol kustom.

iotmi_client_request_t

Merupakan permintaan untuk dipublikasikan ke komponen integrasi terkelola.

MessageType

Jenis pesan (CommonTypes: :C2 MIMessage Type). Daftar berikut menunjukkan nilai yang valid.

- C2MI_DEVICE_ONBOARDED: Menunjukkan pesan orientasi perangkat dengan muatan terkait.

- `C2MI_DE_PROVISIONING_PRE_ASSOCIATED_COMPLETE`: Menunjukkan pemberitahuan lengkap tugas de-penyediaan untuk perangkat yang telah dikaitkan sebelumnya.
- `C2MI_DE_PROVISIONING_ACTIVATED_COMPLETE`: Menunjukkan pemberitahuan lengkap tugas de-penyediaan untuk perangkat yang diaktifkan.
- `C2MI_DE_PROVISIONING_COMPLETE_RESPONSE`: Menunjukkan respons lengkap tugas de-penyediaan.
- `C2MI_CONTROL_EVENT`: Menunjukkan peristiwa kontrol dengan potensi perubahan status perangkat.
- `C2MI_CONTROL_SEND_COMMAND`: Menunjukkan perintah kontrol dari pengontrol lokal.
- `C2MI_CONTROL_SEND_DEVICE_STATE_QUERY`: Menunjukkan kueri status perangkat kontrol dari pengontrol lokal.

ReqPayload

Payload permintaan, biasanya string berformat JSON.

RSPPayload

Payload respons, diisi oleh komponen integrasi terkelola.

`iotmi_client_event_t`

Merupakan peristiwa yang diterima dari komponen integrasi terkelola.

`event_id`

Pengidentifikasi unik acara tersebut.

`panjang`

Panjang data acara.

`data`

Sebuah pointer ke data peristiwa, termasuk. `messageType` Daftar berikut menunjukkan nilai yang mungkin.

- `C2MI_PROVISION_UGS_TASK`: Menunjukkan tugas penyediaan untuk aliran UGS.
- `C2MI_PROVISION_SS_TASK`: Menunjukkan tugas penyediaan untuk SimpleSetup aliran.
- `C2MI_DE_PROVISION_PRE_ASSOCIATED_TASK`: Menunjukkan tugas de-penyediaan untuk perangkat yang telah dikaitkan sebelumnya.

- `C2MI_DE_PROVISION_ACTIVATED_TASK`: Menunjukkan tugas de-penyediaan untuk perangkat yang diaktifkan.
- `C2MI_DEVICE_ONBOARDED_RESPONSE`: Menunjukkan respons orientasi perangkat.
- `C2MI_CONTROL_TASK`: Menunjukkan tugas kontrol.
- `C2MI_CONTROL_EVENT_NOTIFICATION`: Menunjukkan pemberitahuan peristiwa kontrol untuk pengontrol lokal.

ctx

Konteks khusus yang terkait dengan acara tersebut.

Kontrol hub

Kontrol hub adalah ekstensi ke integrasi terkelola SDK perangkat akhir yang memungkinkannya berinteraksi dengan `MQTTPROXY` komponen di Hub SDK. Dengan kontrol hub, Anda dapat menerapkan kode menggunakan SDK perangkat Akhir dan mengontrol hub Anda melalui cloud integrasi terkelola sebagai perangkat terpisah. SDK kontrol hub akan disediakan sebagai paket terpisah dengan Hub SDK, diberi label sebagai `iot-managed-integrations-hub-control-x.x.x`

Topik

- [Prasyarat](#)
- [Akhir komponen SDK perangkat](#)
- [Integrasikan dengan SDK perangkat Akhir](#)
- [Contoh: Membangun kontrol hub](#)
- [Contoh yang didukung](#)
- [Platform yang didukung](#)

Prasyarat

Untuk mengatur kontrol hub, Anda memerlukan yang berikut ini:

- Hub onboard ke [Hub SDK](#), versi 0.4.0 atau lebih tinggi.
- Unduh versi terbaru [SDK perangkat Akhir](#) dari file. Konsol Manajemen AWS
- Komponen [proxy MQTT](#) yang berjalan di hub, versi 0.5.0 atau lebih tinggi.

Akhiri komponen SDK perangkat

Gunakan komponen berikut dari [End device SDK](#):

- Generator kode untuk model data
- Penangan model data

Karena Hub SDK sudah memiliki proses orientasi dan koneksi ke cloud, Anda tidak memerlukan komponen berikut:

- Penyedia
- Antarmuka PKCS
- Penangan pekerjaan
- Agen MQTT

Integrasikan dengan SDK perangkat Akhir

1. Ikuti petunjuk di [Generator Kode untuk Model Data](#) untuk menghasilkan kode C tingkat rendah.
2. Ikuti petunjuk dalam [Mengintegrasikan SDK perangkat Akhir](#) ke:
 - a. Siapkan lingkungan build

Buat kode di Amazon Linux 2023/x86_64 sebagai host pengembangan Anda. Instal dependensi build yang diperlukan:

```
dnf install make gcc gcc-c++ cmake
```

- b. Kembangkan fungsi panggilan balik perangkat keras

Sebelum menerapkan fungsi callback perangkat keras, pahami cara kerja API. Contoh ini menggunakan On/Off cluster dan OnOff atribut untuk mengontrol fungsi perangkat. Untuk detail API, lihat [Fungsi C tingkat rendah APIs](#).

```
struct DeviceState
{
    struct iotmiDev_Agent *agent;
    struct iotmiDev_Endpoint *endpointLight;
    /* This simulates the HW state of OnOff */
}
```

```

bool hwState;
};

/* This implementation for OnOff getter just reads
the state from the DeviceState */
iotmiDev_DMStatus exampleGetOnOff(bool *value, void *user)
{
    struct DeviceState *state = (struct DeviceState *) (user);
    *value = state->hwState;
    return iotmiDev_DMStatusOk;
}

```

- c. Siapkan titik akhir dan kaitkan fungsi panggilan balik perangkat keras

Setelah mengimplementasikan fungsi, buat titik akhir dan daftarkan callback Anda. Selesaikan tugas-tugas ini:

- i. Buat agen perangkat
- ii. Isi poin fungsi callback untuk setiap struct cluster yang ingin Anda dukung
- iii. Siapkan titik akhir dan daftarkan cluster yang didukung

```

struct DeviceState
{
    struct iotmiDev_Agent * agent;
    struct iotmiDev_Endpoint *endpoint1;

    /* OnOff cluster states*/
    bool hwState;
};

/* This implementation for OnOff getter just reads
the state from the DeviceState */
iotmiDev_DMStatus exampleGetOnOff( bool * value, void * user )
{
    struct DeviceState * state = ( struct DeviceState * ) ( user );
    *value = state->hwState;
    printf( "%s(): state->hwState: %d\n", __func__, state->hwState );
    return iotmiDev_DMStatusOk;
}

iotmiDev_DMStatus exampleGetOnTime( uint16_t * value, void * user )

```

```
{
    *value = 0;
    printf( "%s(): OnTime is %u\n", __func__, *value );
    return iotmiDev_DMStatusOk;
}

iotmiDev_DMStatus exampleGetStartupOnOff( iotmiDev_OnOff_StartUpOnOffEnum *
value, void * user )
{
    *value = iotmiDev_OnOff_StartUpOnOffEnum_Off;
    printf( "%s(): StartupOnOff is %d\n", __func__, *value );
    return iotmiDev_DMStatusOk;
}

void setupOnOff( struct DeviceState *state )
{
    struct iotmiDev_clusterOnOff clusterOnOff = {
        .getOnOff = exampleGetOnOff,
        .getOnTime = exampleGetOnTime,
        .getStartupOnOff = exampleGetStartupOnOff,
    };
    iotmiDev_OnOffRegisterCluster( state->endpoint1,
        &clusterOnOff,
        ( void * ) state);
}

/* Here is the sample setting up an endpoint 1 with OnOff
cluster. Note all error handling code is omitted. */
void setupAgent(struct DeviceState *state)
{
    struct iotmiDev_Agent_Config config = {
        .thingId = IOTMI_DEVICE_MANAGED_THING_ID,
        .clientId = IOTMI_DEVICE_CLIENT_ID,
    };
    iotmiDev_Agent_InitDefaultConfig(&config);

    /* Create a device agent before calling other SDK APIs */
    state->agent = iotmiDev_Agent_new(&config);

    /* Create endpoint#1 */
    state->endpoint1 = iotmiDev_Agent_addEndpoint( state->agent,
        1,
```

```
Device",  
    { "Camera" },  
    setupOnOff(state);  
}  
"Data Model Handler Test  
(const char*[])  
1 );
```

Contoh: Membangun kontrol hub

Kontrol hub disediakan sebagai bagian dari paket Hub SDK. Sub-paket kontrol hub diberi label `iot-managed-integrations-hub-control-x.x.x` dan berisi pustaka yang berbeda dari SDK perangkat yang tidak dimodifikasi.

1. Pindahkan kode yang dihasilkan file ke `example` folder:

```
cp codegen/out/* example/dm
```

2. Untuk membangun kontrol hub, jalankan perintah berikut:

```
cd <hub-control-root-folder>
```

```
mkdir build
```

```
cd build
```

```
cmake -DBUILD_EXAMPLE_WITH_MQTT_PROXY=ON -  
DIOTMI_USE_MANAGED_INTEGRATIONS_DEVICE_LOG=ON ..
```

```
cmake -build .
```

3. Jalankan contoh dengan `MQTTProxy` komponen di hub, dengan `MQTTProxy` komponen `HubOnboarding` dan berjalan.

```
./examples/iotmi_device_sample_camera/iotmi_device_sample_camera
```

Lihat [Model data integrasi terkelola](#) untuk model data. Ikuti Langkah 5 [Memulai dengan End device SDK](#) untuk mengatur titik akhir dan mengelola komunikasi antara pengguna akhir dan `iot-managed-integrations`

Contoh yang didukung

Contoh berikut telah dibangun dan diuji:

- `iotmi_device_dm_air_purifier_demo`
- `iotmi_device_basic_diagnostics`
- `iotmi_device_dm_camera_demo`

Platform yang didukung

Tabel berikut menampilkan platform yang didukung untuk kontrol hub.

Arsitektur	Sistem operasi	Versi GCC	Versi Binutils
X86_64	Linux	10.5.0	2.37
aarch64	Linux	10.5.0	2.37

Aktifkan CloudWatch Log

Hub SDK menyediakan fungsionalitas pencatatan yang komprehensif. Secara default, Hub SDK menulis log ke sistem file lokal. Namun, Anda dapat memanfaatkan cloud API untuk mengonfigurasi streaming CloudWatch log ke Log, yang menawarkan:

- **Pantau kinerja perangkat:** Tangkap log runtime terperinci untuk manajemen perangkat proaktif. Aktifkan analisis log lanjutan dan pemantauan di seluruh armada perangkat Anda
- **Memecahkan masalah:** Hasilkan entri log granular untuk analisis diagnostik cepat. Rekam sistem dan acara tingkat aplikasi untuk penyelidikan mendalam.
- **Pencatatan yang fleksibel dan terpusat:** Manajemen log jarak jauh tanpa akses perangkat langsung. Agregat log dari beberapa perangkat dalam satu repositori yang dapat dicari.

Prasyarat

- Onboard perangkat yang dikelola ke cloud. Lihat [Pengaturan orientasi hub](#) untuk detail.
- Verifikasi startup agen Hub dan inialisasi yang berhasil. Lihat [Instal dan validasi integrasi terkelola Hub SDK](#) untuk detail.

Note

Untuk membuat konfigurasi logging, lihat [PutRuntimeLogConfiguration API](#) untuk detailnya.

Warning

Mengaktifkan log dihitung terhadap pengukuran kuota berjenjang. Peningkatan level log akan menghasilkan volume pesan yang lebih tinggi dan biaya tambahan.

Pengaturan konfigurasi log SDK Hub

Konfigurasi setelan log SDK hub dengan memanggil API untuk menyiapkan konfigurasi log runtime.

Example contoh Permintaan API

```
aws iot-managed-integrations put-runtime-log-configuration \  
  --managed-thing-id MANAGED_THING_ID \  
  --runtime-log-configurations LogLevel=DEBUG,UploadLog=TRUE
```

RuntimeLogConfigurations atribut

Atribut berikut bersifat opsional dan dapat dikonfigurasi di RuntimeLogConfigurations API.

LogLevel

Menetapkan tingkat keparahan minimum untuk jejak runtime. Nilai: DEBUG, ERROR, INFO, WARN

Default: WARN (rilis build)

LogFlushLevel

Menentukan tingkat keparahan untuk pembilasan data langsung ke penyimpanan lokal. Nilai: DEBUG, ERROR, INFO, WARN

Default: DISABLED

LocalStoreLocation

Menentukan lokasi penyimpanan untuk jejak runtime. Default: `/var/log/awsiotmi`

- Log aktif: `/var/log/awsiotmi/ManagedIntegrationsDeviceSdkHub.log`
- Log yang diputar: `/var/log/awsiotmi/ManagedIntegrationsDeviceSdkHub.N.log` (N menunjukkan urutan rotasi)

LocalStoreFileRotationMaxBytes

Memicu rotasi file ketika file saat ini melebihi ukuran yang ditentukan.

Important

Untuk efisiensi optimal, pertahankan ukuran file di bawah 125 KB. Nilai di atas 125 KB akan dibatasi secara otomatis.

LocalStoreFileRotationMaxFiles,

Menetapkan jumlah maksimum file rotasi yang diizinkan oleh daemon log.

UploadLog

Mengontrol transfer jejak runtime ke cloud. Log disimpan dalam grup `/aws/iotmanagedintegration` CloudWatch Log.

Default: `false`.

UploadPeriodMinutes

Mendefinisikan frekuensi unggahan runtime trace. Default: 5

DeleteLocalStoreAfterUpload

Mengontrol penghapusan file setelah diunggah. Default: `true`

Note

Jika disetel ke false, file yang diunggah diganti namanya menjadi: `/var/log/awsiotmi/ManagedIntegrationsDeviceSdkHub.uploaded.{uploaded_timestamp}`

Contoh file log

Lihat contoh file CloudWatch Log di bawah ini:

Jenis perangkat Zigbee dan Z-Wave yang didukung

Halaman ini mencantumkan jenis perangkat yang terhubung ke hub yang telah diuji dengan integrasi terkelola dan didukung. Integrasi terkelola mendukung keduanya [Pengaturan sederhana \(SS\)](#) dan [Pengaturan yang dipandu pengguna \(UGS\)](#) untuk perangkat ini.

Tabel ini mencantumkan perangkat Zigbee yang didukung.

Jenis perangkat Zigbee	Kemampuan yang didukung
Bola lampu pintar/Lampu yang dapat diredupkan/Lampu RGB	OnOff, LevelControl, ColorControl
Steker pintar	OnOff
Sakelar pintar	OnOff
Strip LED	OnOff, LevelControl, ColorControl
Katup air	OnOff
Katup radiator	Termostat OnOff, Timer
Termostat	Termostat,, FanControl OnOff, Timer
Pembuka pintu garasi	WindowCovering, OnOff, LevelControl

Jenis perangkat Zigbee	Kemampuan yang didukung
Alarm asap	BooleanState, OnOff, TemperatureMeasurement, Timer, Asap COAlarm
Sensor gerak	BooleanState
Sensor hunian/keberadaan manusia	BooleanState, OccupancySensing
Sensor pintu dan jendela	BooleanState
Sensor kebocoran air	BooleanState
Sensor getaran	BooleanState
Sensor suhu dan kelembaban	TemperatureMeasurement, RelativeHumidityMeasurement

Tabel ini mencantumkan perangkat Z-Wave yang didukung.

Jenis perangkat Z-Wave	Kemampuan yang didukung
Bola lampu pintar/Lampu yang dapat diredupkan	OnOff, LevelControl
Steker pintar	OnOff
Pengontrol pintu garasi	OnOff, LevelControl
Meteran energi	ElectricalEnergyMeasurement, ElectricalPowerMeasurement
Baterai	LevelControl
Sirene	LevelControl
Sensor gerak	BooleanState
Sensor pintu dan jendela	BooleanState

Jenis perangkat Z-Wave	Kemampuan yang didukung
Sensor kebocoran air	BooleanState
Sensor suhu	TemperatureMeasurement
Sensor CO	Asap COAlarm
Sensor asap	Asap COAlarm

Jalankan integrasi terkelola pada Raspberry Pi

Note

Implementasi AWS IoT Hub SDK pada Raspberry Pi ini adalah proyek demonstrasi yang ditujukan untuk tujuan pembelajaran dan pengujian saja dan tidak dimaksudkan untuk digunakan di lingkungan produksi. Untuk keperluan demo ini, atur konfigurasi berikut untuk kemudahan pengembangan:

AWS penyimpanan kredensial: Hanya untuk tujuan demo, kredensial dan sertifikat disimpan di lokasi yang dapat diakses untuk pengujian dan pengembangan yang lebih mudah. Lingkungan produksi harus menggunakan solusi penyimpanan aman seperti AWS Secrets Manager, atau Systems Manager Parameter Store. Mereka harus menerapkan enkripsi saat istirahat, dan mengikuti pedoman AWS IoT keamanan.

Hak istimewa kontainer: Demo berjalan dengan hak istimewa yang ditingkatkan untuk memungkinkan akses tidak terbatas ke sumber daya host dan menyederhanakan alur kerja pengembangan. Dalam produksi, kontainer harus beroperasi dengan hak istimewa minimal yang diperlukan.

Konfigurasi jembatan jaringan: Demo menggunakan konfigurasi jembatan jaringan yang mengekspos lalu lintas jaringan internal untuk memudahkan debugging dan pemantauan. Dalam lingkungan produksi, terapkan isolasi dan segmentasi jaringan yang tepat untuk mencegah akses tidak sah ke lalu lintas jaringan internal.

Izin perangkat USB: Akses perangkat USB yang tidak dibatasi diaktifkan untuk memudahkan koneksi periferal pengembangan dan perangkat pengujian. Untuk produksi, terapkan kontrol dan validasi perangkat USB yang ketat untuk mencegah serangan spoofing perangkat.

Konfigurasi ini memungkinkan pengujian langsung dan tidak boleh digunakan di lingkungan produksi. Saat menerapkan ke produksi, ikuti praktik terbaik keamanan untuk mencegah kompromi sistem host dan akses tidak sah ke kredensial.

Sebagai prasyarat, Anda harus mengatur dongle USB Sonoff Zigbee sebelum mengatur Raspberry Pi.

Flash firmware ke dongle USB Sonoff Zigbee

Prasyarat

- [Dongle USB Sonoff Zigbee](#)
- Windows: Instal [CP210x Driver Windows Universal](#)

Flash firmware

1. Unduh [Zigbee Dongle Firmware](#) Build 7.4.1.0.
2. Buka [Silabs Firmware Flasher](#).
3. Hubungkan Sonoff Zigbee USB Dongle ke komputer Anda.
4. Gulir dan temukan ZBDongle-E.
5. Pilih Hubungkan.
6. Tunggu hingga perangkat terhubung.
7. Pilih Ubah firmware.
8. Pilih Unggah firmware Anda sendiri.
9. Temukan lokasi [Zigbee Dongle Firmware Build 7.4.1.0](#) unduh dan pilih.
10. Klik Instal.
11. Tunggu firmware dipasang.
12. Pilih Lanjutkan saat instalasi selesai.

Dongle sekarang siap digunakan.

Pilih di antara opsi yang tercantum di bawah ini untuk menjalankan integrasi terkelola Hub SDK pada Raspberry Pi Anda. Langkah-langkah penyiapan dan validasi untuk kedua pendekatan tercantum di bawah ini.

Topik

- [Integrasi terkelola gambar Hub SDK pada Raspberry Pi](#)
- [Integrasi terkelola wadah Hub SDK Docker di Raspberry Pi](#)
- [Aplikasi demo integrasi terkelola](#)

Integrasi terkelola gambar Hub SDK pada Raspberry Pi

Note

Implementasi AWS IoT Hub SDK pada Raspberry Pi ini adalah proyek demonstrasi yang ditujukan hanya untuk tujuan pembelajaran dan pengujian dan tidak dimaksudkan untuk digunakan di lingkungan produksi. Untuk keperluan demo ini, atur konfigurasi berikut untuk kemudahan pengembangan:

AWS penyimpanan kredensial: Hanya untuk tujuan demo, kredensial dan sertifikat disimpan di lokasi yang dapat diakses untuk pengujian dan pengembangan yang lebih mudah. Lingkungan produksi harus menggunakan solusi penyimpanan aman seperti AWS Secrets Manager, atau Systems Manager Parameter Store. Mereka harus menerapkan enkripsi saat istirahat, dan mengikuti pedoman AWS IoT keamanan.

Hak istimewa kontainer: Demo berjalan dengan hak istimewa yang ditingkatkan untuk memungkinkan akses tidak terbatas ke sumber daya host dan menyederhanakan alur kerja pengembangan. Dalam produksi, kontainer harus beroperasi dengan hak istimewa minimal yang diperlukan.

Konfigurasi jembatan jaringan: Demo menggunakan konfigurasi jembatan jaringan yang mengekspos lalu lintas jaringan internal untuk memudahkan debugging dan pemantauan. Dalam lingkungan produksi, terapkan isolasi dan segmentasi jaringan yang tepat untuk mencegah akses tidak sah ke lalu lintas jaringan internal.

Izin perangkat USB: Akses perangkat USB yang tidak dibatasi diaktifkan untuk memudahkan koneksi periferal pengembangan dan perangkat pengujian. Untuk produksi, terapkan kontrol dan validasi perangkat USB yang ketat untuk mencegah serangan spoofing perangkat.

Konfigurasi ini memungkinkan pengujian langsung dan tidak boleh digunakan di lingkungan produksi. Saat menerapkan ke produksi, ikuti praktik terbaik keamanan untuk mencegah kompromi sistem host dan akses tidak sah ke kredensi.

Prasyarat

Lengkapi persyaratan ini sebelum menerapkan gambar Raspberry Pi:

- Unduh dan instal [Raspberry Pi imager](#).
- Dapatkan [Kartu SD](#).
- Siapkan [Raspberry Pi 5 dengan CPU quad-core 2.4Ghz 64-bit \(RAM 8GB\)](#).
- Hubungkan [Dongle USB Sonoff Zigbee](#).
- [Flash firmware ke dongle USB Sonoff Zigbee](#).
- Hubungkan Dongle [Silicon Labs SLUSB001A](#).
- [Mendaftar untuk sebuah AWS akun](#).
- Instal versi terbaru [AWS CLI dari Referensi AWS CLI Perintah Integrasi Terkelola](#).

Flash gambar Raspberry Pi pada kartu SD baru

Flash gambar Integrasi terkelola ke kartu SD Anda menggunakan langkah-langkah berikut:

1. Unduh [Integrasi Terkelola Raspberry Pi Hub SDK Image](#).
2. Luncurkan Raspberry Pi Imager di desktop Anda.
3. Masukkan kartu SD ke pembaca kartu SD bawaan komputer Anda, atau pembaca kartu USB eksternal.
4. Pilih Pilih Perangkat → Raspberry Pi 5.
5. Pilih Pilih OS → Gunakan kustom → Temukan lotMI-HubSDK-RPi-Image file -v1.0.0.img.gz → Buka.
6. Pilih Pilih penyimpanan → Pilih Pembaca Kartu SD Anda.
7. Verifikasi konfigurasi Anda cocok dengan layar berikut:
8. Klik Berikutnya.
9. Konfigurasi pengaturan kustomisasi OS:
 - Nama host: Pilih raspberrypi.

- Nama Pengguna dan Kata Sandi:
 - Aktifkan Tetapkan nama pengguna dan kata sandi:
 - Untuk Nama Pengguna:, masukanhub123456.
 - Untuk Kata Sandi:, masukansh123456.
- LAN nirkabel:
 - Aktifkan Konfigurasi LAN nirkabel.
 - Masukkan SSID router Anda, dan kata sandi.

Contoh pengaturan:

- SSID: iotmi-tplink
- Kata sandi: ***** (Minimal 8 karakter)
- Tetapkan Negara: keUS.
- Setel Pengaturan Lokal:
 - Atur Zona waktu: keAmerica/Los Angeles.
 - Setel Tata Letak Keyboard: keUS.
- SSH:
 - Pilih tab layanan.
 - Periksa Aktifkan SSH.
 - Pilih Gunakan otentikasi kata sandi.

10.Konfirmasikan semua popup untuk kustomisasi OS dan penghapusan data.

11.Tunggu proses penulisan selesai.

12.Verifikasi penyelesaian yang berhasil dengan layar berikut:

13.Klik Lanjutkan.

14.Keluarkan kartu SD dan masukkan ke dalam Raspberry Pi Anda.

Jalankan Hub SDK pada Raspberry Pi

Mulai layanan Hub SDK pada Raspberry Pi Anda yang dikonfigurasi:

1. Masukkan kartu SD yang disiapkan ke perangkat Raspberry Pi 5.

2. Hubungkan Sonoff Zigbee USB Dongle dan Silicon Labs SLUSB001A Dongle ke Raspberry Pi.


3. Nyalakan Raspberry Pi.
4. Pastikan Raspberry Pi dan komputer Anda (dari mana Anda SSH) berada di jaringan yang sama.
5. SSH ke Raspberry Pi menggunakan kredensial yang Anda tetapkan selama penerapan gambar.

```
ssh username@hostname
```

6. Arahkan ke direktori SDK hub:

```
cd /data/aws/iotmi
```

7. Selesaikan [pengaturan orientasi Hub](#) untuk menambahkan materi otentikasi dan konfigurasi.

 Note

Anda harus berada di YUL atau DUB wilayah untuk melakukan langkah ini.

8. Jalankan Hub SDK:

```
cd /data/aws/iotmi
bash start_hub_sdk.sh
```

Sistem menampilkan respons berikut untuk memulai Hub SDK yang berhasil:

```
-----Stopping SDK running processes---
-----Starting Hub SDK-----
-----Creating logs directory-----
Logs directory created.
-----Verifying Middleware paths-----
All middleware libraries exist
-----Verifying Middleware pre reqs---
AIPC and KVstroage directories exist
-----Starting HubOnboarding-----
-----Starting MQTT Proxy-----
-----Staring Log Daemon---
-----Starting Event Manager-----
-----Starting Zigbee Service-----
--Checking Zigbee network information--
-----Starting Zwave Service-----
/data/aws/iotmi/middleware/AceZwave/bin /data/aws/iotmi
/data/aws/iotmi
```

```
-----Starting CDMB-----
-----Starting Agent-----
-----Starting Provisioner-----
-----Checking SDK status-----
hub1234+    1780  0.2  0.1 1093936 16368 pts/1    Sl+  16:34   0:00 ./iotmi_mqtt_proxy -
C /data/aws/iotmi/config/iotmi_config.json
Process 'iotmi_mqtt_proxy' is running.
hub1234+    1884  0.0  0.0 236272  2624 pts/1    Sl+  16:34   0:00 ./middleware/
AceCommon/bin/ace_eventmgr
Process 'ace_eventmgr' is running.
hub1234+    1892  9.1  0.1 393040  8352 pts/1    Sl+  16:34   0:04 ./middleware/
AceZigbee/bin/ace_zigbee_service
Process 'ace_zigbee_service' is running.
hub1234+    1923  0.0  0.1 1570736 12736 pts/1    Sl+  16:34   0:00 ./zwave_svc
Process 'zwave_svc' is running.
hub1234+    1958  0.0  0.0 1067632 5776 pts/1    Sl+  16:34   0:00 ./iotmi_cdmb
Process 'iotmi_cdmb' is running.
hub1234+    2001  0.2  0.2 2017712 21264 pts/1    Sl+  16:35   0:00 ./iotmi_device_agent
Process 'iotmi_device_agent' is running.
hub1234+    2045  0.0  0.1 1457824 12624 pts/1    Sl+  16:35   0:00 ./
iotmi_lpw_provisioner
Process 'iotmi_lpw_provisioner' is running.
hub1234+    1813  0.0  0.0 875152  6848 pts/1    Sl+  16:34   0:00 ./iotmi_log_daemon
Process 'iotmi_log_daemon' is running.
-----Successfully Started Hub SDK----
```

Langkah berikutnya

Setelah berhasil memulai Hub SDK, lanjutkan dengan orientasi dan pengelolaan perangkat di

[Penyiapan yang dipandu pengguna untuk onboard dan mengoperasikan perangkat](#)

Integrasi terkelola wadah Hub SDK Docker di Raspberry Pi

Note

Implementasi AWS IoT Hub SDK pada Raspberry Pi ini adalah proyek demonstrasi yang ditujukan untuk tujuan pembelajaran dan pengujian saja dan tidak dimaksudkan untuk digunakan di lingkungan produksi. Untuk keperluan demo ini, atur konfigurasi berikut untuk kemudahan pengembangan:

AWS penyimpanan kredensial: Hanya untuk tujuan demo, kredensial dan sertifikat disimpan di lokasi yang dapat diakses untuk pengujian dan pengembangan yang lebih mudah. Lingkungan produksi harus menggunakan solusi penyimpanan yang aman seperti

AWS Secrets Manager, atau Systems Manager Parameter Store. Mereka harus menerapkan enkripsi saat istirahat, dan mengikuti pedoman AWS IoT keamanan.

Hak istimewa kontainer: Demo berjalan dengan hak istimewa yang ditingkatkan untuk memungkinkan akses tidak terbatas ke sumber daya host dan menyederhanakan alur kerja pengembangan. Dalam produksi, kontainer harus beroperasi dengan hak istimewa minimal yang diperlukan.

Konfigurasi jembatan jaringan: Demo menggunakan konfigurasi jembatan jaringan yang mengekspos lalu lintas jaringan internal untuk memudahkan debugging dan pemantauan. Dalam lingkungan produksi, terapkan isolasi dan segmentasi jaringan yang tepat untuk mencegah akses tidak sah ke lalu lintas jaringan internal.

Izin perangkat USB: Akses perangkat USB yang tidak dibatasi diaktifkan untuk memudahkan koneksi periferal pengembangan dan perangkat pengujian. Untuk produksi, terapkan kontrol dan validasi perangkat USB yang ketat untuk mencegah serangan spoofing perangkat.

Konfigurasi ini memungkinkan pengujian langsung dan tidak boleh digunakan di lingkungan produksi. Saat menerapkan ke produksi, ikuti praktik terbaik keamanan untuk mencegah kompromi sistem host dan akses tidak sah ke kredensi.

Prasyarat

Prasyarat berikut diperlukan untuk wadah docker.

- Unduh dan instal [Raspberry Pi imager](#).
- Dapatkan [Kartu SD](#).
- Siapkan [Raspberry Pi 5 dengan CPU quad-core 2.4Ghz 64-bit \(RAM 8GB\)](#).
- Hubungkan [Dongle USB Sonoff Zigbee](#).
- [Flash firmware ke dongle USB Sonoff Zigbee](#).
- Hubungkan Dongle [Silicon Labs SLUSB001A](#).
- [Mendaftar untuk sebuah AWS akun](#).
- Instal versi terbaru [AWS CLI dari Referensi AWS CLI Perintah Integrasi Terkelola](#).
- Akses SSH ke Raspberry Pi dengan alamat IP atau nama host.

Gunakan integrasi terkelola wadah Hub SDK Docker di Raspberry Pi

1. Unduh [integrasi terkelola Raspberry Pi Hub SDK Docker](#).

2. Salin file ke Raspberry Pi menggunakan SCP:

```
scp ~/path/to/IotMI-HubSDK-Docker-v1.0.0.tar.gz [username]@raspberrypi.local:~
```

3. Connect ke Raspberry Pi melalui SSH:

```
ssh hub123456@raspberrypi.local
```

4. Instal Docker jika tidak ada:

```
# Install Docker
cd
curl -fsSL https://get.docker.com | sudo sh

# Add your user to docker group
sudo usermod -aG docker $USER
exit # exit ssh

# Log in again
```

5. Instal Docker Compose jika tidak ada:

```
# Install Docker Compose
sudo apt-get update
sudo apt-get install -y docker-compose-plugin
```

6. Ekstrak file Hub SDK:

```
# Navigate to the home directory
cd

# Extract the hub-docker.tar.gz file
tar -xzf IotMI-HubSDK-Docker-v1.0.0.tar.gz
```

7. Arahkan ke direktori hub-docker:

```
cd IotMI-HubSDK-Docker
```

8. Selesaikan [pengaturan orientasi Hub](#) untuk mengonfigurasi otentikasi dan pengaturan.


```
hub-docker/iotmi
```

- Untuk memulai ulang kontainer dengan aman sambil menyimpan data, lakukan:

```
docker compose down  
docker compose up -d  
docker compose logs -f
```

Aplikasi demo integrasi terkelola

Note

Implementasi AWS IoT Hub SDK pada Raspberry Pi ini adalah proyek demonstrasi yang ditujukan untuk tujuan pembelajaran dan pengujian saja dan tidak dimaksudkan untuk digunakan di lingkungan produksi. Untuk keperluan demo ini, atur konfigurasi berikut untuk kemudahan pengembangan:

AWS penyimpanan kredensial: Hanya untuk tujuan demo, kredensial dan sertifikat disimpan di lokasi yang dapat diakses untuk pengujian dan pengembangan yang lebih mudah. Lingkungan produksi harus menggunakan solusi penyimpanan yang aman seperti AWS Secrets Manager, atau Systems Manager Parameter Store. Mereka harus menerapkan enkripsi saat istirahat, dan mengikuti pedoman AWS IoT keamanan.

Hak istimewa kontainer: Demo berjalan dengan hak istimewa yang ditingkatkan untuk memungkinkan akses tidak terbatas ke sumber daya host dan menyederhanakan alur kerja pengembangan. Dalam produksi, kontainer harus beroperasi dengan hak istimewa minimal yang diperlukan.

Konfigurasi jembatan jaringan: Demo menggunakan konfigurasi jembatan jaringan yang mengekspos lalu lintas jaringan internal untuk memudahkan debugging dan pemantauan. Dalam lingkungan produksi, terapkan isolasi dan segmentasi jaringan yang tepat untuk mencegah akses tidak sah ke lalu lintas jaringan internal.

Izin perangkat USB: Akses perangkat USB yang tidak dibatasi diaktifkan untuk memudahkan koneksi periferal pengembangan dan perangkat pengujian. Untuk produksi, terapkan kontrol dan validasi perangkat USB yang ketat untuk mencegah serangan spoofing perangkat.

Konfigurasi ini memungkinkan pengujian langsung dan tidak boleh digunakan di lingkungan produksi. Saat menerapkan ke produksi, ikuti praktik terbaik keamanan untuk mencegah kompromi sistem host dan akses tidak sah ke kredensi.

Aplikasi demo adalah aplikasi demo berbasis React yang menampilkan kemampuan integrasi Terkelola untuk manajemen perangkat rumah pintar. Aplikasi ini menunjukkan orientasi perangkat, kontrol, dan pemantauan untuk perangkat Z-Wave dan Zigbee melalui antarmuka web modern.

Prasyarat

- [Mendaftar untuk sebuah AWS akun.](#)
- [Buat loker kredenal dan tambahkan loker kredenal ke](#) hub Anda.
- Selesaikan [pengaturan orientasi Hub](#).
- [Node.js 18+ dan npm.](#)
- Instal versi terbaru [AWS CLI dari Referensi AWS CLI Perintah Integrasi Terkelola](#).
- Peramban web modern (Chrome, Firefox, Safari, Edge)

Instal dan konfigurasi Aplikasi

1. Unduh [aplikasi demo integrasi terkelola](#).
2. Ekstrak paket:

```
cd ~/Downloads
tar -xzf IotMI-HubSDK-DemoApp-v1.0.0.tar.gz
cd IotManagedIntegrations-DemoApp
```

3. Instal dependensi:

```
npm install
```

4. Buat `.env` file di direktori root:

```
# AWS Configuration
REACT_APP_AWS_REGION=your_region
REACT_APP_AWS_ACCESS_KEY_ID=your_access_key
REACT_APP_AWS_SECRET_ACCESS_KEY=your_secret_key
REACT_APP_AWS_SESSION_TOKEN=your_session_token

# IoT Managed Integrations Endpoint
REACT_APP_IOT_ENDPOINT=https://your-iot-endpoint.amazonaws.com

# Hub Configuration
```

```
REACT_APP_HUB_MANAGED_THING_ID=your_hub_id  
REACT_APP_CREDENTIAL_LOCKER_ID=your_credential_locker_id
```

5. Bangun dan mulai aplikasi:

```
npm start
```

6. Akses aplikasi di:

```
http://localhost:3000
```

Untuk informasi harga, lihat [bagian Integrasi terkelola di halaman harga Manajemen AWS IoT Perangkat](#).

Hub integrasi terkelola offboard

Ikhtisar proses offboard Hub SDK

Proses offboarding hub menghapus hub dari sistem AWS Cloud manajemen. Ketika cloud mengirimkan [DeleteManagedThing](#) permintaan, proses menyelesaikan dua tujuan utama:

Tindakan sisi perangkat:

- Setel ulang status internal hub
- Hapus semua data yang disimpan secara lokal
- Siapkan perangkat untuk orientasi ulang future potensial

Tindakan sisi awan:

- Hapus semua sumber daya cloud yang terkait dengan hub
- Pemutusan lengkap dari akun sebelumnya

Pelanggan biasanya memulai hub offboarding ketika:

- Mengubah akun terkait hub
- Mengganti hub yang ada dengan perangkat baru

Proses ini memastikan transisi yang bersih dan aman antara konfigurasi hub, memungkinkan manajemen perangkat yang mulus dan fleksibilitas akun.

Prasyarat

- Anda harus memiliki hub yang terpasang. Untuk petunjuk, lihat [Pengaturan orientasi Hub](#).
- Dalam `iotmi_config.json` file yang terletak di `/data/aws/iotmi/config/`, verifikasi yang `iot_provisioning_state` menunjukkan `PROVISIONED`.
- Konfirmasikan bahwa sertifikat dan kunci permanen yang direferensikan `iotmi_config.json` ada di jalur yang ditentukan.
- Pastikan bahwa HubOnboarding, Agen, Penyedia, dan proxy MQTT dikonfigurasi dan berjalan dengan benar.
- Verifikasi bahwa hub tidak memiliki perangkat anak. Gunakan [DeleteManagedThing](#) API untuk menghapus semua perangkat anak sebelum melanjutkan.

Proses offboard Hub SDK

Ikuti langkah-langkah ini untuk melepaskan hub:

Ambil ID `hub_managed_thing`

`iotmi_config.json` file ini digunakan untuk menyimpan ID hal terkelola untuk hub integrasi Terkelola. Pengenal ini adalah bagian penting dari informasi yang memungkinkan hub untuk berkomunikasi dengan layanan Integrasi AWS IoT Terkelola. ID hal yang dikelola disimpan dalam bagian `rw` (baca-tulis) dari file JSON, di bawah bidang `managed_thing_id` Ini terlihat pada konfigurasi sampel berikut:

```
{
  "ro": {
    "iot_provisioning_method": "FLEET_PROVISIONING",
    "iot_claim_cert_path": "PATH",
    "iot_claim_pk_path": "PATH",
    "UPC": "UPC",
    "sh_endpoint_url": "ENDPOINT_URL",
    "SN": "SN",
    "fp_template_name": "TEMPLATENAME"
  },
}
```

```
    "rw": {
      "iot_provisioning_state": "PROVISIONED",
      "client_id": "ID",
      "managed_thing_id": "ID",
      "iot_permanent_cert_path": "CERT_PATH",
      "iot_permanent_pk_path": "KEY",
      "metadata": {
        "last_updated_epoch_time": 1747766125
      }
    }
  }
}
```

Kirim perintah ke hub offboard

Gunakan kredensi akun Anda dan jalankan perintah dengan yang `managed_thing_id` diambil di bagian sebelumnya:

```
aws iot-managed-integrations delete-managed-thing \
  --identifier HUB_MANAGED_THING_ID
```

Verifikasi hub telah di-offboard

Gunakan kredensi akun Anda dan jalankan perintah dengan yang `managed_thing_id` diambil di bagian sebelumnya:

```
aws iot-managed-integrations get-managed-thing \
  --identifier HUB_MANAGED_THING_ID
```

Skenario sukses dan gagal

Skenario sukses

Jika perintah untuk melepaskan hub berhasil, respons sampel berikut diharapkan:

```
{
  "Message" : "Managed Thing resource not found."
}
```

Selain itu, sampel berikut `iotmi_config.json` akan diamati jika perintah hub offboarding berhasil. Verifikasi bahwa bagian `rw` hanya berisi metadata **`iot_provisioning_state`** dan opsional. Tidak adanya metadata dapat diterima. `iot_provisioning_state` harus `NOT_PROVISIONED`.

```
{
  "ro": {
    "iot_provisioning_method": "FLEET_PROVISIONING",
    "iot_claim_cert_path": "PATH",
    "iot_claim_pk_path": "PATH",
    "UPC": "1234567890101",
    "sh_endpoint_url": "ENDPOINT_URL",
    "SN": "1234567890101",
    "fp_template_name": "test-template"
  },
  "rw": {
    "iot_provisioning_state": "NOT_PROVISIONED",
    "metadata": {
      "last_updated_epoch_time": 1747766125
    }
  }
}
```

Skenario kegagalan

Jika perintah untuk melepaskan hub tidak berhasil, respons sampel berikut diharapkan:

```
{
  "Arn" : "ARN",
  "CreatedAt" : 1.748968266655E9,
  "Id" : "ID",
  "ProvisioningStatus" : "DELETE_IN_PROGRESS",
  "Role" : "CONTROLLER",
  "SerialNumber" : "SERIAL_NO",
  "Tags" : { },
  "UniversalProductCode" : "UPC",
  "UpdatedAt" : 1.748968272107E9
}
```

- Jika ProvisioningStatusyaDELETE_IN_PROGRESS, ikuti petunjuk dalam [pemulihan Hub](#).
- Jika ProvisioningStatustidakDELETE_IN_PROGRESS, perintah untuk melepaskan hub gagal di cloud Integrasi terkelola, atau tidak diterima oleh cloud integrasi Terkelola. Ikuti petunjuk dalam [pemulihan Hub](#).
- Jika offboarding tidak berhasil, `iotmi_config.json` file Anda akan terlihat seperti file contoh di bawah ini.

```
{
  "ro": {
    "iot_provisioning_method": "FLEET_PROVISIONING",
    "iot_claim_cert_path": "PATH",
    "iot_claim_pk_path": "PATH",
    "UPC": "123456789101",
    "sh_endpoint_url": "ENDPOINT_URL",
    "SN": "123456789101",
    "fp_template_name": "test-template"
  },
  "rw": {
    "iot_provisioning_state": "PROVISIONED",
    "client_id": "ID",
    "managed_thing_id": "ID",
    "iot_permanent_cert_path": "PATH",
    "iot_permanent_pk_path": "PATH",
    "metadata": {
      "last_updated_epoch_time": 1747766125
    }
  }
}
```

(Opsional) Setelah offboarding Hub SDK

Important

Skenario berikut mencantumkan tindakan opsional yang harus dilakukan setelah offboarding Hub SDK gagal, atau jika Anda ingin melakukan onboard ulang hub setelah offboarding.

Masuk kembali

Jika offboarding berhasil, ikuti Hub SDK Anda dengan mengikuti [Langkah 3: Buat hal yang dikelola \(penyediaan armada\)](#), dan proses onboard lainnya.

Pemulihan hub

Keberhasilan offboarding hub perangkat dan offboarding Cloud gagal

Jika panggilan [GetManagedThing](#) API tidak mengembalikan Managed Thing resource not found pesan, tetapi file tersebut `iotmi_config.json` di-offboard. Lihat [skenario Sukses](#) untuk contoh file json.

Untuk memulihkan dari skenario ini, lihat [Penghapusan paksa](#).

Offboarding hub perangkat gagal

Skenario ini adalah ketika file `iotmi_config.json` tidak di-offboard dengan benar. Lihat [Skenario kegagalan](#) untuk contoh file json.

Untuk memulihkan dari skenario ini, lihat [Penghapusan paksa](#). Jika `iotmi_config.json` masih belum di-offboard, hub harus di-reset pabrik.

Offboarding hub perangkat dan offboarding Cloud gagal

Dalam skenario ini, `iotmi_config.json` masih belum di-offboard, dan status hub adalah `ACTIVATED`, atau `DISCOVERED`

Untuk memulihkan dari skenario ini, lihat [Penghapusan paksa](#). Jika penghapusan paksa gagal, atau `iotmi_config.json` masih belum di-offboard, hub harus disetel ulang pabrik.

Hub sedang offline dan status hub adalah `DELETE_IN_PROGRESS`

Dalam skenario ini, hub sedang offline dan cloud menerima perintah offboarding.

Untuk memulihkan dari skenario ini, lihat [Penghapusan paksa](#).

Penghapusan paksa

Untuk menghapus sumber daya cloud tanpa offboarding hub perangkat yang berhasil, ikuti langkah-langkah berikut. Operasi ini dapat mengakibatkan ketidakkonsistenan antara status cloud dan perangkat, yang berpotensi menyebabkan masalah dengan operasi masa depan.

Panggil [DeleteManagedThing](#) API dengan `hub_managed_thing_id` dan parameter gaya:

```
aws iot-managed-integrations delete-managed-thing \  
  --identifier HUB_MANAGED_THING_ID \  
  --force
```

Selanjutnya, panggil [GetManagedThing](#) API dan verifikasi bahwa itu kembali `Managed Thing resource not found`. Ini menegaskan bahwa sumber daya cloud dihapus.

Note

Pendekatan ini tidak disarankan, karena dapat menyebabkan inkonsistensi antara cloud dan status perangkat. Umumnya lebih baik untuk memastikan offboarding hub perangkat yang sukses sebelum mencoba menghapus sumber daya cloud.

Middleware khusus protokol

⚠ Important

Dokumentasi dan kode yang disediakan di sini menjelaskan implementasi referensi middleware. Ini tidak diberikan kepada Anda sebagai bagian dari SDK.

Middleware khusus protokol memiliki peran penting dalam berinteraksi dengan tumpukan protokol yang mendasarinya. Komponen onboarding perangkat dan kontrol perangkat dari integrasi terkelola Hub SDK menggunakannya untuk berinteraksi dengan perangkat akhir.

Middleware melakukan fungsi-fungsi berikut.

- Abstraksi APIs dari tumpukan protokol perangkat dari vendor yang berbeda dengan menyediakan satu set umum. APIs
- Menyediakan manajemen eksekusi perangkat lunak seperti penjadwal utas, manajemen antrian acara, dan cache data.

Arsitektur middleware

Diagram blok di bawah ini mewakili arsitektur middleware Zigbee. Arsitektur middlewares dari protokol lain seperti Z-Wave juga serupa.

Middleware khusus protokol memiliki tiga komponen utama.

- ACS Zigbee DPK: Zigbee Device Porting Kit (DPK) digunakan untuk memberikan abstraksi dari perangkat keras dan sistem operasi yang mendasarinya, sehingga memungkinkan portabilitas. Pada dasarnya ini dapat dianggap sebagai lapisan abstraksi perangkat keras (HAL), yang menyediakan satu set umum APIs untuk mengontrol dan berkomunikasi dengan radio Zigbee dari vendor yang berbeda. Middleware Zigbee berisi implementasi API DPK untuk kerangka Aplikasi Silicon Labs Zigbee.
- ACS Zigbee Service: Layanan Zigbee berjalan sebagai daemon khusus. Ini termasuk penanganan API yang melayani panggilan API dari aplikasi klien melalui saluran IPC. AIPC digunakan sebagai saluran IPC antara adaptor Zigbee dan layanan Zigbee. Ini menyediakan fungsi lain seperti menangani kedua async/sync perintah, menangani peristiwa dari HAL, dan menggunakan ACS Event Manager untuk pendaftaran/penerbitan acara.

- **Adaptor Zigbee ACS:** Adaptor Zigbee adalah perpustakaan yang berjalan dalam proses aplikasi (dalam hal ini, aplikasi adalah plugin CDMB). Adaptor Zigbee menyediakan satu set APIs yang dikonsumsi oleh aplikasi klien seperti plugin CDMB/Provisioner protokol untuk mengontrol dan berkomunikasi dengan perangkat akhir.

End-to-end contoh alur perintah middleware

Berikut adalah contoh aliran perintah melalui middleware Zigbee.

Berikut adalah contoh aliran perintah melalui middleware Z-Wave.

Organisasi kode middleware khusus protokol

Bagian ini berisi informasi tentang lokasi kode untuk setiap komponen di dalam IotManagedIntegrationsDeviceSDK-Middleware repositori. Berikut ini adalah contoh struktur folder dalam repositori ini.

```
./IotManagedIntegrationsDeviceSDK-Middleware
|- greengrass
|- example-iot-ace-dpk
|- example-iot-ace-general
|- example-iot-ace-project
|- example-iot-ace-z3-gateway
|- example-iot-ace-zware
|- example-iot-ace-zwave-mw
```

Topik

- [Organisasi kode middleware Zigbee](#)
- [Organisasi kode middleware Z-Wave](#)

Organisasi kode middleware Zigbee

Berikut ini menunjukkan organisasi kode middleware referensi Zigbee.

Topik

- [ACS Zigbee DPK](#)
- [Laboratorium Silikon Zigbee SDK](#)
- [Layanan ACS Zigbee](#)
- [ACS Zigbee Adaptor](#)

ACS Zigbee DPK

Kode untuk Zigbee DPK terletak di dalam direktori yang tercantum dalam contoh di bawah ini:

```
./IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-dpk/example/dpk/ace_hal/  
|- common  
|-   |- fxnDBusClient  
|-   |- include  
|- kvs  
|- log  
|- wifi  
|-   |- include  
|-   |- src  
|-   |- wifid  
|-       |- fxnWifiClient  
|-       |- include  
|- zigbee  
|-   |- include  
|-   |- src  
|-   |- zigbeed  
|-       |- ember  
|-       |- include  
|- zwave  
|-   |- include  
|-   |- src  
|-   |- zwaved  
|-       |- fxnZwaveClient  
|-       |- include  
|-       |- zware
```

Laboratorium Silikon Zigbee SDK

Silicon Labs SDK disajikan di dalam IotManagedIntegrationsDeviceSDK-Middleware/*example*-iot-ace-z3-gateway folder. Lapisan ACS Zigbee DPK ini diimplementasikan untuk Silicon Labs SDK ini.

```
./IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-zz3-gateway/  
|- autogen  
|- config  
|- gecko_sdk_4.3.2  
|-   |- platform  
|-   |- protocol  
|-   |- util
```

Layanan ACS Zigbee

Kode untuk Layanan Zigbee terletak di dalam folder. `IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-general/middleware/zigbee/ includeSubfolder src` dan di lokasi ini berisi semua file yang terkait dengan layanan ACS Zigbee.

```
IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-general/middleware/zigbee/  
src/  
|- zb_alloc.c  
|- zb_callbacks.c  
|- zb_database.c  
|- zb_discovery.c  
|- zb_log.c  
|- zb_main.c  
|- zb_region_info.c  
|- zb_server.c  
|- zb_svc.c  
|- zb_svc_pwr.c  
|- zb_timer.c  
|- zb_util.c  
|- zb_zdo.c  
|- zb_zts.c  
IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-general/middleware/zigbee/  
include/  
|- init.zigbeeservice.rc  
|- zb_ace_log_uhl.h  
|- zb_alloc.h  
|- zb_callbacks.h  
|- zb_client_aipc.h  
|- zb_client_event_handler.h  
|- zb_database.h  
|- zb_discovery.h  
|- zb_log.h  
|- zb_region_info.h  
|- zb_server.h
```

```
|– zb_svc.h
|– zb_svc_pwr.h
|– zb_timer.h
|– zb_util.h
|– zb_zdo.h
|– zb_zts.h
```

ACS Zigbee Adaptor

Kode untuk Adaptor ACS Zigbee terletak di dalam folder. `IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-general/middleware/zigbee/api` includeSubfolder `src` dan di lokasi ini berisi semua file yang terkait dengan pustaka Adaptor Zigbee ACS.

```
IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-general/middleware/zigbee/
api/src/
|– zb_client_aipc.c
|– zb_client_api.c
|– zb_client_event_handler.c
|– zb_client_zcl.c
IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-general/middleware/zigbee/
api/include/
|– ace
|– |– zb_adapter.h
|– |– zb_command.h
|– |– zb_network.h
|– |– zb_types.h
|– |– zb_zcl.h
|– |– zb_zcl_cmd.h
|– |– zb_zcl_color_control.h
|– |– zb_zcl_hvac.h
|– |– zb_zcl_id.h
|– |– zb_zcl_identify.h
|– |– zb_zcl_level.h
|– |– zb_zcl_measure_and_sensing.h
|– |– zb_zcl_onoff.h
|– |– zb_zcl_power.h
```

Organisasi kode middleware Z-Wave

Berikut ini menunjukkan organisasi kode middleware referensi gelombang-Z.

Topik

- [ACS Z-Gelombang DPK](#)
- [Silicon Labs ZWare dan Zip Gateway](#)
- [Layanan ACS Z-Wave](#)
- [ACS Z-Wave Adaptor](#)

ACS Z-Gelombang DPK

Kode untuk Z-Wave DPK terletak di dalam folder. `IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-dpk/example/dpk/ace_hal/zwave`

```
./IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-dpk/example/dpk/ace_hal/  
|- common  
|-   |- fxnDbusClient  
|-   |- include  
|- kvs  
|- log  
|- wifi  
|-   |- include  
|-   |- src  
|-   |- wifid  
|-       |- fxnWifiClient  
|-       |- include  
|- zibgee  
|-   |- include  
|-   |- src  
|-   |- zigbeed  
|-       |- ember  
|-       |- include  
|- zwave  
|-   |- include  
|-   |- src  
|-   |- zwaved  
|-       |- fxnZwaveClient  
|-       |- include  
|-       |- zware
```

Silicon Labs ZWare dan Zip Gateway

Kode untuk laboratorium Silicon ZWare dan Zip Gateway terletak di dalam `IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-z3-gateway` folder. Lapisan ACS Z-Wave DPK ini diimplementasikan untuk gateway Z-Wave C- APIs dan Zip.

```
./IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-z3-gateway/  
|- autogen  
|- config  
|- gecko_sdk_4.3.2  
|-   |- platform  
|-   |- protocol  
|-   |- util
```

Layanan ACS Z-Wave

Kode untuk Layanan Z-Wave terletak di dalam folder yang tercantum dalam `IotManagedIntegrationsMiddlewares/exampleiot-ace-zwave-mw/` folder. `includeFolder src` dan di lokasi ini berisi semua file yang terkait dengan layanan ACS Z-Wave.

```
IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-zwave-mw/src/  
|- zwave_mgr.c  
|- zwave_mgr_cc.c  
|- zwave_mgr_ipc_aipc.c  
|- zwave_svc.c  
|- zwave_svc_dispatcher.c  
|- zwave_svc_hsm.c  
|- zwave_svc_ipc_aipc.c  
|- zwave_svc_main.c  
|- zwave_svc_publish.c  
IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-zwave-mw/include/  
|- ace  
|-   |- zwave_common_cc.h  
|-   |- zwave_common_cc_battery.h  
|-   |- zwave_common_cc_doorlock.h  
|-   |- zwave_common_cc_firmware.h  
|-   |- zwave_common_cc_meter.h  
|-   |- zwave_common_cc_notification.h  
|-   |- zwave_common_cc_sensor.h  
|-   |- zwave_common_cc_switch.h  
|-   |- zwave_common_cc_thermostat.h  
|-   |- zwave_common_cc_version.h  
|-   |- zwave_common_types.h  
|-   |- zwave_mgr.h  
|-   |- zwave_mgr_cc.h  
|- zwave_log.h  
|- zwave_mgr_internal.h  
|- zwave_mgr_ipc.h  
|- zwave_svc_hsm.h
```

```
|– zwave_svc_internal.h
|– zwave_utils.h
```

ACS Z-Wave Adaptor

Kode untuk Adaptor ACS Zigbee terletak di dalam folder. `IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-zwave-mw/cli/ includeFolder src` dan di lokasi ini berisi semua file yang terkait dengan pustaka Adaptor Z-Wave ACS.

```
IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-zwave-mw/cli/
|– include
|–   |– zwave_cli.h
|– src
|–   |– zwave_cli.yaml
|–   |– zwave_cli_cc.c
|–   |– zwave_cli_event_monitor.c
|–   |– zwave_cli_main.c
|–   |– zwave_cli_net.c
```

Integrasikan middleware dengan SDK

Integrasi middleware pada hub baru dibahas di bagian berikut.

Topik

- [Integrasi API kit porting perangkat \(DPK\)](#)
- [Implementasi referensi dan organisasi kode](#)

Integrasi API kit porting perangkat (DPK)

Untuk mengintegrasikan SDK vendor chipset apa pun dengan middleware, antarmuka API standar disediakan oleh lapisan DPK (Device porting kit) di tengah. Integrasi terkelola penyedia layanan atau ODMs perlu mengimplementasikannya APIs berdasarkan SDK vendor yang didukung oleh chipset Zigbee/Z-wave/Wi -Fi yang digunakan pada Hub IoT mereka.

Implementasi referensi dan organisasi kode

Kecuali middleware, semua komponen Device SDK lainnya, seperti integrasi terkelola Device Agent dan Common Data Model Bridge (CDMB) dapat digunakan tanpa modifikasi apa pun dan hanya perlu dikompilasi silang.

Implementasi middleware didasarkan pada Silicon Labs SDK untuk Zigbee dan Z-Wave. Jika chipset Z-Wave dan Zigbee yang digunakan di hub baru didukung oleh Silicon Labs SDK yang ada di middleware, maka middleware referensi dapat digunakan tanpa modifikasi apa pun. Anda hanya perlu mengkompilasi silang middleware dan kemudian dapat dijalankan di hub baru.

DPK (Device porting kit) APIs untuk Zigbee dapat ditemukan di `acehal_zigbee.c`, dan implementasi referensi APIs DPK hadir di dalam folder. `zigbee`

```
IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-dpk/example/dpk/ace_hal/  
zigbee/  
|- CMakeLists.txt  
|- include  
|-   |- zigbee_log.h  
|- src  
|-   |- acehal_zigbee.c  
|- zigbeed  
|-   |- CMakeLists.txt  
|-   |- ember  
|-     |- ace_ember_common.c  
|-     |- ace_ember_ctrl.c  
|-     |- ace_ember_hal_callbacks.c  
|-     |- ace_ember_network_creator.c  
|-     |- ace_ember_power_settings.c  
|-     |- ace_ember_zts.c  
|-     |- include  
|-       |- zbd_api.h  
|-       |- zbd_callbacks.h  
|-       |- zbd_common.h  
|-       |- zbd_network_creator.h  
|-       |- zbd_power_settings.h  
|-       |- zbd_zts.h
```

DPK APIs untuk Z-Wave dapat ditemukan di `acehal_zwave.c` dan referensi implementasi DPK APIs hadir di dalam folder. `zwaved`

```
IotManagedIntegrationsDeviceSDK-Middleware/example-iot-ace-dpk/example/dpk/ace_hal/  
zwave/  
|- CMakeLists.txt  
|- include  
|-   |- zwave_log.h  
|- src  
|-   |- acehal_zwave.c
```

```
|- zwaved
|-   |- CMakeLists.txt
|-   |- fxnZwaveClient
|-   |-   |- zwave_client.c
|-   |-   |- zwave_client.h
|-   |- include
|-   |-   |- zwaved_cc_intf_api.h
|-   |-   |- zwaved_common_utils.h
|-   |-   |- zwaved_ctrl_api.h
|-   |- zware
|-   |-   |- ace_zware_cc_intf.c
|-   |-   |- ace_zware_common_utils.c
|-   |-   |- ace_zware_ctrl.c
|-   |-   |- ace_zware_debug.c
|-   |-   |- ace_zware_debug.h
|-   |-   |- ace_zware_internal.h
```

Sebagai titik awal untuk mengimplementasikan lapisan DPK untuk SDK vendor yang berbeda, implementasi referensi dapat digunakan dan dimodifikasi. Berikut dua modifikasi akan diperlukan untuk mendukung SDK vendor yang berbeda:

1. Ganti SDK vendor saat ini dengan SDK vendor baru di repositori.
2. Menerapkan middleware DPK (Device porting kit) APIs sesuai dengan SDK vendor baru.

Integrasi terkelola Akhiri perangkat SDK

Bangun platform IoT yang menghubungkan perangkat pintar ke integrasi terkelola dan memproses perintah melalui antarmuka kontrol terpadu. SDK perangkat Akhir terintegrasi dengan firmware perangkat Anda dan menyediakan pengaturan yang disederhanakan dengan komponen tepi SDK, serta konektivitas yang aman ke AWS IoT Core dan AWS IoT Manajemen Perangkat. Unduh versi terbaru SDK perangkat Akhir dari Konsol Manajemen AWS

Panduan ini menjelaskan cara mengimplementasikan SDK perangkat Akhir di firmware Anda. Tinjau arsitektur, komponen, dan langkah-langkah integrasi untuk mulai membangun implementasi Anda.

Topik

- [Apa itu SDK Perangkat Akhir?](#)
- [Arsitektur dan komponen SDK perangkat akhir](#)
- [Provisionee](#)
- [Over-the-Air pemutakhiran](#)
- [Generator kode model data](#)
- [Fungsi C tingkat rendah APIs](#)
- [Interaksi fitur dan perangkat dalam integrasi terkelola](#)
- [Memulai dengan End device SDK](#)

Apa itu SDK Perangkat Akhir?

Apa itu SDK Perangkat Akhir?

SDK perangkat Akhir adalah kumpulan kode sumber, pustaka, dan alat yang disediakan oleh AWS IoT. Dibangun untuk lingkungan terbatas sumber daya, SDK mendukung perangkat dengan RAM hanya 512 KB dan memori flash 4 MB, seperti kamera dan pembersih udara yang berjalan pada Linux tertanam dan sistem operasi real-time (RTOS). Unduh versi terbaru SDK perangkat Akhir dari [Konsol AWS IoT Manajemen](#).

Komponen inti

SDK menggabungkan agen MQTT untuk komunikasi cloud, penanganan pekerjaan untuk manajemen tugas, dan integrasi terkelola, Data Model Handler. Komponen-komponen ini bekerja sama untuk

menyediakan konektivitas yang aman dan terjemahan data otomatis antara perangkat Anda dan integrasi terkelola.

Untuk persyaratan teknis terperinci, lihat [Referensi teknis](#).

Arsitektur dan komponen SDK perangkat akhir

Bagian ini menjelaskan arsitektur End device SDK dan bagaimana komponennya berinteraksi dengan C-Functions tingkat rendah Anda. Diagram berikut menggambarkan komponen inti dan hubungannya dalam kerangka SDK.

Akhiri komponen SDK perangkat

Arsitektur End device SDK berisi komponen-komponen ini untuk integrasi fitur integrasi terkelola:

Provisionee

Membuat sumber daya perangkat di cloud integrasi terkelola, termasuk sertifikat perangkat dan kunci pribadi untuk komunikasi MQTT yang aman. Kredensial ini membangun koneksi tepercaya antara perangkat Anda dan integrasi terkelola.

Agen MQTT

Mengelola koneksi MQTT melalui pustaka klien C thread-safe. Proses latar belakang ini menangani antrian perintah di lingkungan multi-utas, dengan ukuran antrian yang dapat dikonfigurasi untuk perangkat yang dibatasi memori. Rute pesan melalui integrasi terkelola untuk diproses.

Penangan pekerjaan

Proses over-the-air (OTA) pembaruan untuk firmware perangkat, patch keamanan, dan pengiriman file. Layanan bawaan ini mengelola pembaruan perangkat lunak untuk semua perangkat terdaftar.

Penangan Model Data

Menerjemahkan operasi antara integrasi terkelola dan C-Functions Tingkat Rendah Anda menggunakan AWS'implementasi Model Data Matter. Untuk informasi lebih lanjut, lihat [dokumentasi Materi](#) di GitHub.

Kunci dan sertifikat

[Mengelola operasi kriptografi melalui PKCS #11 API, mendukung modul keamanan perangkat keras dan implementasi perangkat lunak seperti inti. PKCS11](#) API ini menangani operasi sertifikat untuk komponen seperti Provisionee dan Agen MQTT selama koneksi TLS.

Provisionee

Provisionee adalah komponen integrasi terkelola yang memungkinkan penyediaan armada dengan klaim. Dengan provisionee, Anda menyediakan perangkat Anda dengan aman. SDK menciptakan sumber daya yang diperlukan untuk penyediaan perangkat, yang mencakup sertifikat perangkat dan kunci pribadi yang diperoleh dari cloud integrasi terkelola. Saat Anda ingin menyediakan perangkat Anda, atau jika ada perubahan yang mengharuskan Anda untuk menyediakan kembali perangkat Anda, Anda dapat menggunakan provisionee.

Topik

- [Alur kerja penyediaan](#)
- [Tetapkan variabel lingkungan](#)
- [Daftarkan titik akhir kustom](#)
- [Buat profil penyediaan](#)
- [Buat hal yang dikelola](#)
- [Penyediaan Wi-Fi pengguna SDK](#)
- [Penyediaan armada dengan klaim](#)
- [Kemampuan Hal yang Dikelola](#)

Alur kerja penyediaan

Proses ini membutuhkan pengaturan di kedua sisi cloud dan perangkat. Pelanggan mengonfigurasi persyaratan cloud seperti titik akhir khusus, profil penyediaan, dan hal-hal terkelola. Pada perangkat pertama dihidupkan, penyediaannya:

1. Terhubung ke titik akhir integrasi terkelola menggunakan sertifikat klaim
2. Memvalidasi parameter perangkat melalui kait penyediaan armada
3. Memperoleh dan menyimpan sertifikat permanen dan kunci pribadi pada perangkat
4. Perangkat menggunakan sertifikat permanen untuk menyambung kembali

5. Menemukan dan mengunggah kemampuan perangkat ke integrasi terkelola

Setelah penyediaan berhasil, perangkat berkomunikasi langsung dengan integrasi terkelola. Provisioner mengaktifkan hanya untuk tugas penyediaan ulang.

Tetapkan variabel lingkungan

Tetapkan AWS kredensial berikut di lingkungan cloud Anda:

```
$ export AWS_ACCESS_KEY_ID=YOUR-ACCOUNT-ACCESS-KEY-ID
$ export AWS_SECRET_ACCESS_KEY=YOUR-ACCOUNT-SECRET-ACCESS-KEY
$ export AWS_DEFAULT_REGION=YOUR-DEFAULT-REGION
```

Daftarkan titik akhir kustom

Gunakan perintah [RegisterCustomEndpoint](#) API di lingkungan cloud Anda untuk membuat titik akhir khusus untuk device-to-cloud komunikasi.

```
aws iot-managed-integrations register-custom-endpoint
```

Contoh respon

```
{ "EndpointAddress": "[ACCOUNT-PREFIX]-ats.iot.AWS-REGION.amazonaws.com" }
```

Note

Simpan alamat titik akhir untuk mengonfigurasi parameter penyediaan. gunakan [GetCustomEndpoint](#) API, untuk mengembalikan informasi titik akhir. Untuk informasi selengkapnya, lihat [GetCustomEndpoint](#) API, dan [RegisterCustomEndpoint](#) API di Panduan Referensi API integrasi Terkelola.

Buat profil penyediaan

Buat profil penyediaan yang menentukan metode penyediaan armada Anda. Jalankan [CreateProvisioningProfile](#) API di lingkungan cloud Anda untuk mengembalikan sertifikat klaim dan kunci pribadi untuk otentikasi perangkat:

```
aws iot-managed-integrations create-provisioning-profile \
```

```
--provisioning-type "FLEET_PROVISIONING" \  
--name "PROVISIONING-PROFILE-NAME"
```

Contoh respon

```
{ "Arn":"arn:aws:iot-managed-integrations:AWS-REGION:YOUR-ACCOUNT-ID:provisioning-  
profile/PROFILE_NAME",  
  "ClaimCertificate":"string",  
  "ClaimCertificatePrivateKey":"string",  
  "Name":"ProfileName",  
  "ProvisioningType":"FLEET_PROVISIONING"}
```

Anda dapat mengimplementasikan pustaka abstraksi PKCS11 platform inti (PAL) untuk membuat PKCS11 pustaka inti berfungsi dengan perangkat Anda. Port PKCS11 PAL inti harus menyediakan lokasi untuk menyimpan sertifikat klaim dan kunci pribadi. Dengan menggunakan fitur ini, Anda dapat menyimpan kunci pribadi dan sertifikat perangkat dengan aman. Anda dapat menyimpan kunci pribadi dan sertifikat pada modul keamanan perangkat keras (HSM) atau modul platform tepercaya (TPM).

Buat hal yang dikelola

Daftarkan perangkat Anda dengan cloud integrasi terkelola menggunakan [CreateManagedThing](#) API. Sertakan nomor seri (SN) dan kode produk universal (UPC) perangkat Anda:

```
aws iot-managed-integrations create-managed-thing --role DEVICE \  
--authentication-material-type WIFI_SETUP_QR_BAR_CODE \  
--authentication-material "SN:DEVICE-SN;UPC:DEVICE-UPC;"
```

Berikut ini menampilkan contoh respons API.

```
{  
  "Arn":"arn:aws:iot-managed-integrations:AWS-REGION:ACCOUNT-ID:managed-  
thing/59d3c90c55c4491192d841879192d33f",  
  "CreatedAt":1.730960226491E9,  
  "Id":"59d3c90c55c4491192d841879192d33f"  
}
```

API mengembalikan ID Managed thing yang dapat digunakan untuk validasi penyediaan. Anda harus memberikan nomor seri perangkat (SN) dan kode produk universal (UPC), yang dicocokkan dengan

hal terkelola yang disetujui selama transaksi penyediaan. Transaksi mengembalikan hasil yang mirip dengan berikut ini:

```
/**
 * @brief Device info structure.
 */
typedef struct iotmiDev_DeviceInfo
{
    char serialNumber[ IOTMI_DEVICE_MAX_SERIAL_NUMBER_LENGTH + 1U ];
    char universalProductCode[ IOTMI_DEVICE_MAX_UPC_LENGTH + 1U ];
    char internationalArticleNumber[ IOTMI_DEVICE_MAX_EAN_LENGTH + 1U ];
} iotmiDev_DeviceInfo_t;
```

Penyediaan Wi-Fi pengguna SDK

Produsen perangkat dan penyedia solusi memiliki layanan penyediaan Wi-Fi milik mereka sendiri untuk menerima dan mengonfigurasi kredensial Wi-Fi. Layanan penyediaan Wi-Fi melibatkan penggunaan aplikasi seluler khusus, koneksi Bluetooth Low Energy (BLE), dan protokol eksklusif lainnya untuk mentransfer kredensial Wi-Fi secara aman untuk proses penyiapan awal.

Konsumen SDK perangkat Akhir harus menerapkan layanan penyediaan Wi-Fi dan perangkat dapat terhubung ke jaringan Wi-Fi.

Penyediaan armada dengan klaim

Menggunakan provisionee, pengguna akhir dapat memberikan sertifikat unik dan mendaftarkannya dengan integrasi terkelola menggunakan penyediaan dengan klaim.

ID klien dapat diperoleh baik dari respons templat penyediaan atau sertifikat perangkat <common name>“_”<serial number>

Kemampuan Hal yang Dikelola

Penyedia menemukan kemampuan hal yang dikelola, kemudian mengunggah kemampuan ke integrasi terkelola. Itu membuat kemampuan tersedia untuk aplikasi dan layanan lain untuk diakses. Perangkat, klien web lainnya, dan layanan dapat memperbarui kemampuan dengan menggunakan MQTT dan topik MQTT yang dicadangkan, atau HTTP menggunakan REST API.

Over-the-Air pemutakhiran

Ikhtisar arsitektur OTA

Proses pembaruan Over-the-Air (OTA) melibatkan beberapa komponen yang bekerja sama untuk mengirimkan pembaruan firmware ke perangkat Anda. Diagram berikut menggambarkan bagaimana permintaan pembaruan OTA ditangani melalui interaksi antara SDK perangkat Akhir, SDK Hub, dan fitur tersebut.

Arsitektur pembaruan OTA terdiri dari komponen-komponen berikut:

- Pelanggan: Mengunggah dokumen pekerjaan ke bucket S3 dan memulai pembaruan melalui API
- Layanan OTA: Menangani penciptaan lapangan kerja, validasi, dan manajemen
- AWS IoT Pekerjaan: Mengelola pelaksanaan pekerjaan dan pengiriman ke perangkat
- Perangkat: Menerima dan menerapkan pembaruan menggunakan Harmony SDK

Prasyarat

Sebelum membuat tugas OTA, Anda harus mengonfigurasi prasyarat berikut:

Konfigurasi akses Amazon S3

Untuk mengaktifkan pembaruan OTA, Anda harus mengunggah dokumen pekerjaan ke bucket Amazon S3 dan mengonfigurasi izin akses yang sesuai:

1. Unggah dokumen pekerjaan OTA Anda ke bucket S3
2. Tambahkan kebijakan bucket Amazon S3 yang memberikan akses integrasi terkelola ke dokumen pekerjaan Anda:

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PolicyForS3JobDocument",
      "Effect": "Allow",
```

```
"Principal": {
  "Service": "iotmanagedintegrations.amazonaws.com"
},
"Action": "s3:GetObject",
"Resource": [
  "arn:aws:s3:::YOUR_BUCKET/*",
  "arn:aws:s3:::YOUR_BUCKET/ota_job_document.json",
  "arn:aws:s3:::YOUR_BUCKET"
]
}
]
```

Melaksanakan tugas Over-the-Air (OTA)

Anda dapat membuat tugas OTA dengan dua cara, tergantung pada persyaratan pembaruan dan strategi penargetan perangkat Anda:

Pembaruan tugas OTA satu kali

Tugas OTA satu kali berisi daftar statis target (ManagedThings) untuk melakukan pembaruan OTA. Anda dapat menambahkan hingga 100 target sekaligus. Alur kerja menggunakan AWS IoT Pekerjaan dengan Pengindeksan Armada sambil mempertahankan lapisan abstraksi integrasi terkelola.

Gunakan contoh berikut untuk membuat tugas OTA satu kali:

```
aws iotmanagedintegrations create-ota-task \
  --description "One-time OTA update" \
  --s3-url "s3://test-job-document-bucket/ota-job-document.json" \
  --protocol HTTP \
  --target ["arn:aws:iotmanagedintegrations:region:account id:managed-thing/managed thing id"] \
  --ota-mechanism PUSH \
  --ota-type ONE_TIME \
  --client-token "foo" \
  --tags '{"key1":"foo","key2":"foo"}'
```

Pembaruan tugas OTA berkelanjutan

Alur kerja pengelompokan OTA (Over-the-Air) memungkinkan Anda menerapkan pembaruan firmware ke grup perangkat berdasarkan atribut tertentu, menggunakan AWS IoT Pekerjaan dengan

Pengindeksan Armada sambil mempertahankan lapisan abstraksi integrasi terkelola. Tugas OTA berkelanjutan menggunakan string kueri alih-alih target tertentu. Semua perangkat yang cocok dengan kriteria kueri menjalani pembaruan OTA, dan kriteria kueri terus dievaluasi ulang. Target yang cocok akan memiliki penerapan pekerjaan.

Konfigurasi prasyarat

Sebelum membuat tugas OTA berkelanjutan, selesaikan prasyarat ini:

1. Buat hal yang dikelola dengan memanggil [CreateManagedThing](#) API dan melakukan penyediaan armada.
2. Tambahkan atribut metadata ke hal-hal terkelola untuk penargetan kueri.

Tambahkan atribut dan metadata untuk ManagedThing menggunakan API: [UpdateManagedThing](#)

```
aws iotmanagedintegrations update-managed-thing \  
  --managed-thing-id "YOUR_MANAGED_THING_ID" \  
  --meta-data '{"owner":"managedintegrations","version":"1.0"}'
```

Gunakan contoh berikut untuk membuat tugas OTA berkelanjutan:

```
aws iotmanagedintegrations create-ota-task \  
  --description "Continuous OTA update" \  
  --s3-url "s3://test-job-document-bucket/ota-job-document.json" \  
  --protocol HTTP \  
  --ota-mechanism PUSH \  
  --ota-type CONTINUOUS \  
  --client-token "foo" \  
  --ota-target-query-string "attributes.owner=managedintegrations" \  
  --tags '{"key1":"foo","key2":"foo"}'
```

Memahami alur kerja OTA berkelanjutan

Alur kerja pembaruan OTA berkelanjutan mengikuti langkah-langkah ini:

1. Anda memperbarui hal-hal terkelola dengan atribut menggunakan [UpdateManagedThing](#) API.
2. Buat pekerjaan OTA dengan string kueri yang menargetkan atribut perangkat tertentu.
3. Layanan OTA membuat Thing Group dinamis AWS IoT Core berdasarkan atribut kueri
4. Pekerjaan IoT mengeksekusi pembaruan pada perangkat yang cocok

5. Anda memantau kemajuan melalui pemberitahuan [ListOtaTaskExecutions](#) API atau OTA melalui aliran Kinesis (jika diaktifkan).

Perbedaan antara Integrasi Terkelola OTA dan Pekerjaan IoT

Perbedaan mendasar antara Integrasi Terkelola OTA dan Pekerjaan IoT terletak pada orkestrasi layanan dan otomatisasi. Integrasi terkelola OTA menyediakan solusi layanan tunggal yang mengabstraksi kompleksitas koordinasi multi-layanan.

Integrasi Terkelola yang dilakukan OTA secara otomatis:

- Pembuatan Dynamic Thing Group: Secara otomatis menghasilkan grup AWS IoT Core hal berdasarkan kriteria kueri Anda.
- Resolusi target: Menerjemahkan string kueri (Contoh: `attributes.owner=managedintegrations`) ke dalam target perangkat yang sebenarnya.
- Integrasi layanan: Berkoordinasi dengan mulus antara, Pekerjaan AWS IoT Core IoT, dan layanan Pengindeksan Armada.
- Manajemen siklus hidup: Menangani seluruh alur kerja OTA mulai dari pembuatan hingga pemantauan eksekusi.

Apa yang dihilangkan MI OTA:

- Membuat grup benda di AWS IoT Core.
- Menambahkan sesuatu ke grup.
- Membuat Pekerjaan IoT.

Integrasi terkelola OTA menangani ketiga operasi secara internal berdasarkan string kueri Anda, secara otomatis menemukan perangkat yang sesuai dengan kriteria Anda, membuat Pekerjaan IoT di bawah tenda, dan mengatur alur kerja OTA lengkap tanpa mengharuskan Anda berinteraksi dengan beberapa layanan secara langsung. AWS

Pengaturan konfigurasi tugas OTA

Anda dapat membuat konfigurasi untuk pembaruan OTA untuk mengontrol cara pembaruan diluncurkan ke perangkat, mengatur kondisi batal, dan mengonfigurasi batas waktu.

Contoh: CreateOtaTaskConfiguration

Gunakan contoh berikut untuk membuat konfigurasi tugas OTA:

```
aws iotmanagedintegrations create-ota-task-configuration \  
  --description "OTA configuration" \  
  --name "MyOtaConfig" \  
  --push-config '{  
    "AbortConfig": {  
      "AbortConfigCriteriaList": [  
        {  
          "Action": "CANCEL",  
          "FailureType": "FAILED",  
          "MinNumberOfExecutedThings": 1,  
          "ThresholdPercentage": 90.0  
        }  
      ]  
    },  
    "RolloutConfig": {  
      "ExponentialRolloutRate": {  
        "BaseRatePerMinute": 1,  
        "IncrementFactor": 3.0,  
        "RateIncreaseCriteria": {  
          "numberOfNotifiedThings": 1  
        }  
      },  
      "MaximumPerMinute": 1  
    },  
    "TimeoutConfig": {  
      "InProgressTimeoutInMinutes": 100  
    }  
  }' \  
  --client-token "foo"
```

Terapkan pengaturan konfigurasi ke tugas OTA

Setelah konfigurasi dibuat, Anda akan menerima `taskConfigurationId` yang ditambahkan ke `CreateOtaTask` permintaan Anda bersama dengan konfigurasi tambahan:

```
aws iotmanagedintegrations create-ota-task \  
  --description "OTA with configuration" \  
  --s3-url "s3://test-job-document-bucket/ota-job-document.json" \  
  --task-configuration-id <taskConfigurationId>
```

```
--protocol HTTP \  
--target ["arn:aws:iotmanagedintegrations:region:account id:managed-thing/managed  
thing id"] \  
--ota-mechanism PUSH \  
--ota-type ONE_TIME \  
--client-token "foo" \  
--task-configuration-id "ae4f49352c5443369f43ad6c3a7f1580" \  
--ota-scheduling-config '{  
  "EndBehavior": "STOP_ROLLOUT",  
  "EndTime": "2024-10-23T17:00",  
  "StartTime": "2024-10-20T17:00"  
}' \  
--ota-task-execution-retry-config '{  
  "RetryConfigCriteria": [  
    {  
      "FailureType": "FAILED",  
      "MinNumberOfRetries": 1  
    }  
  ]  
}' \  
--tags '{"key1":"foo","key2":"foo"}'
```

Pantau pemberitahuan OTA

Anda dapat memantau pembaruan OTA menggunakan dua metode berbeda:

Pemberitahuan push melalui Kinesis Data Streams

Saat pemberitahuan OTA diaktifkan, peristiwa status pembaruan secara otomatis didorong ke aliran Kinesis Anda. Ini memberikan visibilitas real-time ke dalam kemajuan pembaruan firmware di seluruh perangkat.

Monitor dengan ListOtaTaskExecutions API

Anda dapat menggunakan [ListOtaTaskExecutions](#) API untuk memeriksa status pembaruan OTA secara manual untuk hal-hal yang Anda kelola:

```
aws iotmanagedintegrations list-ota-task-executions \  
--task-id "task-123456789" \  
--max-results 25
```

Respons memberikan status eksekusi terperinci untuk setiap hal yang dikelola:

```
{
  "taskExecutionSummaries": [
    {
      "taskExecutionSummary": {
        "executionNumber": 1,
        "lastUpdatedAt": 1634567890,
        "queuedAt": 1634567800,
        "startedAt": 1634567830,
        "status": "SUCCEEDED",
        "retryAttempt": 0
      },
      "managedThingId": "device-001"
    },
    {
      "taskExecutionSummary": {
        "executionNumber": 1,
        "lastUpdatedAt": 1634567920,
        "queuedAt": 1634567800,
        "startedAt": 1634567840,
        "status": "IN_PROGRESS",
        "retryAttempt": 0
      },
      "managedThingId": "device-002"
    }
  ],
  "nextToken": "NEXT_TOKEN"
}
```

API ini memungkinkan Anda untuk mengambil status eksekusi terperinci untuk setiap hal terkelola yang ditargetkan oleh tugas OTA tertentu, termasuk cap waktu dan status saat ini.

Memproses dokumen pekerjaan

Saat Anda membuat tugas OTA, penangan pekerjaan menjalankan langkah-langkah berikut di perangkat Anda. Ketika pembaruan tersedia, ia meminta dokumen pekerjaan melalui MQTT.

1. Berlangganan topik notifikasi MQTT.
2. Memanggil [StartNextPendingJobExecution](#) API untuk pekerjaan yang tertunda.
3. Menerima dokumen pekerjaan yang tersedia.
4. Memproses pembaruan berdasarkan batas waktu yang ditentukan.

Dengan menggunakan penanganan pekerjaan, aplikasi dapat menentukan apakah akan segera mengambil tindakan atau menunggu hingga periode waktu tunggu yang ditentukan.

Menerapkan agen OTA

Ketika Anda menerima dokumen pekerjaan dari integrasi terkelola, Anda harus memiliki implementasi agen OTA Anda sendiri yang memproses dokumen pekerjaan, mengunduh pembaruan, dan melakukan operasi instalasi apa pun. Agen OTA perlu melakukan langkah-langkah berikut:

1. Mengurai dokumen pekerjaan untuk firmware Amazon URLs S3.
2. Unduh pembaruan firmware melalui HTTP.
3. Verifikasi tanda tangan digital.
4. Instal pembaruan yang divalidasi.
5. Panggilan `iotmi_JobsHandler_updateJobStatus` dengan `SUCCESS` atau `FAILED` status.

Ketika perangkat Anda berhasil menyelesaikan operasi OTA, perangkat harus memanggil `iotmi_JobsHandler_updateJobStatus` API dengan status `JobSucceeded` untuk melaporkan pekerjaan yang berhasil.

```
/**
 * @brief Enumeration of possible job statuses.
 */
typedef enum{
    JobQueued,          /** The job is in the queue, waiting to be processed. */
    JobInProgress,     /** The job is currently being processed. */
    JobFailed,         /** The job processing failed. */
    JobSucceeded,      /** The job processing succeeded. */
    JobRejected        /** The job was rejected, possibly due to an error or invalid
request. */
} iotmi_JobCurrentStatus_t;

/**
 * @brief Update the status of a job with optional status details.
 *
 * @param[in] pJobId Pointer to the job ID string.
 * @param[in] jobIdLength Length of the job ID string.
 * @param[in] status The new status of the job.
 * @param[in] statusDetails Pointer to a string containing additional details about the
job status.
```

```
*           This can be a JSON-formatted string or NULL if no details
are needed.
* @param[in] statusDetailsLength Length of the status details string. Set to 0 if
`statusDetails` is NULL.
*
* @return 0 on success, non-zero on failure.
*/
int iotmi_JobsHandler_updateJobStatus( const char * pJobId,
                                     size_t jobIdLength,
                                     iotmi_JobCurrentStatus_t status,
                                     const char * statusDetails,
                                     size_t statusDetailsLength );
```

Generator kode model data

Pelajari cara menggunakan generator kode untuk model data. Kode yang dihasilkan dapat digunakan untuk membuat serial dan deserialisasi model data yang dipertukarkan antara cloud dan perangkat.

Repositori proyek berisi alat pembuatan kode untuk membuat penanganan model data kode C. Topik berikut menjelaskan pembuat kode dan alur kerja.

Topik

- [Proses pembuatan kode](#)
- [Pengaturan lingkungan](#)
- [Hasilkan kode untuk perangkat](#)

Proses pembuatan kode

Generator kode membuat file sumber C dari tiga input utama: AWS'implementasi Model Data Matter (file.matter) dari Platform Lanjutan Perpustakaan Cluster Zigbee (ZCL), plugin Python yang menangani preprocessing, dan template Jinja2 yang menentukan struktur kode. Selama pembuatan, plugin Python memproses file.matter Anda dengan menambahkan definisi tipe global, mengatur tipe data berdasarkan dependensinya, dan memformat informasi untuk rendering template.

Gambar berikut menjelaskan generator kode membuat file sumber C.

SDK perangkat Akhir menyertakan plugin Python dan templat Jinja2 yang berfungsi dalam proyek. [codegen.pyconnectedhomeip](#) Kombinasi ini menghasilkan beberapa file C untuk setiap cluster berdasarkan masukan file.matter Anda.

Subtopik berikut menjelaskan file-file ini.

- [Plugin Python](#)
- [Jinja2 template](#)
- [\(Opsional\) Skema kustom](#)

Plugin Python

Generator kode, `codegen.py`, mem-parsing file.matter, dan mengirimkan informasi sebagai objek Python ke plugin. File plugin `iotmi_data_model.py` memproses data ini dan membuat sumber dengan templat yang disediakan. Preprocessing meliputi:

1. Menambahkan informasi yang tidak tersedia dari `codegen.py`, seperti tipe global
2. Melakukan pengurutan topologi pada tipe data untuk menetapkan urutan definisi yang benar

Note

Urutan topologi memastikan tipe dependen didefinisikan setelah dependensinya, terlepas dari urutan aslinya.

Jinja2 template

SDK perangkat Akhir menyediakan template Jinja2 yang disesuaikan untuk penanganan model data dan C-Functions tingkat rendah.

Jinja2 template

Templat	Sumber yang dihasilkan	Keterangan
<code>cluster.h.jinja</code>	<code>iotmi_device_<cluster>.h</code>	Membuat file header fungsi C tingkat rendah.

Templat	Sumber yang dihasilkan	Keterangan
<code>cluster.c.jinja</code>	<code>iotmi_device_<cluster>.c</code>	Menerapkan dan mendaftarkan pointer fungsi callback dengan Data Model Handler.
<code>cluster_type_helpers.h.jinja</code>	<code>iotmi_device_type_helpers_<cluster>.h</code>	Mendefinisikan prototipe fungsi untuk tipe data.
<code>cluster_type_helpers.c.jinja</code>	<code>iotmi_device_type_helpers_<cluster>.c</code>	Menghasilkan prototipe fungsi tipe data untuk enumerasi, bitmap, daftar, dan struktur khusus cluster.
<code>iot_device_dm_types.h.jinja</code>	<code>iotmi_device_dm_types.h</code>	Mendefinisikan tipe data C untuk tipe data global.
<code>iot_device_type_helpers_global.h.jinja</code>	<code>iotmi_device_type_helpers_global.h</code>	Mendefinisikan tipe data C untuk operasi global.
<code>iot_device_type_helpers_global.c.jinja</code>	<code>iotmi_device_type_helpers_global.c</code>	Mendeklarasikan tipe data standar termasuk boolean, integer, floating point, string, bitmap, daftar, dan struktur.

(Opsional) Skema kustom

SDK perangkat akhir menggabungkan proses pembuatan kode standar dengan skema khusus. Ini memungkinkan perpanjangan Model Data Matter untuk perangkat dan perangkat lunak perangkat Anda. Skema khusus dapat membantu menggambarkan kemampuan perangkat untuk device-to-cloud komunikasi.

Untuk informasi rinci tentang model data integrasi terkelola, termasuk format, struktur, dan persyaratan, lihat [Model data integrasi terkelola](#).

Gunakan `codegen.py` alat untuk menghasilkan file sumber C untuk skema kustom, sebagai berikut:

Note

Setiap cluster kustom memerlukan ID cluster yang sama untuk tiga file berikut.

- Buat skema kustom dalam JSON format yang menyediakan representasi cluster untuk pelaporan kemampuan untuk membuat cluster kustom baru di cloud. File sampel terletak di `codegen/custom_schemas/custom.SimpleLighting@1.0`.
- Buat file definisi ZCL (Zigbee Cluster Library) dalam XML format yang berisi informasi yang sama dengan skema kustom. Gunakan alat ZAP untuk menghasilkan file Matter IDL Anda dari ZCL XHTML. File sampel terletak di `codegen/zcl/custom.SimpleLighting.xml`.
- Output dari alat ZAP adalah Matter IDL File (`.matter`) dan mendefinisikan cluster Matter yang sesuai dengan skema kustom Anda. Ini adalah input untuk `codegen.py` alat untuk menghasilkan file sumber C untuk SDK perangkat Akhir. File sampel terletak di `codegen/matter_files/custom-light.matter`.

Untuk petunjuk terperinci tentang cara mengintegrasikan model data integrasi terkelola kustom ke dalam alur kerja pembuatan kode Anda, lihat [Hasilkan kode untuk perangkat](#)

Pengaturan lingkungan

Pelajari cara mengonfigurasi lingkungan Anda untuk menggunakan pembuat `codegen.py` kode.

Topik

- [Prasyarat](#)
- [Konfigurasi lingkungan Anda](#)

Prasyarat

Instal item berikut sebelum Anda mengonfigurasi lingkungan Anda:

- Git
- Python 3.10 atau lebih tinggi
- Pui 1.2.0 atau lebih tinggi

Konfigurasi lingkungan Anda

Gunakan prosedur berikut untuk mengkonfigurasi lingkungan Anda untuk menggunakan generator kode codegen.py.

1. Unduh versi terbaru [SDK perangkat Akhir](#) dari file. Konsol Manajemen AWS
2. Siapkan lingkungan Python. Proyek codegen berbasis python dan menggunakan Pui untuk manajemen ketergantungan.
 - Instal dependensi proyek menggunakan pui di direktori: codegen

```
poetry run poetry install --no-root
```

3. Siapkan repositori Anda.

- a. Kloning `connectedhomeip` repositori. Ini menggunakan `codegen.py` skrip yang terletak di `connectedhomeip/scripts/` folder untuk pembuatan kode. Untuk informasi selengkapnya, lihat [connectedhomeip](#) on. GitHub

```
git clone -b v1.4.0.0 https://github.com/project-chip/connectedhomeip.git
```

- b. Kloning pada tingkat yang sama dengan folder `IoT-managed-integrations-End-Device-SDK` root Anda. Struktur folder Anda harus cocok dengan yang berikut:

```
| -connectedhomeip  
| -IoT-managed-integrations-End-Device-SDK
```

Note

Anda tidak perlu mengkloning submodul secara rekursif.

Hasilkan kode untuk perangkat

Buat kode C yang disesuaikan untuk perangkat Anda menggunakan alat pembuatan kode integrasi terkelola. Bagian ini menjelaskan cara menghasilkan kode dari file sampel yang disertakan dengan SDK atau dari spesifikasi Anda sendiri. Pelajari cara menggunakan skrip pembuatan, memahami proses alur kerja, dan membuat kode yang sesuai dengan kebutuhan perangkat Anda.

Topik

- [Prasyarat](#)
- [Hasilkan kode untuk file.matter khusus](#)
- [Alur kerja pembuatan kode](#)

Prasyarat

1. Python 3.10 atau lebih tinggi.
2. Mulailah dengan file.matter untuk pembuatan kode. SDK perangkat Akhir menyediakan dua file sampel dicodgen/matter_files folder:
 - custom-air-purifier.matter
 - aws_camera.matter

Note

File sampel ini menghasilkan kode untuk kluster aplikasi demo.

Hasilkan kode

Jalankan perintah ini untuk menghasilkan kode di folder keluar:

```
bash ./gen-data-model-api.sh
```

Hasilkan kode untuk file.matter khusus

Untuk menghasilkan kode untuk .matter file tertentu atau menyediakan file Anda sendiri .matter, lakukan tugas-tugas berikut.

Untuk menghasilkan kode untuk file.matter kustom

1. Siapkan file.matter Anda
2. Jalankan perintah generasi:

```
./codegen.sh [--format] configs/dm_basic.json path-to-matter-file output-directory
```

(Opsional) Untuk menghasilkan kode dengan skema khusus

1. Siapkan skema kustom Anda dalam format JSON
2. Jalankan perintah generasi:

```
./codegen.sh [--format] configs/dm_basic.json path-to-matter-file output-directory  
--custom-schemas-dir path-to-custom-schema-directory
```

Perintah di atas menggunakan beberapa komponen untuk mengubah `.matter` file Anda menjadi C kode:

- `codegen.py` dari proyek ConnectedHomeIP
- Plugin Python terletak di `codegen/py_scripts/iotmi_data_model.py`
- Template Jinja2 dari folder `codegen/py_scripts/templates`

Plugin mendefinisikan variabel untuk diteruskan ke template Jinja2, yang kemudian digunakan untuk menghasilkan output kode C akhir. Menambahkan `--format` bendera menerapkan format Dentang ke kode yang dihasilkan.

Alur kerja pembuatan kode

Proses pembuatan kode mengatur struktur data file.matter Anda menggunakan fungsi utilitas dan penyortiran topologi. `topsort.py` Ini memastikan urutan yang tepat dari tipe data dan dependensinya.

Script kemudian menggabungkan spesifikasi file.matter Anda dengan pemrosesan plugin Python untuk mengekstrak dan memformat informasi yang diperlukan. Akhirnya, ini menerapkan format template Jinja2 untuk membuat output kode C akhir.

Alur kerja ini memastikan bahwa persyaratan khusus perangkat Anda dari file.matter diterjemahkan secara akurat ke dalam kode C fungsional yang terintegrasi dengan sistem integrasi terkelola.

Fungsi C tingkat rendah APIs

Integrasikan kode khusus perangkat Anda dengan integrasi terkelola menggunakan C-Function tingkat rendah yang disediakan. APIs Bagian ini menjelaskan operasi API yang tersedia untuk

setiap cluster dalam model AWS data untuk interaksi perangkat ke cloud yang efisien. Pelajari cara menerapkan fungsi callback, memancarkan peristiwa, memberi tahu perubahan atribut, dan mendaftarkan cluster untuk titik akhir perangkat Anda.

Komponen API utama meliputi:

1. Struktur penunjuk fungsi callback untuk atribut dan perintah
2. Fungsi emisi peristiwa
3. Fungsi pemberitahuan perubahan atribut
4. Fungsi registrasi cluster

Dengan menerapkan ini APIs, Anda membuat jembatan antara operasi fisik perangkat Anda dan fitur cloud integrasi terkelola, memastikan komunikasi dan kontrol yang mulus.

Bagian berikut menggambarkan API [OnOffcluster](#).

OnOff API kluster

[OnOff.xml](#) Cluster mendukung atribut dan perintah ini:.

- Atribut:
 - OnOff (boolean)
 - GlobalSceneControl (boolean)
 - OnTime (int16u)
 - OffWaitTime (int16u)
 - StartUpOnOff (StartUpOnOffEnum)
- Perintah:
 - Off : () -> Status
 - On : () -> Status
 - Toggle : () -> Status
 - OffWithEffect : (EffectIdentifier: EffectIdentifierEnum, EffectVariant: enum8) -> Status
 - OnWithRecallGlobalScene : () -> Status
 - OnWithTimedOff : (OnOffControl: OnOffControlBitmap, OnTime: int16u, OffWaitTime: int16u) -> Status

Untuk setiap perintah, kami menyediakan pointer fungsi yang dipetakan 1:1 yang dapat Anda gunakan untuk mengaitkan implementasi Anda.

Semua callback untuk atribut dan perintah didefinisikan dalam struct C dinamai cluster.

Contoh C struct

```
struct iotmiDev_clusterOnOff
{
    /*
     - Each attribute has a getter callback if it's readable

     - Each attribute has a setter callback if it's writable

     - The type of `value` are derived according to the data type of
       the attribute.

     - `user` is the pointer passed during an endpoint setup

     - The callback should return iotmiDev_DMStatus to report success or not.

     - For unsupported attributes, just leave them as NULL.
    */
    iotmiDev_DMStatus (*getOnTime)(uint16_t *value, void *user);
    iotmiDev_DMStatus (*setOnTime)(uint16_t value, void *user);
    /*
     - Each command has a command callback

     - If a command takes parameters, the parameters will be defined in a struct
       such as `iotmiDev_OnOff_OnWithTimedOffRequest` below.

     - `user` is the pointer passed during an endpoint setup

     - The callback should return iotmiDev_DMStatus to report success or not.

     - For unsupported commands, just leave them as NULL.
    */
    iotmiDev_DMStatus (*cmdOff)(void *user);
    iotmiDev_DMStatus (*cmdOnWithTimedOff)(const iotmiDev_OnOff_OnWithTimedOffRequest
    *request, void *user);
};
```

Selain struct C, fungsi pelaporan perubahan atribut didefinisikan untuk semua atribut.

```
/* Each attribute has a report function for the customer to report
   an attribute change. An attribute report function is thread-safe.
   */
void iotmiDev_OnOff_OnTime_report_attr(struct iotmiDev_Endpoint *endpoint, uint16_t
newValue, bool immediate);
```

Fungsi pelaporan peristiwa didefinisikan untuk semua peristiwa khusus cluster. Karena OnOff cluster tidak mendefinisikan peristiwa apa pun, di bawah ini adalah contoh dari CameraAvStreamManagement cluster.

```
/* Each event has a report function for the customer to report
   an event. An event report function is thread-safe.
   The iotmiDev_CameraAvStreamManagement_VideoStreamChangedEvent struct is
   derived from the event definition in the cluster.
   */
void iotmiDev_CameraAvStreamManagement_VideoStreamChanged_report_event(struct
iotmiDev_Endpoint *endpoint, const
iotmiDev_CameraAvStreamManagement_VideoStreamChangedEvent *event, bool immediate);
```

Setiap cluster juga memiliki fungsi register.

```
iotmiDev_DMStatus iotmiDev_OnOffRegisterCluster(struct iotmiDev_Endpoint *endpoint,
const struct iotmiDev_clusterOnOff *cluster, void *user);
```

Pointer pengguna yang diteruskan ke fungsi register akan diteruskan ke fungsi callback.

Interaksi fitur dan perangkat dalam integrasi terkelola

Bagian ini menjelaskan peran implementasi C-Function dan interaksi antara perangkat dan fitur perangkat integrasi terkelola.

Topik

- [Menangani perintah jarak jauh](#)
- [Menangani peristiwa yang tidak diminta](#)

Menangani perintah jarak jauh

Perintah jarak jauh ditangani oleh interaksi antara SDK perangkat Akhir dan fitur. Tindakan berikut menjelaskan contoh bagaimana Anda dapat menyalakan bola lampu menggunakan interaksi ini.

Klien MQTT menerima payload dan diteruskan ke Data Model Handler

Saat Anda mengirim perintah jarak jauh, klien MQTT menerima pesan dari integrasi terkelola dalam format JSON. Kemudian meneruskan payload ke handler model data. Misalnya, Anda ingin menggunakan integrasi terkelola untuk menyalakan bola lampu. Bola lampu memiliki titik akhir #1 yang mendukung OnOff cluster. Dalam hal ini, ketika Anda mengirim perintah untuk menyalakan bola lampu, integrasi terkelola mengirimkan permintaan melalui MQTT ke perangkat, yang mengatakan bahwa ia ingin memanggil perintah On pada titik akhir #1.

Data Model Handler memeriksa fungsi callback dan memanggilnya

Data Model Handler mem-parsing permintaan JSON. Jika permintaan berisi properti atau tindakan, Data Model Handler akan menemukan titik akhir dan secara berurutan memanggil fungsi callback yang sesuai. Misalnya, dalam kasus bola lampu, ketika Data Model Handler menerima pesan MQTT, ia memeriksa apakah fungsi callback yang sesuai dengan perintah On yang ditentukan dalam OnOff cluster terdaftar pada endpoint #1.

Implementasi Handler dan C-Function menjalankan perintah

Data Model Handler memanggil fungsi callback yang sesuai yang ditemukan dan memanggilnya. Implementasi C-Function kemudian memanggil fungsi perangkat keras yang sesuai untuk mengontrol perangkat keras fisik dan mengembalikan hasil eksekusi. Misalnya, dalam kasus bola lampu, Data Model Handler memanggil fungsi callback dan menyimpan hasil eksekusi. Fungsi callback kemudian menyalakan bola lampu sebagai hasilnya.

Data Model Handler mengembalikan hasil eksekusi

Setelah semua fungsi callback dipanggil, Data Model Handler menggabungkan semua hasil. Kemudian mengemas respons dalam format JSON dan menerbitkan hasilnya ke cloud integrasi terkelola menggunakan klien MQTT. Dalam kasus bola lampu, pesan MQTT dalam respons akan berisi hasil bahwa bola lampu dihidupkan oleh fungsi panggilan balik.

Menangani peristiwa yang tidak diminta

Peristiwa yang tidak diminta juga ditangani oleh interaksi antara SDK perangkat Akhir dan fitur. Tindakan berikut menjelaskan caranya.

Perangkat mengirimkan pemberitahuan ke Data Model Handler

Ketika perubahan properti atau peristiwa terjadi, seperti ketika tombol fisik telah ditekan pada perangkat, implementasi C-Function menghasilkan pemberitahuan peristiwa yang tidak diminta dan memanggil fungsi pemberitahuan yang sesuai untuk mengirim pemberitahuan ke Data Model Handler.

Data Model Handler menerjemahkan pemberitahuan

Data Model Handler menangani notifikasi yang diterima dan menerjemahkannya ke model AWS data.

Data Model Handler menerbitkan notifikasi ke Cloud

Data Model Handler kemudian menerbitkan peristiwa yang tidak diminta ke cloud integrasi terkelola menggunakan klien MQTT.

Memulai dengan End device SDK

Ikuti langkah-langkah berikut untuk menjalankan End device SDK pada perangkat Linux. Bagian ini memandu Anda melalui pengaturan lingkungan, konfigurasi jaringan, implementasi fungsi perangkat keras, dan konfigurasi titik akhir.

Important

Aplikasi demonstrasi dalam `examples` direktori dan implementasi Platform Abstraction Layer (PAL) di dalamnya hanya `platform/posix` untuk referensi. Jangan gunakan ini di lingkungan produksi.

Tinjau setiap langkah dari prosedur berikut dengan hati-hati untuk memastikan integrasi perangkat yang tepat dengan integrasi terkelola.

Integrasikan SDK perangkat Akhir

1. Siapkan EC2 instans Amazon

Masuk ke Konsol Manajemen AWS dan luncurkan EC2 instans Amazon menggunakan AMI Amazon Linux. Lihat [Memulai Amazon EC2](#) di [Panduan Pengguna Amazon Elastic Container Registry](#).

2. Siapkan lingkungan build

Buat kode di Amazon Linux 2023/x86_64 sebagai host pengembangan Anda. Instal dependensi build yang diperlukan:

```
dnf install make gcc gcc-c++ cmake
```

3. (Opsional) Siapkan jaringan

SDK perangkat akhir paling baik digunakan dengan perangkat keras fisik. Jika menggunakan Amazon EC2, jangan ikuti langkah ini.

Jika Anda tidak menggunakan Amazon EC2 sebelum menggunakan aplikasi sampel, inisialisasi jaringan dan sambungkan perangkat Anda ke jaringan Wi-Fi yang tersedia. Selesaikan pengaturan jaringan sebelum penyediaan perangkat:

```
/* Provisioning the device PKCS11 with claim credential. */  
status = deviceCredentialProvisioning();
```

4. Konfigurasi parameter penyediaan

Note

Ikuti [Provisioning](#) untuk mendapatkan sertifikat klaim dan kunci pribadi sebelum melanjutkan lebih lanjut.

Ubah file konfigurasi `example/project_name/device_config.sh` dengan parameter penyediaan berikut:

Parameter penyediaan

Parameter makro	Deskripsi	Cara mendapatkan informasi ini
IOTMI_R00 T_CA_PATH	File sertifikat CA root.	Anda dapat mengunduh file ini dari bagian Unduh sertifikat Amazon Root CA di panduan AWS IoT Core pengembang.

Parameter makro	Deskripsi	Cara mendapatkan informasi ini
IOTMI_CLAIM_CERTIFICATE_PATH	Jalur ke file sertifikat klaim.	Untuk mendapatkan sertifikat klaim dan kunci pribadi, buat profil penyediaan menggunakan API. CreateProvisioningProfile Untuk petunjuk, lihat Buat profil penyediaan .
IOTMI_PRIVATE_KEY_PATH	Jalur ke file kunci pribadi klaim.	
IOTMI_INTEGRATION_ENDPOINT	URL titik akhir untuk integrasi terkelola.	Untuk mendapatkan endpoint integrasi terkelola, gunakan API. RegisterCustomEndpoint Untuk petunjuk, lihat Daftarkan titik akhir kustom .
IOTMI_INTEGRATION_ENDPOINT_PORT	Nomor port untuk titik akhir integrasi terkelola	Secara default, port 8883 digunakan untuk operasi publikasi dan berlangganan MQTT. Port 443 diatur untuk ekstensi TLS Application Layer Protocol Negotiation (ALPN) yang digunakan perangkat.

5. Membangun dan menjalankan aplikasi demo

Bagian ini menunjukkan dua aplikasi demo Linux: kamera keamanan sederhana dan pembersih udara, keduanya digunakan CMake sebagai sistem build.

a. Aplikasi kamera keamanan sederhana

Untuk membangun dan menjalankan aplikasi, jalankan perintah ini:

```
>cd <path-to-code-drop>
# If you didn't generate cluster code earlier
>(cd codegen && poetry run poetry install --no-root && ./gen-data-model-api.sh)
>mkdir build
>cd build
>cmake ..
>cmake -build .
>./examples/iotmi_device_sample_camera/iotmi_device_sample_camera
```

Demo ini mengimplementasikan C-Functions tingkat rendah untuk kamera simulasi dengan RTC Session Controller dan Recording cluster. Selesaikan aliran yang disebutkan [Alur kerja penyediaan](#) sebelum menjalankan.

Contoh output dari aplikasi demo:

```
[2406832727][MAIN][INFO] ===== Device initialization and WIFI provisioning
=====
[2406832728][MAIN][INFO] fleetProvisioningTemplateName: XXXXXXXXXXXX
[2406832728][MAIN][INFO] managedintegrationsEndpoint: XXXXXXXXXXXX.account-prefix-
ats.iot.region.amazonaws.com
[2406832728][MAIN][INFO] pDeviceSerialNumber: XXXXXXXXXXXX
[2406832728][MAIN][INFO] universalProductCode: XXXXXXXXXXXX
[2406832728][MAIN][INFO] rootCertificatePath: XXXXXXXXXXXX
[2406832728][MAIN][INFO] pClaimCertificatePath: XXXXXXXXX
[2406832728][MAIN][INFO] pClaimKeyPath: XXXXXXXXXXXXXXXXXXXX
[2406832728][MAIN][INFO] deviceInfo.serialNumber XXXXXXXXXXXX
[2406832728][MAIN][INFO] deviceInfo.universalProductCode XXXXXXXXXXXXXXXXXXXX
[2406832728][PKCS11][INFO] PKCS #11 successfully initialized.
[2406832728][MAIN][INFO] ===== Start certificate provisioning
=====
[2406832728][PKCS11][INFO] ===== Loading Root CA and claim credentials
through PKCS#11 interface =====
[2406832728][PKCS11][INFO] Writing certificate into label "Root Cert".
[2406832728][PKCS11][INFO] Creating a 0x1 type object.
[2406832728][PKCS11][INFO] Writing certificate into label "Claim Cert".
[2406832728][PKCS11][INFO] Creating a 0x1 type object.
[2406832728][PKCS11][INFO] Creating a 0x3 type object.
[2406832728][MAIN][INFO] ===== Fleet-provisioning-by-Claim =====
[2025-01-02 01:43:11.404995144][iotmi_device_sdkLog][INFO] [2406832728]
[MQTT_AGENT][INFO]
[2025-01-02 01:43:11.405106991][iotmi_device_sdkLog][INFO] Establishing a TLS
session to XXXXXXXXXXXXXXXXXXXX.account-prefix-ats.iot.region.amazonaws.com
[2025-01-02 01:43:11.405119166][iotmi_device_sdkLog][INFO]
[2025-01-02 01:43:11.844812513][iotmi_device_sdkLog][INFO] [2406833168]
[MQTT_AGENT][INFO]
[2025-01-02 01:43:11.844842576][iotmi_device_sdkLog][INFO] TLS session
connected
[2025-01-02 01:43:11.844852105][iotmi_device_sdkLog][INFO]
[2025-01-02 01:43:12.296421687][iotmi_device_sdkLog][INFO] [2406833620]
[MQTT_AGENT][INFO]
[2025-01-02 01:43:12.296449663][iotmi_device_sdkLog][INFO] Session present: 0.
```

```

[2025-01-02 01:43:12.296458997][iotmi_device_sdkLog][INFO]
[2025-01-02 01:43:12.296467793][iotmi_device_sdkLog][INFO] [2406833620]
[MQTT_AGENT][INFO]
[2025-01-02 01:43:12.296476275][iotmi_device_sdkLog][INFO] MQTT connect with
clean session.
[2025-01-02 01:43:12.296484350][iotmi_device_sdkLog][INFO]
[2025-01-02 01:43:13.171056119][iotmi_device_sdkLog][INFO] [2406834494]
[FLEET_PROVISIONING][INFO]
[2025-01-02 01:43:13.171082442][iotmi_device_sdkLog][INFO] Received accepted
response from Fleet Provisioning CreateKeysAndCertificate API.
[2025-01-02 01:43:13.171092740][iotmi_device_sdkLog][INFO]
[2025-01-02 01:43:13.171122834][iotmi_device_sdkLog][INFO] [2406834494]
[FLEET_PROVISIONING][INFO]
[2025-01-02 01:43:13.171132400][iotmi_device_sdkLog][INFO] Received privatekey
and certificate with Id: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
[2025-01-02 01:43:13.171141107][iotmi_device_sdkLog][INFO]
[2406834494][PKCS11][INFO] Creating a 0x3 type object.
[2406834494][PKCS11][INFO] Writing certificate into label "Device Cert".
[2406834494][PKCS11][INFO] Creating a 0x1 type object.
[2025-01-02 01:43:18.584615126][iotmi_device_sdkLog][INFO] [2406839908]
[FLEET_PROVISIONING][INFO]
[2025-01-02 01:43:18.584662031][iotmi_device_sdkLog][INFO] Received accepted
response from Fleet Provisioning RegisterThing API.
[2025-01-02 01:43:18.584671912][iotmi_device_sdkLog][INFO]
[2025-01-02 01:43:19.100030237][iotmi_device_sdkLog][INFO] [2406840423]
[FLEET_PROVISIONING][INFO]
[2025-01-02 01:43:19.100061720][iotmi_device_sdkLog][INFO] Fleet-provisioning
iteration 1 is successful.
[2025-01-02 01:43:19.100072401][iotmi_device_sdkLog][INFO]
[2406840423][MQTT][ERROR] MQTT Connection Disconnected Successfully
[2025-01-02 01:43:19.216938181][iotmi_device_sdkLog][INFO] [2406840540]
[MQTT_AGENT][INFO]
[2025-01-02 01:43:19.216963713][iotmi_device_sdkLog][INFO] MQTT agent thread
leaves thread loop for iotmiDev_MQTTAgentStop.
[2025-01-02 01:43:19.216973740][iotmi_device_sdkLog][INFO]
[2406840540][MAIN][INFO] iotmiDev_MQTTAgentStop is called to break thread loop
function.
[2406840540][MAIN][INFO] Successfully provision the device.
[2406840540][MAIN][INFO] Client ID :
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
[2406840540][MAIN][INFO] Managed thing ID : XXXXXXXXXXXXXXXXXXXXXXXXXXXX
[2406840540][MAIN][INFO] ===== application loop
=====

```

```
[2025-01-02 01:43:19.217094828][iotmi_device_sdkLog][INFO] [2406840540]
[MQTT_AGENT][INFO]
[2025-01-02 01:43:19.217124600][iotmi_device_sdkLog][INFO] Establishing a TLS
session to XXXXXXXXXX.account-prefix-ats.iot.region.amazonaws.com:8883
[2025-01-02 01:43:19.217138724][iotmi_device_sdkLog][INFO]
[2406840540][Cluster OnOff][INFO] exampleOnOffInitCluster() for endpoint#1
[2406840540][MAIN][INFO] Press Ctrl+C when you finish testing...
[2406840540][Cluster ActivatedCarbonFilterMonitoring][INFO]
exampleActivatedCarbonFilterMonitoringInitCluster() for endpoint#1
[2406840540][Cluster AirQuality][INFO] exampleAirQualityInitCluster() for
endpoint#1
[2406840540][Cluster CarbonDioxideConcentrationMeasurement][INFO]
exampleCarbonDioxideConcentrationMeasurementInitCluster() for endpoint#1
[2406840540][Cluster FanControl][INFO] exampleFanControlInitCluster() for
endpoint#1
[2406840540][Cluster HepaFilterMonitoring][INFO]
exampleHepaFilterMonitoringInitCluster() for endpoint#1
[2406840540][Cluster Pm1ConcentrationMeasurement][INFO]
examplePm1ConcentrationMeasurementInitCluster() for endpoint#1
[2406840540][Cluster Pm25ConcentrationMeasurement][INFO]
examplePm25ConcentrationMeasurementInitCluster() for endpoint#1
[2406840540][Cluster TotalVolatileOrganicCompoundsConcentrationMeasurement]
[INFO]
exampleTotalVolatileOrganicCompoundsConcentrationMeasurementInitCluster() for
endpoint#1
[2025-01-02 01:43:19.648185488][iotmi_device_sdkLog][INFO] [2406840971]
[MQTT_AGENT][INFO]
[2025-01-02 01:43:19.648211988][iotmi_device_sdkLog][INFO] TLS session
connected
[2025-01-02 01:43:19.648225583][iotmi_device_sdkLog][INFO]

[2025-01-02 01:43:19.938281231][iotmi_device_sdkLog][INFO] [2406841261]
[MQTT_AGENT][INFO]
[2025-01-02 01:43:19.938304799][iotmi_device_sdkLog][INFO] Session present: 0.
[2025-01-02 01:43:19.938317404][iotmi_device_sdkLog][INFO]
```

b. Aplikasi pembersih udara sederhana

Untuk membangun dan menjalankan aplikasi, jalankan perintah berikut:

```
>cd <path-to-code-drop>
# If you didn't generate cluster code earlier
>(cd codegen && poetry run poetry install --no-root && ./gen-data-model-api.sh)
>mkdir build
```

```
>cd build
>cmake ..
>cmake --build .
>./examples/iotmi_device_dm_air_purifier/iotmi_device_dm_air_purifier_demo
```

Demo ini mengimplementasikan C-Functions tingkat rendah untuk pembersih udara simulasi dengan 2 titik akhir dan cluster yang didukung berikut:

Cluster yang didukung untuk titik akhir pembersih udara

Titik akhir	klaster
Titik Akhir #1: Pembersih Udara	OnOff
	Kontrol Kipas
	Pemantauan Filter HEPA
	Pemantauan Filter Karbon Aktif
Titik Akhir #2: Sensor Kualitas Udara	Kualitas Udara
	Pengukuran Konsentrasi Karbon Dioksida
	Pengukuran Konsentrasi Formaldehida
	Pengukuran Konsentrasi Pm25
	Pengukuran Konsentrasi Pm1
	Pengukuran Konsentrasi Senyawa Organik Volatile Total

Outputnya mirip dengan aplikasi demo kamera, dengan berbagai cluster yang didukung.

6. Langkah selanjutnya:

Integrasi terkelola SDK perangkat akhir dan aplikasi demo sekarang berjalan di instans Amazon EC2 Anda. Ini memungkinkan Anda untuk mengembangkan dan menguji aplikasi Anda pada perangkat keras fisik Anda sendiri. Dengan pengaturan ini, Anda dapat memanfaatkan layanan Integrasi terkelola untuk mengontrol AWS IoT perangkat Anda.

a. Kembangkan fungsi panggilan balik perangkat keras

Sebelum menerapkan fungsi callback perangkat keras, pahami cara kerja API. Contoh ini menggunakan On/Off cluster dan OnOff atribut untuk mengontrol fungsi perangkat. Untuk detail API, lihat [Fungsi C tingkat rendah APIs](#).

```
struct DeviceState
{
    struct iotmiDev_Agent *agent;
    struct iotmiDev_Endpoint *endpointLight;
    /* This simulates the HW state of OnOff */
    bool hwState;
};

/* This implementation for OnOff getter just reads
the state from the DeviceState */
iotmiDev_DMStatus exampleGetOnOff(bool *value, void *user)
{
    struct DeviceState *state = (struct DeviceState *) (user);
    *value = state->hwState;
    return iotmiDev_DMStatusOk;
}
```

b. Siapkan titik akhir dan kaitkan fungsi panggilan balik perangkat keras

Setelah mengimplementasikan fungsi, buat titik akhir dan daftarkan callback Anda. Lakukan hal-hal berikut:

- i. Buat agen perangkat.
 - A. Buat agen perangkat menggunakan `iotmiDev_Agent_new()` sebelum Anda menjalankan fungsi SDK lainnya.
 - B. Minimal, konfigurasi Anda harus menyertakan parameter ThingId dan ClientID.
 - C. Gunakan `iotmiDev_Agent_initDefaultConfig()` fungsi untuk menetapkan nilai default yang wajar untuk parameter seperti ukuran antrian dan titik akhir maksimum.
 - D. Ketika Anda selesai menggunakan sumber daya, bebaskan mereka dengan `iotmiDev_Agent_free()` fungsi. Ini mencegah kebocoran memori dan memastikan manajemen sumber daya yang tepat dalam aplikasi Anda.

- ii. Isi pointer fungsi callback untuk setiap struktur cluster yang ingin Anda dukung.
- iii. Siapkan titik akhir dan daftarkan kluster yang didukung.

Buat titik akhir dengan `iotmiDev_Agent_addEndpoint()`, yang membutuhkan:

- A. ID titik akhir yang unik.
- B. Nama titik akhir deskriptif
- C. Satu atau lebih jenis perangkat yang sesuai dengan definisi model AWS data.
- D. Setelah membuat titik akhir, daftarkan cluster menggunakan fungsi registrasi khusus cluster yang sesuai.
- E. Setiap registrasi cluster memerlukan fungsi callback untuk atribut dan perintah. Sistem meneruskan pointer konteks pengguna Anda ke callback untuk mempertahankan status di antara panggilan.

```
struct DeviceState
{
    struct iotmiDev_Agent * agent;
    struct iotmiDev_Endpoint *endpoint1;

    /* OnOff cluster states*/
    bool hwState;
};

/* This implementation for OnOff getter just reads
the state from the DeviceState */
iotmiDev_DMStatus exampleGetOnOff( bool * value, void * user )
{
    struct DeviceState * state = ( struct DeviceState * ) ( user );
    *value = state->hwState;
    printf( "%s(): state->hwState: %d\n", __func__, state->hwState );
    return iotmiDev_DMStatusOk;
}

iotmiDev_DMStatus exampleGetOnTime( uint16_t * value, void * user )
{
    *value = 0;
    printf( "%s(): OnTime is %u\n", __func__, *value );
    return iotmiDev_DMStatusOk;
}
```

```

}

iotmiDev_DMStatus exampleGetStartupOnOff( iotmiDev_OnOff_StartUpOnOffEnum *
value, void * user )
{
    *value = iotmiDev_OnOff_StartUpOnOffEnum_Off;
    printf( "%s(): StartupOnOff is %d\n", __func__, *value );
    return iotmiDev_DMStatusOk;
}

void setupOnOff( struct DeviceState *state )
{
    struct iotmiDev_clusterOnOff clusterOnOff = {
        .getOnOff = exampleGetOnOff,
        .getOnTime = exampleGetOnTime,
        .getStartupOnOff = exampleGetStartupOnOff,
    };
    iotmiDev_OnOffRegisterCluster( state->endpoint1,
        &clusterOnOff,
        ( void * ) state);
}

/* Here is the sample setting up an endpoint 1 with OnOff
cluster. Note all error handling code is omitted. */
void setupAgent(struct DeviceState *state)
{
    struct iotmiDev_Agent_Config config = {
        .thingId = IOTMI_DEVICE_MANAGED_THING_ID,
        .clientId = IOTMI_DEVICE_CLIENT_ID,
    };
    iotmiDev_Agent_InitDefaultConfig(&config);

    /* Create a device agent before calling other SDK APIs */
    state->agent = iotmiDev_Agent_new(&config);

    /* Create endpoint#1 */
    state->endpoint1 = iotmiDev_Agent_addEndpoint( state->agent,
        1,
        "Data Model Handler Test
Device",
        (const char*[])
{ "Camera" },
        1 );
}

```

```

    setupOnOff(state);
}

```

c. Gunakan penanganan pekerjaan untuk mendapatkan dokumen pekerjaan

i. Memulai panggilan ke aplikasi OTA Anda:

```

static iotmi_JobCurrentStatus_t processOTA( iotmi_JobData_t * pJobData )
{
    iotmi_JobCurrentStatus_t jobCurrentStatus = JobSucceeded;

    ...
    // This function should create OTA tasks
    jobCurrentStatus = YOUR_OTA_FUNCTION(iotmi_JobData_t * pJobData);
    ...

    return jobCurrentStatus;
}

```

- ii. Panggilan `iotmi_JobsHandler_start` untuk menginisialisasi penanganan pekerjaan.
- iii. Panggilan `iotmi_JobsHandler_getJobDocument` untuk mengambil Dokumen pekerjaan dari integrasi terkelola.
- iv. Ketika Dokumen Pekerjaan berhasil diperoleh, tulis operasi OTA khusus Anda dalam `processOTA` fungsi dan kembalikan `JobSucceeded` status.

```

static void prvJobsHandlerThread( void * pParam )
{
    JobsHandlerStatus_t status = JobsHandlerSuccess;
    iotmi_JobData_t jobDocument;
    iotmiDev_DeviceRecord_t * pThreadParams = ( iotmiDev_DeviceRecord_t * )
pParam;
    iotmi_JobsHandler_config_t config = { .pManagedThingID = pThreadParams-
>pManagedThingID, .jobsQueueSize = 10 };

    status = iotmi_JobsHandler_start( &config );

    if( status != JobsHandlerSuccess )
    {
        LogError( ( "Failed to start Jobs Handler." ) );
        return;
    }
}

```

```
while( !bExit )
{
    status = iotmi_JobsHandler_getJobDocument( &jobDocument, 30000 );

    switch( status )
    {
        case JobsHandlerSuccess:
        {
            LogInfo( ( "Job document received." ) );
            LogInfo( ( "Job ID: %.*s", ( int ) jobDocument.jobIdLength,
jobDocument.pJobId ) );
            LogInfo( ( "Job document: %.*s", ( int )
jobDocument.jobDocumentLength, jobDocument.pJobDocument ) );

            /* Process the job document */
            iotmi_JobCurrentStatus_t jobStatus =
processOTA( &jobDocument );

            iotmi_JobsHandler_updateJobStatus( jobDocument.pJobId,
jobDocument.jobIdLength, jobStatus, NULL, 0 );

            iotmiJobsHandler_destroyJobDocument(&jobDocument);

            break;
        }
        case JobsHandlerTimeout:
        {
            LogInfo( ( "No job document available. Polling for job
document." ) );

            iotmi_JobsHandler_pollJobDocument();

            break;
        }
        default:
        {
            LogError( ( "Failed to get job document." ) );
            break;
        }
    }
}

while( iotmi_JobsHandler_getJobDocument( &jobDocument, 0 ) ==
JobsHandlerSuccess )
```

```
{
    /* Before stopping the Jobs Handler, process all the remaining
    jobs. */

    LogInfo( ( "Job document received before stopping." ) );
    LogInfo( ( "Job ID: %.*s", ( int ) jobDocument.jobIdLength,
    jobDocument.pJobId ) );
    LogInfo( ( "Job document: %.*s", ( int )
    jobDocument.jobDocumentLength, jobDocument.pJobDocument ) );

    storeJobs( &jobDocument );

    iotmiJobsHandler_destroyJobDocument(&jobDocument);
}

iotmi_JobsHandler_stop();

LogInfo( ( "Job handler thread end." ) );
}
```

Port SDK perangkat Akhir ke perangkat Anda

Port SDK perangkat Akhir ke platform perangkat Anda. Ikuti langkah-langkah berikut untuk menghubungkan perangkat Anda dengan Manajemen AWS IoT Perangkat.

Unduh dan verifikasi SDK perangkat Akhir

1. Unduh versi terbaru SDK perangkat Akhir dari konsol [integrasi terkelola](#).
2. Verifikasi bahwa platform Anda ada dalam daftar platform yang didukung di [Referensi: Platform yang didukung](#).

Note

SDK perangkat Akhir telah diuji pada platform yang ditentukan. Platform lain mungkin berfungsi, tetapi belum diuji.

3. Ekstrak (unzip) file SDK ke ruang kerja Anda.
4. Konfigurasi lingkungan build Anda dengan setelan berikut:

- Jalur file sumber
 - Direktori file header
 - Pustaka yang dibutuhkan
 - Flag compiler dan linker
5. Sebelum Anda mem-port Platform Abstraction Layer (PAL), pastikan fungsionalitas dasar platform Anda diinisialisasi. Fungsionalitas meliputi:
- Tugas sistem operasi
 - Periferal
 - Antarmuka jaringan
 - Persyaratan khusus platform

Port PAL ke perangkat Anda

1. Buat direktori baru untuk implementasi khusus platform Anda di direktori platform yang ada. Misalnya, jika Anda menggunakan FreeRTOS, buat direktori di `platform/freertos`

Example Struktur direktori SDK

```
### <SDK_ROOT_FOLDER>
#   ### CMakeLists.txt
#   ### LICENSE.txt
#   ### cmake
#   ### commonDependencies
#   ### components
#   ### docs
#   ### examples
#   ### include
#   ### lib
#   ### platform
#   ### test
#   ### tools
```

2. Salin file implementasi referensi POSIX (.c dan.h) dari folder posix ke direktori platform baru Anda. File-file ini menyediakan template untuk fungsi yang perlu Anda terapkan.
- Manajemen memori flash untuk penyimpanan kredensi

- Implementasi PKCS #11
 - Antarmuka transportasi jaringan
 - Sinkronisasi waktu
 - Fungsi reboot dan reset sistem
 - Mekanisme pencatatan log
 - Konfigurasi khusus perangkat
3. Siapkan otentikasi Transport Layer Security (TLS) dengan Mbed TLS.
- Gunakan implementasi POSIX yang disediakan jika Anda sudah memiliki versi Mbed TLS yang cocok dengan versi SDK di platform Anda.
 - Dengan versi TLS yang berbeda, Anda menerapkan kait transportasi untuk tumpukan TLS Anda dengan tumpukan. TCP/IP
4. Bandingkan konfigurasi mBEDTLS platform Anda dengan persyaratan SDK di. `platform/posix/mbedtls/mbedtls_config.h` Pastikan semua opsi yang diperlukan diaktifkan.
5. SDK bergantung pada CoreMQTT untuk berinteraksi dengan cloud. Oleh karena itu, Anda harus menerapkan lapisan transport jaringan yang menggunakan struktur berikut:

```
typedef struct TransportInterface
{
    TransportRecv_t recv;
    TransportSend_t send;
    NetworkContext_t * pNetworkContext;
} TransportInterface_t;
```

Untuk informasi selengkapnya, lihat [dokumentasi antarmuka Transport](#) di situs web FreeRTOS.

6. (Opsional) SDK menggunakan PCS #11 API untuk menangani operasi sertifikat. CorePKCS adalah implementasi PKCS #11 non-hardware khusus untuk pembuatan prototipe. Kami menyarankan Anda menggunakan kriptoprosesor aman seperti Trusted Platform Module (TPM), Hardware Security Module (HSM), atau Secure Element di lingkungan produksi Anda:
- Tinjau contoh implementasi PKCS #11 yang menggunakan sistem file Linux untuk manajemen kredensial di. `platform/posix/corePKCS11-mbedtls`
 - Implementasikan layer PKCS #11 PAL di. `commonDependencies/core_pkcs11/corePKCS11/source/include/core_pkcs11.h`

- Menerapkan sistem file Linux diplatform/posix/corePKCS11-mbedtls/source/iotmi_pal_Pkcs110perations.c.
- Menerapkan fungsi penyimpanan dan muat dari jenis penyimpanan Anda diplatform/include/iotmi_pal_Nvm.h.
- Menerapkan akses file standar diplatform/posix/source/iotmi_pal_Nvm.c.

Untuk petunjuk porting terperinci, lihat [Mem-porting PKCS11 pustaka inti di Panduan Pengguna FreeRTOS](#).

7. Tambahkan pustaka statis SDK ke lingkungan build Anda:
 - Siapkan jalur pustaka untuk menyelesaikan masalah tautan atau konflik simbol
 - Verifikasi semua dependensi ditautkan dengan benar

Uji port Anda

Anda dapat menggunakan aplikasi contoh yang ada untuk menguji port Anda. Kompilasi harus selesai tanpa kesalahan atau peringatan.

Note

Kami menyarankan Anda memulai dengan aplikasi multitasking yang paling sederhana. Aplikasi contoh menyediakan setara multitasking.

1. Temukan contoh aplikasi diexamples/[device_type_sample].
2. Konversikan main.c file ke proyek Anda, dan tambahkan entri untuk memanggil fungsi main () yang ada.
3. Verifikasi bahwa Anda dapat mengkompilasi aplikasi demo dengan sukses.

Referensi teknis

Topik

- [Referensi: Platform yang didukung](#)
- [Referensi: Persyaratan teknis](#)
- [Referensi: API Umum](#)

Referensi: Platform yang didukung

Tabel berikut menampilkan platform yang didukung untuk SDK.

Platform yang didukung

Platform	Arsitektur	Sistem operasi
Linux x86_64	x86_64	Linux
Ambarella	Armv8 () AArch64	Linux
AmeBad	Armv8-M 32 bit	FreeRTOS
ESP32S3	Xtensa 32 bit LX7	FreeRTOS

Referensi: Persyaratan teknis

Tabel berikut menunjukkan persyaratan teknis untuk SDK, termasuk ruang RAM. SDK perangkat Akhir sendiri membutuhkan sekitar 5 hingga 10 MB ruang ROM saat menggunakan konfigurasi yang sama.

Ruang RAM

SDK dan komponen	Persyaratan ruang (byte digunakan)
Akhiri SDK perangkat itu sendiri	180 KB
Antrian perintah Agen MQTT default	480 byte (dapat dikonfigurasi)
Antrian masuk Agen MQTT default	320 byte (dapat dikonfigurasi)

Referensi: API Umum

Bagian ini adalah daftar operasi API yang tidak spesifik untuk klaster.

```
/* return code for data model related API */
enum iotmiDev_DMStatus
{
    /* The operation succeeded */

```

```
iotmiDev_DMStatusOk = 0,
/* The operation failed without additional information */
iotmiDev_DMStatusFail = 1,
/* The operation has not been implemented yet. */
iotmiDev_DMStatusNotImplement = 2,
/* The operation is to create a resource, but the resource already exists. */
iotmiDev_DMStatusExist = 3,
}

/* The opaque type to represent a instance of device agent. */
struct iotmiDev_Agent;

/* The opaque type to represent an endpoint. */
struct iotmiDev_Endpoint;

/* A device agent should be created before calling other API */
struct iotmiDev_Agent* iotmiDev_create_agent();

/* Destroy the agent and free all occupied resources */
void iotmiDev_destroy_agent(struct iotmiDev_Agent *agent);

/* Add an endpoint, which starts with empty capabilities */
struct iotmiDev_Endpoint* iotmiDev_addEndpoint(struct iotmiDev_Agent *handle, uint16
id, const char *name);

/* Test all clusters registered within an endpoint.
Note: this API might exist only for early drop. */
void iotmiDev_testEndpoint(struct iotmiDev_Endpoint *endpoint);
```

Keamanan dalam integrasi terkelola untuk AWS IoT Device Management

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

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

- Keamanan cloud — AWS bertanggung jawab untuk melindungi infrastruktur yang menjalankan AWS layanan di AWS Cloud. AWS juga memberi Anda layanan yang dapat Anda gunakan dengan aman. Auditor pihak ketiga secara teratur menguji dan memverifikasi efektivitas keamanan kami sebagai bagian dari [Program AWS Kepatuhan Program AWS Kepatuhan](#) . Untuk mempelajari tentang program kepatuhan yang berlaku untuk integrasi terkelola, lihat [AWS Layanan dalam Lingkup oleh AWS Layanan Program Kepatuhan](#) .
- Keamanan di cloud — Tanggung jawab Anda ditentukan oleh AWS layanan yang Anda gunakan. Anda juga bertanggung jawab atas faktor lain, yang mencakup sensitivitas data Anda, persyaratan perusahaan Anda, serta undang-undang dan peraturan yang berlaku.

Dokumentasi ini membantu Anda memahami cara menerapkan model tanggung jawab bersama saat menggunakan integrasi terkelola. Topik berikut menunjukkan cara mengonfigurasi integrasi terkelola untuk memenuhi tujuan keamanan dan kepatuhan Anda. Anda juga mempelajari cara menggunakan AWS layanan lain yang membantu Anda memantau dan mengamankan sumber daya integrasi terkelola Anda.

Topik

- [Perlindungan data dalam integrasi terkelola](#)
- [Manajemen identitas dan akses untuk integrasi terkelola](#)
- [Gunakan AWS Secrets Manager untuk perlindungan data untuk alur kerja C2C](#)
- [Validasi kepatuhan untuk integrasi terkelola](#)
- [Gunakan integrasi terkelola dengan titik akhir VPC antarmuka](#)
- [Connect ke integrasi terkelola untuk endpoint AWS IoT Device Management FIPS](#)

Perlindungan data dalam integrasi terkelola

[Model tanggung jawab AWS bersama model](#) berlaku untuk perlindungan data dalam integrasi Terkelola untuk AWS IoT Device Management. Seperti yang dijelaskan dalam model AWS ini, bertanggung jawab untuk melindungi infrastruktur global yang menjalankan semua AWS Cloud. Anda bertanggung jawab untuk mempertahankan kendali atas konten yang di-host pada infrastruktur ini. Anda juga bertanggung jawab atas tugas-tugas konfigurasi dan manajemen keamanan untuk Layanan AWS yang Anda gunakan. Lihat informasi yang lebih lengkap tentang privasi data dalam [Pertanyaan Umum Privasi Data](#). Lihat informasi tentang perlindungan data di Eropa di pos blog [Model Tanggung Jawab Bersama dan GDPR AWS](#) di Blog Keamanan AWS .

Untuk tujuan perlindungan data, kami menyarankan Anda melindungi Akun AWS kredensial dan mengatur pengguna individu dengan AWS IAM Identity Center atau AWS Identity and Access Management (IAM). Dengan cara itu, setiap pengguna hanya diberi izin yang diperlukan untuk memenuhi tanggung jawab tugasnya. Kami juga menyarankan supaya Anda mengamankan data dengan cara-cara berikut:

- Gunakan autentikasi multi-faktor (MFA) pada setiap akun.
- Gunakan SSL/TLS untuk berkomunikasi dengan AWS sumber daya. Kami mensyaratkan TLS 1.2 dan menganjurkan TLS 1.3.
- Siapkan API dan pencatatan aktivitas pengguna dengan AWS CloudTrail. Untuk informasi tentang penggunaan CloudTrail jejak untuk menangkap AWS aktivitas, lihat [Bekerja dengan CloudTrail jejak](#) di AWS CloudTrail Panduan Pengguna.
- Gunakan solusi AWS enkripsi, bersama dengan semua kontrol keamanan default di dalamnya Layanan AWS.
- Gunakan layanan keamanan terkelola tingkat lanjut seperti Amazon Macie, yang membantu menemukan dan mengamankan data sensitif yang disimpan di Amazon S3.
- Jika Anda memerlukan modul kriptografi tervalidasi FIPS 140-3 saat mengakses AWS melalui antarmuka baris perintah atau API, gunakan titik akhir FIPS. Lihat informasi selengkapnya tentang titik akhir FIPS yang tersedia di [Standar Pemrosesan Informasi Federal \(FIPS\) 140-3](#).

Kami sangat merekomendasikan agar Anda tidak pernah memasukkan informasi identifikasi yang sensitif, seperti nomor rekening pelanggan Anda, ke dalam tanda atau bidang isian bebas seperti bidang Nama. Ini termasuk saat Anda bekerja dengan integrasi terkelola untuk AWS IoT Device Management atau lainnya Layanan AWS menggunakan konsol, API AWS CLI, atau AWS SDKs. Data apa pun yang Anda masukkan ke dalam tanda atau bidang isian bebas yang digunakan untuk

nama dapat digunakan untuk log penagihan atau log diagnostik. Saat Anda memberikan URL ke server eksternal, kami sangat menganjurkan supaya Anda tidak menyertakan informasi kredensial di dalam URL untuk memvalidasi permintaan Anda ke server itu.

Enkripsi data saat istirahat untuk integrasi terkelola

Integrasi terkelola untuk AWS IoT Device Management mengenkripsi data pelanggan sensitif saat istirahat secara default menggunakan kunci enkripsi.

Ada dua jenis kunci enkripsi yang digunakan untuk melindungi data sensitif bagi pelanggan integrasi terkelola:

Kunci terkelola pelanggan (CMK)

Integrasi terkelola mendukung penggunaan kunci terkelola pelanggan simetris yang dapat Anda buat, miliki, dan kelola. Anda memiliki kontrol penuh atas kunci KMS ini, termasuk membuat dan memelihara kebijakan utama mereka, kebijakan IAM, dan hibah, mengaktifkan dan menonaktifkannya, memutar materi kriptografi mereka, menambahkan tag, membuat alias yang merujuk ke kunci KMS, dan menjadwalkan kunci KMS untuk dihapus.

AWS kunci yang dimiliki

Integrasi terkelola menggunakan kunci ini secara default untuk mengenkripsi data pelanggan yang sensitif secara otomatis. Anda tidak dapat melihat, mengelola, atau mengaudit penggunaannya. Anda tidak perlu mengambil tindakan apa pun atau mengubah program apa pun untuk melindungi kunci yang mengenkripsi data Anda. Enkripsi data saat istirahat secara default membantu mengurangi overhead operasional dan kompleksitas yang terlibat dalam melindungi data sensitif. Pada saat yang sama, ini memungkinkan Anda untuk membangun aplikasi aman yang memenuhi kepatuhan enkripsi yang ketat dan persyaratan peraturan.


Kunci enkripsi default yang digunakan adalah kunci AWS milik. Atau, API opsional untuk memperbarui kunci enkripsi Anda adalah [PutDefaultEncryptionConfiguration](#).

Untuk informasi selengkapnya tentang jenis kunci AWS KMS enkripsi, lihat [AWS KMS kunci](#).

AWS KMS penggunaan untuk integrasi terkelola

Integrasi terkelola mengenkripsi dan mendekripsi semua data pelanggan menggunakan enkripsi amplop. Jenis enkripsi ini akan mengambil data teks biasa Anda dan mengenkripsi dengan kunci data. Selanjutnya, kunci enkripsi yang disebut kunci pembungkus akan mengenkripsi kunci data asli

yang digunakan untuk mengenkripsi data plaintext Anda. Dalam enkripsi amplop, kunci pembungkus tambahan dapat digunakan untuk mengenkripsi kunci pembungkus yang ada yang lebih dekat dalam derajat pemisahan dari kunci data asli. Karena kunci data asli dienkripsi oleh kunci pembungkus yang disimpan secara terpisah, Anda dapat menyimpan kunci data asli dan data plaintext terenkripsi di lokasi yang sama. Keyring digunakan untuk menghasilkan, mengenkripsi, dan mendekripsi kunci data selain kunci pembungkus yang digunakan untuk mengenkripsi dan mendekripsi kunci data.

 Note

AWS Database Encryption SDK menyediakan enkripsi amplop untuk implementasi enkripsi sisi klien Anda. Untuk informasi selengkapnya tentang SDK Enkripsi AWS Database, lihat [Apa itu SDK Enkripsi AWS Database?](#)

[Untuk informasi selengkapnya tentang enkripsi amplop, kunci data, kunci pembungkus, dan gantungan kunci, lihat Enkripsi amplop, Kunci data, Kunci pembungkus, dan gantungan kunci.](#)

Integrasi terkelola memerlukan layanan untuk menggunakan kunci terkelola pelanggan Anda untuk operasi internal berikut:

- Kirim `DescribeKey` permintaan AWS KMS untuk memverifikasi bahwa ID kunci terkelola pelanggan simetris disediakan saat melakukan rotasi kunci data.
- Kirim `GenerateDataKeyWithoutPlaintext` permintaan AWS KMS untuk menghasilkan kunci data yang dienkripsi oleh kunci terkelola pelanggan Anda.
- Kirim `ReEncrypt*` permintaan AWS KMS untuk mengenkripsi ulang kunci data dengan kunci terkelola pelanggan Anda.
- Kirim `Decrypt` permintaan AWS KMS untuk mendekripsi data dengan kunci terkelola pelanggan Anda.

Jenis data yang dienkripsi menggunakan kunci enkripsi

Integrasi terkelola menggunakan kunci enkripsi untuk mengenkripsi beberapa jenis data yang disimpan saat istirahat. Daftar berikut menguraikan jenis data yang dienkripsi saat istirahat menggunakan kunci enkripsi:

- Peristiwa konektor cloud-to-cloud (C2C) seperti penemuan perangkat dan pembaruan status perangkat.

- Pembuatan Hal Terkelola yang mewakili perangkat fisik dan profil perangkat yang berisi kemampuan untuk jenis perangkat tertentu. Untuk informasi selengkapnya tentang profil perangkat dan perangkat, lihat [Perangkat](#) dan [Perangkat](#).
- Pemberitahuan integrasi terkelola pada berbagai aspek implementasi perangkat Anda. Untuk informasi selengkapnya tentang notifikasi integrasi terkelola, lihat [Siapkan notifikasi integrasi terkelola](#).
- Informasi Identifikasi Pribadi (PII) dari pengguna akhir seperti materi otentikasi perangkat, nomor seri perangkat, nama pengguna akhir, pengenal perangkat, dan Nama Sumber Daya Amazon perangkat (arn).

Bagaimana integrasi terkelola menggunakan kebijakan utama di AWS KMS

Untuk rotasi kunci cabang dan panggilan asinkron, integrasi terkelola memerlukan kebijakan kunci untuk menggunakan kunci enkripsi Anda. Kebijakan utama digunakan untuk alasan berikut:

- Secara terprogram mengotorisasi penggunaan kunci enkripsi ke prinsipal lain. AWS

Untuk contoh kebijakan kunci yang digunakan untuk mengelola akses ke kunci enkripsi Anda dalam integrasi terkelola, lihat [Buat kunci enkripsi](#)

Note

Untuk kunci yang AWS dimiliki, kebijakan kunci tidak diperlukan karena kunci yang AWS dimiliki dimiliki oleh AWS dan Anda tidak dapat melihat, mengelola, atau menggunakannya. Integrasi terkelola menggunakan kunci yang AWS dimiliki secara default untuk mengenkripsi data pelanggan sensitif Anda secara otomatis.

Selain menggunakan kebijakan utama untuk mengelola konfigurasi enkripsi Anda dengan AWS KMS kunci, integrasi terkelola menggunakan kebijakan IAM. Untuk informasi selengkapnya tentang kebijakan IAM, lihat [Kebijakan dan izin](#) di AWS Identity and Access Management

Buat kunci enkripsi

Anda dapat membuat kunci enkripsi dengan menggunakan Konsol Manajemen AWS atau AWS KMS APIs.

Untuk membuat kunci enkripsi

Ikuti langkah-langkah untuk [Membuat kunci KMS](#) di Panduan AWS Key Management Service Pengembang.

Kebijakan utama

Pernyataan kebijakan kunci mengontrol akses ke AWS KMS kunci. Setiap AWS KMS kunci hanya akan berisi satu kebijakan kunci. Kebijakan kunci tersebut menentukan AWS prinsipal mana yang dapat menggunakan kunci tersebut dan bagaimana mereka dapat menggunakannya. Untuk informasi selengkapnya tentang mengelola akses dan penggunaan AWS KMS kunci menggunakan pernyataan kebijakan utama, lihat [Mengelola akses menggunakan kebijakan](#).

Berikut ini adalah contoh pernyataan kebijakan kunci yang dapat Anda gunakan untuk mengelola akses dan penggunaan AWS KMS kunci yang disimpan dalam integrasi Akun AWS terkelola Anda:

```
{
  "Statement" : [
    {
      "Sid" : "Allow access to principals authorized to use managed integrations",
      "Effect" : "Allow",
      "Principal" : {
        //Note: Both role and user are acceptable.
        "AWS" : "arn:aws:iam::111122223333:user/username",
        "AWS" : "arn:aws:iam::111122223333:role/roleName"
      },
      "Action" : [
        "kms:GenerateDataKeyWithoutPlaintext",
        "kms:Decrypt",
        "kms:ReEncrypt*"
      ],
      "Resource" : "arn:aws:kms:region:111122223333:key/key_ID",
      "Condition" : {
        "StringEquals" : {
          "kms:ViaService" : "iotmanagedintegrations.amazonaws.com"
        },
        "ForAnyValue:StringEquals": {
          "kms:EncryptionContext:aws-crypto-ec:iotmanagedintegrations": "111122223333"
        },
        "ArnLike": {
          "aws:SourceArn": [
            "arn:aws:iotmanagedintegrations:<region>:<accountId>:managed-thing/
            <managedThingId>",
            "arn:aws:iotmanagedintegrations:<region>:<accountId>:credential-locker/
            <credentialLockerId>",
```

```

        "arn:aws:iotmanagedintegrations:<region>:<accountId>:provisioning-profile/
<provisioningProfileId>",
        "arn:aws:iotmanagedintegrations:<region>:<accountId>:ota-task/<otaTaskId>"
    ]
}
},
{
    "Sid" : "Allow access to principals authorized to use managed integrations for
async flow",
    "Effect" : "Allow",
    "Principal" : {
        "Service": "iotmanagedintegrations.amazonaws.com"
    },
    "Action" : [
        "kms:GenerateDataKeyWithoutPlaintext",
        "kms:Decrypt",
        "kms:ReEncrypt*"
    ],
    "Resource" : "arn:aws:kms:region:111122223333:key/key_ID",
    "Condition" : {
        "ForAnyValue:StringEquals": {
            "kms:EncryptionContext:aws-crypto-ec:iotmanagedintegrations": "111122223333"
        },
        "ArnLike": {
            "aws:SourceArn": [
                "arn:aws:iotmanagedintegrations:<region>:<accountId>:managed-thing/
<managedThingId>",
                "arn:aws:iotmanagedintegrations:<region>:<accountId>:credential-locker/
<credentialLockerId>",
                "arn:aws:iotmanagedintegrations:<region>:<accountId>:provisioning-profile/
<provisioningProfileId>",
                "arn:aws:iotmanagedintegrations:<region>:<accountId>:ota-task/<otaTaskId>"
            ]
        }
    }
},
{
    "Sid" : "Allow access to principals authorized to use managed integrations for
describe key",
    "Effect" : "Allow",
    "Principal" : {
        "AWS": "arn:aws:iam::111122223333:user/username"
    },

```

```
"Action" : [
  "kms:DescribeKey",
],
"Resource" : "arn:aws:kms:region:111122223333:key/key_ID",
"Condition" : {
  "StringEquals" : {
    "kms:ViaService" : "iotmanagedintegrations.amazonaws.com"
  }
}
},
{
  "Sid": "Allow access for key administrators",
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:root"
  },
  "Action" : [
    "kms:*"
  ],
  "Resource": "*"
}
]
```

Untuk informasi selengkapnya tentang toko-toko utama, lihat [Toko utama](#).

Memperbarui konfigurasi enkripsi

Kemampuan untuk memperbarui konfigurasi enkripsi Anda dengan mulus sangat penting untuk mengelola implementasi enkripsi data Anda untuk integrasi terkelola. Ketika Anda awalnya onboard dengan integrasi terkelola, Anda akan diminta untuk memilih konfigurasi enkripsi Anda. Opsi Anda akan berupa kunci yang AWS dimiliki default atau membuat AWS KMS kunci Anda sendiri.

Konsol Manajemen AWS

Untuk memperbarui konfigurasi enkripsi Anda di Konsol Manajemen AWS, buka beranda AWS IoT layanan dan kemudian navigasikan ke Integrasi Terkelola untuk Kontrol Terpadu > Pengaturan > Enkripsi. Di jendela Pengaturan enkripsi, Anda dapat memperbarui konfigurasi enkripsi Anda dengan memilih AWS KMS kunci baru untuk perlindungan enkripsi tambahan. Pilih Sesuaikan pengaturan enkripsi (lanjutan) untuk memilih AWS KMS kunci yang ada atau Anda dapat memilih Buat AWS KMS kunci untuk membuat kunci terkelola pelanggan Anda sendiri.

Perintah API

Ada dua yang APIs digunakan untuk mengelola konfigurasi AWS KMS kunci enkripsi Anda dalam integrasi terkelola: `PutDefaultEncryptionConfiguration` dan `GetDefaultEncryptionConfiguration`.

Untuk memperbarui konfigurasi enkripsi default, hubungi `PutDefaultEncryptionConfiguration`. Untuk informasi selengkapnya tentang `PutDefaultEncryptionConfiguration`, lihat [PutDefaultEncryptionConfiguration](#).

Untuk melihat konfigurasi enkripsi default, panggil `GetDefaultEncryptionConfiguration`. Untuk informasi selengkapnya tentang `GetDefaultEncryptionConfiguration`, lihat [GetDefaultEncryptionConfiguration](#).

Manajemen identitas dan akses untuk integrasi terkelola

AWS Identity and Access Management (IAM) adalah Layanan AWS yang membantu administrator mengontrol akses ke AWS sumber daya dengan aman. Administrator IAM mengontrol siapa yang dapat diautentikasi (masuk) dan diberi wewenang (memiliki izin) untuk menggunakan sumber daya integrasi terkelola. IAM adalah Layanan AWS yang dapat Anda gunakan tanpa biaya tambahan.

Topik

- [Audiens](#)
- [Mengautentikasi dengan identitas](#)
- [Mengelola akses menggunakan kebijakan](#)
- [AWS kebijakan terkelola untuk integrasi terkelola](#)
- [Bagaimana integrasi terkelola bekerja dengan IAM](#)
- [Contoh kebijakan berbasis identitas untuk integrasi terkelola](#)
- [Pemecahan masalah identitas dan akses integrasi terkelola](#)
- [Menggunakan peran terkait layanan untuk integrasi terkelola](#)

Audiens

Cara Anda menggunakan AWS Identity and Access Management (IAM) berbeda berdasarkan peran Anda:

- Pengguna layanan - minta izin dari administrator Anda jika Anda tidak dapat mengakses fitur (lihat [Pemecahan masalah identitas dan akses integrasi terkelola](#))
- Administrator layanan - tentukan akses pengguna dan mengirimkan permintaan izin (lihat [Bagaimana integrasi terkelola bekerja dengan IAM](#))
- Administrator IAM - tulis kebijakan untuk mengelola akses (lihat [Contoh kebijakan berbasis identitas untuk integrasi terkelola](#))

Mengautentikasi dengan identitas

Otentikasi adalah cara Anda masuk AWS menggunakan kredensi identitas Anda. Anda harus diautentikasi sebagai Pengguna root akun AWS, pengguna IAM, atau dengan mengasumsikan peran IAM.

Anda dapat masuk sebagai identitas federasi menggunakan kredensial dari sumber identitas seperti AWS IAM Identity Center (Pusat Identitas IAM), autentikasi masuk tunggal, atau kredensial Google/Facebook Untuk informasi selengkapnya tentang cara masuk, lihat [Cara masuk ke Akun AWS Anda](#) dalam Panduan Pengguna AWS Sign-In .

Untuk akses terprogram, AWS sediakan SDK dan CLI untuk menandatangani permintaan secara kriptografis. Untuk informasi selengkapnya, lihat [AWS Signature Version 4 untuk permintaan API](#) dalam Panduan Pengguna IAM.

Akun AWS pengguna root

Saat Anda membuat Akun AWS, Anda mulai dengan satu identitas masuk yang disebut pengguna Akun AWS root yang memiliki akses lengkap ke semua Layanan AWS dan sumber daya. Kami sangat menyarankan agar Anda tidak menggunakan pengguna root untuk tugas sehari-hari. Untuk tugas yang memerlukan kredensial pengguna root, lihat [Tugas yang memerlukan kredensial pengguna root](#) dalam Panduan Pengguna IAM.

Identitas terfederasi

Sebagai praktik terbaik, mewajibkan pengguna manusia untuk menggunakan federasi dengan penyedia identitas untuk mengakses Layanan AWS menggunakan kredensi sementara.

Identitas federasi adalah pengguna dari direktori perusahaan Anda, penyedia identitas web, atau Directory Service yang mengakses Layanan AWS menggunakan kredensi dari sumber identitas. Identitas terfederasi mengambil peran yang memberikan kredensial sementara.

Untuk manajemen akses terpusat, kami menyarankan AWS IAM Identity Center. Untuk informasi selengkapnya, lihat [Apa itu Pusat Identitas IAM?](#) dalam Panduan Pengguna AWS IAM Identity Center

Pengguna dan grup IAM

[Pengguna IAM](#) adalah identitas dengan izin khusus untuk satu orang atau aplikasi. Sebaiknya gunakan kredensial sementara alih-alih pengguna IAM dengan kredensial jangka panjang. Untuk informasi selengkapnya, lihat [Mewajibkan pengguna manusia untuk menggunakan federasi dengan penyedia identitas untuk mengakses AWS menggunakan kredensial sementara](#) di Panduan Pengguna IAM.

[Grup IAM](#) menentukan kumpulan pengguna IAM dan mempermudah pengelolaan izin untuk pengguna dalam jumlah besar. Untuk mempelajari selengkapnya, lihat [Kasus penggunaan untuk pengguna IAM](#) dalam Panduan Pengguna IAM.

Peran IAM

[Peran IAM](#) adalah identitas dengan izin khusus yang menyediakan kredensial sementara. Anda dapat mengambil peran dengan [beralih dari pengguna ke peran IAM \(konsol\)](#) atau dengan memanggil operasi AWS CLI atau AWS API. Untuk informasi selengkapnya, lihat [Metode untuk mengambil peran](#) dalam Panduan Pengguna IAM.

Peran IAM berguna untuk akses pengguna gabungan, izin pengguna IAM sementara, akses lintas akun, akses lintas layanan, dan aplikasi yang berjalan di Amazon. EC2 Untuk informasi selengkapnya, lihat [Akses sumber daya lintas akun di IAM](#) dalam Panduan Pengguna IAM.

Mengelola akses menggunakan kebijakan

Anda mengontrol akses AWS dengan membuat kebijakan dan melampirkannya ke AWS identitas atau sumber daya. Kebijakan menentukan izin saat dikaitkan dengan identitas atau sumber daya. AWS mengevaluasi kebijakan ini ketika kepala sekolah membuat permintaan. Sebagian besar kebijakan disimpan AWS sebagai dokumen JSON. Untuk informasi selengkapnya tentang dokumen kebijakan JSON, lihat [Gambaran umum kebijakan JSON](#) dalam Panduan Pengguna IAM.

Menggunakan kebijakan, administrator menentukan siapa yang memiliki akses ke apa dengan mendefinisikan principal mana yang dapat melakukan tindakan pada sumber daya apa, dan dalam kondisi apa.

Secara default, pengguna dan peran tidak memiliki izin. Administrator IAM membuat kebijakan IAM dan menambahkannya ke peran, yang kemudian dapat diambil oleh pengguna. Kebijakan IAM mendefinisikan izin terlepas dari metode yang Anda gunakan untuk melakukannya.

Kebijakan berbasis identitas

Kebijakan berbasis identitas adalah dokumen kebijakan izin JSON yang Anda lampirkan ke identitas (pengguna, grup, atau peran). Kebijakan ini mengontrol tindakan apa yang bisa dilakukan oleh identitas tersebut, terhadap sumber daya yang mana, dan dalam kondisi apa. Untuk mempelajari cara membuat kebijakan berbasis identitas, lihat [Tentukan izin IAM kustom dengan kebijakan yang dikelola pelanggan](#) dalam Panduan Pengguna IAM.

Kebijakan berbasis identitas dapat berupa kebijakan inline (disematkan langsung ke dalam satu identitas) atau kebijakan terkelola (kebijakan mandiri yang dilampirkan pada banyak identitas). Untuk mempelajari cara memilih antara kebijakan terkelola dan kebijakan inline, lihat [Pilih 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. Contohnya termasuk 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. Anda harus [menentukan principal](#) dalam kebijakan berbasis sumber daya.

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

Jenis-jenis kebijakan lain

AWS mendukung jenis kebijakan tambahan yang dapat menetapkan izin maksimum yang diberikan oleh jenis kebijakan yang lebih umum:

- Batasan izin – Menetapkan izin maksimum yang dapat diberikan oleh kebijakan berbasis identitas kepada entitas IAM. Untuk informasi selengkapnya, lihat [Batasan izin untuk entitas IAM](#) dalam Panduan Pengguna IAM.
- Kebijakan kontrol layanan (SCPs) — Tentukan izin maksimum untuk organisasi atau unit organisasi di AWS Organizations. Untuk informasi selengkapnya, lihat [Kebijakan kontrol layanan](#) dalam Panduan Pengguna AWS Organizations .

- Kebijakan kontrol sumber daya (RCPs) — Tetapkan izin maksimum yang tersedia untuk sumber daya di akun Anda. Untuk informasi selengkapnya, lihat [Kebijakan kontrol sumber daya \(RCPs\)](#) di Panduan AWS Organizations Pengguna.
- Kebijakan sesi – Kebijakan lanjutan yang diteruskan sebagai parameter saat membuat sesi sementara untuk peran atau pengguna terfederasi. Untuk informasi selengkapnya, lihat [Kebijakan sesi](#) dalam Panduan Pengguna IAM.

Berbagai jenis kebijakan

Ketika beberapa jenis kebijakan berlaku pada suatu permintaan, izin yang dihasilkan lebih rumit untuk dipahami. Untuk mempelajari cara AWS menentukan apakah akan mengizinkan permintaan saat beberapa jenis kebijakan terlibat, lihat [Logika evaluasi kebijakan](#) di Panduan Pengguna IAM.

AWS kebijakan terkelola untuk integrasi terkelola

Untuk menambahkan izin ke pengguna, grup, dan peran, lebih mudah menggunakan kebijakan AWS terkelola daripada menulis kebijakan sendiri. Dibutuhkan waktu dan keahlian untuk [membuat kebijakan yang dikelola pelanggan IAM](#) yang hanya memberi tim Anda izin yang mereka butuhkan. Untuk memulai dengan cepat, Anda dapat menggunakan kebijakan AWS terkelola kami. Kebijakan ini mencakup kasus penggunaan umum dan tersedia di Akun AWS Anda. Untuk informasi selengkapnya tentang kebijakan AWS [AWS terkelola](#), lihat [kebijakan terkelola](#) di Panduan Pengguna IAM.

AWS layanan memelihara dan memperbarui kebijakan AWS terkelola. Anda tidak dapat mengubah izin dalam kebijakan AWS terkelola. Layanan terkadang menambahkan izin tambahan ke kebijakan yang dikelola AWS untuk mendukung fitur-fitur baru. Jenis pembaruan ini akan memengaruhi semua identitas (pengguna, grup, dan peran) di mana kebijakan tersebut dilampirkan. Layanan kemungkinan besar akan memperbarui kebijakan yang dikelola AWS saat ada fitur baru yang diluncurkan atau saat ada operasi baru yang tersedia. Layanan tidak menghapus izin dari kebijakan AWS terkelola, sehingga pembaruan kebijakan tidak akan merusak izin yang ada.

Selain itu, AWS mendukung kebijakan terkelola untuk fungsi pekerjaan yang mencakup beberapa layanan. Misalnya, kebijakan ReadOnlyAccess AWS terkelola menyediakan akses hanya-baca ke semua AWS layanan dan sumber daya. Saat layanan meluncurkan fitur baru, AWS tambahkan izin hanya-baca untuk operasi dan sumber daya baru. Untuk melihat daftar dan deskripsi dari kebijakan fungsi tugas, lihat [kebijakan yang dikelola AWS untuk fungsi tugas](#) di Panduan Pengguna IAM.

AWS kebijakan terkelola: AWSIoTManagedIntegrationsFullAccess

Anda dapat melampirkan kebijakan `AWSIoTManagedIntegrationsFullAccess` ke identitas IAM Anda.

Kebijakan ini memberikan izin akses penuh untuk integrasi terkelola dan layanan terkait. Untuk melihat kebijakan ini di Konsol Manajemen AWS, lihat [AWSIoTManagedIntegrationsFullAccess](#).

Detail izin

Kebijakan ini mencakup izin berikut:

- `iotmanagedintegrations`— Menyediakan akses penuh ke integrasi terkelola dan layanan terkait untuk pengguna, grup, dan peran IAM yang Anda tambahkan kebijakan ini.
- `iam`— Memungkinkan pengguna, grup, dan peran IAM yang ditugaskan untuk membuat peran terkait layanan dalam file. Akun AWS

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iotmanagedintegrations:*",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "iam:CreateServiceLinkedRole",
      "Resource": "arn:aws:iam::*:role/aws-service-role/iotmanagedintegrations.amazonaws.com/AWSServiceRoleForIoTManagedIntegrations",
      "Condition": {
        "StringEquals": {
          "iam:AWSServiceName": "iotmanagedintegrations.amazonaws.com"
        }
      }
    }
  ]
}
```

AWS kebijakan terkelola: AWS IoTManagedIntegrationsRolePolicy

Anda dapat melampirkan kebijakan AWS IoTManagedIntegrationsRolePolicy ke identitas IAM Anda.

Kebijakan ini memberikan izin integrasi terkelola untuk mempublikasikan CloudWatch log dan metrik Amazon atas nama Anda.

Untuk melihat kebijakan ini di Konsol Manajemen AWS, lihat [AWSIoTManagedIntegrationsRolePolicy](#).

Detail izin

Kebijakan ini mencakup izin berikut.

- **logs**— Menyediakan kemampuan untuk membuat grup CloudWatch log Amazon dan streaming log ke grup.
- **cloudwatch**— Menyediakan kemampuan untuk mempublikasikan CloudWatch metrik Amazon. Untuk informasi selengkapnya tentang CloudWatch metrik Amazon, lihat [Metrik di Amazon CloudWatch](#)

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CloudWatchLogs",
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup"
      ],
      "Resource": [
        "arn:aws:logs:*:*:log-group:/aws/iotmanagedintegrations/*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:PrincipalAccount": "${aws:ResourceAccount}"
        }
      }
    }
  ]
}
```

```
    },
    {
      "Sid": "CloudWatchStreams",
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": [
        "arn:aws:logs:*:*:log-group:/aws/iotmanagedintegrations/*:log-stream:*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:PrincipalAccount": "${aws:ResourceAccount}"
        }
      }
    },
    {
      "Sid": "CloudWatchMetrics",
      "Effect": "Allow",
      "Action": [
        "cloudwatch:PutMetricData"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "cloudwatch:namespace": [
            "AWS/IoTManagedIntegrations",
            "AWS/Usage"
          ]
        }
      }
    }
  ]
}
```

Integrasi terkelola memperbarui kebijakan AWS terkelola

Lihat detail tentang pembaruan kebijakan AWS terkelola untuk integrasi terkelola sejak layanan ini mulai melacak perubahan ini. Untuk peringatan otomatis tentang perubahan pada halaman ini, berlangganan umpan RSS di halaman riwayat dokumen integrasi terkelola.

Ubah	Deskripsi	Date
Integrasi terkelola mulai melacak perubahan	Integrasi terkelola mulai melacak perubahan untuk kebijakan yang AWS dikelola.	Maret 03, 2025

Bagaimana integrasi terkelola bekerja dengan IAM

Sebelum Anda menggunakan IAM untuk mengelola akses ke integrasi terkelola, pelajari fitur IAM apa yang tersedia untuk digunakan dengan integrasi terkelola.

Fitur IAM yang dapat Anda gunakan dengan integrasi terkelola

Fitur IAM	Dukungan integrasi terkelola
Kebijakan berbasis identitas	Ya
Kebijakan berbasis sumber daya	Tidak
Tindakan kebijakan	Ya
Sumber daya kebijakan	Ya
Kunci kondisi kebijakan	Ya
ACLs	Tidak
ABAC (tanda dalam kebijakan)	Tidak
Kredensial sementara	Ya
Izin principal	Ya
Peran layanan	Ya
Peran terkait layanan	Ya

Untuk mendapatkan tampilan tingkat tinggi tentang cara kerja integrasi terkelola dan AWS layanan lainnya dengan sebagian besar fitur IAM, lihat [AWS layanan yang bekerja dengan IAM di Panduan Pengguna IAM](#).

Kebijakan berbasis identitas untuk integrasi terkelola

Mendukung kebijakan berbasis identitas: Ya

Kebijakan berbasis identitas adalah dokumen kebijakan izin JSON yang dapat Anda lampirkan ke sebuah identitas, seperti pengguna IAM, grup pengguna IAM, atau peran IAM. Kebijakan ini mengontrol jenis tindakan yang dapat dilakukan oleh pengguna dan peran, di sumber daya mana, dan berdasarkan kondisi seperti apa. Untuk mempelajari cara membuat kebijakan berbasis identitas, lihat [Tentukan izin IAM kustom dengan kebijakan terkelola pelanggan](#) dalam Panduan Pengguna IAM.

Dengan kebijakan berbasis identitas IAM, Anda dapat menentukan secara spesifik apakah tindakan dan sumber daya diizinkan atau ditolak, serta kondisi yang menjadi dasar dikabulkannya atau ditolakannya tindakan tersebut. Untuk mempelajari semua elemen yang dapat Anda gunakan dalam kebijakan JSON, lihat [Referensi elemen kebijakan JSON IAM](#) dalam Panduan Pengguna IAM.

Contoh kebijakan berbasis identitas untuk integrasi terkelola

Untuk melihat contoh kebijakan berbasis identitas integrasi terkelola, lihat. [Contoh kebijakan berbasis identitas untuk integrasi terkelola](#)

Kebijakan berbasis sumber daya dalam integrasi terkelola

Mendukung kebijakan berbasis sumber daya: Tidak

Kebijakan berbasis sumber daya adalah dokumen kebijakan JSON yang Anda lampirkan ke sumber daya. Contoh kebijakan berbasis sumber daya adalah kebijakan kepercayaan peran IAM dan kebijakan bucket Amazon S3. Dalam layanan yang mendukung kebijakan berbasis sumber daya, administrator layanan dapat menggunakannya untuk mengontrol akses ke sumber daya tertentu. Untuk sumber daya tempat kebijakan dilampirkan, kebijakan menentukan tindakan apa yang dapat dilakukan oleh principal tertentu pada sumber daya tersebut dan dalam kondisi apa. Anda harus [menentukan principal](#) dalam kebijakan berbasis sumber daya. Prinsipal dapat mencakup akun, pengguna, peran, pengguna federasi, atau. Layanan AWS

Untuk mengaktifkan akses lintas akun, Anda dapat menentukan secara spesifik seluruh akun atau entitas IAM di akun lain sebagai principal dalam kebijakan berbasis sumber daya. Untuk informasi selengkapnya, lihat [Akses sumber daya lintas akun di IAM](#) dalam Panduan Pengguna IAM.

Tindakan kebijakan untuk integrasi terkelola

Mendukung tindakan kebijakan: Ya

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

Elemen `Action` dari kebijakan JSON menjelaskan tindakan yang dapat Anda gunakan untuk mengizinkan atau menolak akses dalam sebuah kebijakan. Sertakan tindakan dalam kebijakan untuk memberikan izin untuk melakukan operasi terkait.

Untuk melihat daftar tindakan integrasi terkelola, lihat [Tindakan yang Ditentukan oleh Integrasi Terkelola](#) dalam Referensi Otorisasi Layanan.

Tindakan kebijakan dalam integrasi terkelola menggunakan awalan berikut sebelum tindakan:

```
iot-mi
```

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

```
"Action": [  
  "iot-mi:action1",  
  "iot-mi:action2"  
]
```

Untuk melihat contoh kebijakan berbasis identitas integrasi terkelola, lihat [Contoh kebijakan berbasis identitas untuk integrasi terkelola](#)

Sumber daya kebijakan untuk integrasi terkelola

Mendukung sumber daya kebijakan: Ya

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

Elemen kebijakan JSON `Resource` menentukan objek yang menjadi target penerapan tindakan. Praktik terbaiknya, tentukan sumber daya menggunakan [Amazon Resource Name \(ARN\)](#). Untuk tindakan yang tidak mendukung izin di tingkat sumber daya, gunakan wildcard (*) untuk menunjukkan bahwa pernyataan tersebut berlaku untuk semua sumber daya.

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

Untuk melihat daftar jenis sumber daya integrasi terkelola dan jenis sumber daya ARNs, lihat Sumber [Daya yang Ditentukan oleh Integrasi Terkelola](#) dalam Referensi Otorisasi Layanan. Untuk mempelajari tindakan mana yang dapat Anda tentukan ARN dari setiap sumber daya, lihat [Tindakan yang Ditentukan oleh Integrasi Terkelola](#).

Untuk melihat contoh kebijakan berbasis identitas integrasi terkelola, lihat. [Contoh kebijakan berbasis identitas untuk integrasi terkelola](#)

Kunci kondisi kebijakan untuk integrasi terkelola

Mendukung kunci kondisi kebijakan khusus layanan: Yes

Administrator dapat menggunakan kebijakan AWS JSON untuk menentukan siapa yang memiliki akses ke apa. Yaitu, principal dapat melakukan tindakan pada suatu sumber daya, dan dalam suatu syarat.

Elemen `Condition` menentukan ketika pernyataan dieksekusi berdasarkan kriteria yang ditetapkan. Anda dapat membuat ekspresi bersyarat yang menggunakan [operator kondisi](#), misalnya sama dengan atau kurang dari, untuk mencocokkan kondisi dalam kebijakan dengan nilai-nilai yang diminta. Untuk melihat semua kunci kondisi AWS global, lihat [kunci konteks kondisi AWS global](#) di Panduan Pengguna IAM.

Untuk melihat daftar kunci kondisi integrasi terkelola, lihat Kunci Kondisi [untuk integrasi Terkelola dalam Referensi](#) Otorisasi Layanan. Untuk mempelajari tindakan dan sumber daya yang dapat Anda gunakan kunci kondisi, lihat [Tindakan yang Ditentukan oleh Integrasi Terkelola](#).

Untuk melihat contoh kebijakan berbasis identitas integrasi terkelola, lihat. [Contoh kebijakan berbasis identitas untuk integrasi terkelola](#)

ACLs dalam integrasi terkelola

Mendukung ACLs: Tidak

Access control lists (ACLs) mengontrol prinsipal mana (anggota akun, pengguna, atau peran) yang memiliki izin untuk mengakses sumber daya. ACLs mirip dengan kebijakan berbasis sumber daya, meskipun mereka tidak menggunakan format dokumen kebijakan JSON.

ABAC dengan integrasi terkelola

Mendukung ABAC (tag dalam kebijakan): Sebagian

Kontrol akses berbasis atribut (ABAC) adalah strategi otorisasi yang menentukan izin berdasarkan atribut tanda. Anda dapat melampirkan tag ke entitas dan AWS sumber daya IAM, lalu merancang kebijakan ABAC untuk mengizinkan operasi saat tag prinsipal cocok dengan tag pada sumber daya.

Untuk mengendalikan akses berdasarkan tanda, berikan informasi tentang tanda di [elemen kondisi](#) dari kebijakan menggunakan kunci kondisi `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, atau `aws:TagKeys`.

Jika sebuah layanan mendukung ketiga kunci kondisi untuk setiap jenis sumber daya, nilainya adalah Ya untuk layanan tersebut. Jika suatu layanan mendukung ketiga kunci kondisi untuk hanya beberapa jenis sumber daya, nilainya adalah Parsial.

Untuk informasi selengkapnya tentang ABAC, lihat [Tentukan izin dengan otorisasi ABAC](#) dalam Panduan Pengguna IAM. Untuk melihat tutorial yang menguraikan langkah-langkah pengaturan ABAC, lihat [Menggunakan kontrol akses berbasis atribut \(ABAC\)](#) dalam Panduan Pengguna IAM.

Menggunakan kredensi sementara dengan integrasi terkelola

Mendukung kredensial sementara: Ya

Kredensi sementara menyediakan akses jangka pendek ke AWS sumber daya dan secara otomatis dibuat saat Anda menggunakan federasi atau beralih peran. AWS merekomendasikan agar Anda secara dinamis menghasilkan kredensi sementara alih-alih menggunakan kunci akses jangka panjang. Untuk informasi selengkapnya, lihat [Kredensial keamanan sementara di IAM](#) dan [Layanan AWS yang berfungsi dengan IAM](#) dalam Panduan Pengguna IAM.

Izin utama lintas layanan untuk integrasi terkelola

Mendukung sesi akses terusan (FAS): Ya

Sesi akses teruskan (FAS) menggunakan izin dari pemanggilan utama Layanan AWS, dikombinasikan dengan permintaan Layanan AWS untuk membuat permintaan ke layanan hilir. Untuk detail kebijakan ketika mengajukan permintaan FAS, lihat [Sesi akses terusan](#).

Peran layanan untuk integrasi terkelola

Mendukung peran layanan: Ya

Peran layanan adalah [peran IAM](#) yang diambil oleh sebuah layanan untuk melakukan tindakan atas nama Anda. Administrator IAM dapat membuat, mengubah, dan menghapus peran layanan dari dalam IAM. Untuk informasi selengkapnya, lihat [Buat sebuah peran untuk mendelegasikan izin ke Layanan AWS](#) dalam Panduan pengguna IAM.

Warning

Mengubah izin untuk peran layanan dapat merusak fungsionalitas integrasi terkelola. Edit peran layanan hanya ketika integrasi terkelola memberikan panduan untuk melakukannya.

Peran terkait layanan untuk integrasi terkelola

Mendukung peran terkait layanan: Ya

Peran terkait layanan adalah jenis peran layanan yang ditautkan ke. Layanan AWS Layanan tersebut dapat menjalankan peran untuk melakukan tindakan atas nama Anda. Peran terkait layanan muncul di Anda Akun AWS dan dimiliki oleh layanan. Administrator IAM dapat melihat, tetapi tidak dapat mengedit izin untuk peran terkait layanan.

Untuk detail tentang pembuatan atau manajemen peran terkait layanan, lihat [Layanan AWS yang berfungsi dengan IAM](#). Cari layanan dalam tabel yang memiliki Yes di kolom Peran terkait layanan. Pilih tautan Ya untuk melihat dokumentasi peran terkait layanan untuk layanan tersebut.

Contoh kebijakan berbasis identitas untuk integrasi terkelola

Secara default, pengguna dan peran tidak memiliki izin untuk membuat atau memodifikasi sumber daya integrasi terkelola. Untuk memberikan izin kepada pengguna untuk melakukan tindakan di sumber daya yang mereka perlukan, administrator IAM dapat membuat kebijakan IAM.

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

Untuk detail tentang tindakan dan jenis sumber daya yang ditentukan oleh integrasi terkelola, termasuk format ARNs untuk setiap jenis sumber daya, lihat [Tindakan, Sumber Daya, dan Kunci Kondisi untuk integrasi Terkelola](#) dalam Referensi Otorisasi Layanan.

Topik

- [Praktik terbaik kebijakan](#)
- [Menggunakan konsol integrasi terkelola](#)
- [Mengizinkan pengguna melihat izin mereka sendiri](#)

Praktik terbaik kebijakan

Kebijakan berbasis identitas menentukan apakah seseorang dapat membuat, mengakses, atau menghapus sumber daya integrasi terkelola di akun Anda. Tindakan ini membuat Akun AWS Anda dikenai biaya. Ketika Anda membuat atau mengedit kebijakan berbasis identitas, ikuti panduan dan rekomendasi ini:

- Mulailah dengan kebijakan AWS terkelola dan beralih ke izin hak istimewa paling sedikit — Untuk mulai memberikan izin kepada pengguna dan beban kerja Anda, gunakan kebijakan AWS terkelola yang memberikan izin untuk banyak kasus penggunaan umum. Mereka tersedia di Akun AWS. Kami menyarankan Anda mengurangi izin lebih lanjut dengan menentukan kebijakan yang dikelola AWS pelanggan yang khusus untuk kasus penggunaan Anda. Untuk informasi selengkapnya, lihat [Kebijakan yang dikelola AWS](#) atau [Kebijakan yang dikelola AWS untuk fungsi tugas](#) dalam Panduan Pengguna IAM.
- Menerapkan izin dengan hak akses paling rendah – Ketika Anda menetapkan izin dengan kebijakan IAM, hanya berikan izin yang diperlukan untuk melakukan tugas. Anda melakukannya dengan mendefinisikan tindakan yang dapat diambil pada sumber daya tertentu dalam kondisi tertentu, yang juga dikenal sebagai izin dengan hak akses paling rendah. Untuk informasi selengkapnya tentang cara menggunakan IAM untuk mengajukan izin, lihat [Kebijakan dan izin dalam IAM](#) dalam Panduan Pengguna IAM.
- Gunakan kondisi dalam kebijakan IAM untuk membatasi akses lebih lanjut – Anda dapat menambahkan suatu kondisi ke kebijakan Anda untuk membatasi akses ke tindakan dan sumber daya. Sebagai contoh, Anda dapat menulis kondisi kebijakan untuk menentukan bahwa semua permintaan harus dikirim menggunakan SSL. Anda juga dapat menggunakan ketentuan untuk memberikan akses ke tindakan layanan jika digunakan melalui yang spesifik Layanan AWS, seperti CloudFormation. Untuk informasi selengkapnya, lihat [Elemen kebijakan JSON IAM: Kondisi](#) dalam Panduan Pengguna IAM.

- Gunakan IAM Access Analyzer untuk memvalidasi kebijakan IAM Anda untuk memastikan izin yang aman dan fungsional – IAM Access Analyzer memvalidasi kebijakan baru dan yang sudah ada sehingga kebijakan tersebut mematuhi bahasa kebijakan IAM (JSON) dan praktik terbaik IAM. IAM Access Analyzer menyediakan lebih dari 100 pemeriksaan kebijakan dan rekomendasi yang dapat ditindaklanjuti untuk membantu Anda membuat kebijakan yang aman dan fungsional. Untuk informasi selengkapnya, lihat [Validasi kebijakan dengan IAM Access Analyzer](#) dalam Panduan Pengguna IAM.
- Memerlukan otentikasi multi-faktor (MFA) - Jika Anda memiliki skenario yang mengharuskan pengguna IAM atau pengguna root di Anda, Akun AWS aktifkan MFA untuk keamanan tambahan. Untuk meminta MFA ketika operasi API dipanggil, tambahkan kondisi MFA pada kebijakan Anda. Untuk informasi selengkapnya, lihat [Amankan akses API dengan MFA](#) dalam Panduan Pengguna IAM.

Untuk informasi selengkapnya tentang praktik terbaik dalam IAM, lihat [Praktik terbaik keamanan di IAM](#) dalam Panduan Pengguna IAM.

Menggunakan konsol integrasi terkelola

Untuk mengakses konsol Integrasi terkelola, Anda harus memiliki set izin minimum. Izin ini harus memungkinkan Anda untuk membuat daftar dan melihat detail tentang sumber daya integrasi terkelola di Anda. Akun AWS Jika Anda membuat kebijakan berbasis identitas yang lebih ketat daripada izin minimum yang diperlukan, konsol tidak akan berfungsi sebagaimana mestinya untuk entitas (pengguna atau peran) dengan kebijakan tersebut.

Anda tidak perlu mengizinkan izin konsol minimum untuk pengguna yang melakukan panggilan hanya ke AWS CLI atau AWS API. Sebagai gantinya, izinkan akses hanya ke tindakan yang sesuai dengan operasi API yang coba mereka lakukan.

Untuk memastikan bahwa pengguna dan peran masih dapat menggunakan konsol integrasi terkelola, lampirkan juga integrasi terkelola *ConsoleAccess* atau kebijakan *ReadOnly* AWS terkelola ke entitas. Untuk informasi selengkapnya, lihat [Menambah izin untuk pengguna](#) dalam Panduan Pengguna IAM.

Mengizinkan pengguna melihat izin mereka sendiri

Contoh ini menunjukkan cara membuat kebijakan yang mengizinkan pengguna IAM melihat kebijakan inline dan terkelola yang dilampirkan ke identitas pengguna mereka. Kebijakan ini

mencakup izin untuk menyelesaikan tindakan ini di konsol atau menggunakan API atau secara terprogram. AWS CLI AWS

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ViewOwnUserInfo",
      "Effect": "Allow",
      "Action": [
        "iam:GetUserPolicy",
        "iam:ListGroupsWithUser",
        "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": "*"
    }
  ]
}
```

Pemecahan masalah identitas dan akses integrasi terkelola

Gunakan informasi berikut untuk membantu Anda mendiagnosis dan memperbaiki masalah umum yang mungkin Anda temui saat bekerja dengan integrasi terkelola dan IAM.

Topik

- [Saya tidak berwenang untuk melakukan tindakan dalam integrasi terkelola](#)
- [Saya tidak berwenang untuk melakukan iam: PassRole](#)
- [Saya ingin mengizinkan orang di luar saya Akun AWS mengakses sumber daya integrasi terkelola saya](#)

Saya tidak berwenang untuk melakukan tindakan dalam integrasi terkelola

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 `iot-mi:GetWidget` rekaan.

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

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

Saya tidak berwenang untuk melakukan iam: PassRole

Jika Anda menerima kesalahan bahwa Anda tidak diizinkan untuk melakukan `iam:PassRole` tindakan, kebijakan Anda harus diperbarui agar Anda dapat meneruskan peran ke integrasi terkelola.

Beberapa Layanan AWS memungkinkan Anda untuk meneruskan peran yang ada ke layanan tersebut alih-alih membuat peran layanan baru atau peran terkait layanan. Untuk melakukannya, Anda harus memiliki izin untuk meneruskan peran ke layanan.

Contoh kesalahan berikut terjadi ketika pengguna IAM bernama `marymajor` mencoba menggunakan konsol untuk melakukan tindakan dalam integrasi terkelola. Namun, tindakan tersebut memerlukan layanan untuk mendapatkan izin yang diberikan oleh peran layanan. Mary tidak memiliki izin untuk meneruskan peran tersebut pada layanan.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
iam:PassRole
```

Dalam kasus ini, kebijakan Mary harus diperbarui agar dia mendapatkan izin untuk melakukan tindakan `iam:PassRole` tersebut.

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

Saya ingin mengizinkan orang di luar saya Akun AWS mengakses sumber daya integrasi terkelola saya

Anda dapat membuat peran yang dapat digunakan pengguna di akun lain atau orang-orang di luar organisasi Anda untuk mengakses sumber daya Anda. Anda dapat menentukan siapa saja yang dipercaya untuk mengambil peran tersebut. Untuk layanan yang mendukung kebijakan berbasis sumber daya atau daftar kontrol akses (ACLs), Anda dapat menggunakan kebijakan tersebut untuk memberi orang akses ke sumber daya Anda.

Untuk mempelajari selengkapnya, periksa referensi berikut:

- Untuk mempelajari apakah integrasi terkelola mendukung fitur-fitur ini, lihat [Bagaimana integrasi terkelola bekerja dengan IAM](#).
- Untuk mempelajari cara menyediakan akses ke sumber daya Anda di seluruh sumber daya Akun AWS yang Anda miliki, lihat [Menyediakan akses ke pengguna IAM di pengguna lain Akun AWS yang Anda miliki](#) di Panduan Pengguna IAM.
- Untuk mempelajari cara menyediakan akses ke sumber daya Anda kepada pihak ketiga Akun AWS, lihat [Menyediakan akses yang Akun AWS dimiliki oleh pihak ketiga](#) dalam Panduan Pengguna IAM.
- Untuk mempelajari cara memberikan akses melalui federasi identitas, lihat [Menyediakan akses ke pengguna terautentikasi eksternal \(federasi identitas\)](#) dalam Panduan Pengguna IAM.
- Untuk mempelajari perbedaan antara menggunakan peran dan kebijakan berbasis sumber daya untuk akses lintas akun, lihat [Akses sumber daya lintas akun di IAM di Panduan Pengguna IAM](#).

Menggunakan peran terkait layanan untuk integrasi terkelola

Integrasi terkelola untuk Manajemen AWS IoT Perangkat menggunakan peran AWS Identity and Access Management terkait [layanan](#) (IAM). Peran terkait layanan adalah jenis unik peran IAM yang

ditautkan langsung ke integrasi terkelola. Peran terkait layanan telah ditentukan sebelumnya oleh integrasi terkelola dan mencakup semua izin yang diperlukan layanan untuk memanggil layanan lain AWS atas nama Anda.

Peran terkait layanan membuat pengaturan integrasi terkelola lebih mudah karena Anda tidak perlu menambahkan izin yang diperlukan secara manual. Integrasi terkelola untuk Manajemen AWS IoT Perangkat menentukan izin peran terkait layanannya, dan kecuali ditentukan lain, hanya integrasi terkelola yang dapat mengambil perannya. Izin yang ditentukan mencakup kebijakan kepercayaan dan kebijakan izin, dan kebijakan izin tersebut tidak dapat dilampirkan ke entitas IAM lainnya.

Anda dapat menghapus peran tertaut layanan hanya setelah menghapus sumber daya terkait terlebih dahulu. Ini melindungi sumber daya integrasi terkelola karena Anda tidak dapat secara tidak sengaja menghapus izin untuk mengakses sumber daya.

Untuk informasi tentang layanan lain yang mendukung peran terkait layanan, silakan lihat [layanan AWS yang bisa digunakan dengan IAM](#) dan carilah layanan yang memiliki opsi Ya di kolom Peran terkait layanan. Pilih Ya dengan sebuah tautan untuk melihat dokumentasi peran terkait layanan untuk layanan tersebut.

Izin peran terkait layanan untuk integrasi terkelola

Integrasi terkelola untuk Manajemen AWS IoT Perangkat menggunakan peran terkait layanan bernama `AWSServiceRoleForIoTManagedIntegrasi` — Menyediakan integrasi terkelola untuk izin Manajemen AWS IoT Perangkat untuk menerbitkan log dan metrik atas nama Anda.

Peran terkait layanan `AWSServiceRoleForIoTManagedIntegrasi` mempercayai layanan berikut untuk mengambil peran:

- `iotmanagedintegrations.amazonaws.com`

Kebijakan izin peran bernama `AWSIoTManagedIntegrationsServiceRolePolicy` memungkinkan integrasi terkelola untuk menyelesaikan tindakan berikut pada sumber daya yang ditentukan:

- Tindakan: `logs:CreateLogGroup`, `logs:DescribeLogGroups`, `logs:CreateLogStream`, `logs:PutLogEvents`, `logs:DescribeLogStreams`, `cloudwatch:PutMetricData` on all of your managed integrations resources.

JSON

```
{
  "Version": "2012-10-17",
  "Statement" : [
    {
      "Sid" : "CloudWatchLogs",
      "Effect" : "Allow",
      "Action" : [
        "logs:CreateLogGroup",
        "logs:DescribeLogGroups"
      ],
      "Resource" : [
        "arn:aws:logs:*:*:log-group:/aws/iotmanagedintegrations/*"
      ]
    },
    {
      "Sid" : "CloudWatchStreams",
      "Effect" : "Allow",
      "Action" : [
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams"
      ],
      "Resource" : [
        "arn:aws:logs:*:*:log-group:/aws/iotmanagedintegrations/*:log-stream:*"
      ]
    },
    {
      "Sid" : "CloudWatchMetrics",
      "Effect" : "Allow",
      "Action" : [
        "cloudwatch:PutMetricData"
      ],
      "Resource" : "*",
      "Condition" : {
        "StringEquals" : {
          "cloudwatch:namespace" : [
            "AWS/IoTManagedIntegrations",
            "AWS/Usage"
          ]
        }
      }
    }
  ]
}
```

```
}  
]  
}
```

Anda harus mengonfigurasi izin agar pengguna, grup, atau peran Anda membuat, mengedit, atau menghapus peran terkait layanan. Untuk informasi selengkapnya, lihat [Izin peran terkait layanan](#) dalam Panduan Pengguna IAM.

Membuat peran terkait layanan untuk integrasi terkelola

Anda tidak perlu membuat peran terkait layanan secara manual. Saat Anda menyebabkan jenis peristiwa seperti memanggil `PutRuntimeLogConfiguration`, `CreateEventLogConfiguration`, atau perintah `RegisterCustomEndpoint` API di Konsol Manajemen AWS, atau AWS API AWS CLI, integrasi terkelola akan membuat peran terkait layanan untuk Anda. Untuk informasi lebih lanjut tentang `PutRuntimeLogConfiguration`, `CreateEventLogConfiguration`, atau `RegisterCustomEndpoint`, lihat [PutRuntimeLogConfiguration](#), [CreateEventLogConfiguration](#), atau [RegisterCustomEndpoint](#).

Jika Anda menghapus peran terkait layanan ini, dan ingin membuatnya lagi, Anda dapat mengulangi proses yang sama untuk membuat kembali peran tersebut di akun Anda. Saat Anda menyebabkan jenis peristiwa seperti memanggil `PutRuntimeLogConfiguration`, atau perintah `RegisterCustomEndpoint` API `CreateEventLogConfiguration`, integrasi terkelola akan membuat peran terkait layanan untuk Anda lagi. Atau, Anda dapat menghubungi AWS Customer Support melalui AWS Support Center Console. Untuk informasi selengkapnya tentang AWS Support Plans, lihat [Membandingkan AWS Support Plans](#).

Anda juga dapat menggunakan konsol IAM untuk membuat peran terkait layanan dengan kasus penggunaan IoT ManagedIntegrations - Managed Role. Di AWS CLI atau AWS API, buat peran terkait layanan dengan nama `iotmanagedintegrations.amazonaws.com` layanan. Untuk informasi selengkapnya, lihat [Membuat peran tertaut layanan](#) dalam Panduan Pengguna IAM. Jika Anda menghapus peran tertaut layanan ini, Anda dapat mengulang proses yang sama untuk membuat peran tersebut lagi.

Mengedit peran terkait layanan untuk integrasi terkelola

Integrasi terkelola tidak memungkinkan Anda mengedit peran terkait layanan `AWSServiceRoleForIoTManagedIntegrations`. Setelah membuat peran terkait layanan, Anda tidak dapat mengubah nama peran karena berbagai entitas mungkin merujuk peran tersebut. Namun, Anda dapat mengedit

penjelasan peran menggunakan IAM. Untuk informasi selengkapnya, lihat [Mengedit peran terkait layanan](#) dalam Panduan Pengguna IAM.

Menghapus peran terkait layanan untuk integrasi terkelola

Jika Anda tidak perlu lagi menggunakan fitur atau layanan yang memerlukan peran terkait layanan, sebaiknya hapus peran tersebut. Dengan begitu, Anda tidak memiliki entitas yang tidak digunakan yang tidak dipantau atau dipelihara secara aktif. Tetapi, Anda harus membersihkan sumber daya peran yang terhubung dengan layanan sebelum menghapusnya secara manual.

Note

Jika integrasi terkelola menggunakan peran saat Anda mencoba menghapus sumber daya, maka penghapusan mungkin gagal. Jika hal itu terjadi, tunggu beberapa menit dan coba mengoperasikannya lagi.

Untuk menghapus peran tertaut layanan secara manual menggunakan IAM

Gunakan konsol IAM, the AWS CLI, atau AWS API untuk menghapus peran terkait layanan AWSService RoleForIoTManaged Integrasi. Untuk informasi selengkapnya, lihat [Menghapus peran terkait layanan](#) dalam Panduan Pengguna IAM.

Wilayah yang Didukung untuk integrasi terkelola peran terkait layanan

Integrasi terkelola untuk Manajemen AWS IoT Perangkat mendukung penggunaan peran terkait layanan di semua Wilayah tempat layanan tersedia. Untuk informasi selengkapnya, lihat [AWS Wilayah dan titik akhir](#).

Gunakan AWS Secrets Manager untuk perlindungan data untuk alur kerja C2C

AWS Secrets Manager adalah layanan penyimpanan rahasia yang dapat Anda gunakan untuk melindungi kredensi database, kunci API, dan informasi rahasia lainnya. Kemudian dalam kode Anda, Anda dapat mengganti kredensi hardcoded dengan panggilan API ke Secrets Manager. Ini membantu memastikan bahwa rahasia tidak dapat dikompromikan oleh seseorang yang memeriksa kode Anda, karena rahasianya tidak ada. Untuk ikhtisar, lihat [Panduan AWS Secrets Manager Pengguna](#).

Secrets Manager mengenkripsi rahasia menggunakan AWS Key Management Service kunci. Untuk informasi selengkapnya, lihat [Enkripsi rahasia dan dekripsi](#) di [AWS Key Management Service](#)

Integrasi terkelola untuk AWS IoT Device Management terintegrasi AWS Secrets Manager sehingga Anda dapat menyimpan data di Secrets Manager dan menggunakan ID rahasia dalam konfigurasi Anda.

Bagaimana integrasi terkelola menggunakan rahasia

Open Authorization (OAuth) adalah standar terbuka untuk otorisasi akses yang didelegasikan, memungkinkan pengguna untuk memberikan situs web atau aplikasi akses ke informasi mereka di situs web lain tanpa membagikan kata sandi mereka. Ini adalah cara aman bagi aplikasi pihak ketiga untuk mengakses data pengguna atas nama pengguna, memberikan alternatif yang lebih aman untuk berbagi kata sandi.

Dalam OAuth, ID klien dan rahasia klien adalah kredensial yang mengidentifikasi dan mengautentikasi aplikasi klien ketika meminta token akses.

Integrasi terkelola untuk AWS IoT Device Management penggunaan OAuth untuk berkomunikasi dengan pelanggan yang menggunakan alur kerja C2C. Pelanggan perlu memberikan ID klien dan rahasia klien untuk berkomunikasi. Integrasi terkelola pelanggan akan menyimpan ID klien dan rahasia klien di AWS akun mereka, dan integrasi terkelola membaca ID klien dan rahasia klien di akun pelanggan kami.

Cara membuat rahasia

Untuk membuat rahasia, ikuti langkah-langkah di [Buat AWS Secrets Manager rahasia](#) di Panduan AWS Secrets Manager Pengguna.

Anda harus membuat rahasia Anda dengan AWS KMS kunci yang dikelola pelanggan untuk integrasi terkelola untuk membaca nilai rahasia. Untuk informasi [selengkapnya, lihat Izin untuk AWS KMS kunci](#) di Panduan AWS Secrets Manager Pengguna.

Anda juga harus menggunakan kebijakan IAM di bagian berikut.

Berikan akses untuk integrasi terkelola AWS IoT Device Management untuk mengambil rahasia

Untuk memungkinkan integrasi terkelola mengambil nilai rahasia dari Secrets Manager, sertakan izin berikut dalam kebijakan sumber daya untuk rahasia saat Anda membuatnya.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [ {
    "Effect": "Allow",
    "Principal": {
      "Service": "iotmanagedintegrations.amazonaws.com"
    },
    "Action": [ "secretsmanager:GetSecretValue" ],
    "Resource": "*",
    "Condition": {
      "StringEquals": {
        "aws:SourceArn": "arn:aws:iotmanagedintegrations:AWS Region:account-
id:account-association:account-association-id"
      }
    }
  } ]
}
```

Tambahkan pernyataan berikut ke kebijakan untuk kunci yang dikelola pelanggan AWS KMS .

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [ {
    "Effect": "Allow",
    "Action": [
      "kms:Decrypt",
      "kms:DescribeKey"
    ],
    "Principal": {
      "Service": [
        "iotmanagedintegrations.amazonaws.com"
      ]
    },
    "Resource": [
      "arn:aws:kms:us-east-1:123456789012:key/*"
    ]
  } ]
}
```

```
    }  
  ]  
}
```

Validasi kepatuhan untuk integrasi terkelola

Untuk mempelajari apakah an Layanan AWS berada dalam lingkup program kepatuhan tertentu, lihat [Layanan AWS di Lingkup oleh Program Kepatuhan Layanan AWS](#) dan pilih program kepatuhan yang Anda minati. Untuk informasi umum, lihat [Program AWS Kepatuhan Program AWS](#) .

Anda dapat mengunduh laporan audit pihak ketiga menggunakan AWS Artifact. Untuk informasi selengkapnya, lihat [Mengunduh Laporan di AWS Artifact](#) .

Tanggung jawab kepatuhan Anda saat menggunakan Layanan AWS ditentukan oleh sensitivitas data Anda, tujuan kepatuhan perusahaan Anda, dan hukum dan peraturan yang berlaku. Untuk informasi selengkapnya tentang tanggung jawab kepatuhan Anda saat menggunakan Layanan AWS, lihat [Dokumentasi AWS Keamanan](#).

Gunakan integrasi terkelola dengan titik akhir VPC antarmuka

Anda dapat membuat koneksi pribadi antara Amazon VPC dan integrasi AWS IoT Terkelola dengan membuat antarmuka titik akhir VPC Amazon. Endpoint antarmuka didukung oleh AWS PrivateLink, teknologi yang memungkinkan Anda mengakses layanan secara pribadi dengan menggunakan alamat IP pribadi. AWS PrivateLink membatasi semua lalu lintas jaringan antara VPC dan Integrasi Terkelola IoT Anda ke jaringan Amazon. Anda tidak memerlukan gateway internet, perangkat NAT, atau koneksi VPN.

Anda tidak diharuskan untuk menggunakannya AWS PrivateLink, tetapi disarankan. Untuk informasi selengkapnya tentang AWS PrivateLink dan titik akhir VPC, lihat [Mengakses AWS layanan melalui AWS PrivateLink Panduan](#).AWS PrivateLink

Topik

- [Pertimbangan untuk Integrasi AWS IoT Terkelola titik akhir VPC](#)
- [Membuat titik akhir VPC antarmuka untuk integrasi Terkelola AWS IoT](#)
- [Menguji titik akhir VPC Anda](#)
- [Mengontrol akses ke layanan melalui titik akhir VPC](#)

- [Penetapan harga](#)
- [Batasan](#)

Pertimbangan untuk Integrasi AWS IoT Terkelola titik akhir VPC

Sebelum Anda menyiapkan titik akhir VPC antarmuka untuk integrasi AWS IoT Terkelola, tinjau [properti dan batasan titik akhir Antarmuka](#) dalam Panduan.AWS PrivateLink

AWS IoT Integrasi terkelola mendukung panggilan ke semua tindakan API-nya dari VPC Anda melalui titik akhir VPC antarmuka.

Titik akhir yang didukung

AWS IoT Integrasi terkelola mendukung titik akhir VPC untuk antarmuka layanan berikut:

- API bidang kontrol: `com.amazonaws.region.iotmanagedintegrations.api`

Titik akhir yang tidak didukung

Titik akhir integrasi AWS IoT Terkelola berikut tidak mendukung titik akhir VPC:

- Titik akhir MQTT: Perangkat MQTT biasanya digunakan di lingkungan pengguna akhir daripada di dalam, membuat integrasi tidak perlu. AWS VPCs AWS PrivateLink
- OAuth titik akhir callback: Banyak platform pihak ketiga tidak beroperasi dalam AWS infrastruktur, mengurangi manfaat AWS PrivateLink dukungan untuk OAuth arus.

Ketersediaan

AWS IoT Integrasi terkelola titik akhir VPC tersedia di Wilayah berikut: AWS

- Canada (Central) - ca-central-1
- Europe (Ireland) - eu-west-1

Wilayah tambahan akan didukung karena integrasi AWS IoT Terkelola memperluas ketersediaannya.

Dukungan dual-stack

AWS IoT Integrasi terkelola Titik akhir VPC mendukung IPv4 keduanya dan lalu lintas. IPv6 Anda dapat membuat titik akhir VPC dengan jenis alamat IP berikut:

- IPv4: Menetapkan IPv4 alamat ke antarmuka jaringan titik akhir
- IPv6: Menetapkan IPv6 alamat ke antarmuka jaringan titik akhir (membutuhkan IPv6 subnet -only)
- Dualstack: Menetapkan keduanya IPv4 dan IPv6 alamat ke antarmuka jaringan titik akhir

Membuat titik akhir VPC antarmuka untuk integrasi Terkelola AWS IoT

Anda dapat membuat titik akhir VPC untuk layanan integrasi AWS IoT Terkelola menggunakan Konsol VPC Amazon atau (CLI). AWS CLI AWS

Untuk membuat antarmuka VPC endpoint untuk integrasi AWS IoT Terkelola (konsol)

1. Buka Konsol VPC Amazon di Konsol [VPC](#) Amazon.
2. Di panel navigasi, pilih Titik Akhir.
3. Pilih Buat Titik Akhir.
4. Untuk kategori Layanan, pilih AWS layanan.
5. Untuk nama Layanan, pilih nama layanan yang sesuai dengan AWS Wilayah Anda. Contoh:
 - `com.amazonaws.ca-central-1.iotmanagedintegrations.api`
 - `com.amazonaws.eu-west-1.iotmanagedintegrations.api`
6. Untuk VPC, pilih VPC tempat Anda akan mengakses Integrasi Terkelola. AWS IoT
7. Untuk Pengaturan tambahan, Aktifkan nama DNS dipilih secara default. Kami menyarankan Anda untuk mempertahankan pengaturan ini. Ini memastikan bahwa permintaan ke titik akhir layanan publik integrasi AWS IoT Terkelola diselesaikan ke titik akhir VPC Amazon Anda.
8. Untuk Subnet, pilih subnet untuk membuat antarmuka jaringan titik akhir. Anda dapat memilih satu subnet per Availability Zone.
9. Untuk jenis alamat IP, pilih dari opsi berikut:
 - IPv4: Tetapkan IPv4 alamat ke antarmuka jaringan titik akhir
 - IPv6: Tetapkan IPv6 alamat ke antarmuka jaringan titik akhir (didukung hanya jika semua subnet yang dipilih hanya -only) IPv6
 - Dualstack: Tetapkan keduanya IPv4 dan IPv6 alamat ke antarmuka jaringan titik akhir

10. Untuk grup Keamanan, pilih grup keamanan untuk dikaitkan dengan antarmuka jaringan titik akhir. Aturan grup keamanan harus memungkinkan komunikasi antara antarmuka jaringan titik akhir dan sumber daya di VPC Anda yang berkomunikasi dengan layanan.
11. Untuk Kebijakan, pilih Akses penuh untuk mengizinkan semua operasi oleh semua prinsipal pada semua sumber daya melalui titik akhir antarmuka. Untuk membatasi akses, pilih Kustom dan tentukan kebijakan.
12. (Opsional) Untuk menambahkan tag, pilih Tambahkan tag baru dan masukkan kunci tag dan nilai.
13. Pilih Buat titik akhir.

Untuk membuat titik akhir VPC antarmuka untuk Integrasi Terkelola IoT (AWS CLI)

Gunakan [create-vpc-endpoint](#) perintah dan tentukan ID VPC, tipe titik akhir VPC (antarmuka), nama layanan, subnet yang akan menggunakan titik akhir, dan grup keamanan untuk dikaitkan dengan antarmuka jaringan titik akhir.

```
aws ec2 create-vpc-endpoint \  
  --vpc-id vpc-12345678 \  
  --route-table-ids rtb-12345678 \  
  --service-name com.amazonaws.ca-central-1.iotmanagedintegrations.api \  
  --vpc-endpoint-type Interface \  
  --subnet-ids subnet-12345678 subnet-87654321 \  
  --security-group-ids sg-12345678
```

Menguji titik akhir VPC Anda

Setelah membuat titik akhir VPC, Anda dapat menguji koneksi dengan melakukan panggilan API ke Integrasi AWS IoT terkelola dari instance EC2 di VPC Anda.

Prasyarat

- Sebuah EC2 instance di subnet pribadi dalam VPC Anda
- Izin IAM yang sesuai untuk operasi integrasi AWS IoT Terkelola
- Aturan grup keamanan yang memungkinkan lalu lintas HTTPS (port 443) ke titik akhir VPC

Menguji koneksi

1. Connect ke EC2 instans Amazon Anda di subnet pribadi.

2. Verifikasi resolusi DNS untuk nama DNS pribadi:

```
dig api.iotmanagedintegrations.region.api.aws
```

3. Uji konektivitas HTTPS:

```
curl -v https://api.iotmanagedintegrations.region.api.aws
```

4. Lakukan panggilan API Integrasi AWS IoT Terkelola:

```
aws iot-managed-integrations list-destinations \  
  --region region \  
  --endpoint-url https://api.iotmanagedintegrations.region.api.aws
```

Ganti `region` dengan AWS Wilayah Anda (misalnya, `ca-central-1`).

Mengontrol akses ke layanan melalui titik akhir VPC

Kebijakan titik akhir VPC adalah kebijakan sumber daya IAM yang Anda lampirkan ke titik akhir VPC antarmuka saat Anda membuat atau memodifikasi titik akhir. Jika Anda tidak melampirkan kebijakan ketika membuat titik akhir, kami melampirkan kebijakan default untuk Anda sehingga memungkinkan akses penuh ke layanan. Kebijakan titik akhir tidak membatalkan atau mengganti kebijakan pengguna IAM atau kebijakan khusus layanan. Ini adalah kebijakan terpisah untuk mengendalikan akses dari titik akhir ke layanan tertentu.

Kebijakan titik akhir harus ditulis dalam format JSON. Untuk informasi selengkapnya, lihat [Mengendalikan Akses ke Layanan dengan Titik Akhir VPC](#) dalam Panduan Pengguna VPC Amazon.

Contoh: Kebijakan titik akhir VPC untuk AWS IoT tindakan integrasi Terkelola

Berikut ini adalah contoh kebijakan endpoint untuk integrasi AWS IoT Terkelola. Kebijakan ini memungkinkan pengguna terhubung ke integrasi AWS IoT Terkelola melalui titik akhir VPC untuk mengakses tujuan tetapi menolak akses ke loker kredensial.

```
{  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Principal": "*",  
      "Action": [  

```

```

        "iotmanagedintegrations:ListDestinations",
        "iotmanagedintegrations:GetDestination",
        "iotmanagedintegrations:CreateDestination",
        "iotmanagedintegrations:UpdateDestination",
        "iotmanagedintegrations>DeleteDestination"
    ],
    "Resource": "*"
},
{
    "Effect": "Deny",
    "Principal": "*",
    "Action": [
        "iotmanagedintegrations:ListCredentialLockers",
        "iotmanagedintegrations:GetCredentialLocker",
        "iotmanagedintegrations:CreateCredentialLocker",
        "iotmanagedintegrations:UpdateCredentialLocker",
        "iotmanagedintegrations>DeleteCredentialLocker"
    ],
    "Resource": "*"
}
]
}

```

Contoh: Kebijakan titik akhir VPC yang membatasi akses ke peran IAM tertentu

Kebijakan titik akhir VPC berikut memungkinkan akses ke integrasi AWS IoT Terkelola hanya untuk prinsipal IAM yang memiliki peran IAM tertentu dalam rantai kepercayaan mereka. Semua kepala sekolah IAM lainnya ditolak aksesnya.

```

{
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "*",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:PrincipalArn": "arn:aws:iam::123456789012:role/
IoTManagedIntegrationsVPCRole"
                }
            }
        }
    ]
}

```

```
]
}
```

Penetapan harga

Anda dikenakan tarif standar untuk membuat dan menggunakan titik akhir VPC antarmuka dengan AWS IoT integrasi Terkelola. Untuk informasi selengkapnya, lihat [harga AWS PrivateLink](#).

Batasan

- [CreateAccountAssociation](#) API, dirancang untuk bekerja OAuth dengan layanan cloud pihak ketiga, yang mengharuskan permintaan untuk meninggalkan jaringan Amazon. Ini penting bagi pelanggan yang menggunakan AWS PrivateLink untuk memuat lalu lintas mereka dalam VPC, karena AWS PrivateLink tidak dapat menyediakan end-to-end penahanan lengkap untuk panggilan API ini.
- Titik akhir VPC untuk integrasi AWS IoT Terkelola tidak tersedia di. AWS GovCloud (US) Regions

Untuk batasan titik akhir VPC umum, lihat [Properti dan batasan titik akhir antarmuka di Panduan Pengguna](#) Amazon VPC.

Connect ke integrasi terkelola untuk endpoint AWS IoT Device Management FIPS

AWS IoT menyediakan titik akhir pesawat kontrol yang mendukung [Federal Information Processing Standard \(FIPS\)](#) 140-2. Titik akhir yang sesuai dengan FIPS berbeda dari titik akhir standar. AWS Untuk berinteraksi dengan integrasi terkelola dengan cara yang sesuai dengan FIPS, Anda harus menggunakan titik akhir yang dijelaskan di bawah ini dengan klien yang sesuai dengan FIPS Anda. AWS IoT Device Management AWS IoT Konsol tidak sesuai dengan FIPS.

Bagian berikut menjelaskan cara mengakses AWS IoT titik akhir yang sesuai dengan FIPS dengan menggunakan REST API, SDK, atau. AWS CLI

Titik akhir bidang kendali

Titik akhir bidang kontrol yang sesuai dengan FIPS yang mendukung operasi integrasi terkelola dan AWS CLI perintah terkaitnya tercantum dalam Titik Akhir [FIPS](#) menurut Layanan. Di [FIPS Endpoints by Service](#), temukan layanan AWS IoT Device Management - Integrasi terkelola, dan cari titik akhir untuk Anda. Wilayah AWS

Untuk menggunakan titik akhir yang sesuai dengan FIPS saat Anda mengakses operasi integrasi tertanam, gunakan AWS SDK atau REST API dengan titik akhir yang sesuai untuk Anda. Wilayah AWS

Untuk menggunakan titik akhir yang sesuai dengan FIPS saat Anda menjalankan perintah CLI integrasi terkelola, tambahkan `--endpoint` parameter dengan titik akhir yang sesuai untuk perintah Anda. Wilayah AWS

Pemantauan Integrasi Terkelola

Pemantauan adalah bagian penting dalam menjaga keandalan, ketersediaan, dan kinerja integrasi Terkelola dan solusi AWS Anda yang lain. AWS menyediakan alat pemantauan berikut untuk melihat integrasi terkelola, melaporkan ketika ada sesuatu yang salah, dan mengambil tindakan otomatis bila perlu:

- AWS CloudTrail menangkap panggilan API dan peristiwa terkait yang dibuat oleh atau atas nama AWS akun Anda dan mengirimkan file log ke bucket Amazon S3 yang Anda tentukan. Anda dapat mengidentifikasi pengguna dan akun mana yang dipanggil AWS, alamat IP sumber dari mana panggilan dilakukan, dan kapan panggilan terjadi. Untuk informasi selengkapnya, lihat [Panduan Pengguna AWS CloudTrail](#).

Logging Integrasi terkelola panggilan API menggunakan AWS CloudTrail

Integrasi terkelola terintegrasi dengan [AWS CloudTrail](#), layanan yang menyediakan catatan tindakan yang diambil oleh pengguna, peran, atau. Layanan AWS CloudTrail menangkap semua panggilan API untuk integrasi terkelola sebagai peristiwa. Panggilan yang diambil mencakup panggilan dari konsol integrasi terkelola dan panggilan kode ke operasi API integrasi terkelola. Dengan menggunakan informasi yang dikumpulkan oleh CloudTrail, Anda dapat menentukan permintaan yang dibuat untuk integrasi terkelola, alamat IP dari mana permintaan dibuat, kapan dibuat, dan detail tambahan.

Setiap entri peristiwa atau log berisi informasi tentang entitas yang membuat permintaan tersebut. Informasi identitas membantu Anda menentukan berikut hal ini:

- Baik permintaan tersebut dibuat dengan kredensial pengguna root atau pengguna.
- Apakah permintaan dibuat atas nama pengguna IAM Identity Center.
- Apakah permintaan tersebut dibuat dengan kredensial keamanan sementara untuk satu peran atau pengguna gabungan.
- Apakah permintaan tersebut dibuat oleh Layanan AWS lain.

CloudTrail aktif di Anda Akun AWS ketika Anda membuat akun dan Anda secara otomatis memiliki akses ke riwayat CloudTrail Acara. Riwayat CloudTrail Acara menyediakan catatan yang dapat

dilihat, dapat dicari, dapat diunduh, dan tidak dapat diubah dari 90 hari terakhir dari peristiwa manajemen yang direkam dalam file. Wilayah AWS Untuk informasi selengkapnya, lihat [Bekerja dengan riwayat CloudTrail Acara](#) di Panduan AWS CloudTrail Pengguna. Tidak ada CloudTrail biaya untuk melihat riwayat Acara.

Untuk catatan acara yang sedang berlangsung dalam 90 hari Akun AWS terakhir Anda, buat jejak atau penyimpanan data acara [CloudTrailDanau](#).

CloudTrail jalan setapak

Jejak memungkinkan CloudTrail untuk mengirimkan file log ke bucket Amazon S3. Semua jalur yang dibuat menggunakan Konsol Manajemen AWS Multi-region. Anda dapat membuat jalur Single-region atau Multi-region dengan menggunakan. AWS CLI Membuat jejak Multi-wilayah disarankan karena Anda menangkap aktivitas Wilayah AWS di semua akun Anda. Jika Anda membuat jejak wilayah Tunggal, Anda hanya dapat melihat peristiwa yang dicatat di jejak. Wilayah AWS Untuk informasi selengkapnya tentang jejak, lihat [Membuat jejak untuk Anda Akun AWS](#) dan [Membuat jejak untuk organisasi](#) di Panduan AWS CloudTrail Pengguna.

Anda dapat mengirimkan satu salinan acara manajemen yang sedang berlangsung ke bucket Amazon S3 Anda tanpa biaya CloudTrail dengan membuat jejak, namun, ada biaya penyimpanan Amazon S3. Untuk informasi selengkapnya tentang CloudTrail harga, lihat [AWS CloudTrail Harga](#). Untuk informasi tentang harga Amazon S3, lihat [Harga Amazon S3](#).

CloudTrail Menyimpan data acara danau

CloudTrail Lake memungkinkan Anda menjalankan kueri berbasis SQL pada acara Anda. CloudTrail [Lake mengonversi peristiwa yang ada dalam format JSON berbasis baris ke format Apache ORC](#). ORC adalah format penyimpanan kolumnar yang dioptimalkan untuk pengambilan data dengan cepat. Peristiwa digabungkan ke dalam penyimpanan data peristiwa, yang merupakan kumpulan peristiwa yang tidak dapat diubah berdasarkan kriteria yang Anda pilih dengan menerapkan pemilih acara [tingkat lanjut](#). Penyeleksi yang Anda terapkan ke penyimpanan data acara mengontrol peristiwa mana yang bertahan dan tersedia untuk Anda kueri. Untuk informasi lebih lanjut tentang CloudTrail Danau, lihat [Bekerja dengan AWS CloudTrail Danau](#) di Panduan AWS CloudTrail Pengguna.

CloudTrail Penyimpanan data acara danau dan kueri menimbulkan biaya. Saat Anda membuat penyimpanan data acara, Anda memilih [opsi harga](#) yang ingin Anda gunakan untuk penyimpanan data acara. Opsi penetapan harga menentukan biaya untuk menelan dan menyimpan peristiwa, dan periode retensi default dan maksimum untuk penyimpanan data acara. Untuk informasi selengkapnya tentang CloudTrail harga, lihat [AWS CloudTrail Harga](#).

Acara manajemen di CloudTrail

[Acara manajemen](#) memberikan informasi tentang operasi manajemen yang dilakukan pada sumber daya di Akun AWS. Ini juga dikenal sebagai operasi pesawat kontrol. Secara default, CloudTrail mencatat peristiwa manajemen.

Integrasi terkelola mencatat operasi bidang kontrol integrasi terkelola berikut CloudTrail sebagai peristiwa manajemen.

- `CreateCloudConnector`
- `UpdateCloudConnector`
- `GetCloudConnector`
- `DeleteCloudConnector`
- `ListCloudConnectors`
- `CreateConnectorDestination`
- `UpdateConnectorDestination`
- `GetConnectorDestination`
- `DeleteConnectorDestination`
- `ListConnectorDestinations`
- `CreateAccountAssociation`
- `UpdateAccountAssociation`
- `GetAccountAssociation`
- `DeleteAccountAssociation`
- `ListAccountAssociations`
- `StartAccountAssociationRefresh`
- `ListManagedThingAccountAssociations`
- `RegisterAccountAssociation`
- `DeregisterAccountAssociation`
- `SendConnectorEvent`
- `ListDeviceDiscoveries`
- `ListDiscoveredDevices`

Contoh acara

Peristiwa mewakili permintaan tunggal dari sumber manapun dan mencakup informasi tentang operasi API yang diminta, tanggal dan waktu operasi, parameter permintaan, dan sebagainya. CloudTrail file log bukanlah jejak tumpukan yang diurutkan dari panggilan API publik, sehingga peristiwa tidak muncul dalam urutan tertentu.

Contoh berikut menunjukkan CloudTrail peristiwa yang menunjukkan operasi `CreateCloudConnector` API yang sukses.

CloudTrail Acara sukses dengan operasi **CreateCloudConnector** API.

```
{
  "eventVersion": "1.09",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "EXAMPLE",
    "arn": "arn:aws:sts::111122223333:assumed-role/Admin/EXAMPLE",
    "accountId": "111122223333",
    "accessKeyId": "EXAMPLEKYSBQSCGRIC",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AR0AZOZQFKYSFZVB2J2GN",
        "arn": "arn:aws:iam::111122223333:role/Admin",
        "accountId": "111122223333",
        "userName": "Admin"
      },
      "attributes": {
        "creationDate": "2025-06-05T18:26:16Z",
        "mfaAuthenticated": "false"
      }
    }
  },
  "eventTime": "2025-06-05T18:30:40Z",
  "eventSource": "iotmanagedintegrations.amazonaws.com",
  "eventName": "CreateCloudConnector",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "192.0.2.0",
  "userAgent": "PostmanRuntime/7.44.0",
  "requestParameters": {
    "EndpointType": "LAMBDA",
    "Description": "Manual testing for C2C CT Validation",
  }
}
```

```

    "ClientToken": "abc7460",
    "EndpointConfig": {
      "lambda": {
        "arn": "arn:aws:lambda:us-
east-1:111122223333:function:LightweightMockConnector7460"
      }
    },
    "Name": "EdenManualTestCloudConnector"
  },
  "responseElements": {
    "X-Frame-Options": "DENY",
    "Access-Control-Expose-Headers": "Content-Length,Content-Type,X-Amzn-
Errortype,X-Amzn-Requestid",
    "Strict-Transport-Security": "max-age:47304000; includeSubDomains",
    "Cache-Control": "no-store, no-cache",
    "X-Content-Type-Options": "nosniff",
    "Content-Security-Policy": "upgrade-insecure-requests; default-src 'none';
object-src 'none'; frame-ancestors 'none'; base-uri 'none'",
    "Pragma": "no-cache",
    "Id": "f7e633e719404c4a933596b4d0cc276e",
    "Arn": "arn:aws:iotmanagedintegrations:us-east-1:111122223333:cloud-connector/
EXAMPLE404c4a933596b4d0cc276e"
  },
  "requestID": "c0071fd1-b8e0-400a-bcc0-EXAMPLE9e4",
  "eventID": "95b318ea-2f63-4183-9c22-EXAMPLE3e",
  "readOnly": false,
  "eventType": "AwsApiCall",
  "managementEvent": true,
  "recipientAccountId": "111122223333",
  "eventCategory": "Management"
}

```

Contoh berikut menunjukkan CloudTrail peristiwa yang menunjukkan operasi `ListDiscoveredDevices` API yang sukses.

CloudTrail Acara sukses dengan operasi **ListDiscoveredDevices** API.

```

{
  "eventVersion": "1.09",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "EZAMPLE",
    "arn": "arn:aws:sts::444455556666:assumed-role/Admin/EXAMPLE",

```

```
    "accountId": "444455556666",
    "accessKeyId": "EXAMPLERJ26PYMH",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "EXAMPLE",
        "arn": "arn:aws:iam::444455556666:role/Admin",
        "accountId": "444455556666",
        "userName": "Admin"
      },
      "attributes": {
        "creationDate": "2025-06-10T23:37:31Z",
        "mfaAuthenticated": "false"
      }
    }
  },
  "eventTime": "2025-06-10T23:38:07Z",
  "eventSource": "iotmanagedintegrations.amazonaws.com",
  "eventName": "ListDiscoveredDevices",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "192.0.2.0",
  "userAgent": "EXAMPLE-runtime/2.4.0",
  "requestParameters": {
    "Identifier": "EXAMPLE4f268483a17d8060f014"
  },
  "responseElements": null,
  "requestID": "27ae1f61-e2e6-43e4-bf17-EXAMPLEa568",
  "eventID": "34734e81-76a8-49a4-9641-EXAMPLE28ed",
  "readOnly": true,
  "eventType": "AwsApiCall",
  "managementEvent": true,
  "recipientAccountId": "444455556666",
  "eventCategory": "Management"
}
```

Untuk informasi tentang konten CloudTrail rekaman, lihat [konten CloudTrail rekaman](#) di Panduan AWS CloudTrail Pengguna.

Riwayat dokumen untuk integrasi terkelola Panduan Pengembang

Tabel berikut menjelaskan rilis dokumentasi untuk integrasi terkelola.

Perubahan	Deskripsi	Tanggal
Rilis ketersediaan umum	Rilis ketersediaan umum dari integrasi terkelola Panduan Pengembang	Juni 25, 2025
Rilis pratinjau awal	Rilis pratinjau awal dari integrasi terkelola Panduan Pengembang	Maret 3, 2025

Terjemahan disediakan oleh mesin penerjemah. Jika konten terjemahan yang diberikan bertentangan dengan versi bahasa Inggris aslinya, utamakan versi bahasa Inggris.