aws

Guia do desenvolvedor, versão 2

AWS IoT Greengrass

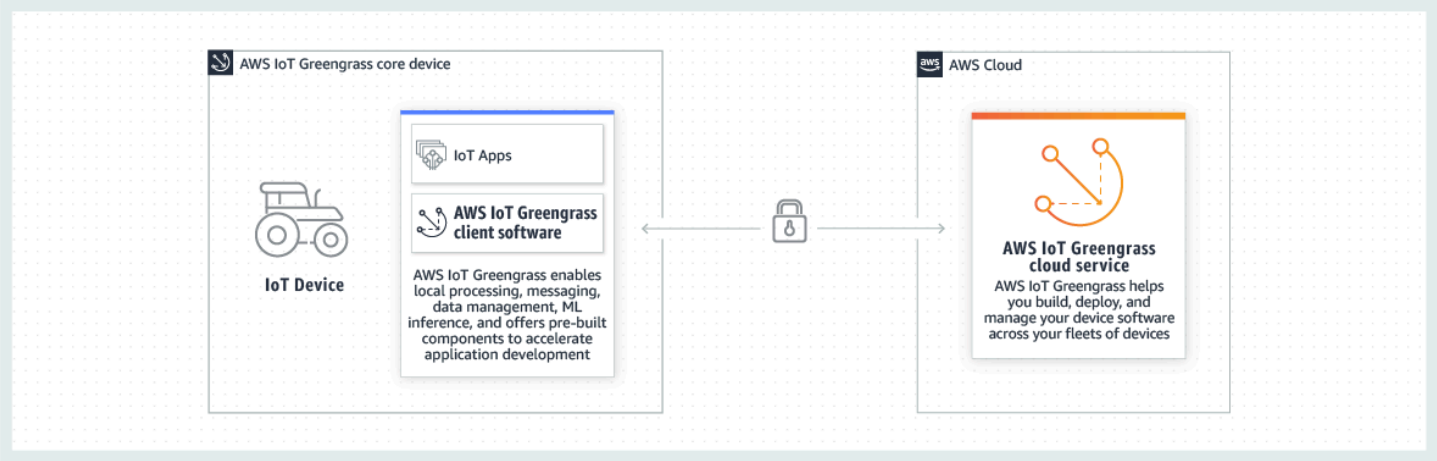
AWS IoT Greengrass: Guia do desenvolvedor, versão 2

Table of Contents

O que é o AWS IoT Greengrass?

AWS IoT Greengrassé um serviço de nuvem e tempo de execução de ponta da Internet das Coisas (IoT) de código aberto que ajuda você a criar, implantar e gerenciar aplicativos de IoT em seus dispositivos. Você pode usar AWS IoT Greengrass para criar um software que permite que seus dispositivos atuem localmente com base nos dados que eles geram, executem previsões com base em modelos de aprendizado de máquina e filtrem e agreguem dados do dispositivo. AWS IoT Greengrasspermite que seus dispositivos coletem e analisem dados mais perto de onde esses dados são gerados, reajam de forma autônoma a eventos locais e se comuniquem com segurança com outros dispositivos na rede local. Os dispositivos Greengrass também podem se comunicar com segurança e AWS IoT Core exportar dados de IoT para o. Nuvem AWS Você pode usar o AWS IoT Greengrass para criar aplicativos de borda usando módulos de software pré-construídos, chamados componentes, que podem conectar seus dispositivos de borda a serviços AWS ou serviços de terceiros. Você também pode usar AWS IoT Greengrass para empacotar e executar seu software usando funções Lambda, contêineres Docker, processos nativos do sistema operacional ou tempos de execução personalizados de sua escolha.

O exemplo a seguir mostra como um AWS IoT Greengrass dispositivo interage com o. Nuvem AWS



Novos atributos

AWS IoT Greengrass V2apresenta novos recursos e melhorias. Veja a seguir mais informações sobre os novos recursos oferecidos na versão 2.

- O que há de novo em AWS IoT Greengrass Version 2

Para usuários iniciantes do AWS IoT Greengrass

Se você é novatoAWS IoT Greengrass, recomendamos que você revise a seção a seguir:

- Como funciona o AWS IoT Greengrass

Em seguida, siga o tutorial de introdução para experimentar os recursos básicos doAWS IoT Greengrass. Neste tutorial, você instala o software AWS IoT Greengrass Core em um dispositivo, desenvolve um componente Hello World e empacota esse componente para implantação.

Para usuários existentes do AWS IoT Greengrass

Para usuários atuais doAWS IoT Greengrass V1, recomendamos os tópicos a seguir para ajudá-lo a entender as diferenças entre a versão 1 e a versão 2 do Greengrass e aprender como migrar da versão 1 para a versão 2:

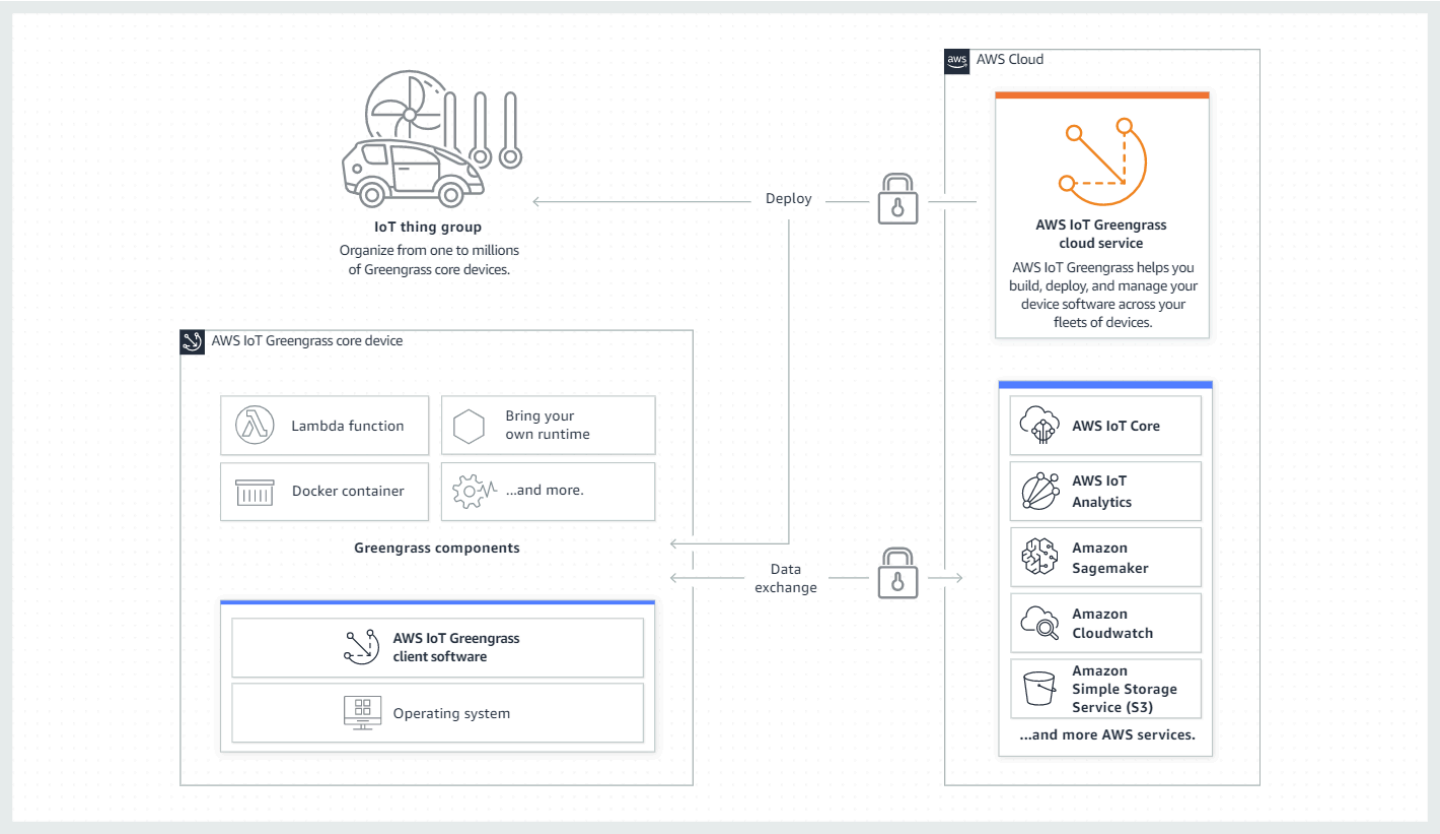
- Migrar da AWS IoT Greengrass versão 1

Como funciona o AWS IoT Greengrass

O software AWS IoT Greengrass cliente, também chamado de software AWS IoT Greengrass Core, é executado em distribuições baseadas em Windows e Linux, como Ubuntu ou Raspberry Pi OS, para dispositivos com arquiteturas ARM ou x86. Com issoAWS IoT Greengrass, você pode programar dispositivos para agir localmente com base nos dados que eles geram, executar previsões com base em modelos de aprendizado de máquina e filtrar e agregar dados do dispositivo. AWS IoT Greengrasspermite a execução local de AWS Lambda funções, contêineres Docker, processos nativos do sistema operacional ou tempos de execução personalizados de sua escolha.

AWS IoT Greengrassfornece módulos de software pré-construídos, chamados de componentes, que permitem ampliar facilmente a funcionalidade do dispositivo de ponta. AWS IoT Greengrassos componentes permitem que você se conecte a AWS serviços e aplicativos de terceiros na borda. Depois de desenvolver seus aplicativos de IoT, AWS IoT Greengrass você pode implantar, configurar e gerenciar remotamente esses aplicativos em sua frota de dispositivos em campo.

O exemplo a seguir mostra como um AWS IoT Greengrass dispositivo interage com o serviço de AWS IoT Greengrass nuvem e outros AWS serviços no Nuvem AWS.



Conceitos principais do AWS IoT Greengrass

A seguir estão os conceitos essenciais para entender e usarAWS IoT Greengrass:

AWS IoTcoisa

Qualquer AWS IoT coisa é uma representação de um dispositivo específico ou entidade lógica. As informações sobre uma coisa são armazenadas no AWS IoT registro.

Dispositivo principal Greengrass

Um dispositivo que executa o software AWS IoT Greengrass Core. Um dispositivo principal do Greengrass é uma coisa de IoT. AWS Você pode adicionar vários dispositivos principais a grupos de AWS IoT coisas para criar e gerenciar grupos de dispositivos principais do Greengrass. Para ter mais informações, consulte Configurando dispositivos AWS IoT Greengrass principais.

Dispositivo cliente Greengrass

Um dispositivo que se conecta e se comunica com um dispositivo principal do Greengrass por meio do MQTT. Um dispositivo cliente Greengrass é uma AWS IoT coisa. O dispositivo principal pode processar, filtrar e agregar dados dos dispositivos clientes que se conectam a ele. Você pode configurar o dispositivo principal para retransmitir mensagens MQTT entre dispositivos

cliente, o serviço de AWS IoT Core nuvem e os componentes do Greengrass. Para ter mais informações, consulte Interaja com dispositivos IoT locais.

Os dispositivos cliente podem executar FreeRTOS ou usar a API de descoberta do AWS IoT Device SDKGreengrass para obter informações sobre os dispositivos principais aos quais eles podem se conectar.

Componente Greengrass

Um módulo de software que é implantado e executado em um dispositivo central do Greengrass. Todo software desenvolvido e implantado com ele AWS IoT Greengrass é modelado como um componente. AWS IoT Greengrassfornece componentes públicos pré-criados que fornecem recursos e funcionalidades que você pode usar em seus aplicativos. Você também pode desenvolver seus próprios componentes personalizados, em seu dispositivo local ou na nuvem. Depois de desenvolver um componente personalizado, você pode usar o serviço de AWS IoT Greengrass nuvem para implantá-lo em dispositivos de núcleo único ou múltiplo. Você pode criar um componente personalizado e implantá-lo em um dispositivo principal. Ao fazer isso, o dispositivo principal baixa os seguintes recursos para executar o componente:

- Receita: um arquivo JSON ou YAML que descreve o módulo de software definindo detalhes, configurações e parâmetros do componente.
- Artifato: o código-fonte, os binários ou os scripts que definem o software que será executado em seu dispositivo. Você pode criar artefatos do zero ou criar um componente usando uma função Lambda, um contêiner do Docker ou um tempo de execução personalizado.
- Dependência: o relacionamento entre componentes que permite impor atualizações ou reinicializações automáticas de componentes dependentes. Por exemplo, você pode ter um componente de processamento seguro de mensagens dependente de um componente de criptografia. Isso garante que todas as atualizações do componente de criptografia atualizem e reiniciem automaticamente o componente de processamento de mensagens.

Para obter mais informações, consulte Desenvolva AWS IoT Greengrass componentes e AWS-componentes fornecidos.

Implantação

O processo para enviar componentes e aplicar a configuração desejada a um dispositivo de destino, que pode ser um único dispositivo principal do Greengrass ou um grupo de dispositivos principais do Greengrass. As implantações aplicam automaticamente todas as configurações de componentes atualizadas ao destino e incluem quaisquer outros componentes definidos como dependências. Você também pode clonar uma implantação existente para criar uma nova

implantação que usa os mesmos componentes, mas é implantada em um destino diferente. As implantações são contínuas, o que significa que todas as atualizações feitas nos componentes ou na configuração do componente de uma implantação são enviadas automaticamente para todos os destinos de destino. Para ter mais informações, consulte Implemente AWS IoT Greengrass componentes em dispositivos.

AWS IoT GreengrassSoftware principal

O conjunto de todos os AWS IoT Greengrass softwares que você instala em um dispositivo principal. AWS IoT Greengrass O software principal compreende o seguinte:

- Núcleo: Esse componente necessário fornece a funcionalidade mínima do software AWS IoT Greengrass Core. O núcleo gerencia implantações, orquestração e gerenciamento do ciclo de vida de outros componentes. Também facilita a comunicação entre AWS IoT Greengrass componentes localmente em um dispositivo individual. Para ter mais informações, consulte Núcleo Greengrass.
- Componentes opcionais: esses componentes configuráveis são fornecidos AWS IoT Greengrass e habilitam recursos adicionais em seus dispositivos de ponta. Dependendo dos seus requisitos, você pode escolher os componentes opcionais que deseja implantar no seu dispositivo, como streaming de dados, inferência de aprendizado de máquina local ou uma interface de linha de comando local. Para ter mais informações, consulte AWS-componentes fornecidos.

Você pode atualizar seu software AWS IoT Greengrass Core implantando novas versões de seus componentes em seu dispositivo.

Atributos do AWS IoT Greengrass

AWS IoT Greengrass Version 2consiste nos seguintes elementos:

- Distribuições de software
  - O componente central do Greengrass, que é a instalação mínima do AWS IoT Greengrass software Core. Esse componente gerencia as implantações, a orquestração e o gerenciamento do ciclo de vida dos componentes do Greengrass.
  - Componentes adicionais AWS fornecidos opcionalmente que se integram a serviços, protocolos e software.
  - Ferramentas de desenvolvimento do Greengrass, que você pode usar para criar, testar, criar, publicar e implantar componentes personalizados do Greengrass.

  - OAWS IoT Device SDK, que contém a biblioteca de comunicação entre processos (IPC) para componentes personalizados do Greengrass e a biblioteca de descoberta do Greengrass para dispositivos cliente.
  - O Stream Manager SDK, que você pode usar para gerenciar fluxos de dados em dispositivos principais.
- Serviço em nuvem
  - API do AWS IoT Greengrass V2
  - Console do AWS IoT Greengrass V2

Software do núcleo do AWS IoT Greengrass

Você pode usar o software AWS IoT Greengrass Core que é executado em seus dispositivos de ponta para fazer o seguinte:

- Processe fluxos de dados no dispositivo local com exportações automáticas para a AWS nuvem. Para ter mais informações, consulte Gerencie fluxos de dados nos dispositivos principais do Greengrass.
- Support mensagens MQTT entre componentes AWS IoT e. Para ter mais informações, consulte Publique/assine mensagens MQTT AWS IoT Core.
- Interaja com dispositivos locais que se conectam e se comunicam pelo MQTT. Para ter mais informações, consulte Interaja com dispositivos IoT locais.
- Support mensagens locais de publicação e assinatura entre componentes. Para ter mais informações, consulte Publique/assine mensagens locais.
- Implemente e invoque componentes e funções Lambda. Para ter mais informações, consulte Implemente AWS IoT Greengrass componentes em dispositivos.
- Gerencie os ciclos de vida dos componentes, por exemplo, com suporte para scripts de instalação e execução. Para ter mais informações, consulte AWS IoT Greengrass referência da receita do componente.
- Execute atualizações de software seguras over-the-air (OTA) do software AWS IoT Greengrass Core e dos componentes personalizados. Para obter mais informações, consulte Implemente AWS IoT Greengrass componentes em dispositivos e Atualize o software AWS IoT Greengrass principal (OTA).
- Forneça armazenamento seguro e criptografado de segredos locais e acesso controlado por componentes. Para ter mais informações, consulte Gerente secreto.

- Conexões seguras entre dispositivos e a AWS nuvem com autenticação e autorização de dispositivos. Para ter mais informações, consulte Autorização e autenticação do dispositivo para o AWS IoT Greengrass.

Você configura e gerencia os principais dispositivos do Greengrass por meio de AWS IoT Greengrass APIs nas quais cria implantações contínuas de software. Para ter mais informações, consulte Implemente AWS IoT Greengrass componentes em dispositivos.

Alguns recursos são compatíveis somente com determinadas plataformas. Para ter mais informações, consulte Compatibilidade de recursos do Greengrass por sistema operacional.

Para obter mais informações sobre plataformas, requisitos e downloads compatíveis, consulteConfigurando dispositivos AWS IoT Greengrass principais.

Ao fazer download desse software, você concorda com o Contrato de licença do software do Greengrass Core.

Compatibilidade de recursos do Greengrass por sistema operacional

AWS IoT Greengrass suporta dispositivos que executam vários sistemas operacionais. Alguns recursos são compatíveis somente com determinados sistemas operacionais. Use as tabelas a seguir para saber quais recursos estão disponíveis para cada sistema operacional compatível. Para obter mais informações sobre sistemas operacionais suportados, requisitos e como configurar os dispositivos principais do Greengrass, consulte. Configurando dispositivos AWS IoT Greengrass principais

Sistema de mensagens

Atributo	Linux	Windows
Troque mensagens MQTT AWS IoT entre componentes	 Sim	 Sim

Atributo	Linux	Windows
Troque mensagens locais de publicação/assinatura entre componentes	 Sim	 Sim
Interaja com dispositivos IoT locais por meio do MQTT	 Sim	 Sim
Interaja com dispositivos Modbus-RTU locais usando o componente Modbus-RTU	 Sim	 Não

Segurança

Atributo	Linux	Windows
Conexões seguras com autenticação e autorização de dispositivos	 Sim	 Sim
Implemente e acesse segredos seguros e criptografados do AWS Secrets Manager	 Sim	 Sim
Use um módulo de segurança de hardware (HSM) para armazenar com segurança a chave privada e o certificado do dispositivo	 Sim	 Não

Atributo	Linux	Windows
Audite os principais dispositivos com AWS IoT Device Defender	Sim	Sim
Use AWS credenciais para interagir com os serviços AWS	Sim	Sim

Instalação

Atributo	Linux	Windows
Instale AWS IoT Greengrass com provisionamento automático	Sim	Sim
Instale AWS IoT Greengrass com provisionamento manual	Sim	Sim
Instale AWS IoT Greengrass com aprovisionamento de AWS IoT frota	Sim	Sim
Instale AWS IoT Greengrass com plug-ins de provisionamento personalizados	Sim	Sim

Atributo	Linux	Windows
Execute AWS IoT Greengrass em um contêiner do Docker usando uma imagem pré-criada do Docker	 Sim	 Não

Manutenção e atualizações remotas

Atributo	Linux	Windows
Execute atualizações de software seguras over-the-air (OTA)	 Sim	 Sim
Gerencie os principais dispositivos com AWS Systems Manager	 Sim	 Não
Conecte-se aos principais dispositivos com AWS IoT tunelamento seguro	 Sim	 Não

Machine learning

Atributo	Linux	Windows
Execute inferências de aprendizado de máquina usando o Amazon SageMaker Edge Manager	 Sim	 Sim

Atributo	Linux	Windows
Execute inferências de aprendizado de máquina usando o Amazon Lookout for Vision	 Sim	 Não
Execute inferência de aprendizado de máquina usando DLR	 Sim	 Sim
Execute inferência de aprendizado de máquina usando TensorFlow	 Sim	 Sim

Características do componente

Atributo	Linux	Windows
Implemente e invoque funções Lambda	 Sim	 Não
Execute contêineres Docker em componentes	 Sim	 Sim
Processe e exporte fluxos de dados de alto volume usando o gerenciador de fluxos	 Sim	 Sim

Atributo	Linux	Windows
Gerencie os ciclos de vida dos componentes com scripts de ciclo de vida	Sim	Sim
Interaja com as sombras do dispositivo	Sim	Sim
Faça upload de registros para o Amazon CloudWatch Logs	Sim	Sim
Faça upload de dados para CloudWatch as métricas da Amazon usando o component e de CloudWatch métricas	Sim	Sim
Publique mensagens no Amazon Simple Notification Service usando o componente Amazon SNS	Sim	Não
Publique dados nos streams de entrega do Amazon Data Firehose usando o gerenciad or de streams	Sim	Sim
Publique dados nos fluxos de entrega do Amazon Data Firehose usando o component e Firehose	Sim	Não

Atributo	Linux	Windows
Reúna e aja de acordo com as métricas de telemetria do sistema em tempo real	Sim	Sim
Configurar limites de recursos do sistema para processos de componentes	Sim	Não
Pausar e retomar processos de componentes	Sim	Não
Integre com AWS IoT SiteWise o uso dos AWS IoT SiteWise componentes	Sim	Sim
Publique streams de vídeo no Amazon Kinesis Video Streams usando o conector de borda para o componente Kinesis Video Streams	Sim	Não

Desenvolvimento de componentes

Atributo	Linux	Windows
Desenvolva componentes localmente nos dispositivos principais	Sim	Sim

Atributo	Linux	Windows
Interaja com um dispositivo principal usando a AWS IoT Greengrass CLI	Sim	Sim
Interaja com um dispositivo principal usando o console de depuração local	Sim	Sim
Use o AWS IoT Device SDK for Python em componentes personalizados	Sim	Sim
Use o AWS IoT Device SDK para C++ em componentes personalizados	Sim	Sim
Use o AWS IoT Device SDK for Java em componentes personalizados	Sim	Sim

Certificação de dispositivos

Atributo	Linux	Windows
Use AWS IoT Device Tester para AWS IoT Greengrass V2 validar dispositivos de IoT	Sim	Sim

O que há de novo em AWS IoT Greengrass Version 2

AWS IoT Greengrass Version 2 é uma versão principal AWS IoT Greengrass que apresenta os seguintes recursos:

- Componentes compatíveis com o Publisher — AWS IoT Greengrass agora oferece componentes compatíveis com o Publisher. Esses componentes são desenvolvidos, oferecidos e atendidos por fornecedores terceirizados. Para ter mais informações, consulte Componentes compatíveis com o editor.
- Operar um dispositivo Greengrass na VPC — A operação de um dispositivo principal do Greengrass na VPC já está disponível. Isso permite que você execute implantações em VPC sem acesso público à Internet. Para ter mais informações, consulte Opere um dispositivo AWS IoT Greengrass principal na VPC.
- Greengrass Testing Framework (GTF) — O GTF for AWS IoT Greengrass Version 2 já está disponível. O GTF é uma coleção de blocos de construção para apoiar a end-to-end automação. Ele permite que os clientes AWS IoT Greengrass Version 2 internos usem a mesma estrutura de testes que a equipe de serviço usa para qualificar alterações de software, aceitação automatizada e fins de garantia de qualidade. Para obter mais informações, consulte Greengrass Testing Framework no Github.
- Certificado pela PSA — as versões AWS IoT Greengrass nucleus 2.7.0 e posteriores agora são certificadas pela Platform Security Architecture (PSA). Para obter mais informações, consulte AWS IoT Greengrass É certificado pela PSA.

AWS IoT Greengrass as notas de lançamento fornecem detalhes sobre os AWS IoT Greengrass lançamentos — novos recursos, atualizações e melhorias, além de correções gerais. AWS IoT Greengrass tem os seguintes tipos de versões:

- Lançamentos de novos recursos para AWS IoT Greengrass
- AWS IoT Greengrass Atualizações de software principais

Esta seção contém todas as notas de AWS IoT Greengrass V2 lançamento, as mais recentes, e inclui grandes mudanças de recursos e correções de erros significativas. Para obter informações sobre pequenas correções adicionais, consulte a organização aws-greengrass em. GitHub

Notas de release

- Versão: atualização do software AWS IoT Greengrass Core v2.12.6 em 24 de maio de 2024
- Versão: atualização do software AWS IoT Greengrass Core v2.12.5 em 25 de abril de 2024
- Versão: atualização do software AWS IoT Greengrass Core v2.12.4 em 02 de abril de 2024
- Versão: atualização do software AWS IoT Greengrass Core v2.12.3 em 27 de março de 2024
- Versão: atualização do software AWS IoT Greengrass Core v2.12.2 em 15 de fevereiro de 2024
- Versão: atualização do software AWS IoT Greengrass Core v2.12.1 em 8 de dezembro de 2023
- Versão: atualização do software AWS IoT Greengrass Core v2.12.0 em 7 de novembro de 2023
- Versão: atualização do software AWS IoT Greengrass Core v2.11.3 em 18 de outubro de 2023
- Versão: atualização do software AWS IoT Greengrass Core v2.11.2 em 9 de agosto de 2023
- Versão: atualização do software AWS IoT Greengrass Core v2.11.1 em 21 de julho de 2023
- Versão: atualização do software AWS IoT Greengrass Core v2.11.0 em 28 de junho de 2023
- Versão: atualização do software AWS IoT Greengrass Core v2.10.3 em 21 de junho de 2023
- Versão: atualização do software AWS IoT Greengrass Core v2.10.2 em 5 de junho de 2023
- Versão: atualização do software AWS IoT Greengrass Core v2.10.1 em 11 de maio de 2023
- Versão: atualização do software AWS IoT Greengrass Core v2.10.0 em 9 de maio de 2023
- Versão: atualização do software AWS IoT Greengrass Core v2.9.6 em 20 de abril de 2023
- Versão: atualização do software AWS IoT Greengrass Core v2.9.5 em 30 de março de 2023
- Versão: atualização do software AWS IoT Greengrass Core v2.9.4 em 24 de fevereiro de 2023
- Versão: atualização do software AWS IoT Greengrass Core v2.9.3 em 01 de fevereiro de 2023
- Versão: atualização do software AWS IoT Greengrass Core v2.9.2 em 22 de dezembro de 2022
- Versão: atualização do software AWS IoT Greengrass Core v2.9.1 em 18 de novembro de 2022
- Versão: atualização do software AWS IoT Greengrass Core v2.9.0 em 15 de novembro de 2022
- Versão: atualização do software AWS IoT Greengrass Core v2.8.1 em 13 de outubro de 2022
- Versão: atualização do software AWS IoT Greengrass Core v2.8.0 em 7 de outubro de 2022
- Versão: atualização do software AWS IoT Greengrass Core v2.7.0 em 28 de julho de 2022
- Versão: atualização do software AWS IoT Greengrass Core v2.6.0 em 27 de junho de 2022
- Versão: atualização do software AWS IoT Greengrass Core v2.5.6 em 31 de maio de 2022
- Versão: atualização do software AWS IoT Greengrass Core v2.5.5 em 6 de abril de 2022
- Versão: atualização do software AWS IoT Greengrass Core v2.5.4 em 23 de março de 2022

Versão: atualização do software AWS IoT Greengrass Core v2.12.6 em 24 de maio de 2024

Esta versão fornece a versão 2.12.6 do componente nucleus do Greengrass e atualizações para os componentes fornecidos. AWS

Data de lançamento: 24 de maio de 2024

Detalhes do lançamento

- Atualizações públicas de componentes

Atualizações públicas de componentes

A tabela a seguir lista os componentes fornecidos por AWS isso, incluindo recursos novos e atualizados.

> **Important**
>
> Quando você implanta um componente, AWS IoT Greengrass instala as versões mais recentes suportadas de todas as dependências desse componente. Por esse motivo, novas versões AWS de patch dos componentes públicos fornecidos podem ser implantadas automaticamente em seus dispositivos principais se você adicionar novos dispositivos

a um grupo de coisas ou atualizar a implantação que visa esses dispositivos. Algumas atualizações automáticas, como a atualização do nucleus, podem fazer com que seus dispositivos reiniciem inesperadamente.
Para evitar atualizações não intencionais para um componente que está sendo executado em seu dispositivo, recomendamos que você inclua diretamente sua versão preferida desse componente ao criar uma implantação. Para obter mais informações sobre o comportamento de atualização AWS IoT Greengrass do software Core, consulteAtualize o software AWS IoT Greengrass principal (OTA).

Componente	Detalhes
Núcleo Greengrass	A versão 2.12.6 do núcleo Greengrass está disponível. Correções de erros e melhorias • Corrige um problema que causa uma falha na inicialização em determinados processadores ARMv8, incluindo o Jetson Nano.
CLI do Greengrass	A versão 2.12.6 da CLI do Greengrass está disponível. Correções de erros e melhorias • Versão atualizada para a versão 2.12.6 do Greengrass nucleus.
Gerente secreto	A versão 2.1.8 do gerenciador secreto está disponível. Correções de erros e melhorias • Corrige um problema em que o gerente secreto não aceita um pagamento parcial.

Versão: atualização do software AWS IoT Greengrass Core v2.12.5 em 25 de abril de 2024

Esta versão fornece a versão 2.12.5 do componente nucleus do Greengrass e atualizações para os componentes fornecidos. AWS

Data de lançamento: 25 de abril de 2024

Detalhes do lançamento

- Atualizações públicas de componentes

Atualizações públicas de componentes

A tabela a seguir lista os componentes fornecidos por AWS isso, incluindo recursos novos e atualizados.

> **Important**
>
> Quando você implanta um componente, AWS IoT Greengrass instala as versões mais recentes suportadas de todas as dependências desse componente. Por esse motivo, novas versões AWS de patch dos componentes públicos fornecidos podem ser implantadas automaticamente em seus dispositivos principais se você adicionar novos dispositivos a um grupo de coisas ou atualizar a implantação que visa esses dispositivos. Algumas atualizações automáticas, como a atualização do nucleus, podem fazer com que seus dispositivos sejam reiniciados inesperadamente.
> Para evitar atualizações não intencionais para um componente que está sendo executado em seu dispositivo, recomendamos que você inclua diretamente sua versão preferida desse componente ao criar uma implantação. Para obter mais informações sobre o comportamento de atualização AWS IoT Greengrass do software Core, consulteAtualize o software AWS IoT Greengrass principal (OTA).

Componente	Detalhes
Núcleo Greengrass	A versão 2.12.5 do núcleo Greengrass está disponível. Correções de erros e melhorias • Corrige um problema em que a reversão da implantação ocasional mente trava ao reverter um componente anteriormente quebrado com dependências rígidas. • Corrige um problema em que o núcleo não publica atualizações de status após o provisionamento da frota. • Adiciona novas tentativas para a `GetDeploymentConfiguration` API após receber erros 404.

Versão: atualização do software AWS IoT Greengrass Core v2.12.4 em 02 de abril de 2024

Esta versão fornece a versão 2.12.4 do componente nucleus do Greengrass e atualizações para os componentes fornecidos. AWS

Data de lançamento: 02 de abril de 2024

Detalhes do lançamento

- Atualizações públicas de componentes

Atualizações públicas de componentes

A tabela a seguir lista os componentes fornecidos por AWS isso, incluindo recursos novos e atualizados.

> **Important**
>
> Quando você implanta um componente, AWS IoT Greengrass instala as versões mais recentes suportadas de todas as dependências desse componente. Por esse motivo, novas versões AWS de patch dos componentes públicos fornecidos podem ser implantadas automaticamente em seus dispositivos principais se você adicionar novos dispositivos a um grupo de coisas ou atualizar a implantação que visa esses dispositivos. Algumas atualizações automáticas, como a atualização do nucleus, podem fazer com que seus dispositivos reiniciem inesperadamente.
>
> Para evitar atualizações não intencionais para um componente que está sendo executado em seu dispositivo, recomendamos que você inclua diretamente sua versão preferida desse componente ao criar uma implantação. Para obter mais informações sobre o comportamento de atualização AWS IoT Greengrass do software Core, consulteAtualize o software AWS IoT Greengrass principal (OTA).

Componente	Detalhes
Núcleo Greengrass	A versão 2.12.4 do núcleo Greengrass está disponível.

Componente	Detalhes
	Correções de erros e melhorias • Corrige um problema em que o núcleo entra em uma condição de impasse durante a inicialização em alguns dispositivos Linux.

Versão: atualização do software AWS IoT Greengrass Core v2.12.3 em 27 de março de 2024

Esta versão fornece a versão 2.12.3 do componente nucleus do Greengrass e atualizações para os componentes fornecidos. AWS

Data de lançamento: 27 de março de 2024

Detalhes do lançamento

- Atualizações públicas de componentes

Atualizações públicas de componentes

A tabela a seguir lista os componentes fornecidos por AWS isso, incluindo recursos novos e atualizados.

Important

Quando você implanta um componente, AWS IoT Greengrass instala as versões mais recentes suportadas de todas as dependências desse componente. Por esse motivo, novas versões AWS de patch dos componentes públicos fornecidos podem ser implantadas automaticamente em seus dispositivos principais se você adicionar novos dispositivos a um grupo de coisas ou atualizar a implantação que visa esses dispositivos. Algumas atualizações automáticas, como a atualização do nucleus, podem fazer com que seus dispositivos sejam reiniciados inesperadamente.
Para evitar atualizações não intencionais para um componente que está sendo executado em seu dispositivo, recomendamos que você inclua diretamente sua versão preferida desse componente ao criar uma implantação. Para obter mais informações sobre o comportamento de atualização AWS IoT Greengrass do software Core, consulteAtualize o software AWS IoT Greengrass principal (OTA).

Componente	Detalhes
Núcleo Greengrass	A versão 2.12.3 do núcleo Greengrass está disponível. Correções de erros e melhorias • Corrige um problema em que o núcleo não relata o status correto do componente após a reinicialização do núcleo e durante a recuperação do componente. • Melhorias e correções de erros gerais.
Gerenciador de sombras	A versão 2.3.7 do componente gerenciador de sombras está disponível. Correções de erros e melhorias Corrige um problema em que o Shadow Manager registra periodicamente um `NullPointerException` erro durante a sincronização do Shadow Manager.
Provisionamento de frotas	A versão 1.2.1 do plug-in de provisionamento de AWS IoT frota está disponível. Correções de erros e melhorias Corrige um problema em que o plug-in de provisionamento de frota fica off-line durante a inicialização do Greengrass Nucleus. O plug-in de provisionamento de frota agora repete indefinidamente as chamadas de conexão MQTT.
Detector IP	A versão 2.1.9 do componente spooler de disco está disponível. Correções de erros e melhorias Ajusta a etapa de aquisição do IP para enviar somente registros no nível do registro de depuração.
Componente de corretor Moquette MQTT 3.1.1	A versão 2.3.6 do componente broker Moquette MQTT 3.1.1 está disponível. Correções de erros e melhorias Melhorias e correções de erros gerais.

Componente	Detalhes
Gerente do Lambda	A versão 2.3.3 do componente Lambda manager está disponível. Correções de erros e melhorias Melhorias e correções de erros gerais.
Console de depuração local	A versão 2.4.2 do componente do console de depuração local está disponível. Correções de erros e melhorias Melhorias e correções de erros gerais.

Versão: atualização do software AWS IoT Greengrass Core v2.12.2 em 15 de fevereiro de 2024

Esta versão fornece a versão 2.12.2 do componente nucleus do Greengrass e atualizações para os componentes fornecidos. AWS

Data de lançamento: 15 de fevereiro de 2024

Detalhes do lançamento

- Atualizações públicas de componentes

Atualizações públicas de componentes

A tabela a seguir lista os componentes fornecidos por AWS isso, incluindo recursos novos e atualizados.

> **Important**
>
> Quando você implanta um componente, AWS IoT Greengrass instala as versões mais recentes suportadas de todas as dependências desse componente. Por esse motivo, novas versões AWS de patch dos componentes públicos fornecidos podem ser implantadas automaticamente em seus dispositivos principais se você adicionar novos dispositivos a um grupo de coisas ou atualizar a implantação que visa esses dispositivos. Algumas

atualizações automáticas, como a atualização do nucleus, podem fazer com que seus dispositivos sejam reiniciados inesperadamente.
Para evitar atualizações não intencionais para um componente que está sendo executado em seu dispositivo, recomendamos que você inclua diretamente sua versão preferida desse componente ao criar uma implantação. Para obter mais informações sobre o comportamento de atualização AWS IoT Greengrass do software Core, consulteAtualize o software AWS IoT Greengrass principal (OTA).

Componente	Detalhes
Núcleo Greengrass	A versão 2.12.2 do núcleo Greengrass está disponível. Correções de erros e melhorias • Corrige um problema em que os registros antigos não eram limpos adequadamente. • Melhorias e correções de erros gerais.
Gerenciador de sombras	A versão 2.3.6 do componente gerenciador de sombras está disponível. Correções de erros e melhorias Corrige um problema em que as propriedades de sombra que são excluídas por meio de Nuvem AWS atualizações enquanto o dispositi vo está off-line continuam existindo na sombra local após recuperar a conectividade.
Lançador Lambda	A versão 2.0.13 do componente lambda launcher está disponível. Correções de erros e melhorias Melhorias e correções de erros gerais.
Spooler de disco	A versão 1.0.3 do componente spooler de disco está disponível. Correções de erros e melhorias Melhora o desempenho ao reutilizar conexões de banco de dados.

Versão: atualização do software AWS IoT Greengrass Core v2.12.1 em 8 de dezembro de 2023

Esta versão fornece a versão 2.12.1 do componente nucleus do Greengrass e atualizações para os componentes fornecidos. AWS

Data de lançamento: 8 de dezembro de 2023

Detalhes do lançamento

- Atualizações públicas de componentes

Atualizações públicas de componentes

A tabela a seguir lista os componentes fornecidos por AWS isso, incluindo recursos novos e atualizados.

> **Important**
>
> Quando você implanta um componente, AWS IoT Greengrass instala as versões mais recentes suportadas de todas as dependências desse componente. Por esse motivo, novas versões AWS de patch dos componentes públicos fornecidos podem ser implantadas automaticamente em seus dispositivos principais se você adicionar novos dispositivos a um grupo de coisas ou atualizar a implantação que visa esses dispositivos. Algumas atualizações automáticas, como a atualização do nucleus, podem fazer com que seus dispositivos reiniciem inesperadamente.
>
> Para evitar atualizações não intencionais para um componente que está sendo executado em seu dispositivo, recomendamos que você inclua diretamente sua versão preferida desse componente ao criar uma implantação. Para obter mais informações sobre o comportamento de atualização AWS IoT Greengrass do software Core, consulteAtualize o software AWS IoT Greengrass principal (OTA).

Componente	Detalhes
Núcleo Greengrass	A versão 2.12.1 do núcleo Greengrass está disponível.

Componente	Detalhes
	Correções de erros e melhorias • Corrige um problema em que o núcleo pode duplicar as assinaturas do MQTT para tópicos de implantação, levando a registros adicionais e publicações do MQTT.
Autenticação do dispositivo cliente	A versão 2.4.5 do componente de autenticação do dispositivo cliente está disponível. Novos atributos Adiciona suporte para prefixos curinga para selecionar nomes de itens com o `selectionRule` parâmetro. Correções de erros e melhorias Corrige um problema em que os certificados não são atualizados com novas informações de conectividade em certos casos.
Spooler de disco	A versão 1.0.2 do componente spooler de disco está disponível. Correções de erros e melhorias Corrige um problema em que o campo de formato de mensagem MQTT não persiste em certos casos.
Ponte MQTT	A versão 2.3.1 do componente spooler de disco está disponível. Correções de erros e melhorias Corrige um problema em que o cliente MQTT local entra em um loop de desconexão.
Gerenciador de streams	A versão 2.1.12 do componente gerenciador de fluxo está disponível. Correções de erros e melhorias Atualiza a ordem em que as credenciais são usadas para que as credenciais do Greengrass sejam preferidas AWS para solicitações de serviço.

Versão: atualização do software AWS IoT Greengrass Core v2.12.0 em 7 de novembro de 2023

Esta versão fornece a versão 2.12.0 do componente nucleus do Greengrass e atualizações para os componentes fornecidos. AWS

Data de lançamento: 7 de novembro de 2023

Destaques do lançamento

- Bootstrap na reversão — AWS IoT Greengrass agora fornece um parâmetro de configuração do núcleo Greengrass chamado. `BootstrapOnRollback` Esse recurso permite que você execute as etapas do ciclo de vida do bootstrap como parte de uma implantação de reversão.

Detalhes do lançamento

- Atualizações públicas de componentes

Atualizações públicas de componentes

A tabela a seguir lista os componentes fornecidos por AWS isso, incluindo recursos novos e atualizados.

> **Important**
>
> Quando você implanta um componente, AWS IoT Greengrass instala as versões mais recentes suportadas de todas as dependências desse componente. Por esse motivo, novas versões AWS de patch dos componentes públicos fornecidos podem ser implantadas automaticamente em seus dispositivos principais se você adicionar novos dispositivos a um grupo de coisas ou atualizar a implantação que visa esses dispositivos. Algumas atualizações automáticas, como a atualização do nucleus, podem fazer com que seus dispositivos sejam reiniciados inesperadamente.
>
> Para evitar atualizações não intencionais para um componente que está sendo executado em seu dispositivo, recomendamos que você inclua diretamente sua versão preferida desse componente ao criar uma implantação. Para obter mais informações sobre o comportamento de atualização AWS IoT Greengrass do software Core, consulteAtualize o software AWS IoT Greengrass principal (OTA).

Componente	Detalhes
Núcleo Greengrass	A versão 2.12.0 do núcleo Greengrass está disponível. Novos atributos • Permite que você execute as etapas do ciclo de vida do bootstrap como parte de uma implantação de reversão.

Versão: atualização do software AWS IoT Greengrass Core v2.11.3 em 18 de outubro de 2023

Esta versão fornece a versão 2.11.3 do componente do núcleo do Greengrass.

Data de lançamento: 18 de outubro de 2023

Detalhes do lançamento

- Atualizações públicas de componentes

Atualizações públicas de componentes

A tabela a seguir lista os componentes fornecidos por AWS isso, incluindo recursos novos e atualizados.

> **Important**
>
> Quando você implanta um componente, AWS IoT Greengrass instala as versões mais recentes suportadas de todas as dependências desse componente. Por esse motivo, novas versões AWS de patch dos componentes públicos fornecidos podem ser implantadas automaticamente em seus dispositivos principais se você adicionar novos dispositivos a um grupo de coisas ou atualizar a implantação que visa esses dispositivos. Algumas atualizações automáticas, como a atualização do nucleus, podem fazer com que seus dispositivos reiniciem inesperadamente.
> Para evitar atualizações não intencionais para um componente que está sendo executado em seu dispositivo, recomendamos que você inclua diretamente sua versão preferida desse componente ao criar uma implantação. Para obter mais informações sobre o comportamento

de atualização AWS IoT Greengrass do software Core, consulteAtualize o software AWS IoT Greengrass principal (OTA).

Componente	Detalhes
Núcleo Greengrass	A versão 2.11.3 do núcleo Greengrass está disponível. Correções de erros e melhorias • Corrige um problema no núcleo em que ele pode iniciar incorretamente um componente quando suas dependências falham. Novos atributos • Adiciona um tipo de endpoint s3 configurável.
Gerente do Lambda	A versão 2.3.1 do componente Lambda manager está disponível. Correções de erros e melhorias • Ajusta os níveis de log para determinados erros.
Console de depuração local	A versão 2.4.0 do componente Lambda Manager está disponível. Novos atributos • Adiciona o console de depuração do gerenciador de streams.
Gerenciador de registros	A versão 2.3.6 do componente gerenciador de registros está disponível. Correções de erros e melhorias • Ajusta os níveis de log para determinados erros.
Gerenciador de sombras	A versão 2.3.4 do componente Shadow manager está disponível. Correções de erros e melhorias • Adiciona suporte para documentos de estado de sombra nulos e vazios.

Versão: atualização do software AWS IoT Greengrass Core v2.11.2 em 9 de agosto de 2023

Esta versão fornece a versão 2.11.2 do componente do núcleo Greengrass.

Data de lançamento: 9 de agosto de 2023

Detalhes do lançamento

- Atualizações públicas de componentes

Atualizações públicas de componentes

A tabela a seguir lista os componentes fornecidos por AWS isso, incluindo recursos novos e atualizados.

> **Important**
>
> Quando você implanta um componente, AWS IoT Greengrass instala as versões mais recentes suportadas de todas as dependências desse componente. Por esse motivo, novas versões AWS de patch dos componentes públicos fornecidos podem ser implantadas automaticamente em seus dispositivos principais se você adicionar novos dispositivos a um grupo de coisas ou atualizar a implantação que visa esses dispositivos. Algumas atualizações automáticas, como a atualização do nucleus, podem fazer com que seus dispositivos reiniciem inesperadamente.
>
> Para evitar atualizações não intencionais para um componente que está sendo executado em seu dispositivo, recomendamos que você inclua diretamente sua versão preferida desse componente ao criar uma implantação. Para obter mais informações sobre o comportamento de atualização AWS IoT Greengrass do software Core, consulteAtualize o software AWS IoT Greengrass principal (OTA).

Componente	Detalhes
Núcleo Greengrass	A versão 2.11.2 do núcleo Greengrass está disponível.

Componente	Detalhes
	Correções de erros e melhorias • Corrige um problema no cliente nucleus MQTT 5 em que ele pode aparecer offline quando um grande número (> 50) de assinaturas está em uso. • Adiciona uma nova tentativa para a falha do docker dial TCP.

Versão: atualização do software AWS IoT Greengrass Core v2.11.1 em 21 de julho de 2023

Esta versão fornece a versão 2.11.1 do componente do núcleo Greengrass.

Data de lançamento: 21 de julho de 2023

Detalhes do lançamento

- Atualizações públicas de componentes

Atualizações públicas de componentes

A tabela a seguir lista os componentes fornecidos por AWS isso, incluindo recursos novos e atualizados.

> **Important**
>
> Quando você implanta um componente, AWS IoT Greengrass instala as versões mais recentes suportadas de todas as dependências desse componente. Por esse motivo, novas versões AWS de patch dos componentes públicos fornecidos podem ser implantadas automaticamente em seus dispositivos principais se você adicionar novos dispositivos a um grupo de coisas ou atualizar a implantação que visa esses dispositivos. Algumas atualizações automáticas, como a atualização do nucleus, podem fazer com que seus dispositivos reiniciem inesperadamente.
> Para evitar atualizações não intencionais para um componente que está sendo executado em seu dispositivo, recomendamos que você inclua diretamente sua versão preferida desse componente ao criar uma implantação. Para obter mais informações sobre o comportamento

de atualização AWS IoT Greengrass do software Core, consulteAtualize o software AWS IoT Greengrass principal (OTA).

Componente	Detalhes
Núcleo Greengrass	A versão 2.11.1 do núcleo Greengrass está disponível. Correções de erros e melhorias • Corrige um problema em que o núcleo não é iniciado se uma tarefa de bootstrap falhar e o arquivo de metadados de implantação estiver corrompido. • Corrige um problema em que os componentes Lambda sob demanda não são relatados nas atualizações de status de implantação. • Adiciona suporte para IDs de política de autorização duplicados.
Gerente do Lambda	A versão 2.2.11 do gerenciador Lambda está disponível. Correções de erros e melhorias • Corrige um problema em que a LegacySubscriptionRouter configuração não é atualizada quando a configuração do Lambda é alterada.

Versão: atualização do software AWS IoT Greengrass Core v2.11.0 em 28 de junho de 2023

Esta versão fornece a versão 2.11.0 do componente do núcleo do Greengrass.

Data de lançamento: 28 de junho de 2023

Destaques do lançamento

- Spooler de disco persistente — AWS IoT Greengrass agora fornece uma implementação de spooler persistente para mensagens enviadas dos principais dispositivos do Greengrass para o. AWS IoT Core Esse componente armazenará essas mensagens de saída no disco. Para ter mais informações, consulte Spooler de disco.

- Melhorias na implantação local — Agora você pode cancelar implantações locais, definir políticas de tratamento de falhas na implantação e obter o status detalhado da implantação.
- Melhorias na velocidade de registro — As velocidades de upload de registros para o componente gerenciador de registros foram aprimoradas.

Detalhes do lançamento

- Atualizações públicas de componentes

Atualizações públicas de componentes

A tabela a seguir lista os componentes fornecidos por AWS isso, incluindo recursos novos e atualizados.

> **Important**
>
> Quando você implanta um componente, AWS IoT Greengrass instala as versões mais recentes suportadas de todas as dependências desse componente. Por esse motivo, novas versões AWS de patch dos componentes públicos fornecidos podem ser implantadas automaticamente em seus dispositivos principais se você adicionar novos dispositivos a um grupo de coisas ou atualizar a implantação que visa esses dispositivos. Algumas atualizações automáticas, como a atualização do nucleus, podem fazer com que seus dispositivos sejam reiniciados inesperadamente.
> Para evitar atualizações não intencionais para um componente que está sendo executado em seu dispositivo, recomendamos que você inclua diretamente sua versão preferida desse componente ao criar uma implantação. Para obter mais informações sobre o comportamento de atualização AWS IoT Greengrass do software Core, consulteAtualize o software AWS IoT Greengrass principal (OTA).

Componente	Detalhes
Núcleo Greengrass	A versão 2.11.0 do núcleo Greengrass está disponível. Novos atributos • Permite que você cancele uma implantação local.

Componente	Detalhes
	• Permite que você configure uma política de tratamento de falhas para uma implantação local. • Adiciona suporte para um plug-in de spooler de disco.
CLI do Greengrass	A versão 2.11.0 da CLI do Greengrass está disponível. Novos atributos • Permite que você cancele uma implantação local. • Permite que você configure uma política de tratamento de falhas para uma implantação local. • Melhora os relatórios detalhados do status de implantação.
Spooler de disco	A versão 1.0.0 do componente spooler de disco está disponível. • O componente spooler de disco fornece armazenamento persistente de mensagens enviadas dos principais dispositivos do Greengrass para o. AWS IoT Core
Gerenciador de registros	A versão 2.3.5 do componente gerenciador de registros está disponível. Melhorias Melhora a velocidade de upload dos registros.

Versão: atualização do software AWS IoT Greengrass Core v2.10.3 em 21 de junho de 2023

Esta versão fornece a versão 2.10.3 do componente do núcleo do Greengrass.

Data de lançamento: 21 de junho de 2023

Detalhes do lançamento

- Atualizações públicas de componentes

Atualizações públicas de componentes

A tabela a seguir lista os componentes fornecidos por AWS isso, incluindo recursos novos e atualizados.

> **Important**
>
> Quando você implanta um componente, AWS IoT Greengrass instala as versões mais recentes suportadas de todas as dependências desse componente. Por esse motivo, novas versões AWS de patch dos componentes públicos fornecidos podem ser implantadas automaticamente em seus dispositivos principais se você adicionar novos dispositivos a um grupo de coisas ou atualizar a implantação que visa esses dispositivos. Algumas atualizações automáticas, como a atualização do nucleus, podem fazer com que seus dispositivos reiniciem inesperadamente.
>
> Para evitar atualizações não intencionais para um componente que está sendo executado em seu dispositivo, recomendamos que você inclua diretamente sua versão preferida desse componente ao criar uma implantação. Para obter mais informações sobre o comportamento de atualização AWS IoT Greengrass do software Core, consulteAtualize o software AWS IoT Greengrass principal (OTA).

Componente	Detalhes
Núcleo Greengrass	A versão 2.10.3 do núcleo Greengrass está disponível. Correções de erros e melhorias • Corrige um problema em que o Greengrass não se inscreve para receber notificações de implantação ao usar o provedor PKCS #11.

Versão: atualização do software AWS IoT Greengrass Core v2.10.2 em 5 de junho de 2023

Esta versão fornece a versão 2.10.2 do componente do núcleo Greengrass.

Data de lançamento: 5 de junho de 2023

Detalhes do lançamento

Atualizações públicas de componentes

A tabela a seguir lista os componentes fornecidos por AWS isso, incluindo recursos novos e atualizados.

> **Important**
>
> Quando você implanta um componente, AWS IoT Greengrass instala as versões mais recentes suportadas de todas as dependências desse componente. Por esse motivo, novas versões AWS de patch dos componentes públicos fornecidos podem ser implantadas automaticamente em seus dispositivos principais se você adicionar novos dispositivos a um grupo de coisas ou atualizar a implantação que visa esses dispositivos. Algumas atualizações automáticas, como a atualização do nucleus, podem fazer com que seus dispositivos reiniciem inesperadamente.
> Para evitar atualizações não intencionais para um componente que está sendo executado em seu dispositivo, recomendamos que você inclua diretamente sua versão preferida desse componente ao criar uma implantação. Para obter mais informações sobre o comportamento de atualização AWS IoT Greengrass do software Core, consulteAtualize o software AWS IoT Greengrass principal (OTA).

Componente	Detalhes
Núcleo Greengrass	A versão 2.10.2 do núcleo Greengrass está disponível. Correções de erros e melhorias • Permite a análise dos ciclos de vida dos componentes sem distinção entre maiúsculas e minúsculas. • Corrige um problema em que a variável PATH do ambiente não foi recriada corretamente. • Corrige a codificação de URI de proxy para componentes, incluindo gerenciador de fluxo para nomes de usuário com caracteres especiais.

Componente	Detalhes
Autenticação do dispositivo cliente	A versão 2.4.2 do componente de autenticação do dispositivo cliente está disponível. Novos atributos Adiciona uma nova opção `startupTimeoutSeconds` de configuração.
Gerente do Lambda	A versão 2.2.9 do componente Lambda manager está disponível. Correções de erros e melhorias Corrige um problema em que o número da porta está corrompido devido a um relógio distorcido.
Gerenciador de registros	A versão 2.3.4 do componente gerenciador de registros está disponível. Correções de erros e melhorias • Adiciona suporte para definir o `periodicUploadIntervalSec` parâmetro para valores fracionários. O mínimo é de 1 microssegundo. • Corrige um problema em que o gerenciador de registros não respeita os CloudWatch `putLogEvents` limites.
Corretor MQTT 3.1 (Moquette)	A versão 2.3.3 do componente broker MQTT 3.1 (Moquette) está disponível. Novos atributos Adiciona uma nova opção `startupTimeoutSeconds` de configuração.
Ponte MQTT	A versão 2.2.6 do componente de ponte MQTT está disponível. Novos atributos Adiciona uma nova opção `startupTimeoutSeconds` de configuração.

Componente	Detalhes
Gerenciador de stream	A versão 2.1.7 do componente gerenciador de fluxo está disponível. Correções de erros e melhorias Corrige um problema em que o gerenciador de fluxo não consegue ler a configuração do proxy corretamente.

Versão: atualização do software AWS IoT Greengrass Core v2.10.1 em 11 de maio de 2023

Esta versão fornece a versão 2.10.1 do componente do núcleo do Greengrass.

Data de lançamento: 11 de maio de 2023

Detalhes do lançamento

- Atualizações públicas de componentes

Atualizações públicas de componentes

A tabela a seguir lista os componentes fornecidos por AWS isso, incluindo recursos novos e atualizados.

> **Important**
>
> Quando você implanta um componente, AWS IoT Greengrass instala as versões mais recentes suportadas de todas as dependências desse componente. Por esse motivo, novas versões AWS de patch dos componentes públicos fornecidos podem ser implantadas automaticamente em seus dispositivos principais se você adicionar novos dispositivos a um grupo de coisas ou atualizar a implantação que visa esses dispositivos. Algumas atualizações automáticas, como a atualização do nucleus, podem fazer com que seus dispositivos sejam reiniciados inesperadamente.
> Para evitar atualizações não intencionais para um componente que está sendo executado em seu dispositivo, recomendamos que você inclua diretamente sua versão preferida desse componente ao criar uma implantação. Para obter mais informações sobre o comportamento

de atualização AWS IoT Greengrass do software Core, consulteAtualize o software AWS IoT Greengrass principal (OTA).

Componente	Detalhes
Núcleo Greengrass	A versão 2.10.1 do núcleo Greengrass está disponível. Correções de erros e melhorias • Corrige um problema que poderia causar uma falha na inicialização em determinados processadores ARMv8, incluindo o Jetson Nano. • O Greengrass não fecha mais o padrão de um componente, isso reverte o comportamento para o comportamento anterior à 2.10.0
Gerenciador de streams	A versão 2.1.6 do novo gerenciador de streams está disponível. Correções de erros e melhorias Corrige um problema que poderia causar uma falha na inicialização em determinados processadores ARMv8, incluindo o Jetson Nano.

Versão: atualização do software AWS IoT Greengrass Core v2.10.0 em 9 de maio de 2023

Esta versão fornece a versão 2.10.0 do componente nucleus do Greengrass e atualizações para os componentes fornecidos. AWS

Data de lançamento: 9 de maio de 2023

Destaques do lançamento

- Suporte ao MQTT5 — AWS IoT Greengrass agora suporta o envio e o recebimento de mensagens AWS IoT Core usando o MQTT5. Para obter mais informações, consulte Publicar mensagens AWS IoT Core MQTT.

Detalhes do lançamento

- Atualizações públicas de componentes

Atualizações públicas de componentes

A tabela a seguir lista os componentes fornecidos por AWS isso, incluindo recursos novos e atualizados.

> **Important**
>
> Quando você implanta um componente, AWS IoT Greengrass instala as versões mais recentes suportadas de todas as dependências desse componente. Por esse motivo, novas versões AWS de patch dos componentes públicos fornecidos podem ser implantadas automaticamente em seus dispositivos principais se você adicionar novos dispositivos a um grupo de coisas ou atualizar a implantação que visa esses dispositivos. Algumas atualizações automáticas, como a atualização do nucleus, podem fazer com que seus dispositivos reiniciem inesperadamente.
> Para evitar atualizações não intencionais para um componente que está sendo executado em seu dispositivo, recomendamos que você inclua diretamente sua versão preferida desse componente ao criar uma implantação. Para obter mais informações sobre o comportamento de atualização AWS IoT Greengrass do software Core, consulteAtualize o software AWS IoT Greengrass principal (OTA).

Componente	Detalhes
Núcleo Greengrass	A versão 2.10.0 do núcleo Greengrass está disponível. Novos atributos • Adiciona `interpolateComponentConfiguration` suporte para a expressão regular vazia. O Greengrass agora interpola a partir do objeto de configuração raiz. • Adiciona suporte para MQTT5. • Adiciona um mecanismo para carregar componentes do plug-in rapidamente sem digitalizar. • Permite que o Greengrass economize espaço em disco excluindo imagens não utilizadas do Docker.

Componente	Detalhes
	Correções de erros e melhorias • Corrige um problema em que a reversão deixa determinados valores de configuração em vigor a partir de uma implantação. • Corrige um problema em que o núcleo do Greengrass valida uma sequência de AWS domínio em terminais de dados e não AWS credenciais personalizados. • Atualiza a resolução de dependências de vários grupos para reresolve r todas as dependências do grupo por meio de Nuvem AWS negociaçã o, em vez de se restringir à versão ativa. Essa atualização também remove o código de erro de implantação`INSTALLED_COMPONEN T_NOT_FOUND` . • Atualiza o núcleo do Greengrass para pular o download de imagens do Docker quando elas já existem localmente. • Atualiza o núcleo do Greengrass para reiniciar uma etapa de instalação do componente antes que o tempo limite expire. • Pequenas correções e melhorias adicionais.
Gerenciador de sombras	A versão 2.3.2 do novo gerenciador de sombras está disponível. Correções de erros e melhorias Corrige um problema em que o gerenciador de sombras entra no `BROKEN` estado quando o banco de dados paralelo local está corrompido.

Versão: atualização do software AWS IoT Greengrass Core v2.9.6 em 20 de abril de 2023

Esta versão fornece a versão 2.9.6 do componente do núcleo do Greengrass.

Data do lançamento: 20 de abril de 2023

Detalhes do lançamento

- Atualizações públicas de componentes

Atualizações públicas de componentes

A tabela a seguir lista os componentes fornecidos por AWS isso, incluindo recursos novos e atualizados.

> **Important**
>
> Quando você implanta um componente, AWS IoT Greengrass instala as versões mais recentes suportadas de todas as dependências desse componente. Por esse motivo, novas versões AWS de patch dos componentes públicos fornecidos podem ser implantadas automaticamente em seus dispositivos principais se você adicionar novos dispositivos a um grupo de coisas ou atualizar a implantação que visa esses dispositivos. Algumas atualizações automáticas, como a atualização do nucleus, podem fazer com que seus dispositivos sejam reiniciados inesperadamente.
> Para evitar atualizações não intencionais para um componente que está sendo executado em seu dispositivo, recomendamos que você inclua diretamente sua versão preferida desse componente ao criar uma implantação. Para obter mais informações sobre o comportamento de atualização AWS IoT Greengrass do software Core, consulteAtualize o software AWS IoT Greengrass principal (OTA).

Componente	Detalhes
Núcleo Greengrass	A versão 2.9.6 do núcleo Greengrass está disponível. Correções de erros e melhorias • Corrige um problema em que uma implantação do Greengrass falha com o erro LAUNCH_DIRECTORY_CORRUPTED e uma reinicialização subsequente do dispositivo falha ao iniciar o Greengrass. Esse erro pode ocorrer quando você move o dispositivo Greengrass entre vários grupos de coisas com implantações que exigem a reinicialização do Greengrass.

Versão: atualização do software AWS IoT Greengrass Core v2.9.5 em 30 de março de 2023

Esta versão fornece a versão 2.9.5 do componente do núcleo do Greengrass.

Data de lançamento: 30 de março de 2023

Detalhes do lançamento

- Atualizações públicas de componentes

Atualizações públicas de componentes

A tabela a seguir lista os componentes fornecidos por AWS isso, incluindo recursos novos e atualizados.

> **Important**
>
> Quando você implanta um componente, AWS IoT Greengrass instala as versões mais recentes suportadas de todas as dependências desse componente. Por esse motivo, novas versões AWS de patch dos componentes públicos fornecidos podem ser implantadas automaticamente em seus dispositivos principais se você adicionar novos dispositivos a um grupo de coisas ou atualizar a implantação que visa esses dispositivos. Algumas atualizações automáticas, como a atualização do nucleus, podem fazer com que seus dispositivos sejam reiniciados inesperadamente.
> Para evitar atualizações não intencionais para um componente que está sendo executado em seu dispositivo, recomendamos que você inclua diretamente sua versão preferida desse componente ao criar uma implantação. Para obter mais informações sobre o comportamento de atualização AWS IoT Greengrass do software Core, consulteAtualize o software AWS IoT Greengrass principal (OTA).

Componente	Detalhes
Núcleo Greengrass	A versão 2.9.5 do núcleo Greengrass está disponível.

Componente	Detalhes
	Novos atributos • Adiciona suporte à verificação de assinatura do software Greengrass nucleus. Correções de erros e melhorias • Corrige um problema em que uma implantação falha quando a região de metadados da receita local não corresponde à região de lançamento do núcleo Greengrass. O núcleo do Greengrass agora renegocia com a nuvem quando isso acontece. • Corrige um problema em que o spooler de mensagens do MQTT é preenchido e nunca remove mensagens. • Pequenas correções e melhorias adicionais.

Versão: atualização do software AWS IoT Greengrass Core v2.9.4 em 24 de fevereiro de 2023

Esta versão fornece a versão 2.9.4 do componente do núcleo Greengrass.

Data de lançamento: 24 de fevereiro de 2023

Detalhes do lançamento

- Atualizações públicas de componentes

Atualizações públicas de componentes

A tabela a seguir lista os componentes fornecidos por AWS isso, incluindo recursos novos e atualizados.

> **Important**
>
> Quando você implanta um componente, AWS IoT Greengrass instala as versões mais recentes suportadas de todas as dependências desse componente. Por esse motivo, novas versões AWS de patch dos componentes públicos fornecidos podem ser implantadas automaticamente em seus dispositivos principais se você adicionar novos dispositivos

a um grupo de coisas ou atualizar a implantação que visa esses dispositivos. Algumas atualizações automáticas, como a atualização do nucleus, podem fazer com que seus dispositivos reiniciem inesperadamente.
Para evitar atualizações não intencionais para um componente que está sendo executado em seu dispositivo, recomendamos que você inclua diretamente sua versão preferida desse componente ao criar uma implantação. Para obter mais informações sobre o comportamento de atualização AWS IoT Greengrass do software Core, consulteAtualize o software AWS IoT Greengrass principal (OTA).

Componente	Detalhes
Núcleo Greengrass	A versão 2.9.4 do núcleo Greengrass está disponível. Correções de erros e melhorias • Verifica se há uma mensagem nula antes de descartar mensagens de QOS 0. • Trunca os valores detalhados do status do trabalho se eles excederem o limite de 1024 caracteres. • Atualiza o script de bootstrap para Windows para ler corretamente o caminho raiz do Greengrass se esse caminho incluir espaços. • Atualiza a assinatura para AWS IoT Core que ela elimine as mensagens do cliente se a resposta da assinatura não for enviada. • Garante que o núcleo carregue sua configuração a partir dos arquivos de backup quando o arquivo de configuração principal estiver corrompid o ou ausente.

Versão: atualização do software AWS IoT Greengrass Core v2.9.3 em 01 de fevereiro de 2023

Esta versão fornece a versão 2.9.3 do componente do núcleo Greengrass.

Data de lançamento: 01 de fevereiro de 2023

Detalhes do lançamento

Atualizações públicas de componentes

A tabela a seguir lista os componentes fornecidos por AWS isso, incluindo recursos novos e atualizados.

> **Important**
>
> Quando você implanta um componente, AWS IoT Greengrass instala as versões mais recentes suportadas de todas as dependências desse componente. Por esse motivo, novas versões AWS de patch dos componentes públicos fornecidos podem ser implantadas automaticamente em seus dispositivos principais se você adicionar novos dispositivos a um grupo de coisas ou atualizar a implantação que visa esses dispositivos. Algumas atualizações automáticas, como a atualização do nucleus, podem fazer com que seus dispositivos reiniciem inesperadamente.
> Para evitar atualizações não intencionais para um componente que está sendo executado em seu dispositivo, recomendamos que você inclua diretamente sua versão preferida desse componente ao criar uma implantação. Para obter mais informações sobre o comportamento de atualização AWS IoT Greengrass do software Core, consulteAtualize o software AWS IoT Greengrass principal (OTA).

Componente	Detalhes
Núcleo Greengrass	A versão 2.9.3 do núcleo Greengrass está disponível. Correções de erros e melhorias • Garante que as IDs do cliente MQTT não sejam duplicadas. • Adiciona leitura e gravação de arquivos mais robustas para evitar e se recuperar da corrupção. • Tenta novamente o docker image pull em caso de erros específicos relacionados à rede. • Adiciona a `noProxyAddresses` opção de conexão MQTT.

Versão: atualização do software AWS IoT Greengrass Core v2.9.2 em 22 de dezembro de 2022

Esta versão fornece a versão 2.9.2 do componente do núcleo Greengrass.

Data de lançamento: 22 de dezembro de 2022

Detalhes do lançamento

- Atualizações públicas de componentes

Atualizações públicas de componentes

A tabela a seguir lista os componentes AWS fornecidos que incluem recursos novos e atualizados.

> **Important**
>
> Quando você implanta um componente, AWS IoT Greengrass instala as versões mais recentes suportadas de todas as dependências desse componente. Por esse motivo, novas versões AWS de patch dos componentes públicos fornecidos podem ser implantadas automaticamente em seus dispositivos principais se você adicionar novos dispositivos a um grupo de coisas ou atualizar a implantação que visa esses dispositivos. Algumas atualizações automáticas, como a atualização do nucleus, podem fazer com que seus dispositivos reiniciem inesperadamente.
> Para evitar atualizações não intencionais para um componente que está sendo executado em seu dispositivo, recomendamos que você inclua diretamente sua versão preferida desse componente ao criar uma implantação. Para obter mais informações sobre o comportamento de atualização AWS IoT Greengrass do software Core, consulteAtualize o software AWS IoT Greengrass principal (OTA).

Componente	Detalhes
Núcleo Greengrass	A versão 2.9.2 do núcleo Greengrass está disponível. Correções de erros e melhorias • Corrige um problema em que a configuração `interpolateComponentConfiguration` não se aplica a uma implantação contínua.

Componente	Detalhes
	• Usa o OSHI para listar todos os processos secundários.

Versão: atualização do software AWS IoT Greengrass Core v2.9.1 em 18 de novembro de 2022

Esta versão fornece a versão 2.9.1 do componente nucleus do Greengrass e atualizações para os componentes fornecidos. AWS

Data da versão: 18 de novembro de 2022

Destaques do lançamento

- Gerenciador de registros — O gerenciador de registros agora processa e carrega diretamente os arquivos de log ativos, em vez de esperar que novos arquivos sejam rotacionados. Essa melhoria reduz significativamente os atrasos nos registros. Para obter mais informações, consulte Gerenciador de registros

Detalhes do lançamento

- Atualizações públicas de componentes

Atualizações públicas de componentes

A tabela a seguir lista os componentes AWS fornecidos que incluem recursos novos e atualizados.

> **Important**
>
> Quando você implanta um componente, AWS IoT Greengrass instala as versões mais recentes suportadas de todas as dependências desse componente. Por esse motivo, novas versões AWS de patch dos componentes públicos fornecidos podem ser implantadas automaticamente em seus dispositivos principais se você adicionar novos dispositivos a um grupo de coisas ou atualizar a implantação que visa esses dispositivos. Algumas atualizações automáticas, como a atualização do nucleus, podem fazer com que seus dispositivos reiniciem inesperadamente.
> Para evitar atualizações não intencionais para um componente que está sendo executado em seu dispositivo, recomendamos que você inclua diretamente sua versão preferida desse

componente ao criar uma implantação. Para obter mais informações sobre o comportamento de atualização AWS IoT Greengrass do software Core, consulteAtualize o software AWS IoT Greengrass principal (OTA).

Componente	Detalhes
Núcleo Greengrass	A versão 2.9.1 do núcleo Greengrass está disponível. Correções de erros e melhorias • Adiciona uma correção em que o Greengrass reinicia se uma implantaç ão remover um componente de plug-in.
Gerenciador de registros	A versão 2.3.0 do novo gerenciador de registros está disponível. **Note** Recomendamos que você atualize para o Greengrass nucleus 2.9.1 ao atualizar para o log manager 2.3.0. Novos atributos • Reduz os atrasos no registro processando e carregando diretamente os arquivos de log ativos, em vez de esperar que novos arquivos sejam alternados. Correções de erros e melhorias • Melhora o suporte à rotação de registros ao girar arquivos com um nome exclusivo. • Pequenas correções e melhorias adicionais.

Versão: atualização do software AWS IoT Greengrass Core v2.9.0 em 15 de novembro de 2022

Esta versão fornece a versão 2.9.0 do componente nucleus do Greengrass e atualizações para os componentes fornecidos. AWS

Data de lançamento: 15 de novembro de 2022

Destaques do lançamento

- Autenticação offline — AWS IoT Greengrass agora oferece suporte à autenticação offline. Você pode configurar seu dispositivo AWS IoT Greengrass principal para que os dispositivos clientes possam se conectar a um dispositivo principal, mesmo quando o dispositivo principal não estiver conectado à nuvem. Para obter mais informações, consulte Autenticação offline.
- Subimplantações — Agora você pode criar subimplantações. Você pode usar uma subimplantação para resolver implantações malsucedidas. Cada subimplantação pode testar uma configuração diferente de uma implantação malsucedida em um subconjunto menor de dispositivos. Para obter mais informações, consulte Criar subimplantações.

Detalhes do lançamento

- Atualizações públicas de componentes

Atualizações públicas de componentes

A tabela a seguir lista os componentes AWS fornecidos que incluem recursos novos e atualizados.

> **Important**
>
> Quando você implanta um componente, AWS IoT Greengrass instala as versões mais recentes suportadas de todas as dependências desse componente. Por esse motivo, novas versões AWS de patch dos componentes públicos fornecidos podem ser implantadas automaticamente em seus dispositivos principais se você adicionar novos dispositivos a um grupo de coisas ou atualizar a implantação que visa esses dispositivos. Algumas atualizações automáticas, como a atualização do nucleus, podem fazer com que seus dispositivos reiniciem inesperadamente.
> Para evitar atualizações não intencionais para um componente que está sendo executado em seu dispositivo, recomendamos que você inclua diretamente sua versão preferida desse

componente ao criar uma implantação. Para obter mais informações sobre o comportamento de atualização AWS IoT Greengrass do software Core, consulteAtualize o software AWS IoT Greengrass principal (OTA).

Componente	Detalhes
Núcleo Greengrass	A versão 2.9.0 do núcleo Greengrass está disponível. Novos atributos • Adiciona a capacidade de criar subimplantações que repetem implantações com um subconjunto menor de dispositivos. Esse recurso cria uma maneira mais eficiente de testar e resolver implantações malsucedidas. Correções de erros e melhorias • Melhora o suporte para sistemas que não têm `useraddgroupadd`, `usermod` e. • Pequenas correções e melhorias adicionais.
Autenticação do dispositivo cliente	A versão 2.3.0 do componente de autenticação do dispositivo cliente está disponível. Novos atributos • Adiciona suporte para autenticação off-line de dispositivos clientes. Com esse recurso, os dispositivos cliente podem continuar se conectando ao dispositivo principal quando o dispositivo principal não estiver conectado à Internet. • Adiciona suporte para autoridades de certificação (CA) fornecidas pelo cliente. Seu dispositivo principal usa uma CA fornecida pelo cliente como certificado raiz para gerar certificados de agente MQTT.
Corretora MQTT 5 (EMQX)	A versão 1.2.0 do Corretora MQTT 5 (EMQX) componente está disponível. Novos atributos Adiciona suporte para cadeias de certificados.

Componente	Detalhes
Corretor Moquette MQTT	A versão 2.3.0 do novo componente de corretor Moquette MQTT está disponível. Novos atributos Adiciona suporte para cadeias de certificados.
Gerente secreto	A versão 2.1.4 do novo gerenciador secreto está disponível. Correções de erros e melhorias Corrige um problema em que segredos em cache eram removidos quando o gerenciador de segredos era implantado e o núcleo do Greengrass era reiniciado.
Gerenciador de streams	A versão 2.1.2 do novo gerenciador de streams está disponível. Correções de erros e melhorias Corrige um problema no sistema operacional Windows que usa um idioma diferente do inglês.

Versão: atualização do software AWS IoT Greengrass Core v2.8.1 em 13 de outubro de 2022

Esta versão fornece a versão 2.8.1 do componente do núcleo do Greengrass.

Data de lançamento: 13 de outubro de 2022

> **Note**
>
> Se você estiver usando o Greengrass nucleus versão 2.8.0, é altamente recomendável que você atualize para o Greengrass nucleus versão 2.8.1.

Detalhes do lançamento

- Atualizações públicas de componentes

Atualizações públicas de componentes

A tabela a seguir lista os componentes AWS fornecidos que incluem recursos novos e atualizados.

> **Important**
>
> Quando você implanta um componente, AWS IoT Greengrass instala as versões mais recentes suportadas de todas as dependências desse componente. Por esse motivo, novas versões AWS de patch dos componentes públicos fornecidos podem ser implantadas automaticamente em seus dispositivos principais se você adicionar novos dispositivos a um grupo de coisas ou atualizar a implantação que visa esses dispositivos. Algumas atualizações automáticas, como a atualização do nucleus, podem fazer com que seus dispositivos sejam reiniciados inesperadamente.
> Para evitar atualizações não intencionais para um componente que está sendo executado em seu dispositivo, recomendamos que você inclua diretamente sua versão preferida desse componente ao criar uma implantação. Para obter mais informações sobre o comportamento de atualização AWS IoT Greengrass do software Core, consulteAtualize o software AWS IoT Greengrass principal (OTA).

Componente	Detalhes
Núcleo Greengrass	A versão 2.8.1 do núcleo Greengrass está disponível. Correções de erros e melhorias • Corrige um problema em que os códigos de erro de implantação não eram gerados corretamente a partir dos erros da API do Greengrass. • Corrige um problema em que as atualizações de status da frota enviam informações imprecisas quando um componente atinge um `ERRORED` estado durante uma implantação. • Corrige um problema em que as implantações não podiam ser concluídas quando o Greengrass tinha mais de 50 assinaturas existente s.

Versão: atualização do software AWS IoT Greengrass Core v2.8.0 em 7 de outubro de 2022

Esta versão fornece a versão 2.8.0 do componente Greengrass nucleus e a versão 1.1.0 do componente MQTT 5 broker (EMQX).

Data de lançamento: 7 de outubro de 2022

Destaques do lançamento

- Códigos de erro de implantação — O núcleo do Greengrass agora relata uma resposta do status de integridade da implantação que inclui códigos de erro detalhados quando a implantação de um componente não pode ser concluída. Para ter mais informações, consulte Códigos de erro de implantação detalhados.
- Status de erro do componente — O núcleo do Greengrass agora relata uma resposta do status de integridade do componente que inclui status de erro detalhados quando um componente entra no estado ou. `BROKEN` `ERRORED` Para ter mais informações, consulte Códigos detalhados de status do componente.

Detalhes do lançamento

- Atualizações públicas de componentes

Atualizações públicas de componentes

A tabela a seguir lista os componentes AWS fornecidos que incluem recursos novos e atualizados.

> **Important**
>
> Quando você implanta um componente, AWS IoT Greengrass instala as versões mais recentes suportadas de todas as dependências desse componente. Por esse motivo, novas versões AWS de patch dos componentes públicos fornecidos podem ser implantadas automaticamente em seus dispositivos principais se você adicionar novos dispositivos a um grupo de coisas ou atualizar a implantação que visa esses dispositivos. Algumas atualizações automáticas, como a atualização do nucleus, podem fazer com que seus dispositivos sejam reiniciados inesperadamente.
> Para evitar atualizações não intencionais para um componente que está sendo executado em seu dispositivo, recomendamos que você inclua diretamente sua versão preferida desse

componente ao criar uma implantação. Para obter mais informações sobre o comportamento de atualização AWS IoT Greengrass do software Core, consulteAtualize o software AWS IoT Greengrass principal (OTA).

Componente	Detalhes
Núcleo Greengrass	A versão 2.8.0 do núcleo Greengrass está disponível. Novos atributos • Atualiza o núcleo do Greengrass para relatar uma resposta do status de integridade da implantação que inclui códigos de erro detalhado s quando há um problema na implantação de componentes em um dispositivo principal. Para ter mais informações, consulte Códigos de erro de implantação detalhados. • Atualiza o núcleo do Greengrass para relatar uma resposta ao status de integridade do componente que inclui códigos de erro detalhados quando um componente entra no `BROKEN` estado ou. `ERRORED` Para ter mais informações, consulte Códigos detalhados de status do component e. • Expande os campos de mensagens de status para melhorar as informações de disponibilidade na nuvem para dispositivos. • Melhora a robustez do serviço de status da frota. Correções de erros e melhorias • Permite que um componente quebrado seja reinstalado quando sua configuração for alterada. • Corrige um problema em que a reinicialização do núcleo durante a implantação do bootstrap causa uma falha na implantação. • Corrige um problema no Windows em que a instalação falha quando um caminho raiz contém espaços. • Corrige um problema em que um componente desligado durante uma implantação usa o script de desligamento da nova versão. • Várias melhorias no desligamento. • Pequenas correções e melhorias adicionais.

Componente	Detalhes
Corretora MQTT 5 (EMQX)	A versão 1.1.0 do Corretora MQTT 5 (EMQX) componente está disponível. Novos atributos • Adiciona suporte às configurações do EMQX, incluindo opções de corretor e plug-ins. Correções de erros e melhorias • Atualiza o EMQX para a versão 4.4.9.

Versão: atualização do software AWS IoT Greengrass Core v2.7.0 em 28 de julho de 2022

Esta versão fornece a versão 2.7.0 do componente Greengrass nucleus, a versão 2.1.0 do componente stream manager e a versão 2.2.5 do componente Lambda manager.

Data do release: 28 de julho de 2022

Destaques do lançamento

- Métricas de telemetria do Stream Manager — O Stream Manager agora envia automaticamente métricas de telemetria para a Amazon EventBridge, para que você possa criar aplicativos na nuvem que monitoram e analisam o volume de dados que seus dispositivos principais carregam. Para ter mais informações, consulte Colete dados de telemetria de integridade do sistema a partir dos dispositivos principais AWS IoT Greengrass.
- Autoridade de certificação personalizada (CA) — Certificados de cliente assinados por uma CA de certificado personalizada, na qual a CA não está registradaAWS IoT, agora são suportados. Para ter mais informações, consulte Use um certificado de dispositivo assinado por uma CA privada.

Detalhes do lançamento

- Atualizações públicas de componentes

Atualizações públicas de componentes

A tabela a seguir lista os componentes AWS fornecidos que incluem recursos novos e atualizados.

> **Important**
>
> Quando você implanta um componente, AWS IoT Greengrass instala as versões mais recentes suportadas de todas as dependências desse componente. Por esse motivo, novas versões AWS de patch dos componentes públicos fornecidos podem ser implantadas automaticamente em seus dispositivos principais se você adicionar novos dispositivos a um grupo de coisas ou atualizar a implantação que visa esses dispositivos. Algumas atualizações automáticas, como a atualização do nucleus, podem fazer com que seus dispositivos sejam reiniciados inesperadamente.
> Para evitar atualizações não intencionais para um componente que está sendo executado em seu dispositivo, recomendamos que você inclua diretamente sua versão preferida desse componente ao criar uma implantação. Para obter mais informações sobre o comportamento de atualização AWS IoT Greengrass do software Core, consulteAtualize o software AWS IoT Greengrass principal (OTA).

Componente	Detalhes
Núcleo Greengrass	A versão 2.7.0 do núcleo Greengrass está disponível. Novos atributos • Atualiza o núcleo do Greengrass para enviar atualizações de status para a AWS IoT Greengrass nuvem quando o dispositivo principal aplica uma implantação local. • Adiciona suporte para certificados de cliente assinados por uma autoridade de certificação (CA) personalizada, na qual a CA não está registradaAWS IoT. Para usar esse recurso, você pode definir a nova opção `greengrassDataPlaneEndpoint` de configuração como`iotdata`. Para ter mais informações, consulte Use um certificado de dispositivo assinado por uma CA privada. Correções de erros e melhorias • Corrige um problema em que o núcleo do Greengrass reverte uma implantação em determinados cenários quando o núcleo é interromp ido ou reiniciado. O núcleo agora retoma a implantação após a reinicial ização do núcleo.

Componente	Detalhes
	• Atualiza o instalador do Greengrass para respeitar o `--start` argumento quando você especifica a configuração do software como um serviço do sistema. • Atualiza o comportamento de SubscribeToComponentUpdates definir o ID de implantação em eventos em que o núcleo atualizou um componente. • Pequenas correções e melhorias adicionais.
Gerenciador de streams	A versão 2.1.0 do componente gerenciador de fluxo está disponível. Novos atributos • Atualiza esse componente para enviar automaticamente métricas de telemetria para a Amazon. EventBridge Para ter mais informações, consulte Colete dados de telemetria de integridade do sistema a partir dos dispositivos principais AWS IoT Greengrass. Esse recurso requer a versão 2.7.0 ou posterior do componente do núcleo do Greengrass. • Versão atualizada para a versão 2.7.0 do Greengrass nucleus.
Gerente do Lambda	A versão 2.2.5 do componente Lambda manager está disponível. Novos atributos • Adiciona suporte para curingas de tópicos do MQTT em fontes de eventos nas quais você assina mensagens locais de publicação/assinatura. Esse recurso requer a versão v2.6.0 ou posterior do componente do núcleo do Greengrass. • Versão atualizada para a versão 2.7.0 do Greengrass nucleus.

Versão: atualização do software AWS IoT Greengrass Core v2.6.0 em 27 de junho de 2022

Esta versão fornece a versão 2.6.0 do componente nucleus do Greengrass, novos componentes fornecidos e AWS atualizações dos componentes fornecidos. AWS

Data de lançamento: 27 de junho de 2022

Destaques do lançamento

- Curingas em tópicos de publicação/assinatura locais — Agora você pode usar curingas do MQTT ao assinar tópicos de publicação/assinatura locais. Para obter mais informações, consulte SubscribeToTopic e Publique/assine mensagens locais.
- Suporte à sombra do dispositivo cliente — Agora você pode interagir com as sombras do dispositivo cliente em componentes personalizados e sincronizar as sombras do dispositivo cliente com. AWS IoT Core Para ter mais informações, consulte Interaja e sincronize as sombras do dispositivo cliente.
- Suporte local ao MQTT 5 para dispositivos clientes — Agora você pode implantar o broker EMQX MQTT 5 para usar os recursos do MQTT 5 na comunicação entre dispositivos cliente e um dispositivo principal. Para obter mais informações, consulte Conecte dispositivos cliente aos dispositivos principais e Corretora MQTT 5 (EMQX).
- Variáveis de receita em configurações de componentes — Agora você pode usar variáveis de receita específicas em configurações de componentes. Você pode usar essas variáveis de receita ao definir a configuração padrão de um componente em uma receita ou ao configurar um componente em uma implantação. Para obter mais informações, consulte Use variáveis de receita em atualizações de mesclagem e Variáveis da receita.
- Curingas nas políticas de autorização de IPC — Agora você pode usar o * curinga para corresponder a qualquer combinação de caracteres nas políticas de autorização de comunicação entre processos (IPC). Esse curinga permite que você permita o acesso a vários recursos em uma única política de autorização. Para ter mais informações, consulte Caracteres curingas nas políticas de autorização.
- Operações de IPC que gerenciam implantações e componentes locais — Agora você pode desenvolver componentes personalizados que gerenciam implantações locais e visualizam detalhes dos componentes. Para obter mais informações, consulte IPC: gerenciar implantações e componentes locais.

- Operações IPC que autenticam e autorizam dispositivos clientes — Agora você pode usar essas operações para criar um componente de agente local personalizado. Para obter mais informações, consulte IPC: autenticar e autorizar dispositivos cliente.

Detalhes do lançamento

- Atualizações públicas de componentes

Atualizações públicas de componentes

A tabela a seguir lista os componentes AWS fornecidos que incluem recursos novos e atualizados.

> **Important**
>
> Quando você implanta um componente, AWS IoT Greengrass instala as versões mais recentes suportadas de todas as dependências desse componente. Por esse motivo, novas versões AWS de patch dos componentes públicos fornecidos podem ser implantadas automaticamente em seus dispositivos principais se você adicionar novos dispositivos a um grupo de coisas ou atualizar a implantação que visa esses dispositivos. Algumas atualizações automáticas, como a atualização do nucleus, podem fazer com que seus dispositivos reiniciem inesperadamente.
> Para evitar atualizações não intencionais para um componente que está sendo executado em seu dispositivo, recomendamos que você inclua diretamente sua versão preferida desse componente ao criar uma implantação. Para obter mais informações sobre o comportamento de atualização AWS IoT Greengrass do software Core, consulteAtualize o software AWS IoT Greengrass principal (OTA).

Componente	Detalhes
Núcleo Greengrass	A versão 2.6.0 do núcleo Greengrass está disponível. Novos atributos • Adiciona suporte para curingas MQTT quando você assina tópicos locais de publicação/assinatura. Para obter mais informações, consulte SubscribeToTopic e Publique/assine mensagens locais.

Componente	Detalhes
	• Adiciona suporte para variáveis de receita em configurações de componentes, além da variável de `component_dependency_name :configuration: json_pointer` receita. Você pode usar essas variáveis de receitas ao definir um componente `DefaultConfiguration` em uma receita ou ao configurar um componente em uma implantação. Para ativar esse recurso, defina a opção interpolateComponentConfigurationde configuração como`true`. Para obter mais informações, consulte Use variáveis de receita em atualizações de mesclagem e Variáveis da receita. • Adiciona suporte total ao caractere * curinga nas políticas de autorização de comunicação entre processos (IPC). Agora você pode especificar o * caractere em uma string de recursos para corresponder a qualquer combinação de caracteres. Para ter mais informações, consulte Caracteres curingas nas políticas de autorização. • Adiciona suporte para componentes personalizados para chamar operações de IPC que a CLI do Greengrass usa. Você pode usar essas operações de IPC para gerenciar implantações locais, visualizar detalhes do componente e gerar uma senha que pode ser usada para entrar no console de depuração local. Para obter mais informações, consulte IPC: gerenciar implantações e componentes locais. Correções de erros e melhorias • Corrige um problema em que componentes dependentes não reagiam quando suas dependências rígidas reiniciavam ou mudavam de estado em determinados cenários. • Melhora as mensagens de erro que o dispositivo principal reporta ao serviço de AWS IoT Greengrass nuvem quando uma implantação falha. • Corrige um problema em que o núcleo do Greengrass aplicava a implantação de uma coisa duas vezes em determinados cenários quando o núcleo era reiniciado. • Pequenas correções e melhorias adicionais. Para obter mais informações, consulte os lançamentos em GitHub.

Componente	Detalhes
Corretora MQTT 5 (EMQX)	A versão 1.0.0 do novo componente broker EMQX MQTT 5 está disponível. Novos atributos • Adiciona suporte para o broker EMQX MQTT 5 local. Os dispositivos clientes podem se conectar a esse broker MQTT para se comunicar com um dispositivo principal usando os recursos do MQTT 5.
Gerenciador de sombras	A versão 2.2.0 do componente shadow manager está disponível. Novos atributos • Adiciona suporte para o serviço paralelo local na interface local de publicação/assinatura. Agora você pode se comunicar com o agente de mensagens local de publicação/assinatura sobre tópicos paralelos do MQTT para obter, atualizar e excluir sombras no dispositivo principal. Esse recurso permite conectar dispositivos cliente ao serviço paralelo local usando a ponte MQTT para retransmitir mensagens sobre tópicos paralelos entre dispositivos cliente e a interface local de publicação/assinatura. Esse recurso requer a versão v2.6.0 ou posterior do componente do núcleo do Greengrass. Para conectar dispositivos cliente ao serviço paralelo local, você também deve usar a versão 2.2.0 ou posterior do componente de ponte MQTT. • Adiciona a `direction` opção que você pode configurar para personalizar a direção para sincronizar sombras entre o serviço de sombra local e o. Nuvem AWS Você pode configurar essa opção para reduzir a largura de banda e as conexões com o. Nuvem AWS

Componente	Detalhes
Autenticação do dispositivo cliente	A versão 2.2.0 do componente de autenticação do dispositivo cliente está disponível. Novos atributos • Adiciona suporte a componentes personalizados para chamar operações de comunicação entre processos (IPC) para autenticar e autorizar dispositivos clientes. Você pode usar essas operações em um componente personalizado do broker MQTT, por exemplo. Para obter mais informações, consulte IPC: autenticar e autorizar dispositivos cliente. • Adiciona as `threadPoolSize` opções `maxActiveAuthTokens` `cloudQueueSize` , e que você pode configurar para ajustar o desempenho desse componente.
Ponte MQTT	A versão 2.2.0 do componente de ponte MQTT está disponível. Novos atributos • Adiciona suporte para curingas de tópicos do MQTT (#e+) quando você especifica publicação/assinatura local como agente de mensagens de origem. Esse recurso requer a versão v2.6.0 ou posterior do componente do núcleo do Greengrass. • Adiciona a `targetTopicPrefix` opção, que você pode especificar para configurar a ponte MQTT para adicionar um prefixo ao tópico de destino ao retransmitir uma mensagem.

Componente	Detalhes
CLI do Greengrass	A versão 2.6.0 da CLI do Greengrass está disponível. Novos atributos • Adiciona suporte para componentes personalizados para chamar operações de comunicação entre processos (IPC) que a CLI do Greengrass usa. Você pode usar essas operações de IPC para gerenciar implantações locais, visualizar detalhes do componente e gerar uma senha que pode ser usada para entrar no console de depuração local. Para obter mais informações, consulte IPC: gerenciar implantações e componentes locais. Correções de erros e melhorias • Pequenas correções e melhorias adicionais.

Versão: atualização do software AWS IoT Greengrass Core v2.5.6 em 31 de maio de 2022

Esta versão fornece a versão 2.5.6 do componente Greengrass nucleus e a versão 2.2.4 do componente gerenciador de registros.

Data de lançamento: 31 de maio de 2022

Detalhes do lançamento

- Atualizações públicas de componentes

Atualizações públicas de componentes

A tabela a seguir lista os componentes AWS fornecidos que incluem recursos novos e atualizados.

> **Important**
>
> Quando você implanta um componente, AWS IoT Greengrass instala as versões mais recentes suportadas de todas as dependências desse componente. Por esse motivo, novas versões AWS de patch dos componentes públicos fornecidos podem ser implantadas automaticamente em seus dispositivos principais se você adicionar novos dispositivos

a um grupo de coisas ou atualizar a implantação que visa esses dispositivos. Algumas atualizações automáticas, como a atualização do nucleus, podem fazer com que seus dispositivos reiniciem inesperadamente.
Para evitar atualizações não intencionais para um componente que está sendo executado em seu dispositivo, recomendamos que você inclua diretamente sua versão preferida desse componente ao criar uma implantação. Para obter mais informações sobre o comportamento de atualização AWS IoT Greengrass do software Core, consulteAtualize o software AWS IoT Greengrass principal (OTA).

Componente	Detalhes
Núcleo Greengrass	A versão 2.5.6 do núcleo Greengrass está disponível. Novos atributos • Adiciona suporte para módulos de segurança de hardware que usam chaves ECC. Você pode usar um módulo de segurança de hardware (HSM) para armazenar com segurança a chave privada e o certifica do do dispositivo. Para ter mais informações, consulte Integração de segurança de hardware. Correções de erros e melhorias • Corrige um problema em que a implantação nunca é concluída quando você implanta um componente com um script de instalação incorreto em determinados cenários. • Melhora o desempenho durante a inicialização. • Pequenas correções e melhorias adicionais.
Gerenciador de registros	A versão 2.2.4 do componente gerenciador de registros está disponível. Correções de erros e melhorias • Melhora a estabilidade ao lidar com configurações inválidas. • Pequenas correções e melhorias adicionais.

Versão: atualização do software AWS IoT Greengrass Core v2.5.5 em 6 de abril de 2022

Esta versão fornece a versão 2.5.5 do componente do núcleo do Greengrass.

Data de lançamento: 6 de abril de 2022

Detalhes do lançamento

- Atualizações públicas de componentes

Atualizações públicas de componentes

A tabela a seguir lista os componentes AWS fornecidos que incluem recursos novos e atualizados.

> **Important**
>
> Quando você implanta um componente, AWS IoT Greengrass instala as versões mais recentes suportadas de todas as dependências desse componente. Por esse motivo, novas versões AWS de patch dos componentes públicos fornecidos podem ser implantadas automaticamente em seus dispositivos principais se você adicionar novos dispositivos a um grupo de coisas ou atualizar a implantação que visa esses dispositivos. Algumas atualizações automáticas, como a atualização do nucleus, podem fazer com que seus dispositivos sejam reiniciados inesperadamente.
> Para evitar atualizações não intencionais para um componente que está sendo executado em seu dispositivo, recomendamos que você inclua diretamente sua versão preferida desse componente ao criar uma implantação. Para obter mais informações sobre o comportamento de atualização AWS IoT Greengrass do software Core, consulteAtualize o software AWS IoT Greengrass principal (OTA).

Componente	Detalhes
Núcleo Greengrass	A versão 2.5.5 do núcleo Greengrass está disponível.

Componente	Detalhes
	Novos atributos • Adiciona a variável de `GG_ROOT_CA_PATH` ambiente para componentes, para que você possa acessar o certificado de autoridade de certificação (CA) raiz em componentes personalizados. Correções de erros e melhorias • Adiciona suporte para dispositivos Windows que usam um idioma de exibição diferente do inglês. • Atualiza como o núcleo do Greengrass analisa os argumentos do instalador booleano, para que você possa especificar um argumento booleano sem um valor booleano para especificar um valor. `true` Por exemplo, agora você pode especificar, `--provision` em vez de `--provision true` instalar, com o provisionamento automático de recursos. • Corrige um problema em que o dispositivo principal não reportava seu status ao serviço de AWS IoT Greengrass nuvem após o provisionamento em determinados cenários. • Pequenas correções e melhorias adicionais.

Versão: atualização do software AWS IoT Greengrass Core v2.5.4 em 23 de março de 2022

Esta versão fornece a versão 2.5.4 do componente Greengrass nucleus e a versão 2.0.10 do componente Lambda launcher.

Data de lançamento: 23 de março de 2022

Detalhes do lançamento

- Atualizações públicas de componentes

Atualizações públicas de componentes

A tabela a seguir lista os componentes AWS fornecidos que incluem recursos novos e atualizados.

> **Important**
>
> Quando você implanta um componente, AWS IoT Greengrass instala as versões mais recentes suportadas de todas as dependências desse componente. Por esse motivo, novas versões AWS de patch dos componentes públicos fornecidos podem ser implantadas automaticamente em seus dispositivos principais se você adicionar novos dispositivos a um grupo de coisas ou atualizar a implantação que visa esses dispositivos. Algumas atualizações automáticas, como a atualização do nucleus, podem fazer com que seus dispositivos reiniciem inesperadamente.
>
> Para evitar atualizações não intencionais para um componente que está sendo executado em seu dispositivo, recomendamos que você inclua diretamente sua versão preferida desse componente ao criar uma implantação. Para obter mais informações sobre o comportamento de atualização AWS IoT Greengrass do software Core, consulteAtualize o software AWS IoT Greengrass principal (OTA).

Componente	Detalhes
Núcleo Greengrass	A versão 2.5.4 do núcleo Greengrass está disponível. Correções de erros e melhorias • Melhorias e correções de erros gerais.
Lançador Lambda	A versão 2.0.10 do componente Lambda launcher está disponível. Correções de erros e melhorias • Melhorias e correções de erros gerais.

Versão: atualização do software AWS IoT Greengrass Core v2.5.3 em 6 de janeiro de 2022

Esta versão fornece a versão 2.5.3 do componente de núcleo do Greengrass e o novo componente provedor PKCS #11.

Data de lançamento: 6 de janeiro de 2022

Destaques do lançamento

- Integração de segurança de hardware — agora você pode configurar o software AWS IoT Greengrass Core para usar uma chave privada e um certificado que você armazena com segurança em um módulo de segurança de hardware (HSM). Para ter mais informações, consulte Integração de segurança de hardware.

Detalhes do lançamento

- Atualizações públicas de componentes

Atualizações públicas de componentes

A tabela a seguir lista os componentes AWS fornecidos que incluem recursos novos e atualizados.

> **Important**
>
> Quando você implanta um componente, AWS IoT Greengrass instala as versões mais recentes suportadas de todas as dependências desse componente. Por esse motivo, novas versões AWS de patch dos componentes públicos fornecidos podem ser implantadas automaticamente em seus dispositivos principais se você adicionar novos dispositivos a um grupo de coisas ou atualizar a implantação que visa esses dispositivos. Algumas atualizações automáticas, como a atualização do nucleus, podem fazer com que seus dispositivos sejam reiniciados inesperadamente.
>
> Para evitar atualizações não intencionais para um componente que está sendo executado em seu dispositivo, recomendamos que você inclua diretamente sua versão preferida desse componente ao criar uma implantação. Para obter mais informações sobre o comportamento de atualização AWS IoT Greengrass do software Core, consulteAtualize o software AWS IoT Greengrass principal (OTA).

Componente	Detalhes
Núcleo Greengrass	A versão 2.5.3 do núcleo Greengrass está disponível.

Componente	Detalhes
	Novos atributos • Adiciona suporte para integração de segurança de hardware. Você pode usar um módulo de segurança de hardware (HSM) para armazenar com segurança a chave privada e o certificado do dispositivo. Para ter mais informações, consulte Integração de segurança de hardware. Correções de erros e melhorias • Corrige um problema com exceções de tempo de execução enquanto o núcleo estabelece conexões MQTT com o. AWS IoT Core
Fornecedor PKCS #11	A versão 2.0.0 do componente provedor PKCS #11 está disponível. Novos atributos • Adiciona suporte para integração de segurança de hardware. Você pode usar um módulo de segurança de hardware (HSM) para armazenar com segurança a chave privada e o certificado do dispositivo. Para ter mais informações, consulte Integração de segurança de hardware.

Versão: atualização do software AWS IoT Greengrass Core v2.5.2 em 3 de dezembro de 2021

Esta versão fornece a versão 2.5.2 do componente do núcleo Greengrass.

Data de lançamento: 3 de dezembro de 2021

Detalhes do lançamento

- Atualizações públicas de componentes

Atualizações públicas de componentes

A tabela a seguir lista os componentes AWS fornecidos que incluem recursos novos e atualizados.

Important

Quando você implanta um componente, AWS IoT Greengrass instala as versões mais recentes suportadas de todas as dependências desse componente. Por esse motivo, novas versões AWS de patch dos componentes públicos fornecidos podem ser implantadas automaticamente em seus dispositivos principais se você adicionar novos dispositivos a um grupo de coisas ou atualizar a implantação que visa esses dispositivos. Algumas atualizações automáticas, como a atualização do nucleus, podem fazer com que seus dispositivos sejam reiniciados inesperadamente.
Para evitar atualizações não intencionais para um componente que está sendo executado em seu dispositivo, recomendamos que você inclua diretamente sua versão preferida desse componente ao criar uma implantação. Para obter mais informações sobre o comportamento de atualização AWS IoT Greengrass do software Core, consulteAtualize o software AWS IoT Greengrass principal (OTA).

Componente	Detalhes
Núcleo Greengrass	A versão 2.5.2 do núcleo Greengrass está disponível. Correções de erros e melhorias • Corrige um problema em que, após as atualizações do núcleo do Greengrass, o serviço Windows não é iniciado novamente depois que você o interrompe ou reinicializa o dispositivo.
AWS IoT Device Defender	A versão 3.0.1 do AWS IoT Device Defendercomponente está disponível. Essa versão do AWS IoT Device Defender componente espera parâmetro s de configuração diferentes da versão 2.x. Se você usar uma configura ção não padrão para a versão 2.x e quiser atualizar da v2.x para a v3.x, deverá atualizar a configuração do componente. Para obter mais informaçõ es, consulte configuração do AWS IoT Device Defender componente. Novos atributos • Adiciona suporte para dispositivos principais que executam o Windows.

Componente	Detalhes
	• Altera o tipo de componente de componente Lambda para componente genérico. Agora, esse componente não depende mais do componente antigo do roteador de assinatura para criar assinaturas. • Adiciona o novo parâmetro `UseInstaller` de configuração que permite desativar opcionalmente o script de instalação que instala as dependências dos componentes.

Versão: atualização do software AWS IoT Greengrass Core v2.5.1 em 23 de novembro de 2021

Esta versão fornece a versão 2.5.1 do componente do núcleo do Greengrass.

Data da versão: 23 de novembro de 2021

Detalhes do lançamento

- Atualizações públicas de componentes

Atualizações públicas de componentes

A tabela a seguir lista os componentes AWS fornecidos que incluem recursos novos e atualizados.

Important

Quando você implanta um componente, AWS IoT Greengrass instala as versões mais recentes suportadas de todas as dependências desse componente. Por esse motivo, novas versões AWS de patch dos componentes públicos fornecidos podem ser implantadas automaticamente em seus dispositivos principais se você adicionar novos dispositivos a um grupo de coisas ou atualizar a implantação que visa esses dispositivos. Algumas atualizações automáticas, como a atualização do nucleus, podem fazer com que seus dispositivos sejam reiniciados inesperadamente.
Para evitar atualizações não intencionais para um componente que está sendo executado em seu dispositivo, recomendamos que você inclua diretamente sua versão preferida desse componente ao criar uma implantação. Para obter mais informações sobre o comportamento

de atualização AWS IoT Greengrass do software Core, consulteAtualize o software AWS IoT Greengrass principal (OTA).

Componente	Detalhes
Núcleo Greengrass	A versão 2.5.1 do núcleo Greengrass está disponível. Correções de erros e melhorias • Adiciona suporte para versões de 32 bits do Java Runtime Environment (JRE) no Windows. • Altera o comportamento de remoção de grupos de coisas para dispositivos principais cuja AWS IoT política não concede a `greengrass:ListThingGroupsForCoreDevice` permissão. Com essa versão, a implantação continua, registra um aviso e não remove componentes quando você remove o dispositivo principal de um grupo de coisas. Para ter mais informações, consulte Implemente AWS IoT Greengrass componentes em dispositivos. • Corrige um problema com as variáveis de ambiente do sistema que o núcleo do Greengrass disponibiliza para os processos de componentes do Greengrass. Agora você pode reiniciar um componente para que ele use as variáveis de ambiente do sistema mais recentes.

Versão: atualização do software AWS IoT Greengrass Core v2.5.0 em 12 de novembro de 2021

Esta versão fornece a versão 2.5.0 do componente nucleus do Greengrass, AWS novos componentes fornecidos e atualizações para os componentes fornecidos. AWS

Data de lançamento: 12 de novembro de 2021

Destaques do lançamento

- Suporte a dispositivos Windows — Agora você pode executar o software AWS IoT Greengrass Core em dispositivos que executam sistemas operacionais Windows. Para obter mais informações,

consulte Compatibilidade de recursos do Greengrass por sistema operacional e Plataformas compatíveis e requisitos.

- Novo comportamento de remoção de grupos de coisas — agora você pode remover um dispositivo principal de um grupo de coisas para remover os componentes desse grupo de coisas na próxima implantação desse dispositivo.

  > **Important**
  >
  > Como resultado dessa alteração, a AWS IoT política de um dispositivo principal deve ter a `greengrass:ListThingGroupsForCoreDevice` permissão. Se você usou o instalador do software AWS IoT Greengrass Core para provisionar recursos, a AWS IoT política padrão permite`greengrass:*`, o que inclui essa permissão. Para ter mais informações, consulte Autorização e autenticação do dispositivo para o AWS IoT Greengrass.

- Suporte de segurança de hardware — Agora você pode configurar o software AWS IoT Greengrass Core para usar um módulo de segurança de hardware (HSM), para poder armazenar com segurança a chave privada e o certificado do dispositivo. Para ter mais informações, consulte Integração de segurança de hardware.
- Suporte a proxy HTTPS — Agora você pode configurar o software AWS IoT Greengrass Core para se conectar por meio de proxies HTTPS. Para ter mais informações, consulte Conectar-se à porta 443 ou por meio de um proxy de rede.

Detalhes do lançamento

- Atualizações de suporte da plataforma
- Atualizações públicas de componentes

Atualizações de suporte da plataforma

Plataforma	Detalhes
Windows	AWS IoT Greengrassagora suporta a execução do software AWS IoT Greengrass Core nas seguintes versões do Windows: • Windows 10 • Windows Server 2019

Plataforma	Detalhes
	Para obter mais informações, consulte Compatibilidade de recursos do Greengrass por sistema operacional e Plataformas compatíveis e requisitos.

Atualizações públicas de componentes

A tabela a seguir lista os componentes AWS fornecidos que incluem recursos novos e atualizados.

> **Important**
>
> Quando você implanta um componente, AWS IoT Greengrass instala as versões mais recentes suportadas de todas as dependências desse componente. Por esse motivo, novas versões AWS de patch dos componentes públicos fornecidos podem ser implantadas automaticamente em seus dispositivos principais se você adicionar novos dispositivos a um grupo de coisas ou atualizar a implantação que visa esses dispositivos. Algumas atualizações automáticas, como a atualização do nucleus, podem fazer com que seus dispositivos sejam reiniciados inesperadamente.
> Para evitar atualizações não intencionais para um componente que está sendo executado em seu dispositivo, recomendamos que você inclua diretamente sua versão preferida desse componente ao criar uma implantação. Para obter mais informações sobre o comportamento de atualização AWS IoT Greengrass do software Core, consulteAtualize o software AWS IoT Greengrass principal (OTA).

Componente	Detalhes
Núcleo Greengrass	A versão 2.5.0 do núcleo Greengrass está disponível. Novos atributos • Adiciona suporte para dispositivos principais que executam o Windows. • Mude o comportamento da remoção de grupos de coisas. Com essa versão, você pode remover um dispositivo principal de um grupo de coisas para desinstalar os componentes desse grupo de coisas na próxima implantação.

Componente	Detalhes
	Como resultado dessa alteração, a AWS IoT política de um dispositivo principal deve ter a `greengrass:ListThingGroupsForCoreDevice` permissão. Se você usou o instalador do software AWS IoT Greengrass Core para provisionar recursos, a AWS IoT política padrão permite `greengrass:*`, o que inclui essa permissão. Para ter mais informações, consulte Autorização e autenticação do dispositivo para o AWS IoT Greengrass. • Adiciona suporte para configurações de proxy HTTPS. Para ter mais informações, consulte Conectar-se à porta 443 ou por meio de um proxy de rede. • Adiciona o novo parâmetro `windowsUser` de configuração. Você pode usar esse parâmetro para especificar o usuário padrão a ser usado para executar componentes em um dispositivo principal do Windows. Para ter mais informações, consulte Configurar o usuário que executa os componentes. • Adiciona as novas opções de `httpClient` configuração que você pode usar para personalizar os tempos limite de solicitação HTTP para melhorar o desempenho em redes lentas. Para obter mais informações, consulte o parâmetro de configuração HttpClient. Correções de erros e melhorias • Corrige a opção de ciclo de vida do bootstrap para reiniciar o dispositivo principal a partir de um componente. • Adiciona suporte para hífens nas variáveis da receita. • Corrige a autorização de IPC para componentes da função Lambda sob demanda. • Melhora as mensagens de registro e altera os registros não críticos de `INFO` um para `DEBUG` outro, então os registros são mais úteis. • Remove a `iot:DescribeCertificate` permissão da função padrão de troca de tokens que o núcleo do Greengrass cria quando você instala o software AWS IoT Greengrass Core com provisionamento automático. Essa permissão não é usada pelo núcleo Greengrass.

Componente	Detalhes
	• Corrige um problema para que o script de provisionamento automático não exija a `iam:GetPolicy` permissão se `iam:CreatePolicy` estiver disponível para a mesma política. • Pequenas correções e melhorias adicionais.
CLI do Greengrass	A versão 2.5.0 da CLI do Greengrass está disponível. Novos atributos • Adiciona suporte para dispositivos principais que executam o Windows. • Adiciona o novo parâmetro de `AuthorizedWindowsGroups` configuração que você pode especificar para autorizar grupos do sistema a usar a CLI do Greengrass em dispositivos Windows. • Adiciona o `windowsUser` parâmetro para implantações locais. Você pode usar esse parâmetro para especificar o usuário a ser usado para executar componentes em um dispositivo principal do Windows.

Componente	Detalhes
CloudWatch métricas	A versão 3.0.0 do componente de CloudWatchmétricas está disponível. Essa versão do componente de CloudWatch métricas espera parâmetro s de configuração diferentes da versão 2.x. Se você usar uma configura ção não padrão para a versão 2.x e quiser atualizar da v2.x para a v3.x, deverá atualizar a configuração do componente. Para obter mais informaçõ es, consulte a configuração do componente de CloudWatch métricas. Novos atributos • Adiciona suporte para dispositivos principais que executam o Windows. • Altera o tipo de componente de componente Lambda para componente genérico. Agora, esse componente não depende mais do componente antigo do roteador de assinatura para criar assinaturas. • Adiciona um novo parâmetro de `InputTopic` configuração para especificar o tópico no qual o componente se inscreve para receber mensagens. • Adiciona um novo parâmetro de `OutputTopic` configuração para especificar o tópico no qual o componente publica respostas de status. • Adiciona um novo parâmetro de `PubSubToIoTCore` configuração para especificar se deseja publicar e assinar tópicos do AWS IoT Core MQTT. • Adiciona o novo parâmetro `UseInstaller` de configuração que permite desativar opcionalmente o script de instalação que instala as dependências dos componentes. Correções de erros e melhorias Adiciona suporte para registros de data e hora duplicados nos dados de entrada.

Componente	Detalhes
Gerente do Lambda	A versão 2.2.0 do componente Lambda Manager está disponível. Correções de erros e melhorias • Corrige um problema em que as funções Lambda não conseguiam gravar registros após uma reinicialização. • Corrige um problema em que o roteador de assinatura antigo envia mensagens duplicadas quando há curingas no tópico. • Corrige um problema em que funções Lambda não fixadas não podiam usar a biblioteca de comunicação entre processos (IPC) do Greengrass no. AWS IoT Device SDK

Versão: atualização do software AWS IoT Greengrass Core v2.4.0 em 3 de agosto de 2021

Esta versão fornece a versão 2.4.0 do componente nucleus do Greengrass, AWS novos componentes fornecidos e atualizações para os componentes fornecidos. AWS

Data de lançamento: 3 de agosto de 2021

Destaques do lançamento

- Limites de recursos do sistema — O componente do núcleo do Greengrass agora suporta os limites de recursos do sistema. Você pode configurar a quantidade máxima de uso de CPU e RAM que os processos de cada componente podem usar no dispositivo principal. Para ter mais informações, consulte Configurar limites de recursos do sistema para componentes.
- Componentes de pausa/retomada — O núcleo do Greengrass agora suporta componentes de pausa e retomada. Você pode usar a biblioteca de comunicação entre processos (IPC) para desenvolver componentes personalizados que pausam e retomam os processos de outros componentes. Para obter mais informações, consulte ResumeComponent e PauseComponent.
- Instalação com provisionamento de AWS IoT frota — use o novo plug-in de provisionamento de AWS IoT frota para instalar o software AWS IoT Greengrass Core em dispositivos que se conectam para provisionar AWS IoT os recursos necessários. AWS Os dispositivos usam um certificado de solicitação para provisionar. Você pode incorporar o certificado de solicitação nos dispositivos durante a fabricação, para que cada dispositivo possa provisionar assim que estiver

on-line. Para ter mais informações, consulte Instale o software AWS IoT Greengrass principal com provisionamento de AWS IoT frota.

- Instalar com provisionamento personalizado — Desenvolva um plug-in de provisionamento personalizado para provisionar AWS os recursos necessários ao instalar o AWS IoT Greengrass software Core nos dispositivos. Você pode criar um aplicativo Java que é executado durante a instalação para configurar os dispositivos principais do Greengrass para seu caso de uso personalizado. Para ter mais informações, consulte Instale o software AWS IoT Greengrass principal com provisionamento personalizado de recursos.

Detalhes do lançamento

- Atualizações públicas de componentes

Atualizações públicas de componentes

A tabela a seguir lista os componentes AWS fornecidos que incluem recursos novos e atualizados.

> **Important**
>
> Quando você implanta um componente, AWS IoT Greengrass instala as versões mais recentes suportadas de todas as dependências desse componente. Por esse motivo, novas versões AWS de patch dos componentes públicos fornecidos podem ser implantadas automaticamente em seus dispositivos principais se você adicionar novos dispositivos a um grupo de coisas ou atualizar a implantação que visa esses dispositivos. Algumas atualizações automáticas, como a atualização do nucleus, podem fazer com que seus dispositivos sejam reiniciados inesperadamente.
>
> Para evitar atualizações não intencionais para um componente que está sendo executado em seu dispositivo, recomendamos que você inclua diretamente sua versão preferida desse componente ao criar uma implantação. Para obter mais informações sobre o comportamento de atualização AWS IoT Greengrass do software Core, consulteAtualize o software AWS IoT Greengrass principal (OTA).

Componente	Detalhes
Núcleo Greengrass	A versão 2.4.0 do núcleo Greengrass está disponível.

Componente	Detalhes
	Novos atributos • Adiciona suporte aos limites de recursos do sistema. Você pode configurar a quantidade máxima de uso de CPU e RAM que os processos de cada componente podem usar no dispositivo principal. Para ter mais informações, consulte Configurar limites de recursos do sistema para componentes. • Adiciona operações de IPC para pausar e retomar componentes. Para obter mais informações, consulte ResumeComponent e PauseComponent. • Adiciona suporte para plug-ins de provisionamento. Você pode especificar um arquivo JAR a ser executado durante a instalação para provisionar AWS os recursos necessários para um dispositivo principal do Greengrass. O núcleo do Greengrass inclui uma interface que você pode implementar para desenvolver plug-ins de provisionamento personalizados. Para ter mais informações, consulte Instale o software AWS IoT Greengrass principal com provisionamento personalizado de recursos. • Adiciona o `thing-name-policy` argumento opcional ao instalador do software AWS IoT Greengrass Core. Você pode usar essa opção para especificar uma AWS IoT política existente ou personalizada ao instalar o software AWS IoT Greengrass Core com provisionamento automático de recursos. Correções de erros e melhorias • Atualiza a configuração de registro na inicialização. Isso corrige um problema em que a configuração de registro não foi aplicada na inicialização. • Atualiza o link simbólico do carregador de núcleo para apontar para o armazenamento de componentes na pasta raiz do Greengrass durante a instalação. Essa atualização permite que você exclua o arquivo JAR e outros artefatos do núcleo que você baixa ao instalar o software AWS IoT Greengrass Core. • Pequenas correções e melhorias adicionais. Para obter mais informações, consulte os lançamentos em GitHub.

Componente	Detalhes
CLI do Greengrass	A versão 2.4.0 da CLI do Greengrass está disponível. Novos atributos • Adiciona suporte aos limites de recursos do sistema. Ao criar uma implantação local, você pode configurar a quantidade máxima de uso de CPU e RAM que os processos de cada componente podem usar no dispositivo principal. Para obter mais informações, consulte Configurar limites de recursos do sistema para componentes e o comando deployment create.
AWS IoTaprovisionamento de frota por reclamação	O provisionamento de AWS IoT frota por meio do plug-in de reclamação já está disponível. Para ter mais informações, consulte Instale o software AWS IoT Greengrass principal com provisionamento de AWS IoT frota. Novos atributos • Adiciona suporte para instalar o software AWS IoT Greengrass Core com provisionamento de AWS IoT frota. Durante a instalação, os dispositivos se conectam AWS IoT para provisionar AWS os recursos necessários e baixar certificados de dispositivos para uso em operações regulares.

Versão: atualização do software AWS IoT Greengrass Core v2.3.0 em 29 de junho de 2021

Esta versão fornece a versão 2.3.0 do componente do núcleo do Greengrass.

Data de lançamento: 29 de junho de 2021

Destaques do lançamento

- Suporte a grandes configurações — o componente nucleus do Greengrass agora oferece suporte a documentos de implantação de até 10 MB. Agora você pode implantar atualizações maiores de configuração nos componentes do Greengrass.

> **Note**
>
> Para usar esse recurso, a AWS IoT política de um dispositivo principal deve permitir a `greengrass:GetDeploymentConfiguration` permissão. Se você usou o instalador do software AWS IoT Greengrass Core para provisionar recursos, a AWS IoT política do seu dispositivo principal permite`greengrass:*`, o que inclui essa permissão. Para ter mais informações, consulte Autorização e autenticação do dispositivo para o AWS IoT Greengrass.

Detalhes do lançamento

- Atualizações públicas de componentes

Atualizações públicas de componentes

A tabela a seguir lista os componentes AWS fornecidos que incluem recursos novos e atualizados.

> **Important**
>
> Quando você implanta um componente, AWS IoT Greengrass instala as versões mais recentes suportadas de todas as dependências desse componente. Por esse motivo, novas versões AWS de patch dos componentes públicos fornecidos podem ser implantadas automaticamente em seus dispositivos principais se você adicionar novos dispositivos a um grupo de coisas ou atualizar a implantação que visa esses dispositivos. Algumas atualizações automáticas, como a atualização do nucleus, podem fazer com que seus dispositivos sejam reiniciados inesperadamente.
>
> Para evitar atualizações não intencionais para um componente que está sendo executado em seu dispositivo, recomendamos que você inclua diretamente sua versão preferida desse componente ao criar uma implantação. Para obter mais informações sobre o comportamento de atualização AWS IoT Greengrass do software Core, consulteAtualize o software AWS IoT Greengrass principal (OTA).

Componente	Detalhes
Núcleo Greengrass	A versão 2.3.0 do núcleo Greengrass está disponível. Novos atributos • Adiciona suporte para documentos de configuração de implantação de até 10 MB, acima de 7 KB (para implantações direcionadas a itens) ou 31 KB (para implantações direcionadas a grupos de itens). Para usar esse recurso, a AWS IoT política de um dispositivo principal deve permitir a `greengrass:GetDeploymentConfiguration` permissão. Se você usou o instalador do software AWS IoT Greengrass Core para provisionar recursos, a AWS IoT política do seu dispositivo principal permite`greengrass:*` , o que inclui essa permissão. Para ter mais informações, consulte Autorização e autenticação do dispositivo para o AWS IoT Greengrass. • Adiciona a variável da `iot:thingName` receita. Você pode usar essa variável de receita para obter o nome do dispositivo AWS IoT principal em uma receita. Para ter mais informações, consulte Variáveis da receita. Correções de erros e melhorias • Pequenas correções e melhorias adicionais. Para obter mais informaçõ es, consulte os lançamentos em GitHub.

Versão: atualização do software AWS IoT Greengrass Core v2.2.0 em 18 de junho de 2021

Esta versão fornece a versão 2.2.0 do componente nucleus do Greengrass, AWS novos componentes fornecidos e atualizações para os componentes fornecidos. AWS

Data de lançamento: 18 de junho de 2021

Destaques do lançamento

- Suporte ao dispositivo cliente — Os novos componentes do dispositivo cliente AWS fornecidos permitem que você conecte dispositivos cliente aos seus dispositivos principais usando a

descoberta na nuvem. Você pode sincronizar dispositivos cliente AWS IoT Core e interagir com dispositivos clientes nos componentes do Greengrass. Para ter mais informações, consulte Interaja com dispositivos IoT locais.

- Serviço paralelo local — O novo componente gerenciador de sombra ativa o serviço paralelo local em seus dispositivos principais. Você pode usar esse serviço de sombra para interagir com sombras locais enquanto estiver off-line usando as bibliotecas de comunicação entre processos (IPC) do Greengrass no. AWS IoT Device SDK Você também pode usar o componente do gerenciador de sombras para sincronizar estados de sombra locais com o. AWS IoT Core Para ter mais informações, consulte Interaja com as sombras do dispositivo.

Detalhes do lançamento

- Atualizações públicas de componentes

Atualizações públicas de componentes

A tabela a seguir lista os componentes AWS fornecidos que incluem recursos novos e atualizados.

> **Important**
>
> Quando você implanta um componente, AWS IoT Greengrass instala as versões mais recentes suportadas de todas as dependências desse componente. Por esse motivo, novas versões AWS de patch dos componentes públicos fornecidos podem ser implantadas automaticamente em seus dispositivos principais se você adicionar novos dispositivos a um grupo de coisas ou atualizar a implantação que visa esses dispositivos. Algumas atualizações automáticas, como a atualização do nucleus, podem fazer com que seus dispositivos sejam reiniciados inesperadamente.
>
> Para evitar atualizações não intencionais para um componente que está sendo executado em seu dispositivo, recomendamos que você inclua diretamente sua versão preferida desse componente ao criar uma implantação. Para obter mais informações sobre o comportamento de atualização AWS IoT Greengrass do software Core, consulteAtualize o software AWS IoT Greengrass principal (OTA).

Componente	Detalhes
Núcleo Greengrass	A versão 2.2.0 do núcleo Greengrass está disponível. Novos atributos • Adiciona operações de IPC para gerenciamento local de sombras. Correções de erros e melhorias • Reduz o tamanho do arquivo JAR. • Reduz o uso da memória. • Corrige problemas em que a configuração do log não foi atualizada em certos casos. • Pequenas correções e melhorias adicionais. Para obter mais informaçõ es, consulte os lançamentos em GitHub.
Gerenciador de sombras	A versão 2.0.0 do novo componente do gerenciador de sombras está disponível. Novos atributos • Adiciona suporte para sombras clássicas e nomeadas. • Adiciona suporte ao gerenciamento local de sombras usando IPC. • Adiciona suporte para sincronização de sombras comAWS IoT Core.
Autenticação do dispositivo cliente	A versão 2.0.0 do novo componente de autenticação do dispositivo cliente está disponível. Novos atributos • Adiciona suporte para dispositivos cliente Greengrass, que são dispositi vos IoT locais que se conectam a um dispositivo principal via MQTT. • Adiciona suporte para autenticação e autorização de dispositivos clientes e suas ações de MQTT.
Corretor Moquette MQTT	A versão 2.0.0 do novo componente de corretor Moquette MQTT está disponível.

Componente	Detalhes
	Novos atributos • Adiciona suporte para um broker Moquette MQTT local que lida com a comunicação com dispositivos clientes.
Ponte MQTT	A versão 2.0.0 do novo componente de ponte MQTT está disponível. Novos atributos • Adiciona suporte para retransmitir mensagens entre o agente local do MQTT, o agente local de publicação/assinatura do Greengrass e o agente do MQTT. AWS IoT Core
Detector IP	A versão 2.0.0 do novo componente detector de IP está disponível. Novos atributos • Adiciona suporte para reportar os endpoints locais do broker MQTT de um dispositivo principal ao serviço de AWS IoT Greengrass nuvem para que os dispositivos clientes se conectem.
Gerenciador de registros	A versão 2.1.1 do componente gerenciador de registros está disponível. Correções de erros e melhorias • Corrige um problema em que a configuração do log do sistema não foi atualizada em alguns casos.
Detecção de objetos DLR	A versão 2.1.2 da detecção de objetos DLR está disponível. Correções de erros e melhorias • Corrige um problema de escala de imagem que resultou em caixas delimitadoras imprecisas nos resultados de inferência de detecção de objetos DLR da amostra.

Componente	Detalhes
TensorFlow Detecção leve de objetos	A versão 2.1.1 da detecção de objetos TensorFlow Lite está disponível. Correções de erros e melhorias • Corrige um problema de escala de imagem que resultou em caixas delimitadoras imprecisas nos resultados de inferência de detecção de objetos TensorFlow Lite de amostra.

Versão: atualização do software AWS IoT Greengrass Core v2.1.0 em 26 de abril de 2021

Esta versão fornece a versão 2.1.0 do componente nucleus do Greengrass e AWS atualiza os componentes fornecidos.

Data de lançamento: 26 de abril de 2021

Destaques do lançamento

- Integração entre o Docker Hub e o Amazon Elastic Container Registry (Amazon ECR) — O novo componente do gerenciador de aplicativos Docker permite que você baixe imagens públicas ou privadas do Amazon ECR. Você também pode usar esse componente para baixar imagens públicas do Docker Hub e. AWS Marketplace Para ter mais informações, consulte Execute um contêiner Docker.
- Dockerfile e imagens do Docker para o software AWS IoT Greengrass Core — Você pode usar a imagem Docker do Greengrass para executar em AWS IoT Greengrass um contêiner do Docker que usa o Amazon Linux 2 como sistema operacional básico. Você também pode usar o AWS IoT Greengrass Dockerfile para criar sua própria imagem do Greengrass. Para ter mais informações, consulte Execute AWS IoT Greengrass o software Core em um contêiner Docker.
- Support para estruturas e plataformas adicionais de aprendizado de máquina — Você pode implantar amostras de componentes de inferência de aprendizado de máquina que usam modelos pré-treinados para realizar a classificação de imagens de amostra e a detecção de objetos usando o TensorFlow Lite 2.5.0 e o DLR 1.6.0. Esta versão também amplia exemplos de suporte de aprendizado de máquina para dispositivos Armv8 (AArch64). Para ter mais informações, consulte Executar a inferência de machine learning.

Detalhes do lançamento

- Atualizações de suporte da plataforma
- Atualizações públicas de componentes

Atualizações de suporte da plataforma

Plataforma	Detalhes
Docker	Um Dockerfile e uma imagem Docker para já AWS IoT Greengrass estão disponíveis. Dockerfile AWS IoT Greengrassfornece um Dockerfile para criar uma imagem de contêiner que tenha o software AWS IoT Greengrass principal e as dependências instaladas em uma imagem base do Amazon Linux 2 (x86_64). Você pode modificar a imagem base no Dockerfile para ser executada AWS IoT Greengrass em uma arquitetura de plataforma diferente. Docker image (Imagem do Docker) AWS IoT Greengrassfornece uma imagem Docker pré-criada que tem o software AWS IoT Greengrass principal e as dependências instaladas em uma imagem base do Amazon Linux 2 (x86_64). Para ter mais informações, consulte Execute AWS IoT Greengrass o software Core em um contêiner Docker.

Atualizações públicas de componentes

A tabela a seguir lista os componentes AWS fornecidos que incluem recursos novos e atualizados.

> **Important**
>
> Quando você implanta um componente, AWS IoT Greengrass instala as versões mais recentes suportadas de todas as dependências desse componente. Por esse motivo, novas

versões AWS de patch dos componentes públicos fornecidos podem ser implantadas automaticamente em seus dispositivos principais se você adicionar novos dispositivos a um grupo de coisas ou atualizar a implantação que visa esses dispositivos. Algumas atualizações automáticas, como a atualização do nucleus, podem fazer com que seus dispositivos reiniciem inesperadamente.
Para evitar atualizações não intencionais para um componente que está sendo executado em seu dispositivo, recomendamos que você inclua diretamente sua versão preferida desse componente ao criar uma implantação. Para obter mais informações sobre o comportamento de atualização AWS IoT Greengrass do software Core, consulteAtualize o software AWS IoT Greengrass principal (OTA).

Componente	Detalhes
Núcleo Greengrass	A versão 2.1.0 do núcleo Greengrass está disponível. Novos atributos • Suporta o download de imagens do Docker de repositórios privados no Amazon ECR. • Adiciona os seguintes parâmetros para personalizar a configuração do MQTT nos dispositivos principais: • `maxInFlightPublishes` — O número máximo de mensagens de QoS 1 não confirmadas do MQTT que podem estar em andamento ao mesmo tempo. • `maxPublishRetry` — O número máximo de vezes para repetir uma mensagem que não foi publicada. • Adiciona o parâmetro de `fleetstatusservice` configuração para configurar o intervalo no qual o dispositivo principal publica o status do dispositivo noNuvem AWS. • Pequenas correções e melhorias adicionais. Para obter mais informaçõ es, consulte os lançamentos em GitHub. Correções de erros e melhorias • Corrige um problema que fazia com que as implantações de sombra fossem duplicadas quando o núcleo era reiniciado.

Componente	Detalhes
	• Corrige um problema que fazia com que o núcleo falhasse ao encontrar uma exceção de carga de serviço. • Melhora a resolução de dependências de componentes para falhar em uma implantação que inclui uma dependência circular. • Corrige um problema que impedia que um componente de plug-in fosse reimplantado se esse componente tivesse sido removido anteriormente do dispositivo principal. • Corrija um problema que fazia com que a variável de `HOME` ambiente fosse definida no */greengrass/v2* `/work` diretório dos componentes do Lambda ou dos componentes executados como raiz. Agora, a `HOME` variável está definida corretamente no diretório inicial do usuário que executa o componente. • Pequenas correções e melhorias adicionais. Para obter mais informaçõ es, consulte os lançamentos em GitHub.
Gerenciador de aplicativos Docker	A versão 2.0.0 do novo componente do gerenciador de aplicativos Docker está disponível. Novos atributos • Gerencia credenciais para baixar imagens de repositórios privados no Amazon ECR. • Faz o download de imagens públicas do Amazon ECR, Docker Hub e. AWS Marketplace
Lançador Lambda	A versão 2.0.4 do componente Lambda launcher está disponível. Correções de erros e melhorias • Corrige um problema em que o componente não passa corretamente `AddGroupOwner` para o contêiner da função Lambda.

Componente	Detalhes
Roteador de assinatura antigo	A versão 2.1.0 do componente antigo do roteador de assinatura está disponível. Correções de erros e melhorias • Adiciona suporte para especificar nomes de componentes em vez de ARNs para `source` e. `target` Se você especificar um nome de componente para uma assinatura, não precisará reconfigurar a assinatura sempre que a versão da função Lambda for alterada.
Console de depuração local	A versão 2.1.0 do componente do console de depuração local está disponível. Novos atributos • Usa HTTPS para proteger sua conexão com o console de depuração local. O HTTPS está habilitado por padrão. Correções de erros e melhorias • Você pode ignorar as mensagens da barra de flash no editor de configuração.
Gerenciador de registros	A versão 2.1.0 do componente gerenciador de registros está disponível. Correções de erros e melhorias • Use padrões para `logFileDirectoryPath` e `logFileRegex` que funcionem para componentes do Greengrass que imprimem na saída padrão (stdout) e no erro padrão (stderr). • Direcione corretamente o tráfego por meio de um proxy de rede configurado ao fazer o upload dos registros para o CloudWatch Logs. • Manipule corretamente os caracteres de dois pontos (`:`) nos nomes dos fluxos de log. CloudWatch Os nomes dos fluxos de registro de registros não oferecem suporte a dois pontos. • Simplifique os nomes do fluxo de log removendo os nomes dos grupos de coisas do fluxo de log. • Remova uma mensagem de registro de erros que é impressa durante o comportamento normal.

Componente	Detalhes
Classificação de imagens DLR	A versão 2.1.1 do componente de classificação de imagem DLR está disponível. Novos atributos • Use o Deep Learning Runtime v1.6.0. • Adicione suporte para classificação de imagens de amostra nas plataformas Armv8 (AArch64). Isso amplia o suporte ao aprendizado de máquina para os principais dispositivos do Greengrass que executam o NVIDIA Jetson, como o Jetson Nano. • Ative a integração da câmera para inferência de amostras. Use o novo parâmetro de `UseCamera` configuração para permitir que o código de inferência de amostra acesse a câmera em seu dispositivo principal do Greengrass e execute a inferência localmente na imagem capturada. • Adicione suporte para publicação de resultados de inferência noNuvem AWS. Use o novo parâmetro de `PublishResultsOnTopic` configuração para especificar o tópico no qual você deseja publicar os resultados. • Adicione o novo parâmetro de `ImageDirectory` configuração que permite especificar um diretório personalizado para a imagem na qual você deseja realizar a inferência. Correções de erros e melhorias • Grave os resultados da inferência no arquivo de log do componente em vez de em um arquivo de inferência separado. • Use o módulo de registro do software AWS IoT Greengrass Core para registrar a saída do componente. • Use o AWS IoT Device SDK para ler a configuração do componente e aplicar as alterações na configuração.

Componente	Detalhes
Detecção de objetos DLR	A versão 2.1.1 do componente de detecção de objetos DLR está disponível. Novos atributos • Use o Deep Learning Runtime v1.6.0. • Adicione suporte para detecção de objetos de amostra nas plataform as Armv8 (AArch64). Isso amplia o suporte ao aprendizado de máquina para os principais dispositivos do Greengrass que executam o NVIDIA Jetson, como o Jetson Nano. • Ative a integração da câmera para inferência de amostras. Use o novo parâmetro de `UseCamera` configuração para permitir que o código de inferência de amostra acesse a câmera em seu dispositivo principal do Greengrass e execute a inferência localmente na imagem capturada. • Adicione suporte para publicação de resultados de inferência noNuvem AWS. Use o novo parâmetro de `PublishResultsOnTopic` configuração para especificar o tópico no qual você deseja publicar os resultados. • Adicione o novo parâmetro de `ImageDirectory` configuração que permite especificar um diretório personalizado para a imagem na qual você deseja realizar a inferência. Correções de erros e melhorias • Grave os resultados da inferência no arquivo de log do componente em vez de em um arquivo de inferência separado. • Use o módulo de registro do software AWS IoT Greengrass Core para registrar a saída do componente. • Use o AWS IoT Device SDK para ler a configuração do componente e aplicar as alterações na configuração.

Componente	Detalhes
Armazenamento de modelos de classificação de imagens DLR	A versão 2.1.1 do componente de armazenamento de modelos de classificação de imagem DLR está disponível. Novos atributos • Adicione um modelo de classificação de imagem de amostra ResNet -50 para plataformas Armv8 (AArch64). Isso amplia o suporte ao aprendizado de máquina para os principais dispositivos do Greengrass que executam o NVIDIA Jetson, como o Jetson Nano.
Armazenamento de modelos de detecção de objetos DLR	A versão 2.1.1 do componente de armazenamento de modelos de detecção de objetos DLR está disponível. Novos atributos • Adicione uma amostra do modelo de detecção de objetos YOLOv3 para plataformas Armv8 (AArch64). Isso amplia o suporte ao aprendizado de máquina para os principais dispositivos do Greengrass que executam o NVIDIA Jetson, como o Jetson Nano.
Instalador DLR	A versão 1.6.1 do componente DLR está disponível. Novos atributos • Instale o Deep Learning Runtime v1.6.0 e suas dependências. • Adicione suporte para instalação de DLR em plataformas Armv8 (AArch64). Isso amplia o suporte ao aprendizado de máquina para os principais dispositivos do Greengrass que executam o NVIDIA Jetson, como o Jetson Nano. Correções de erros e melhorias • Instale o AWS IoT Device SDK no ambiente virtual para ler a configuração do componente e aplicar as alterações na configuração. • Correções e melhorias adicionais de pequenos bugs.

Componente	Detalhes
TensorFlow Classificação de imagens Lite	A versão 2.1.0 do novo componente de classificação de imagens TensorFlow Lite está disponível. Novos atributos • Adicione suporte para inferência de classificação de imagens de amostra usando o TensorFlow Lite.
TensorFlow Detecção leve de objetos	A versão 2.1.0 do novo componente de detecção de objetos TensorFlow Lite está disponível. Novos atributos • Adicione suporte para inferência de detecção de objetos de amostra usando o TensorFlow Lite.
TensorFlow Loja de modelos de classificação de imagens Lite	A versão 2.1.0 do novo componente de armazenamento de modelos de classificação de imagens TensorFlow Lite está disponível. Novos atributos • Forneça um modelo quantizado MobileNet v1 pré-treinado para inferênci a de classificação de imagens de amostra usando o Lite. TensorFlow
TensorFlow Loja de modelos de detecção de objetos Lite	A versão 2.1.0 do novo componente de armazenamento de modelos de detecção de objetos TensorFlow Lite está disponível. Novos atributos • Forneça um MobileNet modelo pré-treinado de detecção de disparo único (SSD) treinado no conjunto de dados COCO para inferência de detecção de objetos de amostra usando o Lite. TensorFlow
TensorFlow Leve	A versão 2.5.0 do novo componente TensorFlow Lite está disponível. Novos atributos • Instale o TensorFlow Lite v1.6.0 e suas dependências em um ambiente virtual nas plataformas Armv7, Armv8 (AArch64) e x86_64.

Versão: atualização do software AWS IoT Greengrass Core v2.0.5 em 09 de março de 2021

Esta versão fornece a versão 2.0.5 do componente nucleus do Greengrass e AWS atualiza os componentes fornecidos. Ele corrige um problema com o suporte de proxy de rede e um problema com o endpoint do plano de dados Greengrass nas AWS regiões da China.

Data de lançamento: 09 de março de 2021

Atualizações públicas de componentes

A tabela a seguir lista os componentes AWS fornecidos que incluem recursos novos e atualizados.

> **Important**
>
> Quando você implanta um componente, AWS IoT Greengrass instala as versões mais recentes suportadas de todas as dependências desse componente. Por esse motivo, novas versões AWS de patch dos componentes públicos fornecidos podem ser implantadas automaticamente em seus dispositivos principais se você adicionar novos dispositivos a um grupo de coisas ou atualizar a implantação que visa esses dispositivos. Algumas atualizações automáticas, como a atualização do nucleus, podem fazer com que seus dispositivos sejam reiniciados inesperadamente.
> Para evitar atualizações não intencionais para um componente que está sendo executado em seu dispositivo, recomendamos que você inclua diretamente sua versão preferida desse componente ao criar uma implantação. Para obter mais informações sobre o comportamento de atualização AWS IoT Greengrass do software Core, consulteAtualize o software AWS IoT Greengrass principal (OTA).

Componente	Detalhes
Núcleo Greengrass	A versão 2.0.5 do núcleo Greengrass está disponível. Correções de erros e melhorias • Roteia corretamente o tráfego por meio de um proxy de rede configura do ao baixar os componentes AWS fornecidos.

Componente	Detalhes
	• Use o endpoint correto do plano de dados Greengrass nas AWS regiões da China.

Versão: atualização do software AWS IoT Greengrass Core v2.0.4 em 04 de fevereiro de 2021

Esta versão fornece a versão 2.0.4 do componente do núcleo do Greengrass. Ele inclui o novo `greengrassDataPlanePort` parâmetro para configurar a comunicação HTTPS pela porta 443 e corrige bugs. A política mínima do IAM agora exige o `iam:GetPolicy` e `sts:GetCallerIdentity` quando o instalador do software AWS IoT Greengrass Core é executado`--provision true`.

Data de lançamento: 04 de fevereiro de 2021

Atualizações públicas de componentes

A tabela a seguir lista os componentes AWS fornecidos que incluem recursos novos e atualizados.

> **Important**
>
> Quando você implanta um componente, AWS IoT Greengrass instala as versões mais recentes suportadas de todas as dependências desse componente. Por esse motivo, novas versões AWS de patch dos componentes públicos fornecidos podem ser implantadas automaticamente em seus dispositivos principais se você adicionar novos dispositivos a um grupo de coisas ou atualizar a implantação que visa esses dispositivos. Algumas atualizações automáticas, como a atualização do nucleus, podem fazer com que seus dispositivos sejam reiniciados inesperadamente.
> Para evitar atualizações não intencionais para um componente que está sendo executado em seu dispositivo, recomendamos que você inclua diretamente sua versão preferida desse componente ao criar uma implantação. Para obter mais informações sobre o comportamento de atualização AWS IoT Greengrass do software Core, consulteAtualize o software AWS IoT Greengrass principal (OTA).

Componente	Detalhes
Núcleo Greengrass	A versão 2.0.4 do núcleo Greengrass está disponível. Novos atributos • Habilita o tráfego HTTPS pela porta 443. Você pode usar o novo parâmetro de `greengrassDataPlanePort` configuração para a versão 2.0.4 do componente nucleus para configurar a comunicação HTTPS para viajar pela porta 443 em vez da porta padrão 8443. Para ter mais informações, consulte Configurar HTTPS pela porta 443. • Adiciona a variável de receita do caminho de trabalho. Você pode usar essa variável de receita para obter o caminho para as pastas de trabalho dos componentes, que você pode usar para compartilhar arquivos entre componentes e suas dependências. Para obter mais informações, consulte a variável de receita do caminho de trabalho. Correções de erros e melhorias • Impede a criação da política de função de troca de tokens AWS Identity and Access Management (IAM) se uma política de função já existir. Como resultado dessa alteração, o instalador agora exige o `iam:GetPolicy` e `sts:GetCallerIdentity` quando executado com `--provision true` . Para ter mais informações, consulte Política mínima de IAM para o instalador provisionar recursos. • Lida corretamente com o cancelamento de uma implantação que ainda não foi registrada com sucesso. • Atualiza a configuração para remover entradas mais antigas com carimbos de data/hora mais recentes ao reverter uma implantação. • Pequenas correções e melhorias adicionais. Para obter mais informações, consulte os lançamentos em GitHub.

Migrar da AWS IoT Greengrass versão 1

AWS IoT Greengrass Version 2é uma versão principal do software, das APIs e do console AWS IoT Greengrass Core. AWS IoT Greengrass V2introduz várias melhoriasAWS IoT Greengrass V1, como aplicativos modulares, implantações em grandes frotas de dispositivos e suporte para plataformas adicionais.

> **Note**
>
> Depois de 30 de junho de 2023, AWS IoT Greengrass Version 1 não receberá mais atualizações de recursos, aprimoramentos, correções de erros ou patches de segurança. Para obter mais informações, consulte política de manutenção do AWS IoT Greengrass V1. Se você usaAWS IoT Greengrass V1, é altamente recomendável que você migre para o. AWS IoT Greengrass V2

Siga as instruções deste guia para AWS IoT Greengrass V1 migrar de o. AWS IoT Greengrass V2

Posso executar meus aplicativos V1 na V2?

A maioria dos aplicativos V1 pode ser executada em dispositivos principais V2 sem precisar alterar o código do aplicativo. Se seus aplicativos V1 usarem o recurso a seguir, você não poderá executá-los na V2.

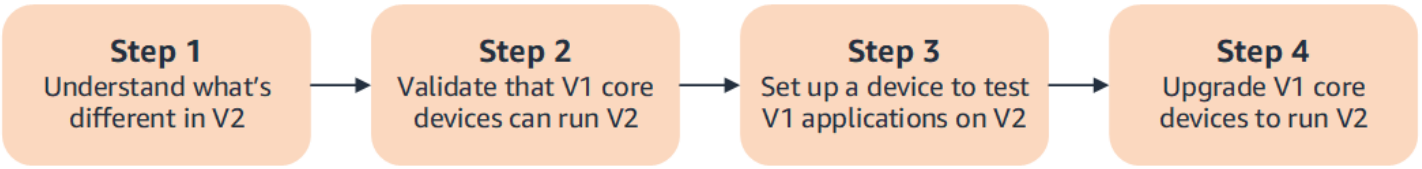
- Os tempos de execução das funções Lambda em C e C++

Se seus aplicativos V1 usarem um dos recursos a seguir, você deverá modificar o código do aplicativo para usar o AWS IoT Device SDK V2 para executar os aplicativos. AWS IoT Greengrass V2

- Interaja com o serviço paralelo local
- Publique mensagens em dispositivos conectados locais (dispositivos Greengrass)

Visão geral da migração

Em um alto nível, você pode usar o procedimento a seguir para atualizar os dispositivos principais de AWS IoT Greengrass V1 paraAWS IoT Greengrass V2. O procedimento exato que você segue depende dos requisitos específicos do seu ambiente.



1. Entenda as diferenças entre V1 e V2

   AWS IoT Greengrass V2introduz novos conceitos fundamentais para frotas de dispositivos e software implantável, e a V2 simplifica vários conceitos da V1.

   O serviço de AWS IoT Greengrass V2 nuvem e o software AWS IoT Greengrass Core v2.x não são compatíveis com versões anteriores do serviço de AWS IoT Greengrass V1 nuvem e do software AWS IoT Greengrass Core v1.x. Como resultado, as atualizações AWS IoT Greengrass V1 over-the-air (OTA) não podem atualizar os dispositivos principais da V1 para a V2.

2. Valide se os dispositivos principais da V1 podem executar a V2

   Verifique se um dispositivo central V1 pode executar o software AWS IoT Greengrass Core v2.x e os recursos. AWS IoT Greengrass V2 AWS IoT Greengrass V2tem requisitos de dispositivo diferentes dosAWS IoT Greengrass V1.

3. Configure um novo dispositivo para testar aplicativos V1 na V2

   Para minimizar o risco de seus dispositivos em produção, crie um novo dispositivo para testar seus aplicativos V1 na V2. Depois de instalar o software AWS IoT Greengrass Core v2.x, você pode criar e implantar AWS IoT Greengrass V2 componentes para migrar e testar seus aplicativos. AWS IoT Greengrass V1

4. Atualize os dispositivos principais da V1 para executar a V2

   Atualize um dispositivo V1 core existente para executar o software AWS IoT Greengrass Core v2.x e os componentes. AWS IoT Greengrass V2 Para migrar uma frota de dispositivos da V1 para a V2, repita essa etapa para cada dispositivo da frota.

Diferenças entre AWS IoT Greengrass V1 e AWS IoT Greengrass V2

AWS IoT Greengrass V2 apresenta novos conceitos fundamentais para dispositivos, frotas e software implantável. Esta seção descreve os conceitos da V1 que são diferentes na V2.

Conceitos e terminologia do Greengrass

Conceito	AWS IoT Greengrass V1	AWS IoT Greengrass V2
Código do aplicativo	Em AWS IoT Greengrass V1, as funções Lambda definem o software que é executado nos dispositivos principais. Em cada grupo do Greengrass, você define assinaturas e recursos locais que a função usa. Para funções Lambda que o software AWS IoT Greengrass Core executa em um ambiente de execução Lambda em contêiner, você define parâmetros de contêiner, como limites de memória.	Em AWS IoT Greengrass V2, os componentes são os módulos de software que são executados nos dispositivos principais. • Cada componente tem uma receita que define os metadados, parâmetros, dependências e scripts do componente a serem executados em cada etapa do ciclo de vida do componente. • A receita também define os artefatos do componente, que são arquivos binários, como scripts, código compilado e recursos estáticos. • Quando você implanta um componente em um dispositivo principal, o dispositivo principal baixa a receita e os artefatos do componente para executar o componente.

Conceito	AWS IoT Greengrass V1	AWS IoT Greengrass V2
		Você pode importar suas funções V1 Lambda como componentes que são executados em um ambiente de execução do Lambda em. AWS IoT Greengrass V2 Ao importar a função Lambda, você especifica as assinaturas, os recursos locais e os parâmetros do contêiner para a função. Para ter mais informações, consulte Etapa 2: criar e implantar AWS IoT Greengrass V2 componentes para migrar aplicativos AWS IoT Greengrass V1. Para obter mais informações sobre como criar componentes personalizados, consulteDesenvolva AWS IoT Greengrass componentes.

Conceito	AWS IoT Greengrass V1	AWS IoT Greengrass V2
AWS IoT Greengrass grupos e implantações	Em AWS IoT Greengrass V1, um grupo define o dispositivo principal, as configurações e o software desse dispositivo principal e a lista de AWS IoT itens que podem se conectar a esse dispositivo principal. Você cria uma implantação para enviar a configuração de um grupo para um dispositivo principal.	Em AWS IoT Greengrass V2, você usa implantações para definir os componentes e as configurações de software que são executados nos dispositivos principais. • Cada implantação tem como alvo um único dispositivo central (o que é uma AWS IoT coisa) ou um grupo de AWS IoT coisas que pode conter vários dispositivos principais. • As implantações em grupos de coisas são contínuas, portanto, quando você adiciona um dispositivo principal a um grupo de coisas, ele recebe a configuração de software desse grupo. Para ter mais informações, consulte Implemente AWS IoT Greengrass componentes em dispositivos. Em AWS IoT Greengrass V2, você também pode criar implantações locais usando a CLI do Greengrass para testar componentes de software personalizados no dispositivo em que você os desenvolv

Conceito	AWS IoT Greengrass V1	AWS IoT Greengrass V2
		e. Para ter mais informaçõ es, consulte Crie AWS IoT Greengrass componentes.
AWS IoT Greengrass Software principal	Em AWS IoT Greengras s V1, o software AWS IoT Greengrass Core é um pacote único que contém o software e todos os seus recursos. O dispositivo periférico no qual você instala o software AWS IoT Greengrass Core é chamado de núcleo Greengrass.	Em AWS IoT Greengras s V2, o software AWS IoT Greengrass Core é modular, para que você possa escolher o que instalar para controlar o espaço ocupado pela memória. • O componente do núcleo do Greengrass é a instalaçã o mínima necessária do AWS IoT Greengrass software Core. O dispositi vo de borda no qual você instala o núcleo é chamado de dispositivo central Greengrass. • O núcleo lida com implantaç ões, orquestração e gerenciamento do ciclo de vida de outros componentes no dispositivo principal. • Recursos como gerenciad or de fluxo, gerenciador secreto e gerenciador de registros são componentes que você implanta somente quando precisa desses recursos. Para ter mais informações, consulte AWS-componentes fornecidos.

Conceito	AWS IoT Greengrass V1	AWS IoT Greengrass V2
Connectors	Em AWS IoT Greengrass V1, os conectores são módulos pré-construídos que você implanta nos dispositivos AWS IoT Greengrass V1 principais para interagir com a infraestrutura local AWS, protocolos de dispositivos e outros serviços em nuvem.	Em AWS IoT Greengrass V2, AWS fornece componentes do Greengrass que implementam a funcionalidade fornecida pelos conectores na V1. Os AWS IoT Greengrass V2 componentes a seguir fornecem a funcionalidade do conector Greengrass V1: • CloudWatch componente de métricas • AWS IoT Device Defender componente • Componente Firehose • Componente adaptador de protocolo Modbus-RTU • Componente Amazon SNS Para ter mais informações, consulte AWS-componentes fornecidos.

Conceito	AWS IoT Greengrass V1	AWS IoT Greengrass V2
Dispositivos conectados (dispositivos Greengrass)	Em AWS IoT Greengrass V1, dispositivos conectados são AWS IoT coisas que você adiciona a um grupo do Greengrass para se conectar ao dispositivo principal desse grupo e se comunicar pelo MQTT. Você deve implantar esse grupo sempre que adicionar ou remover um dispositivo conectado. Você usa assinaturas para retransmitir mensagens entre dispositivos conectados e aplicativos no dispositivo principal. AWS IoT Core	Em AWS IoT Greengrass V2, os dispositivos conectados são chamados de dispositivos cliente Greengrass. • Você associa dispositivos clientes aos dispositivos principais para conectá-los e se comunicar pelo MQTT. • Para autorizar a conexão de dispositivos cliente, você define políticas de autorização que podem ser aplicadas a grupos de dispositivos cliente, portanto, não é necessário criar uma implantação para adicionar ou remover um dispositivo cliente. • Para retransmitir mensagens entre dispositivos cliente e componentes do Greengrass AWS IoT Core, você pode configurar um componente opcional de ponte MQTT. Em ambos AWS IoT Greengrass V1 os casos AWS IoT Greengrass V2, os dispositivos podem executar FreeRTOS ou usar a API de descoberta do AWS IoT Device SDKGreengrass para

Conceito	AWS IoT Greengrass V1	AWS IoT Greengrass V2
		obter informações sobre os principais dispositivos aos quais eles podem se conectar. A API de descoberta do Greengrass é compatível com versões anteriores, portanto, se você tiver dispositivos clientes que se conectam a um dispositivo V1 core, você pode conectá-los a um dispositivo V2 core sem alterar o código. Para obter mais informaçõ es sobre dispositivos cliente, consulteInteraja com dispositivos IoT locais.

Conceito	AWS IoT Greengrass V1	AWS IoT Greengrass V2
Recursos locais	Em AWS IoT Greengrass V1, as funções Lambda executadas em contêineres podem ser configuradas para acessar volumes e dispositivos no sistema de arquivos do dispositivo principal. Esses recursos do sistema de arquivos são conhecidos como recursos locais.	Em AWS IoT Greengrass V2, você pode executar componentes que são funções Lambda, contêineres Docker ou processos nativos do sistema operacional ou tempos de execução personalizados. • Ao importar uma função Lambda em contêiner como um componente, você deve especificar os recursos locais que a função usa. • Funções Lambda não conteinerizadas e componentes não Lambda podem trabalhar diretamente com recursos locais em dispositivos principais, então você não precisa especificar os recursos locais que o componente usa.

Conceito	AWS IoT Greengrass V1	AWS IoT Greengrass V2
Serviço paralelo local	Em AWS IoT Greengras s V1, o serviço de sombra local é ativado por padrão e oferece suporte somente a sombras clássicas sem nome. Você usa o SDK AWS IoT Greengrass principal em suas funções do Lambda para interagir com sombras em seus dispositivos.	Em AWS IoT Greengras s V2, você ativa o serviço paralelo local implantando o componente gerenciador de sombra. • Você pode usar o AWS IoT Device SDK V2 em funções Lambda e component es personalizados para interagir com sombras em seus dispositivos. • O serviço de sombra local oferece suporte a sombras nomeadas. • O serviço de sombra local permite excluir sombras e sincronizar sombras excluídas com. AWS IoT Core Para ter mais informações, consulte Interaja com as sombras do dispositivo.

Conceito	AWS IoT Greengrass V1	AWS IoT Greengrass V2
Assinaturas	Em AWS IoT Greengrass V1, você define assinaturas para um grupo do Greengras s para especificar canais de comunicação entre funções, conectores, dispositivos conectados, o agente AWS IoT Core MQTT e o serviço paralelo local do Lambda. As assinaturas especificam onde as funções Lambda recebem mensagens de eventos para serem consumidas como cargas de função.	Em AWS IoT Greengrass V2, você especifica canais de comunicação sem usar assinaturas. • Os componentes gerenciam seus próprios canais de comunicação para interagir com mensagens locais de publicação/assinatura, mensagens AWS IoT Core MQTT e o serviço paralelo local. • Para desenvolver um componente que reaja às mensagens de outro componente ou do agente AWS IoT Core MQTT, você pode usar interface s de comunicação entre processos (IPC) para mensagens locais de publicação/assinatura e mensagens MQTT. AWS IoT Core • Para desenvolver um componente que interaja com o serviço paralelo local, você pode usar a interface IPC para o serviço paralelo local. • Na configuração do componente, você define políticas de autorizaç

Conceito	AWS IoT Greengrass V1	AWS IoT Greengrass V2
		ão para especificar os tópicos e as sombras locais que o componente tem permissão para usar. • Para configurar canais de comunicação entre dispositivos cliente, o agente local de publicaçã o/assinatura e o agente AWS IoT Core MQTT, você configura e implementa o componente de ponte MQTT. O componente de ponte MQTT permite que você interaja com dispositi vos clientes em component es e retransmita mensagens entre dispositivos clientes e. AWS IoT Core

Conceito	AWS IoT Greengrass V1	AWS IoT Greengrass V2
Acessando outros Serviços da AWS	Em AWS IoT Greengrass V1, você anexa uma função AWS Identity and Access Management (IAM), chamada de função de grupo, a um grupo do Greengrass. A função do grupo define as permissões que as funções e os AWS IoT Greengras s recursos do Lambda no dispositivo principal desse grupo usam para acessar. Serviços da AWS	Em AWS IoT Greengrass V2, você anexa um alias de AWS IoT função a um dispositi vo principal do Greengras s. O alias da função aponta para uma função do IAM chamada função de troca de tokens. A função de troca de tokens define as permissõe s que os componentes do Greengrass no dispositivo principal usam para acessar. Serviços da AWS Para ter mais informações, consulte Autorize os dispositivos principais a interagir com os serviços AWS.

Validate: os dispositivos principais V1 podem executar o software V2

O softwareAWS IoT Greengrass Core v2.x tem requisitos diferentes do softwareAWS IoT Greengrass Core v1.x. Antes de atualizar os dispositivos principais V1 para V2, revise os requisitos do dispositivo paraAWS IoT Greengrass V2. AWS IoT Greengrass V2atualmente não oferece suporte à migração para sistemas personalizados baseados em Linux usando o Projeto Yocto.

Você pode usar AWS IoT Device Tester(IDT) forAWS IoT Greengrass V2 para validar se os dispositivos atendem aos requisitos para executar o softwareAWS IoT Greengrass Core v2.x. O IDT é uma estrutura de teste que pode ser baixada que é executada em seu computador host e se conecta a dispositivos para serem validados. Siga as instruções para usar o IDT para executar o conjuntoAWS IoT Greengrass de qualificações. Ao configurar o IDT, você pode optar por validar se os dispositivos suportam recursos opcionais, como Docker, aprendizado de máquina (ML), gerenciamento de fluxo de dados e integração de segurança de hardware.

Se o IDT relatar falhas ou erros no teste V2 em um dispositivo de núcleo V1, você não poderá atualizar esse dispositivo de V1 para V2.

Configure um novo dispositivo V2 core para testar aplicativos V1

Configure um novo dispositivo AWS IoT Greengrass V2 principal para implantar e testar os componentes e AWS Lambda funções AWS fornecidos para seus AWS IoT Greengrass V1 aplicativos. Você também pode usar esse dispositivo principal V2 para desenvolver e testar componentes adicionais personalizados do Greengrass que executam processos nativos em dispositivos principais. Depois de testar seus aplicativos em um dispositivo V2 core, você pode atualizar seus dispositivos V1 core existentes para V2 e implantar os componentes V2 que fornecem sua funcionalidade V1.

Etapa 1: instalar AWS IoT Greengrass V2 em um novo dispositivo

Instale o software AWS IoT Greengrass Core v2.x em um novo dispositivo. Você pode seguir o tutorial de introdução para configurar um dispositivo e aprender a desenvolver e implantar componentes. Este tutorial usa o provisionamento automático para configurar rapidamente um dispositivo. Ao instalar o software AWS IoT Greengrass Core v2.x, especifique o `--deploy-dev-tools` argumento para implantar a CLI do Greengrass, para que você possa desenvolver, testar e depurar componentes diretamente no dispositivo. Para obter mais informações sobre outras opções de instalação, incluindo como instalar o software AWS IoT Greengrass Core por trás de um proxy ou usando um módulo de segurança de hardware (HSM), consulteInstalar o software do AWS IoT Greengrass Core.

(Opcional) Ativar o registro no Amazon CloudWatch Logs

Para permitir que um dispositivo V2 core faça upload de registros para o Amazon CloudWatch Logs, você pode implantar o componente gerenciador AWS de registros fornecido. Você pode usar o CloudWatch Logs para visualizar os registros de componentes, para poder depurar e solucionar problemas sem acessar o sistema de arquivos do dispositivo principal. Para ter mais informações, consulte Monitore AWS IoT Greengrass os registros.

Etapa 2: criar e implantar AWS IoT Greengrass V2 componentes para migrar aplicativos AWS IoT Greengrass V1

Você pode executar a maioria dos AWS IoT Greengrass V1 aplicativos em AWS IoT Greengrass V2. Você pode importar funções do Lambda como componentes que são executados em AWS

IoT Greengrass V2, e você pode usar componentes AWS fornecidos que oferecem a mesma funcionalidade dos conectores. AWS IoT Greengrass

Você também pode desenvolver componentes personalizados para criar qualquer recurso ou tempo de execução nos dispositivos principais do Greengrass. Para obter informações sobre como desenvolver e testar componentes localmente, consulteCrie AWS IoT Greengrass componentes.

Tópicos

Importar funções V1 Lambda

Você pode importar funções do Lambda como AWS IoT Greengrass V2 componentes. Escolha entre as seguintes abordagens:

- O Import V1 Lambda funciona diretamente como componentes do Greengrass.
- Atualize suas funções do Lambda para usar as bibliotecas do Greengrass na AWS IoT Device SDK v2 e, em seguida, importe as funções do Lambda como componentes do Greengrass.
- Crie componentes personalizados que usam código não Lambda e a AWS IoT Device SDK v2 para implementar a mesma funcionalidade das suas funções do Lambda.

Se sua função Lambda usa recursos, como gerenciador de stream ou segredos locais, você deve definir dependências nos componentes AWS fornecidos que empacotam esses recursos. Quando você implanta o componente da função Lambda, a implantação também inclui o componente para cada recurso que você define como uma dependência. Na implantação, você pode configurar parâmetros, como quais segredos implantar no dispositivo principal. Nem todos os recursos da V1 exigem uma dependência de componente para sua função Lambda na V2. A lista a seguir descreve como usar os recursos da V1 em seu componente de função Lambda da V2.

- Acesse outros AWS serviços

Se sua função Lambda usa AWS credenciais para fazer solicitações a outros AWS serviços, a função de troca de tokens do dispositivo principal deve permitir que o dispositivo principal execute AWS as operações que a função Lambda usa. Para ter mais informações, consulte Autorize os dispositivos principais a interagir com os serviços AWS.

- Gerenciador de streams

  Se sua função Lambda usa o gerenciador de stream, especifique `aws.greengrass.StreamManager` como uma dependência de componente ao importar a função. Ao implantar o componente do gerenciador de fluxo, especifique os parâmetros do gerenciador de fluxo a serem definidos para os dispositivos principais de destino. A função de troca de tokens do dispositivo principal deve permitir que o dispositivo principal acesse os Nuvem AWS destinos que você usa com o gerenciador de streaming. Para ter mais informações, consulte Gerenciador de fluxo.

- Segredos locais

  Se sua função Lambda usa segredos locais, especifique `aws.greengrass.SecretManager` como uma dependência de componente ao importar a função. Ao implantar o componente gerenciador de segredos, especifique os recursos secretos a serem implantados nos dispositivos principais de destino. A função de troca de tokens do dispositivo principal deve permitir que o dispositivo principal recupere os recursos secretos a serem implantados. Para ter mais informações, consulte Gerente secreto.

  Ao implantar seu componente de função Lambda, configure-o para ter uma política de autorização de IPC que conceda permissão para usar a operação de GetSecretValue IPC na V2. AWS IoT Device SDK

- Sombras locais

  Se sua função Lambda interagir com sombras locais, você deverá atualizar o código da função Lambda para usar a V2. AWS IoT Device SDK Você também deve especificar `aws.greengrass.ShadowManager` como uma dependência de componente ao importar a função. Para ter mais informações, consulte Interaja com as sombras do dispositivo.

  Ao implantar seu componente de função Lambda, configure-o para ter uma política de autorização de IPC que conceda permissão para usar as operações de IPC paralelas na V2. AWS IoT Device SDK

- Assinaturas

- Se sua função Lambda assinar mensagens de uma fonte na nuvem, especifique essas assinaturas como fontes de eventos ao importar a função.
- Se sua função Lambda assinar mensagens de outra função Lambda, ou se sua função Lambda publicar mensagens em ou AWS IoT Core outras funções do Lambda, configure e implante o componente legado do roteador de assinatura ao implantar sua função Lambda. Ao implantar o componente legado do roteador de assinatura, especifique as assinaturas que a função Lambda usa.

  > **Note**
  >
  > O componente antigo do roteador de assinatura é necessário somente se sua função Lambda usar a `publish()` função no SDK AWS IoT Greengrass principal. Se você atualizar o código da função Lambda para usar a interface de comunicação entre processos (IPC) na AWS IoT Device SDK V2, não precisará implantar o componente legado do roteador de assinatura. Para obter mais informações, consulte os seguintes serviços de comunicação entre processos:
  >
  > - Publique/assine mensagens locais
  > - Publique/assine mensagens MQTT AWS IoT Core

- Se sua função Lambda assinar mensagens de dispositivos conectados locais, especifique essas assinaturas como fontes de eventos ao importar a função. Você também deve configurar e implantar o componente de ponte MQTT para retransmitir mensagens dos dispositivos conectados para os tópicos locais de publicação/assinatura que você especifica como fontes de eventos.
- Se sua função Lambda publicar mensagens em dispositivos conectados locais, você deverá atualizar o código da função Lambda para usar a AWS IoT Device SDK V2 para publicar mensagens locais de publicação/assinatura. Você também deve configurar e implantar o componente de ponte MQTT para retransmitir mensagens do agente de mensagens de publicação/assinatura local para os dispositivos conectados.

- Volumes e dispositivos locais

  Se sua função Lambda em contêiner acessa volumes ou dispositivos locais, especifique esses volumes e dispositivos ao importar a função Lambda. Esse recurso não exige uma dependência de componente.

Para ter mais informações, consulte Executar AWS Lambda funções.

Use conectores V1

Você pode implantar componentes AWS fornecidos que oferecem a mesma funcionalidade de alguns AWS IoT Greengrass conectores. Ao criar a implantação, você pode configurar os parâmetros dos conectores.

Os AWS IoT Greengrass V2 componentes a seguir fornecem a funcionalidade do conector Greengrass V1:

- CloudWatch componente de métricas
- AWS IoT Device Defender componente
- Componente Firehose
- Componente adaptador de protocolo Modbus-RTU
- Componente Amazon SNS

Execute contêineres Docker

AWS IoT Greengrass V2 não fornece um componente para substituir diretamente o conector de implantação do aplicativo V1 Docker. No entanto, você pode usar o componente do gerenciador de aplicativos Docker para baixar imagens do Docker e, em seguida, criar componentes personalizados que executam contêineres do Docker a partir das imagens baixadas. Para obter mais informações, consulte Gerenciador de aplicativos Docker e Execute um contêiner Docker.

Execute inferência de aprendizado de máquina

AWS IoT Greengrass V2 fornece um componente do Amazon SageMaker Edge Manager que instala o agente do Amazon SageMaker Edge Manager e permite que você use modelos SageMaker compilados pelo NEO como componentes do modelo nos dispositivos principais do Greengrass. AWS IoT Greengrass V2 também fornece componentes que instalam o Deep Learning Runtime e o TensorFlow Lite em seu dispositivo. Você pode usar o modelo DLR e o modelo TensorFlow Lite correspondentes e os componentes de inferência para realizar a classificação da imagem da amostra e a inferência de detecção de objetos. Para usar outras estruturas de aprendizado de máquina, como MXNet TensorFlow e, você pode desenvolver seus próprios componentes personalizados que usam essas estruturas.

Conecte dispositivos Greengrass V1

Os dispositivos conectados em AWS IoT Greengrass V1 são chamados de dispositivos cliente em AWS IoT Greengrass V2. AWS IoT Greengrass V2 o suporte para dispositivos cliente é compatível com versões anteriores AWS IoT Greengrass V1, portanto, você pode conectar dispositivos cliente V1 a dispositivos principais V2 sem alterar o código do aplicativo. Para permitir que os dispositivos cliente se conectem a um dispositivo central V2, implante componentes do Greengrass que habilitem o suporte ao dispositivo cliente e associe os dispositivos cliente ao dispositivo principal. Para retransmitir mensagens entre dispositivos cliente, o serviço de AWS IoT Core nuvem e os componentes do Greengrass (incluindo funções Lambda), implante e configure o componente de ponte MQTT. Você pode implantar o componente detector de IP para detectar automaticamente as informações de conectividade ou gerenciar manualmente os endpoints. Para ter mais informações, consulte Interaja com dispositivos IoT locais.

Ativar o serviço paralelo local

Em AWS IoT Greengrass V2, o serviço paralelo local é implementado pelo componente gerenciador AWS de sombra fornecido. AWS IoT Greengrass V2 também inclui suporte para sombras nomeadas. Para permitir que seus componentes interajam com sombras locais e sincronizem estados de sombra AWS IoT Core, configure e implante o componente do gerenciador de sombras e use as operações IPC de sombra no código do componente. Para ter mais informações, consulte Interaja com as sombras do dispositivo.

Integre com AWS IoT SiteWise

Se você usa seu dispositivo V1 core como AWS IoT SiteWise gateway, siga as instruções para configurar seu novo dispositivo V2 core como gateway. AWS IoT SiteWise AWS IoT SiteWise fornece um script de instalação que implanta os AWS IoT SiteWise componentes para você.

Etapa 3: teste seus AWS IoT Greengrass V2 aplicativos

Depois de criar e implantar componentes V2 em seu novo dispositivo V2 core, verifique se seus aplicativos atendem às suas expectativas. Você pode verificar os registros do dispositivo para ver as mensagens de saída padrão (stdout) e erro padrão (stderr) de seus componentes. Para ter mais informações, consulte Monitore AWS IoT Greengrass os registros.

Se você implantou a CLI do Greengrass no dispositivo principal, poderá usá-la para depurar componentes e suas configurações. Para ter mais informações, consulte Comandos da CLI do Greengrass.

Depois de verificar se seus aplicativos funcionam em um dispositivo V2 core, você pode implantar os componentes do Greengrass do seu aplicativo em outros dispositivos principais. Se você desenvolveu componentes personalizados que executam processos nativos ou contêineres Docker, você deve primeiro publicar esses componentes no AWS IoT Greengrass serviço para implantá-los em outros dispositivos principais.

Atualize os dispositivos principais do Greengrass V1 para o Greengrass V2

Depois de verificar se seus aplicativos e componentes funcionam em um dispositivo AWS IoT Greengrass V2 principal, você pode instalar o software AWS IoT Greengrass Core v2.x em seus dispositivos que atualmente executam a versão v1.x, como dispositivos de produção. Em seguida, implante os componentes do Greengrass V2 para executar seus aplicativos do Greengrass nos dispositivos.

Para atualizar uma frota de dispositivos da V1 para a V2, conclua estas etapas para cada dispositivo a ser atualizado. Você pode usar grupos de coisas para implantar componentes V2 em uma frota de dispositivos principais.

> **Tip**
>
> Recomendamos que você crie um script para automatizar o processo de atualização de uma frota de dispositivos. Se você usa AWS Systems Managerpara gerenciar sua frota, você pode usar o Systems Manager para executar esse script em cada dispositivo para atualizar sua frota da V1 para a V2.
> Você pode entrar em contato com seu representante do AWS Enterprise Support se tiver dúvidas sobre a melhor forma de automatizar o processo de upgrade.

Etapa 1: instalar o software AWS IoT Greengrass Core v2.x

Escolha entre as seguintes opções para instalar o software AWS IoT Greengrass Core v2.x em um dispositivo V1 core:

- Atualize em menos etapas

Para atualizar em menos etapas, você pode desinstalar o software v1.x antes de instalar o software v2.x.

- Atualize com o mínimo de tempo de inatividade

  Para atualizar com o mínimo de tempo de inatividade, você pode instalar as duas versões do software AWS IoT Greengrass Core ao mesmo tempo. Depois de instalar o software AWS IoT Greengrass Core v2.x e verificar se seus aplicativos V2 funcionam corretamente, você desinstala o software AWS IoT Greengrass Core v1.x. Antes de escolher essa opção, considere a RAM adicional necessária para executar as duas versões do software AWS IoT Greengrass Core ao mesmo tempo.

Desinstale o AWS IoT Greengrass Core v1.x antes de instalar a v2.x

Se você quiser atualizar sequencialmente, desinstale o software AWS IoT Greengrass Core v1.x antes de instalar a v2.x em seu dispositivo.

Para desinstalar o software AWS IoT Greengrass Core v1.x

1. Se o software AWS IoT Greengrass Core v1.x estiver sendo executado como um serviço, você deverá interromper, desativar e remover o serviço.

   a. Pare a execução do AWS IoT Greengrass serviço Core software v1.x.

   ```
   sudo systemctl stop greengrass
   ```

   b. Espere até que o serviço pare. Você pode usar o `list` comando para verificar o status do serviço.

   ```
   sudo systemctl list-units --type=service | grep greengrass
   ```

   c. Desative o serviço.

   ```
   sudo systemctl disable greengrass
   ```

   d. Remova o serviço.

   ```
   sudo rm /etc/systemd/system/greengrass.service
   ```

2. Se o software AWS IoT Greengrass Core v1.x não estiver sendo executado como um serviço, use o comando a seguir para interromper o daemon. Substitua *greengrass-root pelo nome* da sua pasta raiz do Greengrass. O local-padrão é `/greengrass`.

```
cd /greengrass-root/ggc/core/
sudo ./greengrassd stop
```

3. (Opcional) Faça backup de sua pasta raiz do Greengrass e, se aplicável, de sua pasta de gravação personalizada, em uma pasta diferente no seu dispositivo.

   a. Use o comando a seguir para copiar a pasta raiz atual do Greengrass para uma pasta diferente e, em seguida, remover a pasta raiz.

```
sudo cp -r /greengrass-root /path/to/greengrass-backup
rm -rf /greengrass-root
```

   b. Use o comando a seguir para mover a pasta de gravação para uma pasta diferente e, em seguida, remover a pasta de gravação.

```
sudo cp -r /write-directory /path/to/write-directory-backup
rm -rf /write-directory
```

Em seguida, você pode usar as instruções de instalação AWS IoT Greengrass V2 para instalar o software em seu dispositivo.

> **Tip**
>
> Para reutilizar a identidade de um dispositivo principal ao migrá-lo da V1 para a V2, siga as instruções para instalar o software AWS IoT Greengrass Core com provisionamento manual. Primeiro, remova o software principal V1 do dispositivo e, em seguida, reutilize o item e o certificado do AWS IoT dispositivo núcleo V1 e atualize AWS IoT as políticas do certificado para conceder as permissões exigidas pelo software v2.x.

Instale o software AWS IoT Greengrass Core v2.x em um dispositivo que já esteja executando a v1.x

Se você instalar o software AWS IoT Greengrass Core v2.x em um dispositivo que já esteja executando o software AWS IoT Greengrass Core v1.x, lembre-se do seguinte:

- O AWS IoT nome do item para seu dispositivo principal V2 deve ser exclusivo. Não use o mesmo nome do seu dispositivo principal V1.
- As portas que você usa para o software AWS IoT Greengrass Core v2.x devem ser diferentes das portas que você usa para a v1.x.
  - Configure o gerenciador de fluxo V1 para usar uma porta diferente de 8088. Para obter mais informações, consulte Configurar o gerenciador de streams.
  - Configure o agente V1 MQTT para usar uma porta diferente da 8883. Para obter mais informações, consulte Configurar a porta MQTT para mensagens locais.
- AWS IoT Greengrass V2não oferece a opção de renomear o serviço do sistema Greengrass. Se você executa o Greengrass como um serviço do sistema, você deve fazer o seguinte para evitar nomes conflitantes de serviços do sistema:
  - Renomeie o serviço Greengrass para v1.x antes de instalar a v2.x.
  - Instale o software AWS IoT Greengrass Core v2.x sem um serviço do sistema e, em seguida, configure manualmente o software como um serviço do sistema com um nome diferente de. `greengrass`

  Para renomear o serviço Greengrass para v1.x

  1. Pare o AWS IoT Greengrass serviço Core software v1.x.

     ```
     sudo systemctl stop greengrass
     ```

  2. Aguarde até que o serviço pare. O serviço pode levar alguns minutos para ser interrompido. Você pode usar o `list-units` comando para verificar se o serviço foi interrompido.

     ```
     sudo systemctl list-units --type=service | grep greengrass
     ```

  3. Desative o serviço.

     ```
     sudo systemctl disable greengrass
     ```

  4. Renomeie o serviço.

```
sudo mv /etc/systemd/system/greengrass.service /etc/systemd/system/greengrass-v1.service
```

5. Recarregue o serviço e inicie-o.

```
sudo systemctl daemon-reload
sudo systemctl reset-failed
sudo systemctl enable greengrass-v1
sudo systemctl start greengrass-v1
```

Em seguida, você pode usar as instruções de instalação AWS IoT Greengrass V2 para instalar o software em seu dispositivo.

> **Tip**
>
> Para reutilizar a identidade de um dispositivo principal ao migrá-lo da V1 para a V2, siga as instruções para instalar o software AWS IoT Greengrass Core com provisionamento manual. Primeiro, remova o software principal V1 do dispositivo e, em seguida, reutilize o item e o certificado do AWS IoT dispositivo núcleo V1 e atualize AWS IoT as políticas do certificado para conceder as permissões exigidas pelo software v2.x.

Etapa 2: implantar AWS IoT Greengrass V2 componentes nos dispositivos principais

Depois de instalar o software AWS IoT Greengrass Core v2.x em seu dispositivo, crie uma implantação que inclua os seguintes recursos. Para implantar componentes em uma frota de dispositivos similares, crie uma implantação para um grupo de coisas que contenha esses dispositivos.

- Componentes da função Lambda que você criou a partir de suas funções Lambda V1. Para ter mais informações, consulte Executar AWS Lambda funções.
- Se você usa assinaturas V1, o componente antigo do roteador de assinatura.
- Se você usa o gerenciador de fluxo, o componente do gerenciador de fluxo. Para ter mais informações, consulte Gerencie fluxos de dados nos dispositivos principais do Greengrass.
- Se você usa segredos locais, o componente gerenciador de segredos.

- Se você usa conectores V1, os componentes do AWSconector fornecidos.
- Se você usa contêineres do Docker, o componente gerenciador de aplicativos do Docker. Para ter mais informações, consulte Execute um contêiner Docker.
- Se você usa inferência de aprendizado de máquina, componentes para suporte ao aprendizado de máquina. Para ter mais informações, consulte Executar a inferência de machine learning.
- Se você usa dispositivos conectados, os componentes do dispositivo cliente são compatíveis. Você também deve ativar o suporte ao dispositivo cliente e associar os dispositivos cliente ao seu dispositivo principal. Para ter mais informações, consulte Interaja com dispositivos IoT locais.
- Se você usa sombras do dispositivo, o componente gerenciador de sombras. Para ter mais informações, consulte Interaja com as sombras do dispositivo.
- Se você fizer upload de registros dos dispositivos principais do Greengrass para o Amazon CloudWatch Logs, o componente do gerenciador de registros. Para ter mais informações, consulte Monitore AWS IoT Greengrass os registros.
- Se você fizer a integração comAWS IoT SiteWise, siga as instruções para configurar o dispositivo principal V2 como um AWS IoT SiteWise gateway. AWS IoT SiteWisefornece um script de instalação que implanta os AWS IoT SiteWise componentes para você.
- Componentes definidos pelo usuário que você desenvolveu para implementar funcionalidades personalizadas.

Para obter informações sobre como criar e revisar implantações, consulte. Implemente AWS IoT Greengrass componentes em dispositivos

Tutorial: Conceitos básicos do AWS IoT Greengrass V2

Você pode concluir este tutorial de introdução para aprender os recursos básicos doAWS IoT Greengrass V2. Neste tutorial, você faz o seguinte:

1. Instale e configure o software AWS IoT Greengrass Core em um dispositivo Linux, como um Raspberry Pi ou um dispositivo Windows. Este dispositivo é um dispositivo principal do Greengrass.
2. Desenvolva um componente Hello World em seu dispositivo principal do Greengrass. Os componentes são módulos de software que são executados nos dispositivos principais do Greengrass.
3. Faça o upload desse componente AWS IoT Greengrass V2 noNuvem AWS.
4. Implante esse componente do Nuvem AWS para o seu dispositivo principal do Greengrass.

> **Note**
>
> Este tutorial descreve como configurar um ambiente de desenvolvimento e explorar os recursos doAWS IoT Greengrass. Para obter mais informações sobre como instalar e configurar dispositivos de produção, consulte o seguinte:
>
> - Configurando dispositivos AWS IoT Greengrass principais
> - Instalar o software do AWS IoT Greengrass Core

Você pode esperar passar de 20 a 30 minutos neste tutorial.

Tópicos

- Próximas etapas

Pré-requisitos

Para concluir este tutorial de conceitos básicos, você precisa:

- Uma Conta da AWS. Se você não tiver uma, consulte Etapa 1: configurar uma AWS conta.
- O uso de um Região da AWSque suporteAWS IoT Greengrass V2. Para obter a lista das regiões com suporte, consulte Endpoints e cotas do AWS IoT Greengrass V2 na Referência geral da AWS.
- Um usuário AWS Identity and Access Management (IAM) com permissões de administrador.
- Um dispositivo a ser configurado como um dispositivo principal do Greengrass, como um Raspberry Pi com sistema operacional Raspberry Pi (anteriormente chamado de Raspbian) ou um dispositivo Windows 10. Você deve ter permissões de administrador neste dispositivo ou a capacidade de adquirir privilégios de administrador, como por meio `sudo` de. Este dispositivo deve ter uma conexão com a internet.

  Você também pode optar por usar um dispositivo diferente que atenda aos requisitos para instalar e executar o software AWS IoT Greengrass Core. Para ter mais informações, consulte Plataformas compatíveis e requisitos.

  Se o seu computador de desenvolvimento atender a esses requisitos, você poderá configurá-lo como seu dispositivo principal do Greengrass neste tutorial.
- Python 3.5 ou posterior instalado para todos os usuários no dispositivo e adicionado à variável de `PATH` ambiente. No Windows, você também deve ter o Python Launcher para Windows instalado para todos os usuários.

  > **Important**
  >
  > No Windows, o Python não é instalado para todos os usuários por padrão. Ao instalar o Python, você deve personalizar a instalação para configurá-la para que o software AWS IoT Greengrass Core execute scripts em Python. Por exemplo, se você usa o instalador gráfico do Python, faça o seguinte:
  >
  > 1. Selecione Instalar lançador para todos os usuários (recomendado).
  > 2. Selecione Customize installation.
  > 3. Selecione Next.

4. Selecione Install for all users.
5. Selecione Add Python to environment variables.
6. Escolha Instalar.

Para obter mais informações, consulte Usando o Python no Windows na documentação do Python 3.

- AWS Command Line Interface(AWS CLI) instalado e configurado com credenciais em seu computador de desenvolvimento e em seu dispositivo. Certifique-se de usar o mesmo Região da AWS para configurar o AWS CLI no seu computador de desenvolvimento e no seu dispositivo. Para usar AWS IoT Greengrass V2 com oAWS CLI, você deve ter uma das seguintes versões ou posteriores:
  - Versão AWS CLI V1 mínima: v1.18.197
  - Versão AWS CLI V2 mínima: v2.1.11

  Tip

  Você pode executar o comando a seguir para verificar a versão do AWS CLI que você tem.

  ```
  aws --version
  ```

  Para obter mais informações, consulte Instalando, atualizando e desinstalando AWS CLI e Configurando o AWS CLI no Guia do AWS Command Line InterfaceUsuário.

  Note

  Se você usa um dispositivo ARM de 32 bits, como um Raspberry Pi com sistema operacional de 32 bits, instale a V1. AWS CLI AWS CLI A V2 não está disponível para dispositivos ARM de 32 bits. Para obter mais informações, consulte Instalando, atualizando e desinstalando a AWS CLI versão 1.

Etapa 1: configurar uma AWS conta

Inscreva-se para um Conta da AWS

Se você não tiver um Conta da AWS, conclua as etapas a seguir para criar um.

Para se inscrever em um Conta da AWS

1. Abra https://portal.aws.amazon.com/billing/signup.
2. Siga as instruções on-line.

   Parte do procedimento de inscrição envolve receber uma chamada telefônica e digitar um código de verificação no teclado do telefone.

   Quando você se inscreve em um Conta da AWS, um Usuário raiz da conta da AWSé criado. O usuário-raiz tem acesso a todos os Serviços da AWS e recursos na conta. Como uma prática recomendada de segurança, atribua o acesso administrativo para um usuário e use somente o usuário-raiz para executar tarefas que requerem o acesso de usuário-raiz.

AWS envia um e-mail de confirmação após a conclusão do processo de inscrição. A qualquer momento, é possível visualizar as atividades da conta atual e gerenciar sua conta acessando https://aws.amazon.com/ e selecionando Minha conta.

Criar um usuário com acesso administrativo

Depois de se inscrever em um Conta da AWS, proteja seu Usuário raiz da conta da AWS AWS IAM Identity Center, habilite e crie um usuário administrativo para que você não use o usuário root nas tarefas diárias.

Proteja seu Usuário raiz da conta da AWS

1. Faça login AWS Management Consolecomo proprietário da conta escolhendo Usuário raiz e inserindo seu endereço de Conta da AWS e-mail. Na próxima página, digite sua senha.

   Para obter ajuda ao fazer login usando o usuário-raiz, consulte Signing in as the root user (Fazer login como usuário-raiz) no Guia do usuário do Início de Sessão da AWS .
2. Habilite a autenticação multifator (MFA) para o usuário-raiz.

Para obter instruções, consulte Habilitar um dispositivo de MFA virtual para seu usuário Conta da AWS raiz (console) no Guia do usuário do IAM.

Criar um usuário com acesso administrativo

1. Habilitar o IAM Identity Center.

   Para obter instruções, consulte Habilitar AWS IAM Identity Center no Guia do usuário do AWS IAM Identity Center .

2. No Centro de Identidade do IAM, conceda o acesso administrativo para um usuário.

   Para ver um tutorial sobre como usar o Diretório do Centro de Identidade do IAM como fonte de identidade, consulte Configurar o acesso do usuário com o padrão Diretório do Centro de Identidade do IAM no Guia AWS IAM Identity Center do usuário.

Iniciar sessão como o usuário com acesso administrativo

- Para fazer login com seu usuário do Centro de Identidade do IAM, use a URL de login que foi enviada ao seu endereço de e-mail quando você criou o usuário do Centro do Usuário do IAM.

  Para obter ajuda para fazer login usando um usuário do IAM Identity Center, consulte Como fazer login no portal de AWS acesso no Guia Início de Sessão da AWS do usuário.

Atribuir acesso para usuários adicionais

1. No Centro de Identidade do IAM, crie um conjunto de permissões que siga as práticas recomendadas de aplicação de permissões com privilégio mínimo.

   Para obter instruções, consulte Create a permission set no Guia do usuário do AWS IAM Identity Center .

2. Atribua usuários a um grupo e, em seguida, atribua o acesso de autenticação única ao grupo.

   Para obter instruções, consulte Add groups no Guia do usuário do AWS IAM Identity Center .

Etapa 2: configurar seu ambiente

Siga as etapas nesta seção para configurar um dispositivo Linux ou Windows para ser usado como seu dispositivo AWS IoT Greengrass principal.

Configurar um dispositivo Linux (Raspberry Pi)

Essas etapas pressupõem que você use um Raspberry Pi com o sistema operacional Raspberry Pi. Se você usa um dispositivo ou sistema operacional diferente, consulte a documentação relevante do seu dispositivo.

Para configurar um Raspberry Pi para AWS IoT Greengrass V2

1. Ative o SSH no seu Raspberry Pi para se conectar remotamente a ele. Para obter mais informações, consulte SSH (Secure shell) na documentação do Raspberry Pi.
2. Encontre o endereço IP do seu Raspberry Pi para se conectar a ele com SSH. Para fazer isso, você pode executar o seguinte comando no seu Raspberry Pi.

   ```
   hostname -I
   ```
3. Conecte-se ao seu Raspberry Pi com SSH.

   No seu computador de desenvolvimento, execute o comando a seguir. *Substitua o nome de usuário pelo nome do usuário a ser conectado e pi-ip-addresssubstitua pelo endereço IP que você encontrou na etapa anterior.*

   ```
   ssh username@pi-ip-address
   ```

   > **Important**
   >
   > Se o seu computador de desenvolvimento usa uma versão anterior do Windows, talvez você não tenha o `ssh` comando ou talvez tenha, `ssh` mas não consiga se conectar ao seu Raspberry Pi. Para se conectar ao seu Raspberry Pi, você pode instalar e configurar o PuTTY, que é um cliente SSH de código aberto e gratuito. Consulte a documentação do PuTTY para se conectar ao seu Raspberry Pi.
4. Instale o Java Runtime, que o software AWS IoT Greengrass Core exige para ser executado. No seu Raspberry Pi, use os seguintes comandos para instalar o Java 11.

```
sudo apt install default-jdk
```

Quando a instalação for concluída, execute o comando a seguir para verificar se o Java é executado no seu Raspberry Pi.

```
java -version
```

O comando imprime a versão do Java que é executada no dispositivo. A saída pode ser semelhante ao exemplo a seguir.

```
openjdk version "11.0.9.1" 2020-11-04
OpenJDK Runtime Environment (build 11.0.9.1+1-post-Debian-1deb10u2)
OpenJDK 64-Bit Server VM (build 11.0.9.1+1-post-Debian-1deb10u2, mixed mode)
```

Dica: defina os parâmetros do kernel em um Raspberry Pi

Se o seu dispositivo for um Raspberry Pi, você poderá concluir as etapas a seguir para visualizar e atualizar os parâmetros do kernel Linux:

1. Abra o arquivo `/boot/cmdline.txt`. Esse arquivo especifica os parâmetros do kernel Linux a serem aplicados quando o Raspberry Pi for inicializado.

   Por exemplo, em um sistema baseado em Linux, você pode executar o seguinte comando para usar o GNU nano para abrir o arquivo.

   ```
   sudo nano /boot/cmdline.txt
   ```

2. Verifique se o `/boot/cmdline.txt` arquivo contém os seguintes parâmetros do kernel. O `systemd.unified_cgroup_hierarchy=0` parâmetro especifica o uso de cgroups v1 em vez de cgroups v2.

   ```
   cgroup_enable=memory cgroup_memory=1 systemd.unified_cgroup_hierarchy=0
   ```

   Se o `/boot/cmdline.txt` arquivo não contiver esses parâmetros ou contiver esses parâmetros com valores diferentes, atualize o arquivo para conter esses parâmetros e valores.

3. Se você atualizou o `/boot/cmdline.txt` arquivo, reinicie o Raspberry Pi para aplicar as alterações.

```
sudo reboot
```

Configurar um dispositivo Linux (outro)

Para configurar um dispositivo Linux para AWS IoT Greengrass V2

1. Instale o Java Runtime, que o software AWS IoT Greengrass Core exige para ser executado. Recomendamos que você use as versões de suporte de longo prazo do Amazon Corretto ou do OpenJDK. É necessária a versão 8 ou superior. Os comandos a seguir mostram como instalar o OpenJDK no seu dispositivo.

   - Para distribuições com base em Debian ou em Ubuntu:

     ```
     sudo apt install default-jdk
     ```

   - Para distribuições com base em Red Hat:

     ```
     sudo yum install java-11-openjdk-devel
     ```

   - Para Amazon Linux 2:

     ```
     sudo amazon-linux-extras install java-openjdk11
     ```

   - Para Amazon Linux 2023:

     ```
     sudo dnf install java-11-amazon-corretto -y
     ```

   Quando a instalação for concluída, execute o comando a seguir para verificar se o Java é executado no seu dispositivo Linux.

   ```
   java -version
   ```

   O comando imprime a versão do Java que é executada no dispositivo. Por exemplo, em uma distribuição baseada em Debian, o resultado pode ser semelhante ao exemplo a seguir.

```
openjdk version "11.0.9.1" 2020-11-04
OpenJDK Runtime Environment (build 11.0.9.1+1-post-Debian-1deb10u2)
OpenJDK 64-Bit Server VM (build 11.0.9.1+1-post-Debian-1deb10u2, mixed mode)
```

2. (Opcional) Crie o usuário e o grupo padrão do sistema que executa componentes no dispositivo. Você também pode optar por permitir que o instalador do software AWS IoT Greengrass Core crie esse usuário e grupo durante a instalação com o argumento do `--component-default-user` instalador. Para ter mais informações, consulte Argumentos de instalação.

```
sudo useradd --system --create-home ggc_user
sudo groupadd --system ggc_group
```

3. Verifique se o usuário que executa o software AWS IoT Greengrass Core (normalmente`root`) tem permissão para executar `sudo` com qualquer usuário e qualquer grupo.

   a. Execute o comando a seguir para abrir o `/etc/sudoers` arquivo.

   ```
   sudo visudo
   ```

   b. Verifique se a permissão para o usuário se parece com o exemplo a seguir.

   ```
   root    ALL=(ALL:ALL) ALL
   ```

4. (Opcional) Para executar funções Lambda em contêineres, você deve habilitar cgroups v1 e habilitar e montar os cgroups de memória e dispositivos. Se você não planeja executar funções Lambda em contêineres, pode pular esta etapa.

   Para habilitar essas opções de cgroups, inicialize o dispositivo com os seguintes parâmetros do kernel Linux.

   ```
   cgroup_enable=memory cgroup_memory=1 systemd.unified_cgroup_hierarchy=0
   ```

   Para obter informações sobre como visualizar e definir os parâmetros do kernel para seu dispositivo, consulte a documentação do sistema operacional e do carregador de inicialização. Siga as instruções para definir permanentemente os parâmetros do kernel.

5. Instale todas as outras dependências necessárias em seu dispositivo, conforme indicado na lista de requisitos emRequisitos do dispositivo.

Configurar um dispositivo Windows

Para configurar um dispositivo Windows para AWS IoT Greengrass V2

1. Instale o Java Runtime, que o software AWS IoT Greengrass Core exige para ser executado. Recomendamos que você use as versões de suporte de longo prazo do Amazon Corretto ou do OpenJDK. É necessária a versão 8 ou superior.
2. Verifique se o Java está disponível na variável de sistema PATH e, caso contrário, adicione-a. A LocalSystem conta executa o software AWS IoT Greengrass Core, então você deve adicionar Java à variável de sistema PATH em vez da variável de usuário PATH para seu usuário. Faça o seguinte:

   a. Pressione a tecla Windows para abrir o menu Iniciar.

   b. Digite **`environment variables`** para pesquisar as opções do sistema no menu Iniciar.

   c. Nos resultados da pesquisa do menu Iniciar, escolha Editar as variáveis de ambiente do sistema para abrir a janela Propriedades do sistema.

   d. Escolha variáveis de ambiente... para abrir a janela Variáveis de ambiente.

   e. Em Variáveis do sistema, selecione Caminho e, em seguida, escolha Editar. Na janela Editar variável de ambiente, você pode visualizar cada caminho em uma linha separada.

   f. Verifique se o caminho para a `bin` pasta da instalação do Java está presente. O caminho pode ser semelhante ao exemplo a seguir.

   ```
   C:\\Program Files\\Amazon Corretto\\jdk11.0.13_8\\bin
   ```

   g. Se a `bin` pasta da instalação do Java estiver ausente do Path, escolha Novo para adicioná-la e, em seguida, escolha OK.
3. Abra o prompt de comando do Windows (`cmd.exe`) como administrador.
4. Crie o usuário padrão na LocalSystem conta no dispositivo Windows. *`Substitua a senha`* por uma senha segura.

   ```
   net user /add ggc_user password
   ```

   > **Tip**
   >
   > Dependendo da configuração do Windows, a senha do usuário pode ser definida para expirar em uma data futura. Para garantir que seus aplicativos Greengrass continuem

operando, monitore quando a senha expira e atualize-a antes que ela expire. Você também pode definir a senha do usuário para nunca expirar.

- Para verificar quando um usuário e sua senha expiram, execute o comando a seguir.

```
net user ggc_user | findstr /C:expires
```

- Para definir que a senha de um usuário nunca expire, execute o comando a seguir.

```
wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
```

- Se você estiver usando o Windows 10 ou posterior, onde o wmiccomando está obsoleto, execute o comando a seguir. PowerShell

```
Get-CimInstance -Query "SELECT * from Win32_UserAccount WHERE name =
 'ggc_user'" | Set-CimInstance -Property @{PasswordExpires="False"}
```

5. Baixe e instale o PsExecutilitário da Microsoft no dispositivo.
6. Use o PsExec utilitário para armazenar o nome de usuário e a senha do usuário padrão na instância do Credential Manager da LocalSystem conta. Substitua a *senha* pela senha do usuário que você definiu anteriormente.

```
psexec -s cmd /c cmdkey /generic:ggc_user /user:ggc_user /pass:password
```

Se for PsExec License Agreementaberto, opte por Acceptconcordar com a licença e execute o comando.

> **Note**
>
> Em dispositivos Windows, a LocalSystem conta executa o núcleo Greengrass, e você deve usar o PsExec utilitário para armazenar as informações padrão do usuário na conta. LocalSystem O uso do aplicativo Credential Manager armazena essas informações na conta do Windows do usuário atualmente conectado, em vez da LocalSystem conta.

Etapa 3: Instalar o software doAWS IoT Greengrass Performance

Siga as etapas desta seção para configurar o Raspberry Pi da como dispositivoAWS IoT Greengrass principal que você pode usar para o desenvolvimento local. Nesta seção, você baixa e executa um instalador que faz o seguinte para configurar o softwareAWS IoT Greengrass Core para seu dispositivo:

- Instala o componente do núcleo Greengrass. O núcleo é um componente obrigatório e é o requisito mínimo para executar o softwareAWS IoT Greengrass Core em um dispositivo. Para obter mais informações, consulte Greengrass do Performance.
- Registra seu dispositivo como umaAWS IoT coisa e baixa um certificado digital que permite que seu dispositivo se conecteAWS. Para obter mais informações, consulte Autorização e autenticação do dispositivo para o AWS IoT Greengrass.
- Adiciona aAWS IoT coisa do dispositivo a um grupo de coisas, que é um grupo ou uma frota deAWS IoT coisas. Os grupos Thing permitem que você gerencie frotas de dispositivos principais do Greengrass. Ao implantar componentes de software em seus dispositivos, você pode optar por implantá-los em dispositivos individuais ou em grupos de dispositivos. Para obter mais informações, consulte Gerenciando dispositivos comAWS IoT o Guia doAWS IoT Core desenvolvedor.
- Cria a função IAM que permite que seu dispositivo principal do Greengrass interaja comAWS os serviços. Por padrão, essa função permite que seu dispositivo interajaAWS IoT e envie registros para o AmazonCloudWatch Logs. Para obter mais informações, consulte Autorize os dispositivos principais a interagir com os serviços AWS.
- Instala a interface de linha deAWS IoT Greengrass comando (`greengrass-cli`), que você pode usar para testar componentes personalizados que você desenvolve no dispositivo principal. Para obter mais informações, consulte Interface de linha de comando do Greengrass.

Instale o software AWS IoT Greengrass Core (console)

1. Faça login no AWS IoT Greengrassconsole.
2. Em Começar com o Greengrass, escolha Configurar um dispositivo principal.
3. Em Etapa 1: Registrar um dispositivo principal do Greengrass, em Nome do dispositivo principal, insira o nome do dispositivo AWS IoT principal do Greengrass. Se a coisa não existir, o instalador a cria.

4. Em Etapa 2: Adicionar a um grupo de coisas para aplicar uma implantação contínua, em Thing group, escolha o grupo de AWS IoT coisas ao qual você deseja adicionar seu dispositivo principal.

   - Se você selecionar Inserir um novo nome de grupo, em Nome do grupo Thing, insira o nome do novo grupo a ser criado. O instalador cria o novo grupo para você.
   - Se você selecionar Selecionar um grupo existente, em Nome do grupo Thing, escolha o grupo existente que você deseja usar.
   - Se você selecionar Nenhum grupo, o instalador não adicionará o dispositivo principal a um grupo de coisas.

5. Em Etapa 3: Instalar o software Greengrass Core, conclua as etapas a seguir.

   a. Escolha o sistema operacional do seu dispositivo principal: Linux ou Windows.

   b. Forneça suas AWS credenciais ao dispositivo para que o instalador possa provisionar os recursos do IAM AWS IoT e do IAM para seu dispositivo principal. Para aumentar a segurança, recomendamos que você obtenha credenciais temporárias para uma função do IAM que permita somente as permissões mínimas necessárias para provisionar. Para ter mais informações, consulte Política mínima de IAM para o instalador provisionar recursos.

      > **Note**
      >
      > O instalador não salva nem armazena suas credenciais.

      No seu dispositivo, faça o seguinte para recuperar as credenciais e disponibilizá-las para o AWS IoT Greengrass instalador do software Core:

      - (Recomendado) Use credenciais temporárias de AWS IAM Identity Center

        i. Forneça o ID da chave de acesso, a chave de acesso secreta e o token da sessão do IAM Identity Center. Para obter mais informações, consulte Atualização manual de credenciais em Obter e atualizar credenciais temporárias no guia do usuário do IAM Identity Center.

        ii. Execute os comandos a seguir para fornecer as credenciais para o software AWS IoT Greengrass principal.

Linux or Unix

```
export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/
bPxRfiCYEXAMPLEKEY
export AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o5OytwEXAMPLE=
```

Windows Command Prompt (CMD)

```
set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
set AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o5OytwEXAMPLE=
```

PowerShell

```
$env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
$env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/
bPxRfiCYEXAMPLEKEY"
$env:AWS_SESSION_TOKEN="AQoDYXdzEJr1K...o5OytwEXAMPLE="
```

- Use credenciais de segurança temporárias de uma função do IAM:

  i. Forneça o ID da chave de acesso, a chave de acesso secreta e o token da sessão de uma função do IAM que você assume. Para obter mais informações sobre como recuperar essas credenciais, consulte Solicitação de credenciais de segurança temporárias no Guia do usuário do IAM.

  ii. Execute os comandos a seguir para fornecer as credenciais para o software AWS IoT Greengrass principal.

  Linux or Unix

```
export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/
bPxRfiCYEXAMPLEKEY
export AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o5OytwEXAMPLE=
```

  Windows Command Prompt (CMD)

```
set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
```

```
set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
set AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o5OytwEXAMPLE=
```

PowerShell

```
$env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
$env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/
bPxRfiCYEXAMPLEKEY"
$env:AWS_SESSION_TOKEN="AQoDYXdzEJr1K...o5OytwEXAMPLE="
```

- Use credenciais de longo prazo de um usuário do IAM:

  i. Forneça o ID da chave de acesso e a chave de acesso secreta para seu usuário do IAM. Você pode criar um usuário do IAM para provisionamento que você excluirá posteriormente. Para saber a política do IAM a ser fornecida ao usuário, consulte Política mínima de IAM para o instalador provisionar recursos. Para obter mais informações sobre como recuperar credenciais de longo prazo, consulte Gerenciamento de chaves de acesso para usuários do IAM no Guia do usuário do IAM.

  ii. Execute os comandos a seguir para fornecer as credenciais para o software AWS IoT Greengrass principal.

  Linux or Unix

  ```
  export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
  export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/
  bPxRfiCYEXAMPLEKEY
  ```

  Windows Command Prompt (CMD)

  ```
  set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
  set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
  ```

  PowerShell

  ```
  $env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
  $env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/
  bPxRfiCYEXAMPLEKEY"
  ```

iii. (Opcional) Se você criou um usuário do IAM para provisionar seu dispositivo Greengrass, exclua o usuário.

iv. (Opcional) Se você usou o ID da chave de acesso e a chave de acesso secreta de um usuário do IAM existente, atualize as chaves do usuário para que elas não sejam mais válidas. Para obter mais informações, consulte Atualização das chaves de acesso no guia AWS Identity and Access Management do usuário.

c. Em Executar o instalador, conclua as etapas a seguir.

i. Em Baixar o instalador, escolha Copiar e execute o comando copiado no seu dispositivo principal. Esse comando baixa a versão mais recente do software AWS IoT Greengrass Core e a descompacta no seu dispositivo.

ii. Em Executar o instalador, escolha Copiar e execute o comando copiado no seu dispositivo principal. Esse comando usa os nomes de AWS IoT coisas e grupos de coisas que você especificou anteriormente para executar o instalador do software AWS IoT Greengrass Core e configurar AWS recursos para seu dispositivo principal.

Esse comando também faz o seguinte:

- Configure o software AWS IoT Greengrass Core como um serviço do sistema que é executado na inicialização. Em dispositivos Linux, isso requer o sistema init Systemd.

> **Important**
>
> Nos dispositivos principais do Windows, você deve configurar o software AWS IoT Greengrass Core como um serviço do sistema.

- Implante o componente AWS IoT Greengrass CLI, que é uma ferramenta de linha de comando que permite desenvolver componentes personalizados do Greengrass no dispositivo principal.
- Especifique o uso do usuário do `ggc_user` sistema para executar componentes de software no dispositivo principal. Em dispositivos Linux, esse comando também especifica o uso do grupo do `ggc_group` sistema, e o instalador cria o usuário e o grupo do sistema para você.

Ao executar esse comando, você deve ver as seguintes mensagens para indicar que o instalador foi bem-sucedido.

```
Successfully configured Nucleus with provisioned resource details!
Configured Nucleus to deploy aws.greengrass.Cli component
Successfully set up Nucleus as a system service
```

> **Note**
>
> Se você tiver um dispositivo Linux e ele não tiver systemd, o instalador não configurará o software como um serviço do sistema e você não verá a mensagem de sucesso da configuração do núcleo como um serviço do sistema.

Instale o software AWS IoT Greengrass principal (CLI)

Para instalar e configurar o software AWS IoT Greengrass Core

1. No seu dispositivo principal do Greengrass, execute o comando a seguir para alternar para o diretório inicial.

   Linux or Unix

   ```
   cd ~
   ```

   Windows Command Prompt (CMD)

   ```
   cd %USERPROFILE%
   ```

   PowerShell

   ```
   cd ~
   ```

2. Em seu dispositivo principal, baixe o software AWS IoT Greengrass Core para um arquivo chamado`greengrass-nucleus-latest.zip`.

Linux or Unix

```
curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-
latest.zip > greengrass-nucleus-latest.zip
```

Windows Command Prompt (CMD)

```
curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-
latest.zip > greengrass-nucleus-latest.zip
```

PowerShell

```
iwr -Uri https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-
latest.zip -OutFile greengrass-nucleus-latest.zip
```

Ao fazer download desse software, você concorda com o Contrato de licença do software do Greengrass Core.

3. Descompacte o software AWS IoT Greengrass Core em uma pasta no seu dispositivo. *GreengrassInstaller*Substitua pela pasta que você deseja usar.

   Linux or Unix

   ```
   unzip greengrass-nucleus-latest.zip -d GreengrassInstaller && rm greengrass-
   nucleus-latest.zip
   ```

   Windows Command Prompt (CMD)

   ```
   mkdir GreengrassInstaller && tar -xf greengrass-nucleus-latest.zip -
   C GreengrassInstaller && del greengrass-nucleus-latest.zip
   ```

   PowerShell

   ```
   Expand-Archive -Path greengrass-nucleus-latest.zip -DestinationPath .\
   \GreengrassInstaller
   rm greengrass-nucleus-latest.zip
   ```

4. Execute o comando a seguir para iniciar o instalador do software AWS IoT Greengrass Core. Esse comando faz o seguinte:

- Crie os AWS recursos que o dispositivo principal precisa para operar.
- Configure o software AWS IoT Greengrass Core como um serviço do sistema que é executado na inicialização. Em dispositivos Linux, isso requer o sistema init Systemd.

> **Important**
>
> Nos dispositivos principais do Windows, você deve configurar o software AWS IoT Greengrass Core como um serviço do sistema.

- Implante o componente AWS IoT Greengrass CLI, que é uma ferramenta de linha de comando que permite desenvolver componentes personalizados do Greengrass no dispositivo principal.
- Especifique o uso do usuário do `ggc_user` sistema para executar componentes de software no dispositivo principal. Em dispositivos Linux, esse comando também especifica o uso do grupo do `ggc_group` sistema, e o instalador cria o usuário e o grupo do sistema para você.

Substitua os valores dos argumentos em seu comando da seguinte maneira.

a. *`/greengrass/v2`*ou *`C:\greengrass\v2`*: o caminho para a pasta raiz a ser usada para instalar o software AWS IoT Greengrass Core.

b. *`GreengrassInstaller`*. O caminho para a pasta em que você descompactou o instalador do software AWS IoT Greengrass Core.

c. *`região`*. O Região da AWS no qual encontrar ou criar recursos.

d. *`MyGreengrassCore`*. O nome da AWS IoT coisa para o seu dispositivo principal do Greengrass. Se a coisa não existir, o instalador a cria. O instalador baixa os certificados para autenticar a AWS IoT coisa. Para ter mais informações, consulte Autorização e autenticação do dispositivo para o AWS IoT Greengrass.

> **Note**
>
> O nome da coisa não pode conter caracteres de dois pontos (`:`).

e. *`MyGreengrassCoreGroup`*. O nome do grupo de AWS IoT coisas do seu dispositivo principal do Greengrass. Se o grupo de coisas não existir, o instalador o cria e adiciona a coisa a ele. Se o grupo de coisas existir e tiver uma implantação ativa, o dispositivo principal baixará e executará o software especificado pela implantação.

> **Note**
>
> O nome do grupo de coisas não pode conter caracteres de dois pontos (`:`).

f. *`Greengrass ThingPolicy`* V2 IoT. O nome da AWS IoT política que permite que os dispositivos principais do Greengrass se comuniquem com e. AWS IoT AWS IoT Greengrass Se a AWS IoT política não existir, o instalador cria uma AWS IoT política permissiva com esse nome. Você pode restringir as permissões dessa política para seu caso de uso. Para ter mais informações, consulte AWS IoTPolítica mínima para dispositivos AWS IoT Greengrass V2 principais.

g. *`Greengrass TokenExchangeRole`* V2. O nome da função do IAM que permite que o dispositivo principal do Greengrass obtenha credenciais temporáriasAWS. Se a função não existir, o instalador a cria, cria e anexa uma política chamada*`GreengrassV2TokenExchangeRole`*`Access`. Para ter mais informações, consulte Autorize os dispositivos principais a interagir com os serviços AWS.

h. *`GreengrassCoreTokenExchangeRoleAlias`*. O alias para a função do IAM que permite que o dispositivo principal do Greengrass obtenha credenciais temporárias posteriormente. Se o alias da função não existir, o instalador o cria e o aponta para a função do IAM especificada por você. Para ter mais informações, consulte Autorize os dispositivos principais a interagir com os serviços AWS.

Linux or Unix

```
sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE \
  -jar ./GreengrassInstaller/lib/Greengrass.jar \
  --aws-region region \
  --thing-name MyGreengrassCore \
  --thing-group-name MyGreengrassCoreGroup \
  --thing-policy-name GreengrassV2IoTThingPolicy \
  --tes-role-name GreengrassV2TokenExchangeRole \
  --tes-role-alias-name GreengrassCoreTokenExchangeRoleAlias \
  --component-default-user ggc_user:ggc_group \
  --provision true \
  --setup-system-service true \
  --deploy-dev-tools true
```

Windows Command Prompt (CMD)

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" ^
  -jar ./GreengrassInstaller/lib/Greengrass.jar ^
  --aws-region region ^
  --thing-name MyGreengrassCore ^
  --thing-group-name MyGreengrassCoreGroup ^
  --thing-policy-name GreengrassV2IoTThingPolicy ^
  --tes-role-name GreengrassV2TokenExchangeRole ^
  --tes-role-alias-name GreengrassCoreTokenExchangeRoleAlias ^
  --component-default-user ggc_user ^
  --provision true ^
  --setup-system-service true ^
  --deploy-dev-tools true
```

PowerShell

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" `
  -jar ./GreengrassInstaller/lib/Greengrass.jar `
  --aws-region region `
  --thing-name MyGreengrassCore `
  --thing-group-name MyGreengrassCoreGroup `
  --thing-policy-name GreengrassV2IoTThingPolicy `
  --tes-role-name GreengrassV2TokenExchangeRole `
  --tes-role-alias-name GreengrassCoreTokenExchangeRoleAlias `
  --component-default-user ggc_user `
  --provision true `
  --setup-system-service true `
  --deploy-dev-tools true
```

Note

Se você estiver executando AWS IoT Greengrass em um dispositivo com memória limitada, poderá controlar a quantidade de memória que o software AWS IoT Greengrass Core usa. Para controlar a alocação de memória, você pode definir as opções de tamanho da pilha da JVM no parâmetro de `jvmOptions` configuração em seu componente de núcleo. Para ter mais informações, consulte Controle a alocação de memória com opções de JVM.

Ao executar esse comando, você deve ver as seguintes mensagens para indicar que o instalador foi bem-sucedido.

```
Successfully configured Nucleus with provisioned resource details!
Configured Nucleus to deploy aws.greengrass.Cli component
Successfully set up Nucleus as a system service
```

> **Note**
>
> Se você tiver um dispositivo Linux e ele não tiver systemd, o instalador não configurará o software como um serviço do sistema e você não verá a mensagem de sucesso da configuração do núcleo como um serviço do sistema.

(Opcional) Execute o software Greengrass (Linux)

Se você instalou o software como um serviço do sistema, o instalador executa o software para você. Caso contrário, você deve executar o software. Para ver se o instalador configurou o software como um serviço do sistema, procure a seguinte linha na saída do instalador.

```
Successfully set up Nucleus as a system service
```

Se você não ver essa mensagem, faça o seguinte para executar o software:

1. Execute o comando a seguir para executar o software.

   ```
   sudo /greengrass/v2/alts/current/distro/bin/loader
   ```

   O software imprime a seguinte mensagem se for iniciado com êxito.

   ```
   Launched Nucleus successfully.
   ```

2. Você deve deixar o shell de comando atual aberto para manter o software AWS IoT Greengrass Core em execução. Se você usa SSH para se conectar ao dispositivo principal, execute o comando a seguir em seu computador de desenvolvimento para abrir uma segunda sessão SSH que você pode usar para executar comandos adicionais no dispositivo principal. *Substitua*

o nome de usuário pelo nome do usuário a ser conectado e pi-ip-addresssubstitua pelo endereço IP do dispositivo.

```
ssh username@pi-ip-address
```

Para obter mais informações sobre como interagir com o serviço do sistema Greengrass, consulte. Configurar o núcleo do Greengrass como um serviço do sistema

Verifique a instalação da CLI do Greengrass no dispositivo

A CLI do Greengrass pode levar até um minuto para ser implantada. Execute o comando a seguir para verificar o status da implantação. Substitua *MyGreengrassCore* pelo nome do seu dispositivo principal.

```
aws greengrassv2 list-effective-deployments --core-device-thing-name MyGreengrassCore
```

`coreDeviceExecutionStatus`Indica o status da implantação no dispositivo principal. Quando o status for`SUCCEEDED`, execute o comando a seguir para verificar se a CLI do Greengrass está instalada e em execução. */greengrass/v2*Substitua pelo caminho para a pasta raiz.

Linux or Unix

```
/greengrass/v2/bin/greengrass-cli help
```

Windows Command Prompt (CMD)

```
C:\greengrass\v2\bin\greengrass-cli help
```

PowerShell

```
C:\greengrass\v2\bin\greengrass-cli help
```

O comando gera informações de ajuda para a CLI do Greengrass. Se `greengrass-cli` não for encontrado, a implantação pode ter falhado ao instalar a CLI do Greengrass. Para ter mais informações, consulte Solução de problemas AWS IoT Greengrass V2.

Você também pode executar o comando a seguir para implantar manualmente a AWS IoT Greengrass CLI no seu dispositivo.

- Substitua a *região* pela Região da AWS que você usa. Certifique-se de usar o mesmo Região da AWS que você usou para configurar o AWS CLI em seu dispositivo.
- Substitua o *ID da conta* pelo seu Conta da AWS ID.
- Substitua *MyGreengrassCore* pelo nome do seu dispositivo principal.

Linux, macOS, or Unix

```
aws greengrassv2 create-deployment \
  --target-arn "arn:aws:iot:region:account-id:thing/MyGreengrassCore" \
  --components '{
    "aws.greengrass.Cli": {
      "componentVersion": "2.12.6"
    }
  }'
```

Windows Command Prompt (CMD)

```
aws greengrassv2 create-deployment ^
  --target-arn "arn:aws:iot:region:account-id:thing/MyGreengrassCore" ^
  --components "{\"aws.greengrass.Cli\":{\"componentVersion\":\"2.12.6\"}}"
```

PowerShell

```
aws greengrassv2 create-deployment `
  --target-arn "arn:aws:iot:region:account-id:thing/MyGreengrassCore" `
  --components '{\"aws.greengrass.Cli\":{\"componentVersion\":\"2.12.6\"}}'
```

> **Tip**
>
> Você pode adicionar */greengrass/v2*/bin (Linux) ou *C:\greengrass\v2*\bin (Windows) à sua variável de PATH ambiente para ser executada `greengrass-cli` sem seu caminho absoluto.

O softwareAWS IoT Greengrass principal e as ferramentas de desenvolvimento local são executados em seu dispositivo. Em seguida, você pode desenvolver umAWS IoT Greengrass componente Hello World em seu dispositivo.

Etapa 4: desenvolver e testar um componente em seu dispositivo

Um componente é um módulo de software executado em dispositivos AWS IoT Greengrass principais. Os componentes permitem que você crie e gerencie aplicativos complexos como blocos de construção discretos que você pode reutilizar de um dispositivo principal do Greengrass para outro. Cada componente é composto por uma receita e artefatos.

- Receitas

  Cada componente contém um arquivo de receita, que define seus metadados. A receita também especifica os parâmetros de configuração do componente, as dependências do componente, o ciclo de vida e a compatibilidade da plataforma. O ciclo de vida do componente define os comandos que instalam, executam e desligam o componente. Para ter mais informações, consulte AWS IoT Greengrass referência da receita do componente.

  Você pode definir receitas no formato JSON ou YAML.

- Artefatos

  Os componentes podem ter qualquer número de artefatos, que são binários de componentes. Os artefatos podem incluir scripts, código compilado, recursos estáticos e quaisquer outros arquivos que um componente consuma. Os componentes também podem consumir artefatos das dependências dos componentes.

Com AWS IoT Greengrass, você pode usar a CLI do Greengrass para desenvolver e testar componentes localmente em um dispositivo principal do Greengrass sem interação com a nuvem. AWS Ao concluir seu componente local, você pode usar a receita e os artefatos do componente para criar esse componente no AWS IoT Greengrass serviço na AWS nuvem e, em seguida, implantá-lo em todos os seus dispositivos principais do Greengrass. Para obter mais informações sobre componentes, consulte Desenvolva AWS IoT Greengrass componentes.

Nesta seção, você aprenderá a criar e executar um componente básico do Hello World localmente em seu dispositivo principal.

Para desenvolver um componente Hello World em seu dispositivo

1. Crie uma pasta para seus componentes com subpastas para receitas e artefatos. Execute os comandos a seguir em seu dispositivo principal do Greengrass para criar essas pastas

e mudar para a pasta do componente. Substitua *~/greengrassv2* ou *%USERPROFILE%\greengrassv2* pelo caminho para a pasta a ser usada no desenvolvimento local.

Linux or Unix

```
mkdir -p ~/greengrassv2/{recipes,artifacts}
cd ~/greengrassv2
```

Windows Command Prompt (CMD)

```
mkdir %USERPROFILE%\greengrassv2\\recipes, %USERPROFILE%\greengrassv2\\artifacts
cd %USERPROFILE%\greengrassv2
```

PowerShell

```
mkdir ~/greengrassv2/recipes, ~/greengrassv2/artifacts
cd ~/greengrassv2
```

2. Use um editor de texto para criar um arquivo de receita que defina os metadados, os parâmetros, as dependências, o ciclo de vida e a capacidade da plataforma do seu componente. Inclua a versão do componente no nome do arquivo da receita para que você possa identificar qual receita reflete qual versão do componente. Você pode escolher o formato YAML ou JSON para sua receita.

   Por exemplo, em um sistema baseado em Linux, você pode executar o seguinte comando para usar o GNU nano para criar o arquivo.

   JSON

   ```
   nano recipes/com.example.HelloWorld-1.0.0.json
   ```

   YAML

   ```
   nano recipes/com.example.HelloWorld-1.0.0.yaml
   ```

> **Note**
>
> AWS IoT Greengrass usa versões semânticas para componentes. As versões semânticas seguem um sistema de numeração principal.secundária.patch. Por exemplo, a versão `1.0.0` representa a primeira versão principal de um componente. Para obter mais informações, consulte a especificação da versão semântica.

3. Cole a seguinte receita no arquivo.

   JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.HelloWorld",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "My first AWS IoT Greengrass component.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "Message": "world"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "run": "python3 -u {artifacts:path}/hello_world.py {configuration:/
Message}"
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "run": "py -3 -u {artifacts:path}/hello_world.py {configuration:/
Message}"
      }
    }
```

```
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.HelloWorld
ComponentVersion: '1.0.0'
ComponentDescription: My first AWS IoT Greengrass component.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    Message: world
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      run: |
        python3 -u {artifacts:path}/hello_world.py "{configuration:/Message}"
  - Platform:
      os: windows
    Lifecycle:
      run: |
        py -3 -u {artifacts:path}/hello_world.py "{configuration:/Message}"
```

A `ComponentConfiguration` seção desta receita define um parâmetro,`Message`, cujo padrão é. `world` A `Manifests` seção define um manifesto, que é um conjunto de instruções e artefatos do ciclo de vida de uma plataforma. Você pode definir vários manifestos para especificar instruções de instalação diferentes para várias plataformas, por exemplo. No manifesto, a `Lifecycle` seção instrui o dispositivo principal do Greengrass a executar o script Hello World com `Message` o valor do parâmetro como argumento.

4. Execute o comando a seguir para criar uma pasta para os artefatos do componente.

   Linux or Unix

   ```
   mkdir -p artifacts/com.example.HelloWorld/1.0.0
   ```

Windows Command Prompt (CMD)

```
mkdir artifacts\com.example.HelloWorld\1.0.0
```

PowerShell

```
mkdir artifacts\com.example.HelloWorld\1.0.0
```

> **Important**
>
> Você deve usar o seguinte formato para o caminho da pasta de artefatos. Inclua o nome e a versão do componente que você especificar na receita.
>
> ```
> artifacts/componentName/componentVersion/
> ```

5. Use um editor de texto para criar um arquivo de artefato de script Python para seu componente Hello World.

   Por exemplo, em um sistema baseado em Linux, você pode executar o seguinte comando para usar o GNU nano para criar o arquivo.

   ```
   nano artifacts/com.example.HelloWorld/1.0.0/hello_world.py
   ```

   Copie e cole o seguinte script Python no arquivo.

   ```
   import sys

   message = "Hello, %s!" % sys.argv[1]

   # Print the message to stdout, which Greengrass saves in a log file.
   print(message)
   ```

6. Use a AWS IoT Greengrass CLI local para gerenciar componentes em seu dispositivo principal do Greengrass.

   Execute o comando a seguir para implantar o componente no AWS IoT Greengrass núcleo. Substitua */greengrass/v2* ou *C:\greengrass\v2* pela pasta AWS IoT Greengrass V2

raiz e substitua *~/greengrassv2 ou %USERPROFILE%\ greengrassv2* pela pasta de desenvolvimento de componentes.

Linux or Unix

```
sudo /greengrass/v2/bin/greengrass-cli deployment create \
  --recipeDir ~/greengrassv2/recipes \
  --artifactDir ~/greengrassv2/artifacts \
  --merge "com.example.HelloWorld=1.0.0"
```

Windows Command Prompt (CMD)

```
C:\greengrass\v2\bin\greengrass-cli deployment create ^
  --recipeDir %USERPROFILE%\greengrassv2\recipes ^
  --artifactDir %USERPROFILE%\greengrassv2\artifacts ^
  --merge "com.example.HelloWorld=1.0.0"
```

PowerShell

```
C:\greengrass\v2\bin\greengrass-cli deployment create `
  --recipeDir ~/greengrassv2/recipes `
  --artifactDir ~/greengrassv2/artifacts `
  --merge "com.example.HelloWorld=1.0.0"
```

Esse comando adiciona o componente que usa a receita em `recipes` e o script Python em. `artifacts` A `--merge` opção adiciona ou atualiza o componente e a versão que você especificar.

7. O software AWS IoT Greengrass Core salva o stdout do processo do componente em arquivos de log na `logs` pasta. Execute o comando a seguir para verificar se o componente Hello World é executado e imprime mensagens.

   Linux or Unix

   ```
   sudo tail -f /greengrass/v2/logs/com.example.HelloWorld.log
   ```

   Windows Command Prompt (CMD)

   ```
   type C:\greengrass\v2\logs\com.example.HelloWorld.log
   ```

O `type` comando grava o conteúdo do arquivo no terminal. Execute esse comando várias vezes para observar as alterações no arquivo.

PowerShell

```
gc C:\greengrass\v2\logs\com.example.HelloWorld.log -Tail 10 -Wait
```

Você deve ver mensagens semelhantes ao exemplo a seguir.

```
Hello, world!
```

> **Note**
>
> Se o arquivo não existir, a implantação local talvez ainda não tenha sido concluída. Se o arquivo não existir em 15 segundos, a implantação provavelmente falhou. Isso pode ocorrer se sua receita não for válida, por exemplo. Execute o comando a seguir para visualizar o arquivo de log AWS IoT Greengrass principal. Esse arquivo inclui registros do serviço de implantação do dispositivo principal do Greengrass.
>
> Linux or Unix
>
> ```
> sudo tail -f /greengrass/v2/logs/greengrass.log
> ```
>
> Windows Command Prompt (CMD)
>
> ```
> type C:\greengrass\v2\logs\greengrass.log
> ```
>
> O `type` comando grava o conteúdo do arquivo no terminal. Execute esse comando várias vezes para observar as alterações no arquivo.
>
> PowerShell
>
> ```
> gc C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
> ```

8. Modifique o componente local para iterar e testar seu código. Abra `hello_world.py` em um editor de texto e adicione o código a seguir na linha 4 para editar a mensagem que o AWS IoT Greengrass núcleo registra.

```
message += " Greetings from your first Greengrass component."
```

O `hello_world.py` script agora deve ter o seguinte conteúdo.

```
import sys

message = "Hello, %s!" % sys.argv[1]
message += " Greetings from your first Greengrass component."

# Print the message to stdout, which Greengrass saves in a log file.
print(message)
```

9. Execute o comando a seguir para atualizar o componente com suas alterações.

   Linux or Unix

   ```
   sudo /greengrass/v2/bin/greengrass-cli deployment create \
     --recipeDir ~/greengrassv2/recipes \
     --artifactDir ~/greengrassv2/artifacts \
     --merge "com.example.HelloWorld=1.0.0"
   ```

   Windows Command Prompt (CMD)

   ```
   C:\greengrass\v2\bin\greengrass-cli deployment create ^
     --recipeDir %USERPROFILE%\greengrassv2\recipes ^
     --artifactDir %USERPROFILE%\greengrassv2\artifacts ^
     --merge "com.example.HelloWorld=1.0.0"
   ```

   PowerShell

   ```
   C:\greengrass\v2\bin\greengrass-cli deployment create `
     --recipeDir ~/greengrassv2/recipes `
     --artifactDir ~/greengrassv2/artifacts `
     --merge "com.example.HelloWorld=1.0.0"
   ```

   Esse comando atualiza o `com.example.HelloWorld` componente com o artefato Hello World mais recente.

10. Execute o comando a seguir para reiniciar o componente. Quando você reinicia um componente, o dispositivo principal usa as alterações mais recentes.

    Linux or Unix

    ```
    sudo /greengrass/v2/bin/greengrass-cli component restart \
      --names "com.example.HelloWorld"
    ```

    Windows Command Prompt (CMD)

    ```
    C:\greengrass\v2\bin\greengrass-cli component restart ^
      --names "com.example.HelloWorld"
    ```

    PowerShell

    ```
    C:\greengrass\v2\bin\greengrass-cli component restart `
      --names "com.example.HelloWorld"
    ```

11. Verifique o registro novamente para verificar se o componente Hello World imprime a nova mensagem.

    Linux or Unix

    ```
    sudo tail -f /greengrass/v2/logs/com.example.HelloWorld.log
    ```

    Windows Command Prompt (CMD)

    ```
    type C:\greengrass\v2\logs\com.example.HelloWorld.log
    ```

    O `type` comando grava o conteúdo do arquivo no terminal. Execute esse comando várias vezes para observar as alterações no arquivo.

    PowerShell

    ```
    gc C:\greengrass\v2\logs\com.example.HelloWorld.log -Tail 10 -Wait
    ```

    Você deve ver mensagens semelhantes ao exemplo a seguir.

```
Hello, world! Greetings from your first Greengrass component.
```

12. Você pode atualizar os parâmetros de configuração do componente para testar configurações diferentes. Ao implantar um componente, você pode especificar uma atualização de configuração, que define como modificar a configuração do componente no dispositivo principal. Você pode especificar quais valores de configuração serão redefinidos para os valores padrão e os novos valores de configuração a serem mesclados no dispositivo principal. Para ter mais informações, consulte Atualizar configurações de componentes.

    Faça o seguinte:

    a. Use um editor de texto para criar um arquivo chamado `hello-world-config-update.json` para conter a atualização de configuração

       Por exemplo, em um sistema baseado em Linux, você pode executar o seguinte comando para usar o GNU nano para criar o arquivo.

       ```
       nano hello-world-config-update.json
       ```

    b. Copie e cole o seguinte objeto JSON no arquivo. Esse objeto JSON define uma atualização de configuração que mescla o valor `friend` ao `Message` parâmetro para atualizar seu valor. Essa atualização de configuração não especifica nenhum valor a ser redefinido. Você não precisa redefinir o `Message` parâmetro porque a atualização de mesclagem substitui o valor existente.

       ```
       {
         "com.example.HelloWorld": {
           "MERGE": {
             "Message": "friend"
           }
         }
       }
       ```

    c. Execute o comando a seguir para implantar a atualização de configuração no componente Hello World.

       Linux or Unix

       ```
       sudo /greengrass/v2/bin/greengrass-cli deployment create \
         --merge "com.example.HelloWorld=1.0.0" \
       ```

```
  --update-config hello-world-config-update.json
```

Windows Command Prompt (CMD)

```
C:\greengrass\v2\bin\greengrass-cli deployment create ^
  --merge "com.example.HelloWorld=1.0.0" ^
  --update-config hello-world-config-update.json
```

PowerShell

```
C:\greengrass\v2\bin\greengrass-cli deployment create `
  --merge "com.example.HelloWorld=1.0.0" `
  --update-config hello-world-config-update.json
```

d. Verifique o registro novamente para verificar se o componente Hello World gera a nova mensagem.

Linux or Unix

```
sudo tail -f /greengrass/v2/logs/com.example.HelloWorld.log
```

Windows Command Prompt (CMD)

```
type C:\greengrass\v2\logs\com.example.HelloWorld.log
```

O `type` comando grava o conteúdo do arquivo no terminal. Execute esse comando várias vezes para observar as alterações no arquivo.

PowerShell

```
gc C:\greengrass\v2\logs\com.example.HelloWorld.log -Tail 10 -Wait
```

Você deve ver mensagens semelhantes ao exemplo a seguir.

```
Hello, friend! Greetings from your first Greengrass component.
```

13. Depois de terminar de testar seu componente, remova-o do seu dispositivo principal. Execute o seguinte comando .

Linux or Unix

```
sudo /greengrass/v2/bin/greengrass-cli deployment create --
remove="com.example.HelloWorld"
```

Windows Command Prompt (CMD)

```
C:\greengrass\v2\bin\greengrass-cli deployment create --
remove="com.example.HelloWorld"
```

PowerShell

```
C:\greengrass\v2\bin\greengrass-cli deployment create --
remove="com.example.HelloWorld"
```

> **Important**
>
> Essa etapa é necessária para que você implante o componente de volta no dispositivo principal depois de carregá-lo no AWS IoT Greengrass. Caso contrário, a implantação falhará com um erro de compatibilidade de versão porque a implantação local especifica uma versão diferente do componente.

Execute o comando a seguir e verifique se o `com.example.HelloWorld` componente não aparece na lista de componentes do seu dispositivo.

Linux or Unix

```
sudo /greengrass/v2/bin/greengrass-cli component list
```

Windows Command Prompt (CMD)

```
C:\greengrass\v2\bin\greengrass-cli component list
```

PowerShell

```
C:\greengrass\v2\bin\greengrass-cli component list
```

Seu componente Hello World está completo e agora você pode carregá-lo no serviço de AWS IoT Greengrass nuvem. Em seguida, você pode implantar o componente nos dispositivos principais do Greengrass.

Etapa 5: Crie seu componente no AWS IoT Greengrass serviço

Ao terminar de desenvolver um componente em seu dispositivo principal, você pode carregá-lo para o AWS IoT Greengrass serviço no Nuvem AWS. Você também pode criar diretamente o componente no AWS IoT Greengrass console. AWS IoT Greengrass fornece um serviço de gerenciamento de componentes que hospeda seus componentes para que você possa implantá-los em dispositivos individuais ou em frotas de dispositivos. Para carregar um componente para o AWS IoT Greengrass serviço, você conclui as seguintes etapas:

- Faça upload de artefatos de componentes em um bucket do S3.
- Adicione o URI do Amazon Simple Storage Service (Amazon S3) de cada artefato à receita do componente.
- Crie um componente a AWS IoT Greengrass partir da receita do componente.

Nesta seção, você conclui essas etapas em seu dispositivo principal do Greengrass para carregar seu componente Hello World no AWS IoT Greengrass serviço.

Crie seu componente em AWS IoT Greengrass (console)

1. Use um bucket do S3 em sua AWS conta para hospedar artefatos de AWS IoT Greengrass componentes. Quando você implanta o componente em um dispositivo principal, o dispositivo baixa os artefatos do componente do bucket.

   Você pode usar um bucket S3 existente ou criar um novo bucket.

   a. No console do Amazon S3, em Buckets, escolha Create bucket.
   b. Em Nome do compartimento, insira um nome de compartimento exclusivo. Por exemplo, você poderá usar o **greengrass-component-artifacts-*region*-*123456789012***.

Substitua *123456789012* pela ID da sua AWS conta e *região* pela Região da AWS que você usa neste tutorial.

c. Para AWS região, selecione a AWS região que você usa para este tutorial.

d. Selecione Criar bucket.

e. Em Buckets, escolha o bucket que você criou e faça o upload do `hello_world.py` script para a `artifacts/com.example.HelloWorld/1.0.0` pasta no bucket. Para obter informações sobre o upload de objetos para buckets do S3, consulte Carregar objetos no Guia do usuário do Amazon Simple Storage Service.

f. Copie o URI do S3 do `hello_world.py` objeto no bucket do S3. Esse URI deve ser semelhante ao exemplo a seguir. Substitua DOC-EXAMPLE-BUCKET pelo nome do bucket S3.

```
s3://DOC-EXAMPLE-BUCKET/artifacts/com.example.HelloWorld/1.0.0/hello_world.py
```

2. Permita que o dispositivo principal acesse artefatos de componentes no bucket do S3.

Cada dispositivo principal tem uma função de IAM do dispositivo principal que permite interagir AWS IoT e enviar registros para a AWS nuvem. Essa função de dispositivo não permite acesso aos buckets do S3 por padrão, portanto, você deve criar e anexar uma política que permita que o dispositivo principal recupere artefatos do componente do bucket do S3.

Se a função do seu dispositivo já permitir o acesso ao bucket do S3, você pode pular esta etapa. Caso contrário, crie uma política do IAM que permita acesso e anexe-a à função, da seguinte forma:

a. No menu de navegação do console do IAM, escolha Políticas e, em seguida, escolha Criar política.

b. Na guia JSON, substitua o conteúdo do espaço reservado pela política a seguir. Substitua DOC-EXAMPLE-BUCKET pelo nome do bucket S3 que contém artefatos de componentes para download do dispositivo principal.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
```

```
      ],
      "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*"
    }
  ]
}
```

c. Escolha Próximo.

d. Na seção Detalhes da política, em Nome, insira**`MyGreengrassV2ComponentArtifactPolicy`**.

e. Escolha Criar política.

f. No menu de navegação do console do IAM, escolha Role e, em seguida, escolha o nome da função para o dispositivo principal. Você especificou esse nome de função ao instalar o software AWS IoT Greengrass Core. Se você não especificou um nome, o padrão será`GreengrassV2TokenExchangeRole`.

g. Em Permissões, escolha Adicionar permissões e, em seguida, escolha Anexar políticas.

h. Na página Adicionar permissões, marque a caixa de seleção ao lado da `MyGreengrassV2ComponentArtifactPolicy` política que você criou e escolha Adicionar permissões.

3. Use a receita do componente para criar um componente no AWS IoT Greengrass console.

   a. No menu de navegação do AWS IoT Greengrass console, escolha Componentes e, em seguida, escolha Criar componente.

   b. Em Informações do componente, escolha Inserir receita como JSON. A receita do espaço reservado deve ser semelhante ao exemplo a seguir.

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.HelloWorld",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "My first AWS IoT Greengrass component.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "Message": "world"
    }
  },
  "Manifests": [
    {
      "Platform": {
```

```
        "os": "linux"
      },
      "Lifecycle": {
        "run": "python3 -u {artifacts:path}/hello_world.py \"{configuration:/
Message}\""
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.HelloWorld/1.0.0/hello_world.py"
        }
      ]
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "run": "py -3 -u {artifacts:path}/hello_world.py \"{configuration:/
Message}\""
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.HelloWorld/1.0.0/hello_world.py"
        }
      ]
    }
  ]
}
```

c. Substitua o URI do espaço reservado em cada `Artifacts` seção pelo URI S3 do seu `hello_world.py` objeto.

d. Escolha Criar componente.

e. No com.example. HelloWorldpágina do componente, verifique se o status do componente é Implantável.

Crie seu componente em AWS IoT Greengrass (AWS CLI)

Para fazer o upload do seu componente Hello World

1. Use um bucket do S3 em seu Conta da AWS para hospedar artefatos de AWS IoT Greengrass componentes. Quando você implanta o componente em um dispositivo principal, o dispositivo baixa os artefatos do componente do bucket.

   Você pode usar um bucket S3 existente ou executar o comando a seguir para criar um bucket. Esse comando cria um bucket com seu Conta da AWS ID e Região da AWS forma um nome de bucket exclusivo. Substitua *123456789012* pela sua Conta da AWS ID e *região* pela Região da AWS que você usa neste tutorial.

   ```
   aws s3 mb s3://greengrass-component-artifacts-123456789012-region
   ```

   O comando exibirá as seguintes informações se a solicitação for bem-sucedida.

   ```
   make_bucket: greengrass-component-artifacts-123456789012-region
   ```

2. Permita que o dispositivo principal acesse artefatos de componentes no bucket do S3.

   Cada dispositivo principal tem uma função IAM do dispositivo principal que permite interagir AWS IoT e enviar registros para Nuvem AWS o. Essa função de dispositivo não permite acesso aos buckets do S3 por padrão, portanto, você deve criar e anexar uma política que permita que o dispositivo principal recupere artefatos do componente do bucket do S3.

   Se a função do dispositivo principal já permitir o acesso ao bucket do S3, você pode pular essa etapa. Caso contrário, crie uma política do IAM que permita acesso e anexe-a à função, da seguinte forma:

   a. Crie um arquivo chamado `component-artifact-policy.json` e copie o seguinte JSON para o arquivo. Essa política permite acesso a todos os arquivos em um bucket do S3. Substitua DOC-EXAMPLE-BUCKET pelo nome do bucket S3.

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

```
        "s3:GetObject"
      ],
      "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*"
    }
  ]
}
```

b. Execute o comando a seguir para criar a política a partir do documento de política em`component-artifact-policy.json`.

Linux or Unix

```
aws iam create-policy \\
  --policy-name MyGreengrassV2ComponentArtifactPolicy \\
  --policy-document file://component-artifact-policy.json
```

Windows Command Prompt (CMD)

```
aws iam create-policy ^
  --policy-name MyGreengrassV2ComponentArtifactPolicy ^
  --policy-document file://component-artifact-policy.json
```

PowerShell

```
aws iam create-policy `
  --policy-name MyGreengrassV2ComponentArtifactPolicy `
  --policy-document file://component-artifact-policy.json
```

Copie a política Amazon Resource Name (ARN) dos metadados da política na saída. Você usa esse ARN para anexar essa política à função do dispositivo principal na próxima etapa.

c. Execute o comando a seguir para anexar a política à função do dispositivo principal. Substitua *GreengrassV2 TokenExchangeRole* pelo nome da função do dispositivo principal. Você especificou esse nome de função ao instalar o software AWS IoT Greengrass Core. Substitua o ARN da política pelo ARN da etapa anterior.

Linux or Unix

```
aws iam attach-role-policy \\
  --role-name GreengrassV2TokenExchangeRole \\
```

```
  --policy-arn
 arn:aws:iam::123456789012:policy/MyGreengrassV2ComponentArtifactPolicy
```

Windows Command Prompt (CMD)

```
aws iam attach-role-policy ^
  --role-name GreengrassV2TokenExchangeRole ^
  --policy-arn
 arn:aws:iam::123456789012:policy/MyGreengrassV2ComponentArtifactPolicy
```

PowerShell

```
aws iam attach-role-policy `
  --role-name GreengrassV2TokenExchangeRole `
  --policy-arn
 arn:aws:iam::123456789012:policy/MyGreengrassV2ComponentArtifactPolicy
```

Se o comando não tiver saída, ele foi bem-sucedido. O dispositivo principal agora pode acessar os artefatos que você carrega nesse bucket do S3.

3. Faça upload do artefato do script Hello World Python para o bucket do S3.

   Execute o comando a seguir para carregar o script no mesmo caminho no bucket em que o script existe no seu AWS IoT Greengrass núcleo. Substitua DOC-EXAMPLE-BUCKET pelo nome do bucket S3.

   Linux or Unix

   ```
   aws s3 cp \
     artifacts/com.example.HelloWorld/1.0.0/hello_world.py \
     s3://DOC-EXAMPLE-BUCKET/artifacts/com.example.HelloWorld/1.0.0/hello_world.py
   ```

   Windows Command Prompt (CMD)

   ```
   aws s3 cp ^
     artifacts/com.example.HelloWorld/1.0.0/hello_world.py ^
     s3://DOC-EXAMPLE-BUCKET/artifacts/com.example.HelloWorld/1.0.0/hello_world.py
   ```

PowerShell

```
aws s3 cp `
  artifacts/com.example.HelloWorld/1.0.0/hello_world.py `
  s3://DOC-EXAMPLE-BUCKET/artifacts/com.example.HelloWorld/1.0.0/hello_world.py
```

O comando gera uma linha que começa com `upload:` se a solicitação for bem-sucedida.

4. Adicione o URI do Amazon S3 do artefato à receita do componente.

   O URI do Amazon S3 é composto pelo nome do bucket e pelo caminho para o objeto de artefato no bucket. O URI do Amazon S3 do seu artefato de script é o URI para o qual você fez o upload do artefato na etapa anterior. Esse URI deve ser semelhante ao exemplo a seguir. Substitua DOC-EXAMPLE-BUCKET pelo nome do bucket S3.

   ```
   s3://DOC-EXAMPLE-BUCKET/artifacts/com.example.HelloWorld/1.0.0/hello_world.py
   ```

   Para adicionar o artefato à receita, adicione uma lista `Artifacts` que contenha uma estrutura com o URI do Amazon S3.

   JSON

   ```
   "Artifacts": [
     {
       "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/com.example.HelloWorld/1.0.0/
   hello_world.py"
     }
   ]
   ```

   Abra o arquivo da receita em um editor de texto.

   Por exemplo, em um sistema baseado em Linux, você pode executar o seguinte comando para usar o GNU nano para criar o arquivo.

   ```
   nano recipes/com.example.HelloWorld-1.0.0.json
   ```

   Adicione o artefato à receita. Seu arquivo de receita deve ser semelhante ao exemplo a seguir.

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.HelloWorld",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "My first AWS IoT Greengrass component.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "Message": "world"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "run": "python3 -u {artifacts:path}/hello_world.py \"{configuration:/
Message}\""
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.HelloWorld/1.0.0/hello_world.py"
        }
      ]
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "run": "py -3 -u {artifacts:path}/hello_world.py \"{configuration:/
Message}\""
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.HelloWorld/1.0.0/hello_world.py"
        }
      ]
    }
  ]
```

```
}
```

YAML

```
Artifacts:
  - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/com.example.HelloWorld/1.0.0/
hello_world.py
```

Abra o arquivo da receita em um editor de texto.

Por exemplo, em um sistema baseado em Linux, você pode executar o seguinte comando para usar o GNU nano para criar o arquivo.

```
nano recipes/com.example.HelloWorld-1.0.0.yaml
```

Adicione o artefato à receita. Seu arquivo de receita deve ser semelhante ao exemplo a seguir.

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.HelloWorld
ComponentVersion: '1.0.0'
ComponentDescription: My first AWS IoT Greengrass component.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    Message: world
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      run: |
        python3 -u {artifacts:path}/hello_world.py "{configuration:/Message}"
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/com.example.HelloWorld/1.0.0/
hello_world.py
  - Platform:
      os: windows
    Lifecycle:
      run: |
        py -3 -u {artifacts:path}/hello_world.py "{configuration:/Message}"
```

```
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/com.example.HelloWorld/1.0.0/
hello_world.py
```

5. Crie um recurso de componente a AWS IoT Greengrass partir da receita. Execute o comando a seguir para criar o componente a partir da receita, que você fornece como um arquivo binário.

   JSON

   ```
   aws greengrassv2 create-component-version --inline-recipe fileb://recipes/
   com.example.HelloWorld-1.0.0.json
   ```

   YAML

   ```
   aws greengrassv2 create-component-version --inline-recipe fileb://recipes/
   com.example.HelloWorld-1.0.0.yaml
   ```

   Se a solicitação for bem-sucedida, a resposta será semelhante ao exemplo a seguir.

   ```
   {
     "arn":
    "arn:aws:greengrass:region:123456789012:components:com.example.HelloWorld:versions:1.0.0",
     "componentName": "com.example.HelloWorld",
     "componentVersion": "1.0.0",
     "creationTimestamp": "Mon Nov 30 09:04:05 UTC 2020",
     "status": {
       "componentState": "REQUESTED",
       "message": "NONE",
       "errors": {}
     }
   }
   ```

   Copie o `arn` da saída para verificar o estado do componente na próxima etapa.

   > **Note**
   >
   > Você também pode ver seu componente Hello World no AWS IoT Greengrass console na página Componentes.

6. Verifique se o componente foi criado e está pronto para ser implantado. Quando você cria um componente, seu estado é`REQUESTED`. Em seguida, AWS IoT Greengrass valida se o componente é implantável. Você pode executar o comando a seguir para consultar o status do componente e verificar se ele é implantável. `arn`Substitua o pelo ARN da etapa anterior.

   ```
   aws greengrassv2 describe-component --arn
    "arn:aws:greengrass:region:123456789012:components:com.example.HelloWorld:versions:1.0.0"
   ```

   Se o componente for validado, a resposta indicará que o estado do componente é`DEPLOYABLE`.

   ```
   {
     "arn":
    "arn:aws:greengrass:region:123456789012:components:com.example.HelloWorld:versions:1.0.0",
     "componentName": "com.example.HelloWorld",
     "componentVersion": "1.0.0",
     "creationTimestamp": "2020-11-30T18:04:05.823Z",
     "publisher": "Amazon",
     "description": "My first Greengrass component.",
     "status": {
       "componentState": "DEPLOYABLE",
       "message": "NONE",
       "errors": {}
     },
     "platforms": [
       {
         "os": "linux",
         "architecture": "all"
       }
     ]
   }
   ```

Seu componente Hello World agora está disponível em AWS IoT Greengrass. Você pode implantá-lo de volta nesse dispositivo principal do Greengrass ou em outros dispositivos principais.

Etapa 6: implantar seu componente

ComAWS IoT Greengrass, você pode implantar componentes em dispositivos individuais ou grupos de dispositivos. Quando você implanta um componente, AWS IoT Greengrass instala e executa o software desse componente em cada dispositivo de destino. Você especifica quais componentes

implantar e a atualização de configuração a ser implantada para cada componente. Você também pode controlar como a implantação é implementada nos dispositivos visados pela implantação. Para ter mais informações, consulte Implemente AWS IoT Greengrass componentes em dispositivos.

Nesta seção, você implanta seu componente Hello World de volta ao seu dispositivo principal do Greengrass.

Implante seu componente (console)

1. No menu de navegação AWS IoT Greengrassdo console, escolha Componentes.
2. Na página Componentes, na guia Meus componentes, escolha com.example.HelloWorld.
3. Na página com.example.HelloWorld, escolha Implantar.
4. Em Adicionar à implantação, escolha Criar nova implantação e, em seguida, escolha Avançar.
5. Na página Especificar detalhes, faça o seguinte:
   a. Na caixa Name (Nome), insira **`Deployment for MyGreengrassCore`**.
   b. Em Destino de implantação, escolha Dispositivo principal e o nome do AWS IoT item para seu dispositivo principal. O valor padrão neste tutorial é *`MyGreengrassCore`*.
   c. Escolha Próximo.
6. Na página Selecionar componentes, em Meus componentes, verifique se o com.example.HelloWorldcomponente está selecionado e escolha Avançar.
7. Na página Configurar componentes com.example.HelloWorld, escolha e faça o seguinte:
   a. Escolha Configurar componente.
   b. Em Atualização de configuração, em Configuração a ser mesclada, insira a configuração a seguir.

      ```
      {
        "Message": "universe"
      }
      ```

      Essa atualização de configuração define o `Message` parâmetro Hello World `universe` para o dispositivo nessa implantação.
   c. Selecione a opção Confirmar.
   d. Escolha Próximo.

8. Na página Definir configurações avançadas, mantenha as configurações padrão e escolha Avançar.
9. Na página Review, escolha Deploy.
10. Verifique se a implantação foi concluída com êxito. A implantação pode levar vários minutos para ser concluída. Verifique o registro do Hello World para verificar a alteração. Execute o comando a seguir em seu dispositivo principal do Greengrass.

    Linux or Unix

    ```
    sudo tail -f /greengrass/v2/logs/com.example.HelloWorld.log
    ```

    Windows Command Prompt (CMD)

    ```
    type C:\greengrass\v2\\logs\\com.example.HelloWorld.log
    ```

    PowerShell

    ```
    gc C:\greengrass\v2\\logs\\com.example.HelloWorld.log -Tail 10 -Wait
    ```

    Você deve ver mensagens semelhantes ao exemplo a seguir.

    ```
    Hello, universe! Greetings from your first Greengrass component.
    ```

    > **Note**
    >
    > Se as mensagens de registro não mudarem, a implantação falhou ou não atingiu o dispositivo principal. Isso pode ocorrer se seu dispositivo principal não estiver conectado à Internet ou não tiver permissões para recuperar artefatos do seu bucket do S3. Execute o comando a seguir em seu dispositivo principal para visualizar o arquivo de log do software AWS IoT Greengrass principal. Esse arquivo inclui registros do serviço de implantação do dispositivo principal do Greengrass.
    >
    > Linux or Unix
    >
    > ```
    > sudo tail -f /greengrass/v2/logs/greengrass.log
    > ```

Windows Command Prompt (CMD)

```
type C:\greengrass\v2\\logs\\greengrass.log
```

O `type` comando grava o conteúdo do arquivo no terminal. Execute esse comando várias vezes para observar as alterações no arquivo.

PowerShell

```
gc C:\greengrass\v2\\logs\\greengrass.log -Tail 10 -Wait
```

Para ter mais informações, consulte Solução de problemas AWS IoT Greengrass V2.

Implante seu componente (AWS CLI)

Para implantar seu componente Hello World

1. No seu computador de desenvolvimento, crie um arquivo chamado `hello-world-deployment.json` e copie o seguinte JSON para o arquivo. Esse arquivo define os componentes e as configurações a serem implantados.

```
{
  "components": {
    "com.example.HelloWorld": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "merge": "{\"Message\":\"universe\"}"
      }
    }
  }
}
```

   Esse arquivo de configuração especifica a implantação `1.0.0` da versão do componente Hello World que você desenvolveu e publicou no procedimento anterior. O `configurationUpdate` especifica a mesclagem da configuração do componente em uma string codificada em JSON. Essa atualização de configuração define o `Message` parâmetro Hello World `universe` para o dispositivo nessa implantação.

2. Execute o comando a seguir para implantar o componente em seu dispositivo principal do Greengrass. Você pode implantar em coisas, que são dispositivos individuais, ou grupos de coisas, que são grupos de dispositivos. *MyGreengrassCore*Substitua pelo nome do AWS IoT item do seu dispositivo principal.

   Linux or Unix

   ```
   aws greengrassv2 create-deployment \
     --target-arn "arn:aws:iot:region:account-id:thing/MyGreengrassCore" \
     --cli-input-json file://hello-world-deployment.json
   ```

   Windows Command Prompt (CMD)

   ```
   aws greengrassv2 create-deployment ^
     --target-arn "arn:aws:iot:region:account-id:thing/MyGreengrassCore" ^
     --cli-input-json file://hello-world-deployment.json
   ```

   PowerShell

   ```
   aws greengrassv2 create-deployment `
     --target-arn "arn:aws:iot:region:account-id:thing/MyGreengrassCore" `
     --cli-input-json file://hello-world-deployment.json
   ```

   O comando gera uma resposta semelhante ao exemplo a seguir.

   ```
   {
     "deploymentId": "deb69c37-314a-4369-a6a1-3dff9fce73a9",
     "iotJobId": "b5d92151-6348-4941-8603-bdbfb3e02b75",
     "iotJobArn": "arn:aws:iot:region:account-id:job/b5d92151-6348-4941-8603-
   bdbfb3e02b75"
   }
   ```

3. Verifique se a implantação foi concluída com êxito. A implantação pode levar vários minutos para ser concluída. Verifique o registro do Hello World para verificar a alteração. Execute o comando a seguir em seu dispositivo principal do Greengrass.

   Linux or Unix

   ```
   sudo tail -f /greengrass/v2/logs/com.example.HelloWorld.log
   ```

Windows Command Prompt (CMD)

```
type C:\greengrass\v2\\logs\\com.example.HelloWorld.log
```

PowerShell

```
gc C:\greengrass\v2\\logs\\com.example.HelloWorld.log -Tail 10 -Wait
```

Você deve ver mensagens semelhantes ao exemplo a seguir.

```
Hello, universe! Greetings from your first Greengrass component.
```

> **Note**
>
> Se as mensagens de registro não mudarem, a implantação falhou ou não atingiu o dispositivo principal. Isso pode ocorrer se seu dispositivo principal não estiver conectado à Internet ou não tiver permissões para recuperar artefatos do seu bucket do S3. Execute o comando a seguir em seu dispositivo principal para visualizar o arquivo de log do software AWS IoT Greengrass principal. Esse arquivo inclui registros do serviço de implantação do dispositivo principal do Greengrass.
>
> Linux or Unix
>
> ```
> sudo tail -f /greengrass/v2/logs/greengrass.log
> ```
>
> Windows Command Prompt (CMD)
>
> ```
> type C:\greengrass\v2\\logs\\greengrass.log
> ```
>
> O `type` comando grava o conteúdo do arquivo no terminal. Execute esse comando várias vezes para observar as alterações no arquivo.
>
> PowerShell
>
> ```
> gc C:\greengrass\v2\\logs\\greengrass.log -Tail 10 -Wait
> ```

Para ter mais informações, consulte Solução de problemas AWS IoT Greengrass V2.

Próximas etapas

Você concluiu este tutorial. O softwareAWS IoT Greengrass Core e seu componente Hello World são executados em seu dispositivo. Além disso, seu componente Hello World está disponível no serviço deAWS IoT Greengrass nuvem para ser implantado em outros dispositivos. Para obter mais informações sobre os tópicos que este tutorial explora, consulte o seguinte:

- Crie AWS IoT Greengrass componentes
- Publish components to deploy to your core devices
- Implemente AWS IoT Greengrass componentes em dispositivos

Configurando dispositivos AWS IoT Greengrass principais

Conclua as tarefas nesta seção para instalar, configurar e executar o software AWS IoT Greengrass principal.

> **Note**
>
> Esta seção descreve a instalação e a configuração avançadas do software AWS IoT Greengrass Core. Se você é usuário iniciante do AWS IoT Greengrass V2, recomendamos que primeiro conclua o tutorial de introdução para configurar um dispositivo principal e explorar os recursos do AWS IoT Greengrass.

Plataformas compatíveis e requisitos

Antes de começar, certifique-se de atender aos seguintes requisitos para instalar e executar o software AWS IoT Greengrass Core.

> **Tip**
>
> Você pode pesquisar dispositivos qualificados AWS IoT Greengrass V2 no Catálogo de dispositivos do AWS parceiro.

Tópicos

Plataformas compatíveis

AWS IoT Greengrass suporta oficialmente dispositivos que executam as seguintes plataformas. Dispositivos com plataformas não incluídas nessa lista podem funcionar, mas AWS IoT Greengrass testes somente nessas plataformas especificadas.

Linux

Arquiteturas:

- Armv7l
- Armv8 (AArch64)
- x86_64

Windows

Arquiteturas:

- x86_64

Versões:

- Windows 10
- Windows 11
- Windows Server 2019
- Windows Server 2022

> **Note**
>
> No momento, alguns AWS IoT Greengrass recursos não são compatíveis com dispositivos Windows. Para obter mais informações, consulte Compatibilidade de recursos do Greengrass por sistema operacional e Considerações sobre recursos para dispositivos Windows.

As plataformas Linux também podem ser executadas AWS IoT Greengrass V2 em um contêiner Docker. Para ter mais informações, consulte Execute AWS IoT Greengrass o software Core em um contêiner Docker.

Para criar um sistema operacional personalizado baseado em Linux, você pode usar a BitBake receita AWS IoT Greengrass V2 do projeto. `meta-aws` O `meta-aws` projeto fornece receitas que você pode usar para criar recursos de software de AWS ponta em sistemas Linux embarcados que são construídos com OpenEmbeddedas estruturas de construção do Projeto Yocto. O Projeto

Yocto é um projeto de colaboração de código aberto que ajuda você a criar sistemas personalizados baseados em Linux para aplicativos embarcados, independentemente da arquitetura de hardware. A BitBake receita para AWS IoT Greengrass V2 instalar, configurar e executar automaticamente o software AWS IoT Greengrass Core em seu dispositivo.

Requisitos do dispositivo

Os dispositivos devem atender aos seguintes requisitos para instalar e executar o software AWS IoT Greengrass Core v2.x.

> **Note**
>
> Você pode usar AWS IoT Device Tester for AWS IoT Greengrass para verificar se seu dispositivo pode executar o software AWS IoT Greengrass Core e se comunicar com Nuvem AWS o. Para ter mais informações, consulte Usando AWS IoT Device Tester para AWS IoT Greengrass V2.

Linux

- O uso de um Região da AWSque suporte AWS IoT Greengrass V2. Para obter a lista de regiões compatíveis, consulte Endpoints e cotas do AWS IoT Greengrass V2 no Referência geral da AWS.
- Espaço mínimo de 256 MB em disco disponível para o software AWS IoT Greengrass Core. Esse requisito não inclui componentes implantados no dispositivo principal.
- Mínimo de 96 MB de RAM alocados para o software AWS IoT Greengrass Core. Esse requisito não inclui componentes que são executados no dispositivo principal. Para ter mais informações, consulte Controle a alocação de memória com opções de JVM.
- Java Runtime Environment (JRE) versão 8 ou superior. O Java deve estar disponível na variável de ambiente PATH no dispositivo. Para usar o Java para desenvolver componentes personalizados, é necessário instalar um Java Development Kit (JDK). Recomendamos que você use as versões de suporte de longo prazo do Amazon Corretto ou do OpenJDK. É necessária a versão 8 ou superior.
- Biblioteca GNU C (glibc) versão 2.25 ou superior.
- Você deve executar o software AWS IoT Greengrass Core como usuário root. Use`sudo`, por exemplo.

- O usuário root que executa o software AWS IoT Greengrass Core, por exemplo`root`, deve ter permissão para executar `sudo` com qualquer usuário e qualquer grupo. O `/etc/sudoers` arquivo deve dar permissão a esse usuário para ser executado `sudo` como outros grupos. A permissão para o usuário entrar `/etc/sudoers` deve ser semelhante ao exemplo a seguir.

  ```
  root    ALL=(ALL:ALL) ALL
  ```

- O dispositivo principal deve ser capaz de realizar solicitações de saída para um conjunto de terminais e portas. Para ter mais informações, consulte Permitir o tráfego de dispositivos por meio de um proxy ou firewall.
- O `/tmp` diretório deve ser montado com `exec` permissões.
- Todos os seguintes comandos de shell:
  - `ps -ax -o pid,ppid`
  - `sudo`
  - `sh`
  - `kill`
  - `cp`
  - `chmod`
  - `rm`
  - `ln`
  - `echo`
  - `exit`
  - `id`
  - `uname`
  - `grep`
- Seu dispositivo também pode exigir os seguintes comandos de shell opcionais:
  - (Opcional) `systemctl`. Esse comando é usado para configurar o software AWS IoT Greengrass principal como um serviço do sistema.
  - (Opcional) `useraddgroupadd`, `usermod` e. Esses comandos são usados para configurar o `ggc_user` usuário e o grupo `ggc_group` do sistema.
  - (Opcional) `mkfifo`. Esse comando é usado para executar funções do Lambda como componentes.

- Para configurar os limites de recursos do sistema para processos de componentes, seu dispositivo deve executar a versão 2.6.24 ou posterior do kernel Linux.
- Para executar funções do Lambda, seu dispositivo deve atender a requisitos adicionais. Para ter mais informações, consulte Requisitos da função do Lambda.

Windows

- O uso de um Região da AWSque suporte AWS IoT Greengrass V2. Para obter a lista de regiões compatíveis, consulte Endpoints e cotas do AWS IoT Greengrass V2 no Referência geral da AWS.
- Espaço mínimo de 256 MB em disco disponível para o software AWS IoT Greengrass Core. Esse requisito não inclui componentes implantados no dispositivo principal.
- Mínimo de 160 MB de RAM alocados para o software AWS IoT Greengrass Core. Esse requisito não inclui componentes que são executados no dispositivo principal. Para ter mais informações, consulte Controle a alocação de memória com opções de JVM.
- Java Runtime Environment (JRE) versão 8 ou superior. O Java deve estar disponível na variável de sistema PATH no dispositivo. Para usar o Java para desenvolver componentes personalizados, é necessário instalar um Java Development Kit (JDK). Recomendamos que você use as versões de suporte de longo prazo do Amazon Corretto ou do OpenJDK. É necessária a versão 8 ou superior.

  > **Note**
  >
  > Para usar a versão 2.5.0 do núcleo Greengrass, você deve usar uma versão de 64 bits do Java Runtime Environment (JRE). A versão 2.5.1 do Greengrass nucleus oferece suporte a JREs de 32 e 64 bits.

- O usuário que instala o software AWS IoT Greengrass Core deve ser um administrador.
- Você deve instalar o software AWS IoT Greengrass Core como um serviço do sistema. Especifique `--setup-system-service true` quando você instala o software.
- Cada usuário que executa processos de componentes deve existir na LocalSystem conta, e o nome e a senha do usuário devem estar na instância do Credential Manager da LocalSystem conta. Você pode configurar esse usuário seguindo as instruções para instalar o software AWS IoT Greengrass Core.

- O dispositivo principal deve ser capaz de realizar solicitações de saída para um conjunto de terminais e portas. Para ter mais informações, consulte Permitir o tráfego de dispositivos por meio de um proxy ou firewall.

Requisitos da função do Lambda

Seu dispositivo deve atender aos seguintes requisitos para executar as funções do Lambda:

- Um sistema operacional baseado em Linux.
- Seu dispositivo deve ter o comando `mkfifo` shell.
- Seu dispositivo deve executar as bibliotecas de linguagem de programação exigidas por uma função Lambda. Você deve instalar as bibliotecas necessárias no dispositivo e adicioná-las à variável de `PATH` ambiente. O Greengrass é compatível com todas as versões compatíveis com o Lambda dos tempos de execução Python, Node.js e Java. O Greengrass não aplica nenhuma restrição adicional às versões obsoletas de tempo de execução do Lambda. Para obter mais informações sobre o AWS IoT Greengrass suporte para tempos de execução do Lambda, consulte. Executar AWS Lambda funções
- Para executar funções Lambda em contêineres, seu dispositivo deve atender aos seguintes requisitos:
  - Kernel Linux versão 4.4 ou posterior.
  - O kernel deve suportar cgroups v1, e você deve habilitar e montar os seguintes cgroups:
    - O cgroup de memória AWS IoT Greengrass para definir o limite de memória para funções Lambda em contêineres.
    - O grupo de dispositivos para funções Lambda em contêineres para acessar dispositivos ou volumes do sistema.

    O software AWS IoT Greengrass Core não é compatível com cgroups v2.

    Para atender a esse requisito, inicialize o dispositivo com os seguintes parâmetros do kernel Linux.

    ```
    cgroup_enable=memory cgroup_memory=1 systemd.unified_cgroup_hierarchy=0
    ```

> **Tip**
>
> Em um Raspberry Pi, edite o `/boot/cmdline.txt` arquivo para definir os parâmetros do kernel do dispositivo.

- Você deve habilitar as seguintes configurações do kernel Linux no dispositivo:
  - Namespace:
    - `CONFIG_IPC_NS`
    - `CONFIG_UTS_NS`
    - `CONFIG_USER_NS`
    - `CONFIG_PID_NS`
  - Cgroups:
    - `CONFIG_CGROUP_DEVICE`
    - `CONFIG_CGROUPS`
    - `CONFIG_MEMCG`
  - Outros:
    - `CONFIG_POSIX_MQUEUE`
    - `CONFIG_OVERLAY_FS`
    - `CONFIG_HAVE_ARCH_SECCOMP_FILTER`
    - `CONFIG_SECCOMP_FILTER`
    - `CONFIG_KEYS`
    - `CONFIG_SECCOMP`
    - `CONFIG_SHMEM`

> **Tip**
>
> Consulte a documentação da sua distribuição Linux para saber como verificar e definir os parâmetros do kernel Linux. Você também pode usar AWS IoT Device Tester for AWS IoT Greengrass para verificar se seu dispositivo atende a esses requisitos. Para ter mais informações, consulte Usando AWS IoT Device Tester para AWS IoT Greengrass V2.

Considerações sobre recursos para dispositivos Windows

No momento, alguns AWS IoT Greengrass recursos não são compatíveis com dispositivos Windows. Analise as diferenças de recursos para confirmar se um dispositivo Windows atende aos seus requisitos. Para ter mais informações, consulte Compatibilidade de recursos do Greengrass por sistema operacional.

Configurar um Conta da AWS

Se você não tiver um Conta da AWS, conclua as etapas a seguir para criar um.

Para se inscrever em um Conta da AWS

1. Abra https://portal.aws.amazon.com/billing/signup.
2. Siga as instruções on-line.

   Parte do procedimento de inscrição envolve receber uma chamada telefônica e digitar um código de verificação no teclado do telefone.

   Quando você se inscreve em um Conta da AWS, um Usuário raiz da conta da AWSé criado. O usuário-raiz tem acesso a todos os Serviços da AWS e recursos na conta. Como uma prática recomendada de segurança, atribua o acesso administrativo para um usuário e use somente o usuário-raiz para executar tarefas que requerem o acesso de usuário-raiz.

Para criar um usuário administrador, selecione uma das opções a seguir.

Selecionar uma forma de gerenciar o administrador	Para	Por	Você também pode
Centro de Identidad	Use credenciais de curto prazo para acessar a AWS.	Seguindo as instruções em Conceitos básicos no Guia	Configure o acesso programático configurando o AWS CLI para uso AWS IAM Identity

Selecionar uma forma de gerenciar o administrador	Para	Por	Você também pode
e do IAM (Recomendado)	Isso está de acordo com as práticas recomendadas de segurança. Para obter informações sobre as práticas recomendadas, consulte Práticas recomendadas de segurança no IAM no Guia do usuário do IAM.	do usuário do AWS IAM Identity Center .	Center no Guia do AWS Command Line Interface usuário.
No IAM (Não recomendado)	Use credenciais de curto prazo para acessar a AWS.	Seguindo as instruções em Criar o seu primeiro usuário administrador e um grupo de usuários do IAM no Guia do usuário do IAM.	Para configurar o acesso programático, consulte Gerenciamento de chaves de acesso de usuários do IAM no Guia do usuário do IAM.

Instalar o software do AWS IoT Greengrass Core

AWS IoT Greengrassse estende AWS aos dispositivos de ponta para que eles possam agir com base nos dados que geram, enquanto os usam Nuvem AWS para gerenciamento, análise e armazenamento durável. Instale o software AWS IoT Greengrass Core em dispositivos de ponta para integração com AWS IoT Greengrass Nuvem AWS o.

> **Important**
>
> Antes de baixar e instalar o software AWS IoT Greengrass Core, verifique se seu dispositivo principal atende aos requisitos para instalar e executar o software AWS IoT Greengrass Core v2.0.

O software AWS IoT Greengrass Core inclui um instalador que configura seu dispositivo como um dispositivo principal do Greengrass. Ao executar o instalador, você pode configurar opções, como a pasta raiz e Região da AWS a ser usada. Você pode optar por fazer com que o instalador crie os recursos necessários AWS IoT e do IAM para você. Você também pode optar por implantar ferramentas de desenvolvimento local para configurar um dispositivo que você usa para desenvolvimento de componentes personalizados.

O software AWS IoT Greengrass principal requer os seguintes recursos AWS IoT e os recursos do IAM para se conectar ao Nuvem AWS e operar:

- Umo objeto do AWS IoT. Quando você registra um dispositivo como uma AWS IoT coisa, esse dispositivo pode usar um certificado digital para se autenticarAWS. Esse certificado permite que o dispositivo se comunique com AWS IoT AWS IoT Greengrass e. Para ter mais informações, consulte Autorização e autenticação do dispositivo para o AWS IoT Greengrass.
- (Opcional) Qualquer grupo de AWS IoT coisas. Você usa grupos de coisas para gerenciar frotas de dispositivos principais do Greengrass. Ao implantar componentes de software em seus dispositivos, você pode optar por implantar em dispositivos individuais ou em grupos de dispositivos. Você pode adicionar um dispositivo a um grupo de coisas para implantar os componentes de software desse grupo de coisas no dispositivo. Para ter mais informações, consulte Implemente AWS IoT Greengrass componentes em dispositivos.
- Um perfil do IAM. Os principais dispositivos do Greengrass usam o provedor de AWS IoT Core credenciais para autorizar chamadas para AWS serviços com uma função do IAM. Essa função permite que seu dispositivo interajaAWS IoT, envie registros para o Amazon CloudWatch Logs e baixe artefatos de componentes personalizados do Amazon Simple Storage Service (Amazon S3). Para ter mais informações, consulte Autorize os dispositivos principais a interagir com os serviços AWS.
- Um alias de AWS IoT função. Os dispositivos principais do Greengrass usam o alias de função para identificar a função do IAM a ser usada. O alias da função permite que você altere a função do IAM, mas mantenha a mesma configuração do dispositivo. Para obter mais informações,

consulte Autorização de chamadas diretas para AWS serviços no Guia do AWS IoT Core desenvolvedor.

Escolha uma das opções a seguir para instalar o software AWS IoT Greengrass Core em seu dispositivo principal.

- Instalação rápida

  Escolha essa opção para configurar um dispositivo principal do Greengrass com o mínimo de etapas possível. O instalador cria os recursos necessários AWS IoT e do IAM para você. Essa opção exige que você forneça AWS credenciais ao instalador para criar recursos no seuConta da AWS.

  Você não pode usar essa opção para instalar atrás de um firewall ou proxy de rede. Se seus dispositivos estiverem protegidos por um firewall ou proxy de rede, considere a instalação manual.

  Para ter mais informações, consulte Instale o software AWS IoT Greengrass principal com provisionamento automático de recursos.

- Instalação manual

  Escolha essa opção para criar os AWS recursos necessários manualmente ou para instalar atrás de um firewall ou proxy de rede. Ao usar uma instalação manual, você não precisa dar permissão ao instalador para criar recursos no seuConta da AWS, pois você cria os recursos necessários AWS IoT e do IAM. Você também pode configurar seu dispositivo para se conectar na porta 443 ou por meio de um proxy de rede. Você também pode configurar o software AWS IoT Greengrass Core para usar uma chave privada e um certificado que você armazena em um módulo de segurança de hardware (HSM), Trusted Platform Module (TPM) ou outro elemento criptográfico.

  Para ter mais informações, consulte Instale o software AWS IoT Greengrass principal com provisionamento manual de recursos.

- Instalação com provisionamento de AWS IoT frota

  Escolha essa opção para criar os AWS recursos necessários a partir de um modelo de provisionamento de AWS IoT frota. Você pode escolher essa opção para criar dispositivos semelhantes em uma frota ou se fabricar dispositivos que seus clientes ativarão posteriormente, como veículos ou dispositivos domésticos inteligentes. Os dispositivos usam certificados de declaração para autenticar e provisionar AWS recursos, incluindo um certificado de cliente

X.509 que o dispositivo usa para se conectar ao Nuvem AWS para operação normal. Você pode incorporar ou atualizar os certificados de solicitação no hardware do dispositivo durante a fabricação e usar o mesmo certificado de solicitação e chave para provisionar vários dispositivos. Você também pode configurar dispositivos para se conectar na porta 443 ou por meio de um proxy de rede.

Para ter mais informações, consulte Instale o software AWS IoT Greengrass principal com provisionamento de AWS IoT frota.

- Instalação com provisionamento personalizado

  Escolha essa opção para desenvolver um aplicativo Java personalizado que provisione os AWS recursos necessários. Você pode escolher essa opção se criar seus próprios certificados de cliente X.509 ou se quiser ter mais controle sobre o processo de provisionamento. AWS IoT Greengrassfornece uma interface que você pode implementar para trocar informações entre seu aplicativo de provisionamento personalizado e o instalador do software AWS IoT Greengrass Core.

  Para ter mais informações, consulte Instale o software AWS IoT Greengrass principal com provisionamento personalizado de recursos.

O AWS IoT Greengrass também fornece ambientes em contêineres que executam o software do AWS IoT Greengrass Core. Você pode usar um Dockerfile para executar AWS IoT Greengrass em um contêiner Docker.

Tópicos

Instale o software AWS IoT Greengrass principal com provisionamento automático de recursos

O software AWS IoT Greengrass Core inclui um instalador que configura seu dispositivo como um dispositivo principal do Greengrass. Para configurar um dispositivo rapidamente, o instalador pode provisionar a AWS IoT AWS IoT coisa, o grupo de coisas, a função do IAM e o alias da AWS

IoT função que o dispositivo principal precisa para operar. O instalador também pode implantar as ferramentas de desenvolvimento local no dispositivo principal, para que você possa usar o dispositivo para desenvolver e testar componentes de software personalizados. O instalador exige AWS credenciais para provisionar esses recursos e criar a implantação.

Se você não puder fornecer AWS credenciais para o dispositivo, poderá provisionar os AWS recursos que o dispositivo principal precisa para operar. Você também pode implantar as ferramentas de desenvolvimento em um dispositivo principal para usar como dispositivo de desenvolvimento. Isso permite que você forneça menos permissões ao dispositivo ao executar o instalador. Para ter mais informações, consulte Instale o software AWS IoT Greengrass principal com provisionamento manual de recursos.

> **Important**
>
> Antes de baixar o software AWS IoT Greengrass Core, verifique se seu dispositivo principal atende aos requisitos para instalar e executar o software AWS IoT Greengrass Core v2.0.

Tópicos

Configurar o ambiente do dispositivo

Siga as etapas nesta seção para configurar um dispositivo Linux ou Windows para ser usado como seu dispositivo AWS IoT Greengrass principal.

Configurar um dispositivo Linux

Para configurar um dispositivo Linux para AWS IoT Greengrass V2

1. Instale o Java Runtime, que o software AWS IoT Greengrass Core exige para ser executado. Recomendamos que você use as versões de suporte de longo prazo do Amazon Corretto ou do OpenJDK. É necessária a versão 8 ou superior. Os comandos a seguir mostram como instalar o OpenJDK no seu dispositivo.

- Para distribuições com base em Debian ou em Ubuntu:

```
sudo apt install default-jdk
```

- Para distribuições com base em Red Hat:

```
sudo yum install java-11-openjdk-devel
```

- Para Amazon Linux 2:

```
sudo amazon-linux-extras install java-openjdk11
```

- Para Amazon Linux 2023:

```
sudo dnf install java-11-amazon-corretto -y
```

Quando a instalação for concluída, execute o comando a seguir para verificar se o Java é executado no seu dispositivo Linux.

```
java -version
```

O comando imprime a versão do Java que é executada no dispositivo. Por exemplo, em uma distribuição baseada em Debian, o resultado pode ser semelhante ao exemplo a seguir.

```
openjdk version "11.0.9.1" 2020-11-04
OpenJDK Runtime Environment (build 11.0.9.1+1-post-Debian-1deb10u2)
OpenJDK 64-Bit Server VM (build 11.0.9.1+1-post-Debian-1deb10u2, mixed mode)
```

2. (Opcional) Crie o usuário e o grupo padrão do sistema que executa componentes no dispositivo. Você também pode optar por permitir que o instalador do software AWS IoT Greengrass Core crie esse usuário e grupo durante a instalação com o argumento do `--component-default-user` instalador. Para ter mais informações, consulte Argumentos de instalação.

```
sudo useradd --system --create-home ggc_user
sudo groupadd --system ggc_group
```

3. Verifique se o usuário que executa o software AWS IoT Greengrass Core (normalmente`root`) tem permissão para executar `sudo` com qualquer usuário e qualquer grupo.

a. Execute o comando a seguir para abrir o `/etc/sudoers` arquivo.

```
sudo visudo
```

b. Verifique se a permissão para o usuário se parece com o exemplo a seguir.

```
root    ALL=(ALL:ALL) ALL
```

4. (Opcional) Para executar funções Lambda em contêineres, você deve habilitar cgroups v1 e habilitar e montar os cgroups de memória e dispositivos. Se você não planeja executar funções Lambda em contêineres, pode pular esta etapa.

   Para habilitar essas opções de cgroups, inicialize o dispositivo com os seguintes parâmetros do kernel Linux.

   ```
   cgroup_enable=memory cgroup_memory=1 systemd.unified_cgroup_hierarchy=0
   ```

   Para obter informações sobre como visualizar e definir os parâmetros do kernel para seu dispositivo, consulte a documentação do sistema operacional e do carregador de inicialização. Siga as instruções para definir permanentemente os parâmetros do kernel.

5. Instale todas as outras dependências necessárias em seu dispositivo, conforme indicado na lista de requisitos emRequisitos do dispositivo.

Configurar um dispositivo Windows

> **Note**
>
> Esse recurso está disponível para a versão 2.5.0 e posterior do componente de núcleo do Greengrass.

Para configurar um dispositivo Windows para AWS IoT Greengrass V2

1. Instale o Java Runtime, que o software AWS IoT Greengrass Core exige para ser executado. Recomendamos que você use as versões de suporte de longo prazo do Amazon Corretto ou do OpenJDK. É necessária a versão 8 ou superior.

2. Verifique se o Java está disponível na variável de sistema PATH e, caso contrário, adicione-a. A LocalSystem conta executa o software AWS IoT Greengrass Core, então você deve adicionar Java à variável de sistema PATH em vez da variável de usuário PATH para seu usuário. Faça o seguinte:

   a. Pressione a tecla Windows para abrir o menu Iniciar.

   b. Digite **`environment variables`** para pesquisar as opções do sistema no menu Iniciar.

   c. Nos resultados da pesquisa do menu Iniciar, escolha Editar as variáveis de ambiente do sistema para abrir a janela Propriedades do sistema.

   d. Escolha variáveis de ambiente... para abrir a janela Variáveis de ambiente.

   e. Em Variáveis do sistema, selecione Caminho e, em seguida, escolha Editar. Na janela Editar variável de ambiente, você pode visualizar cada caminho em uma linha separada.

   f. Verifique se o caminho para a `bin` pasta da instalação do Java está presente. O caminho pode ser semelhante ao exemplo a seguir.

   ```
   C:\\Program Files\\Amazon Corretto\\jdk11.0.13_8\\bin
   ```

   g. Se a `bin` pasta da instalação do Java estiver ausente do Path, escolha Novo para adicioná-la e, em seguida, escolha OK.

3. Abra o prompt de comando do Windows (`cmd.exe`) como administrador.

4. Crie o usuário padrão na LocalSystem conta no dispositivo Windows. *`Substitua a senha`* por uma senha segura.

   ```
   net user /add ggc_user password
   ```

   > **Tip**
   >
   > Dependendo da configuração do Windows, a senha do usuário pode ser definida para expirar em uma data futura. Para garantir que seus aplicativos Greengrass continuem operando, monitore quando a senha expira e atualize-a antes que ela expire. Você também pode definir a senha do usuário para nunca expirar.
   >
   > - Para verificar quando um usuário e sua senha expiram, execute o comando a seguir.
   >
   >   ```
   >   net user ggc_user | findstr /C:expires
   >   ```
   >
   > - Para definir que a senha de um usuário nunca expire, execute o comando a seguir.

```
wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
```

- Se você estiver usando o Windows 10 ou posterior, onde o wmiccomando está obsoleto, execute o comando a seguir. PowerShell

```
Get-CimInstance -Query "SELECT * from Win32_UserAccount WHERE name =
 'ggc_user'" | Set-CimInstance -Property @{PasswordExpires="False"}
```

5. Baixe e instale o PsExecutilitário da Microsoft no dispositivo.
6. Use o PsExec utilitário para armazenar o nome de usuário e a senha do usuário padrão na instância do Credential Manager da LocalSystem conta. Substitua a *senha* pela senha do usuário que você definiu anteriormente.

```
psexec -s cmd /c cmdkey /generic:ggc_user /user:ggc_user /pass:password
```

Se for PsExec License Agreementaberto, escolha Acceptconcordar com a licença e execute o comando.

> **Note**
>
> Em dispositivos Windows, a LocalSystem conta executa o núcleo Greengrass, e você deve usar o PsExec utilitário para armazenar as informações padrão do usuário na conta. LocalSystem O uso do aplicativo Credential Manager armazena essas informações na conta do Windows do usuário atualmente conectado, em vez da LocalSystem conta.

Forneça AWS credenciais para o dispositivo

Forneça suas AWS credenciais ao seu dispositivo para que o instalador possa provisionar os AWS recursos necessários. Para mais informações sobre as permissões necessárias, consulte Política mínima de IAM para o instalador provisionar recursos.

Para fornecer AWS credenciais ao dispositivo

- Forneça suas AWS credenciais ao dispositivo para que o instalador possa provisionar os recursos do IAM AWS IoT e do IAM para seu dispositivo principal. Para aumentar a segurança,

recomendamos que você obtenha credenciais temporárias para uma função do IAM que permita somente as permissões mínimas necessárias para provisionar. Para ter mais informações, consulte Política mínima de IAM para o instalador provisionar recursos.

> **Note**
>
> O instalador não salva nem armazena suas credenciais.

No seu dispositivo, faça o seguinte para recuperar as credenciais e disponibilizá-las para o AWS IoT Greengrass instalador do software Core:

- (Recomendado) Use credenciais temporárias de AWS IAM Identity Center

  a. Forneça o ID da chave de acesso, a chave de acesso secreta e o token da sessão do IAM Identity Center. Para obter mais informações, consulte Atualização manual de credenciais em Obter e atualizar credenciais temporárias no guia do usuário do IAM Identity Center.

  b. Execute os comandos a seguir para fornecer as credenciais para o software AWS IoT Greengrass principal.

  Linux or Unix

  ```
  export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
  export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
  export AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o5OytwEXAMPLE=
  ```

  Windows Command Prompt (CMD)

  ```
  set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
  set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
  set AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o5OytwEXAMPLE=
  ```

  PowerShell

  ```
  $env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
  $env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
  $env:AWS_SESSION_TOKEN="AQoDYXdzEJr1K...o5OytwEXAMPLE="
  ```

- Use credenciais de segurança temporárias de uma função do IAM:

a. Forneça o ID da chave de acesso, a chave de acesso secreta e o token da sessão de uma função do IAM que você assume. Para obter mais informações sobre como recuperar essas credenciais, consulte Solicitação de credenciais de segurança temporárias no Guia do usuário do IAM.

b. Execute os comandos a seguir para fornecer as credenciais para o software AWS IoT Greengrass principal.

Linux or Unix

```
export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
export AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o5OytwEXAMPLE=
```

Windows Command Prompt (CMD)

```
set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
set AWS_SESSION_TOKEN=AQoDYXdzEJr1K...o5OytwEXAMPLE=
```

PowerShell

```
$env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
$env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
$env:AWS_SESSION_TOKEN="AQoDYXdzEJr1K...o5OytwEXAMPLE="
```

- Use credenciais de longo prazo de um usuário do IAM:

a. Forneça o ID da chave de acesso e a chave de acesso secreta para seu usuário do IAM. Você pode criar um usuário do IAM para provisionamento que você excluirá posteriormente. Para saber a política do IAM a ser fornecida ao usuário, consulte Política mínima de IAM para o instalador provisionar recursos. Para obter mais informações sobre como recuperar credenciais de longo prazo, consulte Gerenciamento de chaves de acesso para usuários do IAM no Guia do usuário do IAM.

b. Execute os comandos a seguir para fornecer as credenciais para o software AWS IoT Greengrass principal.

Linux or Unix

```
export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
```

Windows Command Prompt (CMD)

```
set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
```

PowerShell

```
$env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
$env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
```

c. (Opcional) Se você criou um usuário do IAM para provisionar seu dispositivo Greengrass, exclua o usuário.

d. (Opcional) Se você usou o ID da chave de acesso e a chave de acesso secreta de um usuário do IAM existente, atualize as chaves do usuário para que elas não sejam mais válidas. Para obter mais informações, consulte Atualização das chaves de acesso no guia AWS Identity and Access Management do usuário.

Baixe o software AWS IoT Greengrass Core

Você pode baixar a versão mais recente do software AWS IoT Greengrass Core no seguinte local:

- https://d2s8p88vqu9w66.cloudfront.net/releases/ greengrass-nucleus-latest .zip

> **Note**
>
> Você pode baixar uma versão específica do software AWS IoT Greengrass Core no seguinte local. Substitua a *versão* pela versão a ser baixada.
>
> ```
> https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip
> ```

Para baixar o software AWS IoT Greengrass Core

1. Em seu dispositivo principal, baixe o software AWS IoT Greengrass Core para um arquivo chamado`greengrass-nucleus-latest.zip`.

   Linux or Unix

   ```
   curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip > greengrass-nucleus-latest.zip
   ```

   Windows Command Prompt (CMD)

   ```
   curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip > greengrass-nucleus-latest.zip
   ```

   PowerShell

   ```
   iwr -Uri https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip -OutFile greengrass-nucleus-latest.zip
   ```

   Ao fazer download desse software, você concorda com o Contrato de licença do software do Greengrass Core.

2. (Opcional) Para verificar a assinatura do software Greengrass nucleus

   > **Note**
   >
   > Esse recurso está disponível com o Greengrass nucleus versão 2.9.5 e posterior.

   a. Use o comando a seguir para verificar a assinatura do artefato do núcleo Greengrass:

      Linux or Unix

      ```
      jarsigner -verify -certs -verbose greengrass-nucleus-latest.zip
      ```

Windows Command Prompt (CMD)

O nome do arquivo pode parecer diferente dependendo da versão do JDK que você instala. *jdk17.0.6_10*Substitua pela versão do JDK que você instalou.

```
"C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe" -
verify -certs -verbose greengrass-nucleus-latest.zip
```

PowerShell

O nome do arquivo pode parecer diferente dependendo da versão do JDK que você instala. *jdk17.0.6_10*Substitua pela versão do JDK que você instalou.

```
'C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe' -
verify -certs -verbose greengrass-nucleus-latest.zip
```

b. A `jarsigner` invocação produz uma saída que indica os resultados da verificação.

   i. Se o arquivo zip do Greengrass nucleus estiver assinado, a saída conterá a seguinte declaração:

   ```
   jar verified.
   ```

   ii. Se o arquivo zip do Greengrass nucleus não estiver assinado, a saída conterá a seguinte declaração:

   ```
   jar is unsigned.
   ```

c. Se você forneceu a `-certs` opção Jarsigner junto com `-verbose` as opções `-verify` e, a saída também incluirá informações detalhadas do certificado do assinante.

3. Descompacte o software AWS IoT Greengrass Core em uma pasta no seu dispositivo. *GreengrassInstaller*Substitua pela pasta que você deseja usar.

   Linux or Unix

   ```
   unzip greengrass-nucleus-latest.zip -d GreengrassInstaller && rm greengrass-
   nucleus-latest.zip
   ```

Windows Command Prompt (CMD)

```
mkdir GreengrassInstaller && tar -xf greengrass-nucleus-latest.zip -
C GreengrassInstaller && del greengrass-nucleus-latest.zip
```

PowerShell

```
Expand-Archive -Path greengrass-nucleus-latest.zip -DestinationPath .\
\GreengrassInstaller
rm greengrass-nucleus-latest.zip
```

4. (Opcional) Execute o comando a seguir para ver a versão do software AWS IoT Greengrass Core.

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

> **Important**
>
> Se você instalar uma versão do núcleo do Greengrass anterior à v2.4.0, não remova essa pasta depois de instalar o software Core. AWS IoT Greengrass O software AWS IoT Greengrass Core usa os arquivos dessa pasta para ser executado.
> Se você baixou a versão mais recente do software, instale a versão 2.4.0 ou posterior e poderá remover essa pasta depois de instalar o software AWS IoT Greengrass Core.

Instalar o software do AWS IoT Greengrass Core

Execute o instalador com argumentos que especificam o seguinte:

- Crie os AWS recursos que o dispositivo principal precisa para operar.
- Especifique o uso do usuário do `ggc_user` sistema para executar componentes de software no dispositivo principal. Em dispositivos Linux, esse comando também especifica o uso do grupo do `ggc_group` sistema, e o instalador cria o usuário e o grupo do sistema para você.
- Configure o software AWS IoT Greengrass Core como um serviço do sistema que é executado na inicialização. Em dispositivos Linux, isso requer o sistema init Systemd.

> **Important**
>
> Nos dispositivos principais do Windows, você deve configurar o software AWS IoT Greengrass Core como um serviço do sistema.

Para configurar um dispositivo de desenvolvimento com ferramentas de desenvolvimento local, especifique o `--deploy-dev-tools true` argumento. As ferramentas de desenvolvimento local podem levar até um minuto para serem implantadas após a conclusão da instalação.

Para obter mais informações sobre os argumentos que você pode especificar, consulteArgumentos de instalação.

> **Note**
>
> Se você estiver executando AWS IoT Greengrass em um dispositivo com memória limitada, poderá controlar a quantidade de memória que o software AWS IoT Greengrass Core usa. Para controlar a alocação de memória, você pode definir as opções de tamanho da pilha da JVM no parâmetro de `jvmOptions` configuração do componente do núcleo. Para ter mais informações, consulte Controle a alocação de memória com opções de JVM.

Como instalar o software do AWS IoT Greengrass Core

1. Execute o instalador AWS IoT Greengrass Core. Substitua os valores dos argumentos em seu comando da seguinte maneira.

   a. */greengrass/v2*ou *C:\greengrass\v2*: o caminho para a pasta raiz a ser usada para instalar o software AWS IoT Greengrass Core.

   b. *GreengrassInstaller*. O caminho para a pasta em que você descompactou o instalador do software AWS IoT Greengrass Core.

   c. *região*. O Região da AWS no qual encontrar ou criar recursos.

   d. *MyGreengrassCore*. O nome da AWS IoT coisa para o seu dispositivo principal do Greengrass. Se a coisa não existir, o instalador a cria. O instalador baixa os certificados para autenticar a AWS IoT coisa. Para ter mais informações, consulte Autorização e autenticação do dispositivo para o AWS IoT Greengrass.

> **Note**
>
> O nome da coisa não pode conter caracteres de dois pontos (`:`).

e. *`MyGreengrassCoreGroup`*. O nome do grupo de AWS IoT coisas do seu dispositivo principal do Greengrass. Se o grupo de coisas não existir, o instalador o cria e adiciona a coisa a ele. Se o grupo de coisas existir e tiver uma implantação ativa, o dispositivo principal baixará e executará o software especificado pela implantação.

> **Note**
>
> O nome do grupo de coisas não pode conter caracteres de dois pontos (`:`).

f. *`Greengrass ThingPolicy`* V2 IoT. O nome da AWS IoT política que permite que os dispositivos principais do Greengrass se comuniquem com e. AWS IoT AWS IoT Greengrass Se a AWS IoT política não existir, o instalador cria uma AWS IoT política permissiva com esse nome. Você pode restringir as permissões dessa política para seu caso de uso. Para ter mais informações, consulte AWS IoTPolítica mínima para dispositivos AWS IoT Greengrass V2 principais.

g. *`Greengrass TokenExchangeRole`* V2. O nome da função do IAM que permite que o dispositivo principal do Greengrass obtenha credenciais temporáriasAWS. Se a função não existir, o instalador a cria, cria e anexa uma política chamada*`GreengrassV2TokenExchangeRole`*`Access`. Para ter mais informações, consulte Autorize os dispositivos principais a interagir com os serviços AWS.

h. *`GreengrassCoreTokenExchangeRoleAlias`*. O alias para a função do IAM que permite que o dispositivo principal do Greengrass obtenha credenciais temporárias posteriormente. Se o alias da função não existir, o instalador o cria e o aponta para a função do IAM especificada por você. Para ter mais informações, consulte Autorize os dispositivos principais a interagir com os serviços AWS.

Linux or Unix

```
sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE \
  -jar ./GreengrassInstaller/lib/Greengrass.jar \
  --aws-region region \
  --thing-name MyGreengrassCore \
```

```
  --thing-group-name MyGreengrassCoreGroup \
  --thing-policy-name GreengrassV2IoTThingPolicy \
  --tes-role-name GreengrassV2TokenExchangeRole \
  --tes-role-alias-name GreengrassCoreTokenExchangeRoleAlias \
  --component-default-user ggc_user:ggc_group \
  --provision true \
  --setup-system-service true
```

Windows Command Prompt (CMD)

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" ^
  -jar ./GreengrassInstaller/lib/Greengrass.jar ^
  --aws-region region ^
  --thing-name MyGreengrassCore ^
  --thing-group-name MyGreengrassCoreGroup ^
  --thing-policy-name GreengrassV2IoTThingPolicy ^
  --tes-role-name GreengrassV2TokenExchangeRole ^
  --tes-role-alias-name GreengrassCoreTokenExchangeRoleAlias ^
  --component-default-user ggc_user ^
  --provision true ^
  --setup-system-service true
```

PowerShell

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" `
  -jar ./GreengrassInstaller/lib/Greengrass.jar `
  --aws-region region `
  --thing-name MyGreengrassCore `
  --thing-group-name MyGreengrassCoreGroup `
  --thing-policy-name GreengrassV2IoTThingPolicy `
  --tes-role-name GreengrassV2TokenExchangeRole `
  --tes-role-alias-name GreengrassCoreTokenExchangeRoleAlias `
  --component-default-user ggc_user `
  --provision true `
  --setup-system-service true
```

> **Important**
>
> Nos dispositivos principais do Windows, você deve especificar `--setup-system-service true` a configuração do software AWS IoT Greengrass Core como um serviço do sistema.

O instalador imprime as seguintes mensagens se for bem-sucedido:

- Se você especificar`--provision`, o instalador `Successfully configured Nucleus with provisioned resource details` imprimirá se configurou os recursos com êxito.
- Se você especificar`--deploy-dev-tools`, o instalador `Configured Nucleus to deploy aws.greengrass.Cli component` imprimirá se ele criou a implantação com êxito.
- Se você especificar`--setup-system-service true`, o instalador imprimirá `Successfully set up Nucleus as a system service` se configurou e executou o software como um serviço.
- Se você não especificar`--setup-system-service true`, o instalador `Launched Nucleus successfully` imprimirá se foi bem-sucedido e executou o software.

2. Ignore esta etapa se você instalou a Núcleo Greengrass versão 2.0.4 ou posterior. Se você baixou a versão mais recente do software, instalou a versão 2.0.4 ou posterior.

   Execute o comando a seguir para definir as permissões de arquivo necessárias para a pasta raiz do software AWS IoT Greengrass Core. */greengrass/v2*Substitua pela pasta raiz especificada no comando de instalação e substitua */greengrass* pela pasta principal da pasta raiz.

   ```
   sudo chmod 755 /greengrass/v2 && sudo chmod 755 /greengrass
   ```

Se você instalou o software AWS IoT Greengrass Core como um serviço do sistema, o instalador executa o software para você. Caso contrário, você deverá executar o software manualmente. Para ter mais informações, consulte Execute o software AWS IoT Greengrass Core.

> **Note**
>
> Por padrão, a função do IAM criada pelo instalador não permite acesso aos artefatos dos componentes nos buckets do S3. Para implantar componentes personalizados que definem artefatos no Amazon S3, você deve adicionar permissões à função para permitir que seu dispositivo principal recupere artefatos de componentes. Para ter mais informações, consulte Permitir acesso aos buckets do S3 para artefatos de componentes.
> Se você ainda não tem um bucket do S3 para artefatos de componentes, você pode adicionar essas permissões depois de criar um bucket.

Para obter mais informações sobre como configurar e usar o softwareAWS IoT Greengrass, consulte o seguinte:

- Configurar o software AWS IoT Greengrass principal
- Desenvolva AWS IoT Greengrass componentes
- Implemente AWS IoT Greengrass componentes em dispositivos
- Interface de linha de comando do Greengrass

Instale o software AWS IoT Greengrass principal com provisionamento manual de recursos

O software AWS IoT Greengrass Core inclui um instalador que configura seu dispositivo como um dispositivo principal do Greengrass. Para configurar um dispositivo manualmente, você pode criar os recursos necessários AWS IoT e do IAM para o dispositivo usar. Se você criar esses recursos manualmente, não precisará fornecer AWS credenciais ao instalador.

Ao instalar manualmente o software AWS IoT Greengrass Core, você também pode configurar o dispositivo para usar um proxy de rede ou conectar-se AWS à porta 443. Talvez seja necessário especificar essas opções de configuração se o dispositivo estiver protegido por um firewall ou proxy de rede, por exemplo. Para ter mais informações, consulte Conectar-se à porta 443 ou por meio de um proxy de rede.

Você também pode configurar o software AWS IoT Greengrass Core para usar um módulo de segurança de hardware (HSM) por meio da interface PKCS #11. Esse recurso permite que você armazene com segurança arquivos de chave privada e certificado para que eles não sejam expostos ou duplicados no software. Você pode armazenar chaves privadas e certificados em um módulo

de hardware, como um HSM, um Trusted Platform Module (TPM) ou outro elemento criptográfico. Esse recurso está disponível somente em dispositivos Linux. Para obter mais informações sobre a segurança do hardware e os requisitos para usá-lo, consulteIntegração de segurança de hardware.

> **Important**
>
> Antes de baixar o software AWS IoT Greengrass Core, verifique se seu dispositivo principal atende aos requisitos para instalar e executar o software AWS IoT Greengrass Core v2.0.

Tópicos

- Recupere endpoints AWS IoT
- Crie qualquer AWS IoT coisa
- Crie o certificado da coisa
- Configure o certificado da coisa
- Crie uma função de troca de tokens
- Baixe certificados para o dispositivo
- Configurar o ambiente do dispositivo
- Baixe o software AWS IoT Greengrass Core
- Instale o software AWS IoT Greengrass Core

Recupere endpoints AWS IoT

Obtenha os AWS IoT endpoints para você Conta da AWS e salve-os para usar mais tarde. Seu dispositivo usa esses endpoints para se conectar a. AWS IoT Faça o seguinte:

1. Obtenha o endpoint de AWS IoT dados para você Conta da AWS.

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

   A resposta será semelhante ao exemplo a seguir, se a solicitação for bem-sucedida.

   ```
   {
     "endpointAddress": "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
   }
   ```

2. Obtenha o endpoint AWS IoT de credenciais para seu. Conta da AWS

```
aws iot describe-endpoint --endpoint-type iot:CredentialProvider
```

A resposta será semelhante ao exemplo a seguir, se a solicitação for bem-sucedida.

```
{
  "endpointAddress": "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
}
```

Crie qualquer AWS IoT coisa

AWS IoT as coisas representam dispositivos e entidades lógicas que se conectam AWS IoT a. Os principais dispositivos do Greengrass são AWS IoT coisas. Quando você registra um dispositivo como uma AWS IoT coisa, esse dispositivo pode usar um certificado digital para se autenticar AWS.

Nesta seção, você cria AWS IoT algo que representa seu dispositivo.

Para criar qualquer AWS IoT coisa

1. Crie qualquer AWS IoT coisa para o seu dispositivo. No seu computador de desenvolvimento, execute o comando a seguir.

   - Substitua *MyGreengrassCore* pelo nome da coisa a ser usada. Esse nome também é o nome do seu dispositivo principal do Greengrass.

     > **Note**
     >
     > O nome da coisa não pode conter caracteres de dois pontos (`:`).

```
aws iot create-thing --thing-name MyGreengrassCore
```

A resposta será semelhante ao exemplo a seguir, se a solicitação for bem-sucedida.

```
{
  "thingName": "MyGreengrassCore",
  "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
```

```
  "thingId": "8cb4b6cd-268e-495d-b5b9-1713d71dbf42"
}
```

2. (Opcional) Adicione a AWS IoT coisa a um grupo de coisas novo ou existente. Você usa grupos de coisas para gerenciar frotas de dispositivos principais do Greengrass. Ao implantar componentes de software em seus dispositivos, você pode segmentar dispositivos individuais ou grupos de dispositivos. Você pode adicionar um dispositivo a um grupo de coisas com uma implantação ativa do Greengrass para implantar os componentes de software desse grupo de coisas no dispositivo. Faça o seguinte:

   a. (Opcional) Crie um grupo de AWS IoT coisas.

      - *MyGreengrassCoreGroup*Substitua pelo nome do grupo de coisas a ser criado.

      > **Note**
      >
      > O nome do grupo de coisas não pode conter caracteres de dois pontos (`:`).

      ```
      aws iot create-thing-group --thing-group-name MyGreengrassCoreGroup
      ```

      A resposta será semelhante ao exemplo a seguir, se a solicitação for bem-sucedida.

      ```
      {
        "thingGroupName": "MyGreengrassCoreGroup",
        "thingGroupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/
      MyGreengrassCoreGroup",
        "thingGroupId": "4df721e1-ff9f-4f97-92dd-02db4e3f03aa"
      }
      ```

   b. Adicione a AWS IoT coisa a um grupo de coisas.

      - Substitua *MyGreengrassCore* pelo nome da sua AWS IoT coisa.
      - *MyGreengrassCoreGroup*Substitua pelo nome do grupo de coisas.

      ```
      aws iot add-thing-to-thing-group --thing-name MyGreengrassCore --thing-group-
      name MyGreengrassCoreGroup
      ```

      O comando não terá nenhuma saída se a solicitação for bem-sucedida.

Crie o certificado da coisa

Quando você registra um dispositivo como uma AWS IoT coisa, esse dispositivo pode usar um certificado digital para se autenticar AWS. Esse certificado permite que o dispositivo se comunique com AWS IoT AWS IoT Greengrass e.

Nesta seção, você cria e baixa certificados que seu dispositivo pode usar para se conectar AWS.

Se você quiser configurar o software AWS IoT Greengrass Core para usar um módulo de segurança de hardware (HSM) para armazenar com segurança a chave privada e o certificado, siga as etapas para criar o certificado a partir de uma chave privada em um HSM. Caso contrário, siga as etapas para criar o certificado e a chave privada no AWS IoT serviço. O recurso de segurança de hardware está disponível somente em dispositivos Linux. Para obter mais informações sobre a segurança do hardware e os requisitos para usá-lo, consulteIntegração de segurança de hardware.

Crie o certificado e a chave privada no AWS IoT serviço

Para criar o certificado da coisa

1. Crie uma pasta na qual você baixa os certificados da AWS IoT coisa.

   ```
   mkdir greengrass-v2-certs
   ```

2. Crie e baixe os certificados da AWS IoT coisa.

   ```
   aws iot create-keys-and-certificate --set-as-active --certificate-pem-outfile
    greengrass-v2-certs/device.pem.crt --public-key-outfile greengrass-v2-certs/
   public.pem.key --private-key-outfile greengrass-v2-certs/private.pem.key
   ```

   A resposta será semelhante ao exemplo a seguir, se a solicitação for bem-sucedida.

   ```
   {
     "certificateArn": "arn:aws:iot:us-west-2:123456789012:cert/
   aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
     "certificateId":
    "aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
     "certificatePem": "-----BEGIN CERTIFICATE-----
   MIICiTCCAfICCQD6m7oRw0uXOjANBgkqhkiG9w
    0BAQUFADCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZ
    WF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIw
    EAYDVQQDEwlUZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5
   ```

```
 jb20wHhcNMTEwNDI1MjA0NTIxWhcNMTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBh
 MCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBb
 WF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMx
 HzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5jb20wgZ8wDQYJKoZIhvcNAQE
 BBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ21uUSfwfEvySWtC2XADZ4nB+BLYgVI
 k60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9TrDHudUZg3qX4waLG5M43q7Wgc/MbQ
 ITxOUSQv7c7ugFFDzQGBzZswY6786m86gpEIbb3OhjZnzcvQAaRHhdlQWIMm2nr
 AgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4nUhVVxYUntneD9+h8Mg9q6q+auN
 KyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0FkbFFBjvSfpJIlJ00zbhNYS5f6Guo
 EDmFJl0ZxBHjJnyp378OD8uTs7fLvjx79LjSTbNYiytVbZPQUQ5Yaxu2jXnimvw
 3rrszlaEXAMPLE=
-----END CERTIFICATE-----",
  "keyPair": {
    "PublicKey": "-----BEGIN PUBLIC KEY-----\
MIIBIjANBgkqhkEXAMPLEQEFAAOCAQ8AMIIBCgKCAQEAEXAMPLE1nnyJwKSMHw4h\
MMEXAMPLEuuN/dMAS3fyce8DW/4+EXAMPLEyjmoF/YVF/gHr99VEEXAMPLE5VF13\
59VK7cEXAMPLE67GK+y+jikqXOgHh/xJTwo
+sGpWEXAMPLEDz18xOd2ka4tCzuWEXAMPLEahJbYkCPUBSU8opVkR7qkEXAMPLE1DR6sx2HocliOOLtu6Fkw91swQWE
\\GB3ZPrNh0PzQYvjUStZeccyNCx2EXAMPLEvp9mQOUXP6plfgxwKRX2fEXAMPLEDa\
hJLXkX3rHU2xbxJSq7D+XEXAMPLEcw+LyFhI5mgFRl88eGdsAEXAMPLElnI9EesG\
FQIDAQAB\
-----END PUBLIC KEY-----\
",
    "PrivateKey": "-----BEGIN RSA PRIVATE KEY-----\
key omitted for security reasons\
-----END RSA PRIVATE KEY-----\
"
  }
}
```

Salve o Amazon Resource Name (ARN) do certificado para usar na configuração posterior do certificado.

Crie o certificado a partir de uma chave privada em um HSM

> **Note**
>
> Esse recurso está disponível para a versão 2.5.3 e posterior do componente núcleo do Greengrass. AWS IoT Greengrass atualmente não oferece suporte a esse recurso nos dispositivos principais do Windows.

Para criar o certificado da coisa

1. No dispositivo principal, inicialize um token PKCS #11 no HSM e gere uma chave privada. A chave privada deve ser uma chave RSA com um tamanho de chave RSA-2048 (ou maior) ou uma chave ECC.

   > **Note**
   >
   > Para usar um módulo de segurança de hardware com chaves ECC, você deve usar o Greengrass nucleus v2.5.6 ou posterior.
   > Para usar um módulo de segurança de hardware e um gerenciador secreto, você deve usar um módulo de segurança de hardware com chaves RSA.

   Consulte a documentação do seu HSM para saber como inicializar o token e gerar a chave privada. Se seu HSM oferecer suporte a IDs de objeto, especifique uma ID de objeto ao gerar a chave privada. Salve o ID do slot, o PIN do usuário, o rótulo do objeto e o ID do objeto (se o HSM usar um) que você especifica ao inicializar o token e gerar a chave privada. Você usa esses valores posteriormente ao importar o certificado do item para o HSM e configurar o software AWS IoT Greengrass Core.
2. Crie uma solicitação de assinatura de certificado (CSR) a partir da chave privada. AWS IoT usa essa CSR para criar um certificado para a chave privada que você gerou no HSM. Para obter informações sobre como criar uma CSR a partir da chave privada, consulte a documentação do seu HSM. O CSR é um arquivo, como `iotdevicekey.csr`.
3. Copie a CSR do dispositivo para o computador de desenvolvimento. Se o SSH e o SCP estiverem habilitados no computador de desenvolvimento e no dispositivo, você poderá usar o `scp` comando no computador de desenvolvimento para transferir o CSR. Substitua o endereço *IP do dispositivo pelo endereço IP do seu dispositivo e substitua ~/iotdevicekey.csr pelo caminho para o arquivo CSR no dispositivo*.

   ```
   scp device-ip-address:~/iotdevicekey.csr iotdevicekey.csr
   ```
4. No seu computador de desenvolvimento, crie uma pasta na qual você baixa o certificado da AWS IoT coisa.

   ```
   mkdir greengrass-v2-certs
   ```

5. Use o arquivo CSR para criar e baixar o certificado da AWS IoT coisa em seu computador de desenvolvimento.

```
aws iot create-certificate-from-csr --set-as-active --certificate-signing-
request=file://iotdevicekey.csr --certificate-pem-outfile greengrass-v2-certs/
device.pem.crt
```

A resposta será semelhante ao exemplo a seguir, se a solicitação for bem-sucedida.

```
{
  "certificateArn": "arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
  "certificateId":
 "aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
  "certificatePem": "-----BEGIN CERTIFICATE-----
MIICiTCCAfICCQD6m7oRw0uXOjANBgkqhkiG9w
 0BAQUFADCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZ
 WF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIw
 EAYDVQQDEwlUZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5
 jb20wHhcNMTEwNDI1MjA0NTIxWhcNMTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBh
 MCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBb
 WF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMx
 HzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5jb20wgZ8wDQYJKoZIhvcNAQE
 BBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ21uUSfwfEvySWtC2XADZ4nB+BLYgVI
 k60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9TrDHudUZg3qX4waLG5M43q7Wgc/MbQ
 ITxOUSQv7c7ugFFDzQGBzZswY6786m86gpEIbb3OhjZnzcvQAaRHhdlQWIMm2nr
 AgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4nUhVVxYUntneD9+h8Mg9q6q+auN
 KyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0FkbFFBjvSfpJIlJ00zbhNYS5f6Guo
 EDmFJl0ZxBHjJnyp378OD8uTs7fLvjx79LjSTbNYiytVbZPQUQ5Yaxu2jXnimvw
 3rrszlaEXAMPLE=
-----END CERTIFICATE-----"
}
```

Salve o ARN do certificado para usá-lo na configuração posterior do certificado.

Configure o certificado da coisa

Anexe o certificado do AWS IoT item ao que você criou anteriormente e adicione uma AWS IoT política ao certificado para definir as AWS IoT permissões para o dispositivo principal.

Para configurar o certificado da coisa

1. Anexe o certificado à AWS IoT coisa.

   - Substitua *MyGreengrassCore* pelo nome da sua AWS IoT coisa.
   - Substitua o certificado Amazon Resource Name (ARN) pelo ARN do certificado que você criou na etapa anterior.

   ```
   aws iot attach-thing-principal --thing-name MyGreengrassCore
    --principal arn:aws:iot:us-west-2:123456789012:cert/
   aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
   ```

   O comando não terá nenhuma saída se a solicitação for bem-sucedida.

2. Crie e anexe uma AWS IoT política que defina as AWS IoT permissões para seu dispositivo principal do Greengrass. A política a seguir permite acesso a todos os tópicos do MQTT e operações do Greengrass, para que seu dispositivo funcione com aplicativos personalizados e futuras alterações que exijam novas operações do Greengrass. Você pode restringir essa política com base no seu caso de uso. Para ter mais informações, consulte AWS IoTPolítica mínima para dispositivos AWS IoT Greengrass V2 principais.

   Se você já configurou um dispositivo principal do Greengrass, pode anexar sua AWS IoT política em vez de criar uma nova.

   Faça o seguinte:

   a. Crie um arquivo que contenha o documento AWS IoT de política exigido pelos dispositivos principais do Greengrass.

      Por exemplo, em um sistema baseado em Linux, você pode executar o seguinte comando para usar o GNU nano para criar o arquivo.

      ```
      nano greengrass-v2-iot-policy.json
      ```

      Copie o seguinte JSON no arquivo.

      ```
      {
        "Version": "2012-10-17",
        "Statement": [
      ```

```
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish",
        "iot:Subscribe",
        "iot:Receive",
        "iot:Connect",
        "greengrass:*"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

b. Crie uma AWS IoT política a partir do documento de política.

- Substitua *GreengrassV2IoT* pelo nome da ThingPolicy política a ser criada.

```
aws iot create-policy --policy-name GreengrassV2IoTThingPolicy --policy-
document file://greengrass-v2-iot-policy.json
```

A resposta será semelhante ao exemplo a seguir, se a solicitação for bem-sucedida.

```
{
  "policyName": "GreengrassV2IoTThingPolicy",
  "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassV2IoTThingPolicy",
  "policyDocument": "{
    \\"Version\\": \\"2012-10-17\\",
    \\"Statement\\": [
      {
        \\"Effect\\": \\"Allow\\",
        \\"Action\\": [
          \\"iot:Publish\\",
          \\"iot:Subscribe\\",
          \\"iot:Receive\\",
          \\"iot:Connect\\",
          \\"greengrass:*\\"
        ],
        \\"Resource\\": [
```

```
          \\"*\\"
        ]
      }
    ]
  }",
  "policyVersionId": "1"
}
```

c. Anexe a AWS IoT política ao certificado da AWS IoT coisa.

- Substitua *GreengrassV2IoT pelo nome da política ThingPolicy a ser* anexada.
- Substitua o ARN de destino pelo ARN do certificado de sua coisa. AWS IoT

```
aws iot attach-policy --policy-name GreengrassV2IoTThingPolicy
 --target arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
```

O comando não terá nenhuma saída se a solicitação for bem-sucedida.

Crie uma função de troca de tokens

Os dispositivos principais do Greengrass usam uma função de serviço do IAM, chamada função de troca de tokens, para autorizar chamadas para serviços. AWS O dispositivo usa o provedor de AWS IoT credenciais para obter AWS credenciais temporárias para essa função, o que permite que o dispositivo interaja AWS IoT, envie registros para o Amazon CloudWatch Logs e baixe artefatos de componentes personalizados do Amazon S3. Para ter mais informações, consulte Autorize os dispositivos principais a interagir com os serviços AWS.

Você usa um alias de AWS IoT função para configurar a função de troca de tokens para os dispositivos principais do Greengrass. Os aliases de função permitem que você altere a função de troca de tokens de um dispositivo, mas mantenha a mesma configuração do dispositivo. Para obter mais informações, consulte Autorização de chamadas diretas para AWS serviços no Guia do AWS IoT Core desenvolvedor.

Nesta seção, você cria uma função do IAM de troca de tokens e um alias de AWS IoT função que aponta para a função. Se você já configurou um dispositivo principal do Greengrass, pode usar sua função de troca de tokens e seu alias de função em vez de criar novos. Em seguida, você configura o dispositivo AWS IoT para usar essa função e alias.

Para criar uma função do IAM de troca de tokens

1. Crie uma função do IAM que seu dispositivo possa usar como função de troca de tokens. Faça o seguinte:

   a. Crie um arquivo que contenha o documento de política de confiança exigido pela função de troca de tokens.

      Por exemplo, em um sistema baseado em Linux, você pode executar o seguinte comando para usar o GNU nano para criar o arquivo.

      ```
      nano device-role-trust-policy.json
      ```

      Copie o seguinte JSON no arquivo.

      ```
      {
        "Version": "2012-10-17",
        "Statement": [
          {
            "Effect": "Allow",
            "Principal": {
              "Service": "credentials.iot.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
          }
        ]
      }
      ```

   b. Crie a função de troca de tokens com o documento de política de confiança.

      - Substitua a função *GreengrassV2 pelo nome da TokenExchange função* do IAM a ser criada.

      ```
      aws iam create-role --role-name GreengrassV2TokenExchangeRole --assume-role-
      policy-document file://device-role-trust-policy.json
      ```

      A resposta será semelhante ao exemplo a seguir, se a solicitação for bem-sucedida.

      ```
      {
        "Role": {
      ```

```
    "Path": "/",
    "RoleName": "GreengrassV2TokenExchangeRole",
    "RoleId": "AROAZ2YMUHYHK5OKM77FB",
    "Arn": "arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole",
    "CreateDate": "2021-02-06T00:13:29+00:00",
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": "credentials.iot.amazonaws.com"
          },
          "Action": "sts:AssumeRole"
        }
      ]
    }
  }
```

c. Crie um arquivo que contenha o documento de política de acesso exigido pela função de troca de tokens.

   Por exemplo, em um sistema baseado em Linux, você pode executar o seguinte comando para usar o GNU nano para criar o arquivo.

   ```
   nano device-role-access-policy.json
   ```

   Copie o seguinte JSON no arquivo.

   ```
   {
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Action": [
           "logs:CreateLogGroup",
           "logs:CreateLogStream",
           "logs:PutLogEvents",
           "logs:DescribeLogStreams",
           "s3:GetBucketLocation"
         ],
         "Resource": "*"
   ```

```
    }
  ]
}
```

> **Note**
>
> Essa política de acesso não permite acesso a artefatos de componentes em buckets do S3. Para implantar componentes personalizados que definem artefatos no Amazon S3, você deve adicionar permissões à função para permitir que seu dispositivo principal recupere artefatos de componentes. Para ter mais informações, consulte Permitir acesso aos buckets do S3 para artefatos de componentes.
> Se você ainda não tem um bucket do S3 para artefatos de componentes, você pode adicionar essas permissões depois de criar um bucket.

d. Crie a política do IAM a partir do documento de política.

- Substitua *GreengrassV2 TokenExchange RoleAccess* pelo nome da política do IAM a ser criada.

```
aws iam create-policy --policy-name GreengrassV2TokenExchangeRoleAccess --
policy-document file://device-role-access-policy.json
```

A resposta será semelhante ao exemplo a seguir, se a solicitação for bem-sucedida.

```
{
  "Policy": {
    "PolicyName": "GreengrassV2TokenExchangeRoleAccess",
    "PolicyId": "ANPAZ2YMUHYHACI7C5Z66",
    "Arn": "arn:aws:iam::123456789012:policy/
GreengrassV2TokenExchangeRoleAccess",
    "Path": "/",
    "DefaultVersionId": "v1",
    "AttachmentCount": 0,
    "PermissionsBoundaryUsageCount": 0,
    "IsAttachable": true,
    "CreateDate": "2021-02-06T00:37:17+00:00",
    "UpdateDate": "2021-02-06T00:37:17+00:00"
  }
}
```

e. Anexe a política do IAM à função de troca de tokens.

- Substitua a função *GreengrassV2 pelo nome da TokenExchange função* do IAM.
- Substitua o ARN da política pelo ARN da política do IAM que você criou na etapa anterior.

```
aws iam attach-role-policy --role-name GreengrassV2TokenExchangeRole --policy-
arn arn:aws:iam::123456789012:policy/GreengrassV2TokenExchangeRoleAccess
```

O comando não terá nenhuma saída se a solicitação for bem-sucedida.

2. Crie um alias de AWS IoT função que aponte para a função de troca de tokens.

- *GreengrassCoreTokenExchangeRoleAlias*Substitua pelo nome do alias de função a ser criado.
- Substitua o ARN da função pelo ARN da função do IAM que você criou na etapa anterior.

```
aws iot create-role-alias --role-alias GreengrassCoreTokenExchangeRoleAlias --role-
arn arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole
```

A resposta será semelhante ao exemplo a seguir, se a solicitação for bem-sucedida.

```
{
  "roleAlias": "GreengrassCoreTokenExchangeRoleAlias",
  "roleAliasArn": "arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias"
}
```

> **Note**
>
> Para criar um alias de função, você precisa ter permissão para passar a função do IAM de troca de tokens para AWS IoT. Se você receber uma mensagem de erro ao tentar criar um alias de função, verifique se o AWS usuário tem essa permissão. Para obter mais informações, consulte Conceder permissões a um usuário para passar uma função para um AWS serviço no Guia do AWS Identity and Access Management usuário.

3. Crie e anexe uma AWS IoT política que permita que seu dispositivo principal do Greengrass use o alias de função para assumir a função de troca de tokens. Se você já configurou um dispositivo

principal do Greengrass, pode anexar sua AWS IoT política de alias de função em vez de criar uma nova. Faça o seguinte:

a. (Opcional) Crie um arquivo que contenha o documento AWS IoT de política exigido pelo alias da função.

   Por exemplo, em um sistema baseado em Linux, você pode executar o seguinte comando para usar o GNU nano para criar o arquivo.

   ```
   nano greengrass-v2-iot-role-alias-policy.json
   ```

   Copie o seguinte JSON no arquivo.

   - Substitua o ARN do recurso pelo ARN do seu alias de função.

   ```
   {
     "Version":"2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Action": "iot:AssumeRoleWithCertificate",
         "Resource": "arn:aws:iot:us-west-2:123456789012:rolealias/
   GreengrassCoreTokenExchangeRoleAlias"
       }
     ]
   }
   ```

b. Crie uma AWS IoT política a partir do documento de política.

   - Substitua *GreengrassCoreTokenExchangeRoleAliasPolítica* pelo nome da AWS IoT política a ser criada.

   ```
   aws iot create-policy --policy-name GreengrassCoreTokenExchangeRoleAliasPolicy
    --policy-document file://greengrass-v2-iot-role-alias-policy.json
   ```

   A resposta será semelhante ao exemplo a seguir, se a solicitação for bem-sucedida.

   ```
   {
     "policyName": "GreengrassCoreTokenExchangeRoleAliasPolicy",
   ```

```
  "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassCoreTokenExchangeRoleAliasPolicy",
  "policyDocument": "{
    \\"Version\\":\\"2012-10-17\\",
    \\"Statement\\": [
      {
        \\"Effect\\": \\"Allow\\",
        \\"Action\\": \\"iot:AssumeRoleWithCertificate\\",
        \\"Resource\\": \\"arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias\\"
      }
    ]
  }",
  "policyVersionId": "1"
}
```

c. Anexe a AWS IoT política ao certificado da AWS IoT coisa.

- Substitua *GreengrassCoreTokenExchangeRoleAliasPolítica* pelo nome da AWS IoT política de alias da função.
- Substitua o ARN de destino pelo ARN do certificado de sua coisa. AWS IoT

```
aws iot attach-policy --policy-name GreengrassCoreTokenExchangeRoleAliasPolicy
 --target arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
```

O comando não terá nenhuma saída se a solicitação for bem-sucedida.

Baixe certificados para o dispositivo

Anteriormente, você baixou o certificado do seu dispositivo para o seu computador de desenvolvimento. Nesta seção, você copia o certificado para o seu dispositivo principal para configurar o dispositivo com os certificados que ele usa para se conectar AWS IoT. Você também baixa o certificado da autoridade de certificação raiz (CA) da Amazon. Se você usa um HSM, você também importa o arquivo de certificado para o HSM nesta seção.

- Se você criou o certificado e a chave privada do item no AWS IoT serviço anteriormente, siga as etapas para baixar os certificados com a chave privada e os arquivos de certificado.

- Se você criou o certificado do item a partir de uma chave privada em um módulo de segurança de hardware (HSM) anteriormente, siga as etapas para baixar os certificados com a chave privada e o certificado em um HSM.

Baixe certificados com chave privada e arquivos de certificado

Para baixar certificados para o dispositivo

1. Copie o AWS IoT certificado do item do seu computador de desenvolvimento para o dispositivo. Se o SSH e o SCP estiverem habilitados no computador de desenvolvimento e no dispositivo, você poderá usar o `scp` comando no computador de desenvolvimento para transferir o certificado. Substitua o endereço *IP do dispositivo pelo endereço* IP do seu dispositivo.

   ```
   scp -r greengrass-v2-certs/ device-ip-address:~
   ```

2. Crie a pasta raiz do Greengrass no dispositivo. Posteriormente, você instalará o software AWS IoT Greengrass Core nessa pasta.

   Linux or Unix

   - */greengrass/v2*Substitua pela pasta a ser usada.

   ```
   sudo mkdir -p /greengrass/v2
   ```

   Windows Command Prompt

   - Substitua *C:\greengrass\v2* pela pasta a ser usada.

   ```
   mkdir C:\greengrass\v2
   ```

   PowerShell

   - Substitua *C:\greengrass\v2* pela pasta a ser usada.

   ```
   mkdir C:\greengrass\v2
   ```

3. (Somente Linux) Defina as permissões do pai da pasta raiz do Greengrass.

   - Substitua */greengrass* pelo pai da pasta raiz.

   ```
   sudo chmod 755 /greengrass
   ```

4. Copie os certificados do AWS IoT item para a pasta raiz do Greengrass.

   Linux or Unix

   - */greengrass/v2*Substitua pela pasta raiz do Greengrass.

   ```
   sudo cp -R ~/greengrass-v2-certs/* /greengrass/v2
   ```

   Windows Command Prompt

   - Substitua *C:\greengrass\v2* pela pasta a ser usada.

   ```
   robocopy %USERPROFILE%\greengrass-v2-certs C:\greengrass\v2 /E
   ```

   PowerShell

   - Substitua *C:\greengrass\v2* pela pasta a ser usada.

   ```
   cp -Path ~\greengrass-v2-certs\* -Destination C:\greengrass\v2
   ```

5. Baixe o certificado da autoridade de certificação raiz (CA) da Amazon. AWS IoT os certificados são associados ao certificado CA raiz da Amazon por padrão.

   Linux or Unix

   ```
   sudo curl -o /greengrass/v2/AmazonRootCA1.pem https://www.amazontrust.com/
   repository/AmazonRootCA1.pem
   ```

Windows Command Prompt (CMD)

```
curl -o C:\greengrass\v2\\AmazonRootCA1.pem https://www.amazontrust.com/
repository/AmazonRootCA1.pem
```

PowerShell

```
iwr -Uri https://www.amazontrust.com/repository/AmazonRootCA1.pem -OutFile C:
\greengrass\v2\\AmazonRootCA1.pem
```

Baixe certificados com a chave privada e o certificado em um HSM

> **Note**
>
> Esse recurso está disponível para a versão 2.5.3 e posterior do componente núcleo do Greengrass. AWS IoT Greengrass atualmente não oferece suporte a esse recurso nos dispositivos principais do Windows.

Para baixar certificados para o dispositivo

1. Copie o AWS IoT certificado do item do seu computador de desenvolvimento para o dispositivo. Se o SSH e o SCP estiverem habilitados no computador de desenvolvimento e no dispositivo, você poderá usar o `scp` comando no computador de desenvolvimento para transferir o certificado. Substitua o endereço *IP do dispositivo pelo endereço* IP do seu dispositivo.

   ```
   scp -r greengrass-v2-certs/ device-ip-address:~
   ```

2. Crie a pasta raiz do Greengrass no dispositivo. Posteriormente, você instalará o software AWS IoT Greengrass Core nessa pasta.

   Linux or Unix

   - */greengrass/v2*Substitua pela pasta a ser usada.

   ```
   sudo mkdir -p /greengrass/v2
   ```

Windows Command Prompt

- Substitua *C:\greengrass\v2* pela pasta a ser usada.

```
mkdir C:\greengrass\v2
```

PowerShell

- Substitua *C:\greengrass\v2* pela pasta a ser usada.

```
mkdir C:\greengrass\v2
```

3. (Somente Linux) Defina as permissões do pai da pasta raiz do Greengrass.

   - Substitua */greengrass* pelo pai da pasta raiz.

   ```
   sudo chmod 755 /greengrass
   ```

4. Importe o arquivo de certificado da coisa,`~/greengrass-v2-certs/device.pem.crt`, para o HSM. Consulte a documentação do seu HSM para saber como importar certificados para ele. Importe o certificado usando o mesmo token, ID do slot, PIN do usuário, rótulo do objeto e ID do objeto (se o HSM usar um) em que você gerou a chave privada no HSM anteriormente.

   > **Note**
   >
   > Se você gerou a chave privada anteriormente sem uma ID de objeto e o certificado tiver uma ID de objeto, defina a ID do objeto da chave privada com o mesmo valor do certificado. Consulte a documentação do seu HSM para saber como definir o ID do objeto de chave privada.

5. (Opcional) Exclua o arquivo de certificado do item, para que ele exista somente no HSM.

   ```
   rm ~/greengrass-v2-certs/device.pem.crt
   ```

6. Baixe o certificado da autoridade de certificação raiz (CA) da Amazon. AWS IoT os certificados são associados ao certificado CA raiz da Amazon por padrão.

Linux or Unix

```
sudo curl -o /greengrass/v2/AmazonRootCA1.pem https://www.amazontrust.com/
repository/AmazonRootCA1.pem
```

Windows Command Prompt (CMD)

```
curl -o C:\greengrass\v2\\AmazonRootCA1.pem https://www.amazontrust.com/
repository/AmazonRootCA1.pem
```

PowerShell

```
iwr -Uri https://www.amazontrust.com/repository/AmazonRootCA1.pem -OutFile C:
\greengrass\v2\\AmazonRootCA1.pem
```

Configurar o ambiente do dispositivo

Siga as etapas nesta seção para configurar um dispositivo Linux ou Windows para ser usado como seu dispositivo AWS IoT Greengrass principal.

Configurar um dispositivo Linux

Para configurar um dispositivo Linux para AWS IoT Greengrass V2

1. Instale o Java Runtime, que o software AWS IoT Greengrass Core exige para ser executado. Recomendamos que você use as versões de suporte de longo prazo do Amazon Corretto ou do OpenJDK. É necessária a versão 8 ou superior. Os comandos a seguir mostram como instalar o OpenJDK no seu dispositivo.

   - Para distribuições com base em Debian ou em Ubuntu:

     ```
     sudo apt install default-jdk
     ```

   - Para distribuições com base em Red Hat:

     ```
     sudo yum install java-11-openjdk-devel
     ```

   - Para Amazon Linux 2:

```
sudo amazon-linux-extras install java-openjdk11
```

- Para Amazon Linux 2023:

```
sudo dnf install java-11-amazon-corretto -y
```

Quando a instalação for concluída, execute o comando a seguir para verificar se o Java é executado no seu dispositivo Linux.

```
java -version
```

O comando imprime a versão do Java que é executada no dispositivo. Por exemplo, em uma distribuição baseada em Debian, o resultado pode ser semelhante ao exemplo a seguir.

```
openjdk version "11.0.9.1" 2020-11-04
OpenJDK Runtime Environment (build 11.0.9.1+1-post-Debian-1deb10u2)
OpenJDK 64-Bit Server VM (build 11.0.9.1+1-post-Debian-1deb10u2, mixed mode)
```

2. (Opcional) Crie o usuário e o grupo padrão do sistema que executa componentes no dispositivo. Você também pode optar por permitir que o instalador do software AWS IoT Greengrass Core crie esse usuário e grupo durante a instalação com o argumento do `--component-default-user` instalador. Para ter mais informações, consulte Argumentos de instalação.

```
sudo useradd --system --create-home ggc_user
sudo groupadd --system ggc_group
```

3. Verifique se o usuário que executa o software AWS IoT Greengrass Core (normalmente`root`) tem permissão para executar `sudo` com qualquer usuário e qualquer grupo.

   a. Execute o comando a seguir para abrir o `/etc/sudoers` arquivo.

   ```
   sudo visudo
   ```

   b. Verifique se a permissão para o usuário se parece com o exemplo a seguir.

   ```
   root    ALL=(ALL:ALL) ALL
   ```

4. (Opcional) Para executar funções Lambda em contêineres, você deve habilitar cgroups v1 e habilitar e montar os cgroups de memória e dispositivos. Se você não planeja executar funções Lambda em contêineres, pode pular esta etapa.

   Para habilitar essas opções de cgroups, inicialize o dispositivo com os seguintes parâmetros do kernel Linux.

   ```
   cgroup_enable=memory cgroup_memory=1 systemd.unified_cgroup_hierarchy=0
   ```

   Para obter informações sobre como visualizar e definir os parâmetros do kernel para seu dispositivo, consulte a documentação do sistema operacional e do carregador de inicialização. Siga as instruções para definir permanentemente os parâmetros do kernel.
5. Instale todas as outras dependências necessárias em seu dispositivo, conforme indicado na lista de requisitos emRequisitos do dispositivo.

Configurar um dispositivo Windows

> **Note**
>
> Esse recurso está disponível para a versão 2.5.0 e posterior do componente de núcleo do Greengrass.

Para configurar um dispositivo Windows para AWS IoT Greengrass V2

1. Instale o Java Runtime, que o software AWS IoT Greengrass Core exige para ser executado. Recomendamos que você use as versões de suporte de longo prazo do Amazon Corretto ou do OpenJDK. É necessária a versão 8 ou superior.
2. Verifique se o Java está disponível na variável de sistema PATH e, caso contrário, adicione-o. A LocalSystem conta executa o software AWS IoT Greengrass Core, então você deve adicionar Java à variável de sistema PATH em vez da variável de usuário PATH para seu usuário. Faça o seguinte:

   a. Pressione a tecla Windows para abrir o menu Iniciar.
   b. Digite **`environment variables`** para pesquisar as opções do sistema no menu Iniciar.
   c. Nos resultados da pesquisa do menu Iniciar, escolha Editar as variáveis de ambiente do sistema para abrir a janela Propriedades do sistema.

d. Escolha variáveis de ambiente... para abrir a janela Variáveis de ambiente.

e. Em Variáveis do sistema, selecione Caminho e, em seguida, escolha Editar. Na janela Editar variável de ambiente, você pode visualizar cada caminho em uma linha separada.

f. Verifique se o caminho para a `bin` pasta da instalação do Java está presente. O caminho pode ser semelhante ao exemplo a seguir.

```
C:\\Program Files\\Amazon Corretto\\jdk11.0.13_8\\bin
```

g. Se a `bin` pasta da instalação do Java estiver ausente do Path, escolha Novo para adicioná-la e, em seguida, escolha OK.

3. Abra o prompt de comando do Windows (`cmd.exe`) como administrador.

4. Crie o usuário padrão na LocalSystem conta no dispositivo Windows. *`Substitua a senha`* por uma senha segura.

```
net user /add ggc_user password
```

> **Tip**
>
> Dependendo da configuração do Windows, a senha do usuário pode ser definida para expirar em uma data futura. Para garantir que seus aplicativos Greengrass continuem operando, monitore quando a senha expira e atualize-a antes que ela expire. Você também pode definir a senha do usuário para nunca expirar.
>
> - Para verificar quando um usuário e sua senha expiram, execute o comando a seguir.
>
> ```
> net user ggc_user | findstr /C:expires
> ```
>
> - Para definir que a senha de um usuário nunca expire, execute o comando a seguir.
>
> ```
> wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
> ```
>
> - Se você estiver usando o Windows 10 ou posterior, onde o wmiccomando está obsoleto, execute o comando a seguir. PowerShell
>
> ```
> Get-CimInstance -Query "SELECT * from Win32_UserAccount WHERE name =
>  'ggc_user'" | Set-CimInstance -Property @{PasswordExpires="False"}
> ```

5. Baixe e instale o PsExecutilitário da Microsoft no dispositivo.
6. Use o PsExec utilitário para armazenar o nome de usuário e a senha do usuário padrão na instância do Credential Manager da LocalSystem conta. Substitua a *senha* pela senha do usuário que você definiu anteriormente.

   ```
   psexec -s cmd /c cmdkey /generic:ggc_user /user:ggc_user /pass:password
   ```

   Se for PsExec License Agreementaberto, opte por Acceptconcordar com a licença e execute o comando.

   > **Note**
   >
   > Em dispositivos Windows, a LocalSystem conta executa o núcleo Greengrass, e você deve usar o PsExec utilitário para armazenar as informações padrão do usuário na conta. LocalSystem O uso do aplicativo Credential Manager armazena essas informações na conta do Windows do usuário atualmente conectado, em vez da LocalSystem conta.

Baixe o software AWS IoT Greengrass Core

Você pode baixar a versão mais recente do software AWS IoT Greengrass Core no seguinte local:

- https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip

> **Note**
>
> Você pode baixar uma versão específica do software AWS IoT Greengrass Core no seguinte local. Substitua a *versão* pela versão a ser baixada.
>
> ```
> https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip
> ```

Para baixar o software AWS IoT Greengrass Core

1. Em seu dispositivo principal, baixe o software AWS IoT Greengrass Core para um arquivo chamado`greengrass-nucleus-latest.zip`.

Linux or Unix

```
curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-
latest.zip > greengrass-nucleus-latest.zip
```

Windows Command Prompt (CMD)

```
curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-
latest.zip > greengrass-nucleus-latest.zip
```

PowerShell

```
iwr -Uri https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-
latest.zip -OutFile greengrass-nucleus-latest.zip
```

Ao fazer download desse software, você concorda com o Contrato de licença do software do Greengrass Core.

2. (Opcional) Para verificar a assinatura do software Greengrass nucleus

> **Note**
>
> Esse recurso está disponível com o Greengrass nucleus versão 2.9.5 e posterior.

a. Use o comando a seguir para verificar a assinatura do artefato do núcleo Greengrass:

Linux or Unix

```
jarsigner -verify -certs -verbose greengrass-nucleus-latest.zip
```

Windows Command Prompt (CMD)

O nome do arquivo pode parecer diferente dependendo da versão do JDK que você instala. *jdk17.0.6_10*Substitua pela versão do JDK que você instalou.

```
"C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe" -
verify -certs -verbose greengrass-nucleus-latest.zip
```

PowerShell

O nome do arquivo pode parecer diferente dependendo da versão do JDK que você instala. *jdk17.0.6_10*Substitua pela versão do JDK que você instalou.

```
'C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe' -
verify -certs -verbose greengrass-nucleus-latest.zip
```

b. A `jarsigner` invocação produz uma saída que indica os resultados da verificação.

i. Se o arquivo zip do Greengrass nucleus estiver assinado, a saída conterá a seguinte declaração:

```
jar verified.
```

ii. Se o arquivo zip do Greengrass nucleus não estiver assinado, a saída conterá a seguinte declaração:

```
jar is unsigned.
```

c. Se você forneceu a `-certs` opção Jarsigner junto com `-verbose` as opções `-verify` e, a saída também incluirá informações detalhadas do certificado do assinante.

3. Descompacte o software AWS IoT Greengrass Core em uma pasta no seu dispositivo. *GreengrassInstaller*Substitua pela pasta que você deseja usar.

Linux or Unix

```
unzip greengrass-nucleus-latest.zip -d GreengrassInstaller && rm greengrass-
nucleus-latest.zip
```

Windows Command Prompt (CMD)

```
mkdir GreengrassInstaller && tar -xf greengrass-nucleus-latest.zip -
C GreengrassInstaller && del greengrass-nucleus-latest.zip
```

PowerShell

```
Expand-Archive -Path greengrass-nucleus-latest.zip -DestinationPath .\
\GreengrassInstaller
```

```
rm greengrass-nucleus-latest.zip
```

4. (Opcional) Execute o comando a seguir para ver a versão do software AWS IoT Greengrass Core.

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

> **Important**
>
> Se você instalar uma versão do núcleo do Greengrass anterior à v2.4.0, não remova essa pasta depois de instalar o software Core. AWS IoT Greengrass O software AWS IoT Greengrass Core usa os arquivos dessa pasta para ser executado.
> Se você baixou a versão mais recente do software, instale a versão 2.4.0 ou posterior e poderá remover essa pasta depois de instalar o software AWS IoT Greengrass Core.

Instale o software AWS IoT Greengrass Core

Execute o instalador com argumentos que especificam as seguintes ações:

- Instale a partir de um arquivo de configuração parcial que especifica o uso AWS dos recursos e certificados que você criou anteriormente. O software AWS IoT Greengrass Core usa um arquivo de configuração que especifica a configuração de cada componente do Greengrass no dispositivo. O instalador cria um arquivo de configuração completo a partir do arquivo de configuração parcial fornecido por você.
- Especifique o uso do usuário do `ggc_user` sistema para executar componentes de software no dispositivo principal. Em dispositivos Linux, esse comando também especifica o uso do grupo do `ggc_group` sistema, e o instalador cria o usuário e o grupo do sistema para você.
- Configure o software AWS IoT Greengrass Core como um serviço do sistema que é executado na inicialização. Em dispositivos Linux, isso requer o sistema init Systemd.

> **Important**
>
> Nos dispositivos principais do Windows, você deve configurar o software AWS IoT Greengrass Core como um serviço do sistema.

Para obter mais informações sobre os argumentos que você pode especificar, consulteArgumentos de instalação.

> **Note**
>
> Se você estiver executando AWS IoT Greengrass em um dispositivo com memória limitada, poderá controlar a quantidade de memória que o software AWS IoT Greengrass Core usa. Para controlar a alocação de memória, você pode definir as opções de tamanho da pilha da JVM no parâmetro de `jvmOptions` configuração em seu componente de núcleo. Para ter mais informações, consulte Controle a alocação de memória com opções de JVM.

- Se você criou o certificado e a chave privada do item no AWS IoT serviço anteriormente, siga as etapas para instalar o software AWS IoT Greengrass Core com a chave privada e os arquivos de certificado.
- Se você criou o certificado do item a partir de uma chave privada em um módulo de segurança de hardware (HSM) anteriormente, siga as etapas para instalar o software AWS IoT Greengrass Core com a chave privada e o certificado em um HSM.

Instale o software AWS IoT Greengrass Core com arquivos de chave privada e certificado

Para instalar o software AWS IoT Greengrass Core

1. Verifique a versão do software AWS IoT Greengrass Core.

   - *`GreengrassInstaller`*Substitua pelo caminho para a pasta que contém o software.

   ```
   java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
   ```

2. Use um editor de texto para criar um arquivo de configuração chamado `config.yaml` para fornecer ao instalador.

   Por exemplo, em um sistema baseado em Linux, você pode executar o seguinte comando para usar o GNU nano para criar o arquivo.

   ```
   nano GreengrassInstaller/config.yaml
   ```

Copie o seguinte conteúdo YAML para o arquivo. Esse arquivo de configuração parcial especifica os parâmetros do sistema e os parâmetros do núcleo do Greengrass.

```
---
system:
  certificateFilePath: "/greengrass/v2/device.pem.crt"
  privateKeyPath: "/greengrass/v2/private.pem.key"
  rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
  rootpath: "/greengrass/v2"
  thingName: "MyGreengrassCore"
services:
  aws.greengrass.Nucleus:
    componentType: "NUCLEUS"
    version: "2.12.6"
    configuration:
      awsRegion: "us-west-2"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
      iotCredEndpoint: "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
```

Então, faça o seguinte:

- Substitua cada instância do */greengrass/v2* pela pasta raiz do Greengrass.
- Substitua *MyGreengrassCore* pelo nome da AWS IoT coisa.
- Substitua *2.12.6* pela versão do software AWS IoT Greengrass Core.
- Substitua *us-west-2* pelo local onde você Região da AWS criou os recursos.
- *GreengrassCoreTokenExchangeRoleAlias*Substitua pelo nome do alias da função de troca de tokens.
- Substitua o `iotDataEndpoint` por seu endpoint de AWS IoT dados.
- Substitua o `iotCredEndpoint` pelo endpoint de suas AWS IoT credenciais.

> **Note**
>
> Nesse arquivo de configuração, você pode personalizar outras opções de configuração do núcleo, como as portas e o proxy de rede a serem usados, conforme mostrado no

exemplo a seguir. Para obter mais informações, consulte Configuração do [núcleo do Greengrass](#).

```
---
system:
  certificateFilePath: "/greengrass/v2/device.pem.crt"
  privateKeyPath: "/greengrass/v2/private.pem.key"
  rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
  rootpath: "/greengrass/v2"
  thingName: "MyGreengrassCore"
services:
  aws.greengrass.Nucleus:
    componentType: "NUCLEUS"
    version: "2.12.6"
    configuration:
      awsRegion: "us-west-2"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      iotCredEndpoint: "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
      mqtt:
        port: 443
      greengrassDataPlanePort: 443
      networkProxy:
        noProxyAddresses: "http://192.168.0.1,www.example.com"
        proxy:
          url: "https://my-proxy-server:1100"
          username: "Mary_Major"
          password: "pass@word1357"
```

3. Execute o instalador e especifique `--init-config` para fornecer o arquivo de configuração.

   - Substitua */greengrass/v2* ou *C:\greengrass\v2* pela pasta raiz do Greengrass.
   - Substitua cada instância do *GreengrassInstaller*pela pasta em que você descompactou o instalador.

   Linux or Unix

   ```
   sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE \
     -jar ./GreengrassInstaller/lib/Greengrass.jar \
     --init-config ./GreengrassInstaller/config.yaml \
   ```

```
  --component-default-user ggc_user:ggc_group \
  --setup-system-service true
```

Windows Command Prompt (CMD)

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" ^
  -jar ./GreengrassInstaller/lib/Greengrass.jar ^
  --init-config ./GreengrassInstaller/config.yaml ^
  --component-default-user ggc_user ^
  --setup-system-service true
```

PowerShell

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" `
  -jar ./GreengrassInstaller/lib/Greengrass.jar `
  --init-config ./GreengrassInstaller/config.yaml `
  --component-default-user ggc_user `
  --setup-system-service true
```

Important

Nos dispositivos principais do Windows, você deve especificar `--setup-system-service true` a configuração do software AWS IoT Greengrass Core como um serviço do sistema.

Se você especificar`--setup-system-service true`, o instalador `Successfully set up Nucleus as a system service` imprimirá se configurou e executou o software como um serviço do sistema. Caso contrário, o instalador não emitirá nenhuma mensagem se instalar o software com êxito.

Note

Você não pode usar o `deploy-dev-tools` argumento para implantar ferramentas de desenvolvimento local ao executar o instalador sem o `--provision true` argumento. Para obter informações sobre a implantação da CLI do Greengrass diretamente em seu dispositivo, consulte. Interface de linha de comando do Greengrass

4. Verifique a instalação visualizando os arquivos na pasta raiz.

   Linux or Unix

   ```
   ls /greengrass/v2
   ```

   Windows Command Prompt (CMD)

   ```
   dir C:\greengrass\v2
   ```

   PowerShell

   ```
   ls C:\greengrass\v2
   ```

   Se a instalação for bem-sucedida, a pasta raiz conterá várias pastas`config`, como`packages`, e. `logs`

Instale o software AWS IoT Greengrass Core com a chave privada e o certificado em um HSM

> **Note**
>
> Esse recurso está disponível para a versão 2.5.3 e posterior do componente núcleo do Greengrass. AWS IoT Greengrass atualmente não oferece suporte a esse recurso nos dispositivos principais do Windows.

Para instalar o software AWS IoT Greengrass Core

1. Verifique a versão do software AWS IoT Greengrass Core.

   - *`GreengrassInstaller`*Substitua pelo caminho para a pasta que contém o software.

   ```
   java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
   ```

2. Para permitir que o software AWS IoT Greengrass Core use a chave privada e o certificado no HSM, instale o componente do provedor PKCS #11 ao instalar o software AWS IoT Greengrass Core. O componente do provedor PKCS #11 é um plug-in que você pode configurar durante

a instalação. Você pode baixar a versão mais recente do componente provedor PKCS #11 no seguinte local:

- https://d2s8p88vqu9w66.cloudfront.net/releases/Pkcs11Provider/aws.greengrass.crypto.Pkcs11Provider-latest.jar

Baixe o plug-in do provedor PKCS #11 para um arquivo chamado`aws.greengrass.crypto.Pkcs11Provider.jar`. *`GreengrassInstaller`*Substitua pela pasta que você deseja usar.

```
curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/Pkcs11Provider/
aws.greengrass.crypto.Pkcs11Provider-latest.jar > GreengrassInstaller/
aws.greengrass.crypto.Pkcs11Provider.jar
```

Ao fazer download desse software, você concorda com o Contrato de licença do software do Greengrass Core.

3. Use um editor de texto para criar um arquivo de configuração chamado `config.yaml` para fornecer ao instalador.

   Por exemplo, em um sistema baseado em Linux, você pode executar o seguinte comando para usar o GNU nano para criar o arquivo.

   ```
   nano GreengrassInstaller/config.yaml
   ```

   Copie o seguinte conteúdo YAML para o arquivo. Esse arquivo de configuração parcial especifica os parâmetros do sistema, os parâmetros do núcleo do Greengrass e os parâmetros do provedor PKCS #11.

   ```
   ---
   system:
     certificateFilePath: "pkcs11:object=iotdevicekey;type=cert"
     privateKeyPath: "pkcs11:object=iotdevicekey;type=private"
     rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
     rootpath: "/greengrass/v2"
     thingName: "MyGreengrassCore"
   services:
     aws.greengrass.Nucleus:
       componentType: "NUCLEUS"
       version: "2.12.6"
   ```

```
    configuration:
      awsRegion: "us-west-2"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
      iotCredEndpoint: "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
  aws.greengrass.crypto.Pkcs11Provider:
    configuration:
      name: "softhsm_pkcs11"
      library: "/usr/local/Cellar/softhsm/2.6.1/lib/softhsm/libsofthsm2.so"
      slot: 1
      userPin: "1234"
```

Então, faça o seguinte:

- Substitua cada instância de *iotdevicekey* nos URIs do PKCS #11 pelo rótulo do objeto em que você criou a chave privada e importou o certificado.
- Substitua cada instância do */greengrass/v2* pela pasta raiz do Greengrass.
- Substitua *MyGreengrassCore* pelo nome da AWS IoT coisa.
- Substitua *2.12.6* pela versão do software AWS IoT Greengrass Core.
- Substitua *us-west-2* pelo local onde você Região da AWS criou os recursos.
- *GreengrassCoreTokenExchangeRoleAlias*Substitua pelo nome do alias da função de troca de tokens.
- Substitua o `iotDataEndpoint` por seu endpoint de AWS IoT dados.
- Substitua o `iotCredEndpoint` pelo endpoint de suas AWS IoT credenciais.
- Substitua os parâmetros de configuração do `aws.greengrass.crypto.Pkcs11Provider` componente pelos valores da configuração do HSM no dispositivo principal.

> **Note**
>
> Nesse arquivo de configuração, você pode personalizar outras opções de configuração do núcleo, como as portas e o proxy de rede a serem usados, conforme mostrado no exemplo a seguir. Para obter mais informações, consulte Configuração do núcleo do Greengrass.
>
> ```
> ---
> system:
> ```

```
    certificateFilePath: "pkcs11:object=iotdevicekey;type=cert"
    privateKeyPath: "pkcs11:object=iotdevicekey;type=private"
    rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
    rootpath: "/greengrass/v2"
    thingName: "MyGreengrassCore"
services:
  aws.greengrass.Nucleus:
    componentType: "NUCLEUS"
    version: "2.12.6"
    configuration:
      awsRegion: "us-west-2"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
      iotCredEndpoint: "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
      mqtt:
        port: 443
      greengrassDataPlanePort: 443
      networkProxy:
        noProxyAddresses: "http://192.168.0.1,www.example.com"
        proxy:
          url: "https://my-proxy-server:1100"
          username: "Mary_Major"
          password: "pass@word1357"
  aws.greengrass.crypto.Pkcs11Provider:
    configuration:
      name: "softhsm_pkcs11"
      library: "/usr/local/Cellar/softhsm/2.6.1/lib/softhsm/libsofthsm2.so"
      slot: 1
      userPin: "1234"
```

4. Execute o instalador e especifique `--init-config` para fornecer o arquivo de configuração.

   - */greengrass/v2*Substitua pela pasta raiz do Greengrass.
   - Substitua cada instância do *GreengrassInstaller*pela pasta em que você descompactou o instalador.

```
sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE \
  -jar ./GreengrassInstaller/lib/Greengrass.jar \
  --trusted-plugin ./GreengrassInstaller/aws.greengrass.crypto.Pkcs11Provider.jar \
  --init-config ./GreengrassInstaller/config.yaml \
```

```
  --component-default-user ggc_user:ggc_group \
  --setup-system-service true
```

> **Important**
>
> Nos dispositivos principais do Windows, você deve especificar `--setup-system-service true` a configuração do software AWS IoT Greengrass Core como um serviço do sistema.

Se você especificar`--setup-system-service true`, o instalador `Successfully set up Nucleus as a system service` imprimirá se configurou e executou o software como um serviço do sistema. Caso contrário, o instalador não emitirá nenhuma mensagem se instalar o software com êxito.

> **Note**
>
> Você não pode usar o `deploy-dev-tools` argumento para implantar ferramentas de desenvolvimento local ao executar o instalador sem o `--provision true` argumento. Para obter informações sobre a implantação da CLI do Greengrass diretamente em seu dispositivo, consulte. Interface de linha de comando do Greengrass

5. Verifique a instalação visualizando os arquivos na pasta raiz.

   Linux or Unix

   ```
   ls /greengrass/v2
   ```

   Windows Command Prompt (CMD)

   ```
   dir C:\greengrass\v2
   ```

   PowerShell

   ```
   ls C:\greengrass\v2
   ```

Se a instalação for bem-sucedida, a pasta raiz conterá várias pastas`config`, como`packages`, e. `logs`

Se você instalou o software AWS IoT Greengrass Core como um serviço do sistema, o instalador executa o software para você. Caso contrário, você deverá executar o software manualmente. Para ter mais informações, consulte Execute o software AWS IoT Greengrass Core.

Para obter mais informações sobre como configurar e usar o software AWS IoT Greengrass, consulte o seguinte:

- Configurar o software AWS IoT Greengrass principal
- Desenvolva AWS IoT Greengrass componentes
- Implemente AWS IoT Greengrass componentes em dispositivos
- Interface de linha de comando do Greengrass

Instale o software AWS IoT Greengrass principal com provisionamento de AWS IoT frota

Esse recurso está disponível para a versão 2.4.0 e posterior do componente de núcleo do Greengrass.

Com o provisionamento de AWS IoT frotas, você pode configurar AWS IoT para gerar e entregar com segurança certificados de dispositivos X.509 e chaves privadas aos seus dispositivos quando eles se conectarem pela primeira vez. AWS IoT AWS IoT fornece certificados de cliente assinados pela autoridade de certificação (CA) raiz da Amazon. Você também pode configurar AWS IoT para especificar grupos de coisas, tipos de coisas e permissões para os dispositivos principais do Greengrass que você provisiona com o provisionamento de frotas. Você define um modelo de aprovisionamento para definir como AWS IoT provisiona cada dispositivo. O modelo de provisionamento especifica o item, a política e os recursos de certificado a serem criados para um dispositivo durante o provisionamento. Para obter mais informações, consulte Modelos de provisionamento no Guia do AWS IoT Core desenvolvedor.

AWS IoT Greengrass fornece um plug-in de provisionamento de AWS IoT frota que você pode usar para instalar o software AWS IoT Greengrass Core usando AWS recursos criados pelo provisionamento de AWS IoT frota. O plug-in de provisionamento de frota usa o provisionamento

por reclamação. Os dispositivos usam um certificado de declaração de provisionamento e uma chave privada para obter um certificado de dispositivo X.509 e uma chave privada exclusivos que podem ser usados para operações regulares. Você pode incorporar o certificado de solicitação e a chave privada em cada dispositivo durante a fabricação, para que seus clientes possam ativar os dispositivos posteriormente, quando cada dispositivo estiver on-line. Você pode usar o mesmo certificado de solicitação e chave privada para vários dispositivos. Para obter mais informações, consulte Provisionamento por solicitação no Guia do AWS IoT Core desenvolvedor.

> **Note**
>
> Atualmente, o plug-in de provisionamento de frota não oferece suporte ao armazenamento de arquivos de chave privada e certificado em um módulo de segurança de hardware (HSM). Para usar um HSM, instale o software AWS IoT Greengrass Core com provisionamento manual.

Para instalar o software AWS IoT Greengrass Core com provisionamento de AWS IoT frota, você deve configurar os recursos Conta da AWS que são AWS IoT usados para provisionar os dispositivos principais do Greengrass. Esses recursos incluem um modelo de provisionamento, certificados de solicitação e uma função do IAM de troca de tokens. Depois de criar esses recursos, você pode reutilizá-los para provisionar vários dispositivos principais em uma frota. Para ter mais informações, consulte Configure o provisionamento AWS IoT da frota para os principais dispositivos do Greengrass.

> **Important**
>
> Antes de baixar o software AWS IoT Greengrass Core, verifique se seu dispositivo principal atende aos requisitos para instalar e executar o software AWS IoT Greengrass Core v2.0.

Tópicos

- Pré-requisitos
- Recupere endpoints AWS IoT
- Baixe certificados para o dispositivo
- Configurar o ambiente do dispositivo
- Baixe o software AWS IoT Greengrass Core

- Baixe o plug-in de aprovisionamento de AWS IoT frotas
- Instale o software AWS IoT Greengrass Core
- Configure o provisionamento AWS IoT da frota para os principais dispositivos do Greengrass
- Configurar o plug-in de aprovisionamento de AWS IoT frotas
- AWS IoT registro de alterações do plug-in de aprovisionamento de frotas

Pré-requisitos

Para instalar o software AWS IoT Greengrass Core com provisionamento de AWS IoT frota, você deve primeiro configurar o provisionamento de AWS IoT frota para os dispositivos principais do Greengrass. Depois de concluir essas etapas uma vez, você pode usar o provisionamento de frota para instalar o software AWS IoT Greengrass Core em qualquer número de dispositivos.

Recupere endpoints AWS IoT

Obtenha os AWS IoT endpoints para você Conta da AWS e salve-os para usar mais tarde. Seu dispositivo usa esses endpoints para se conectar a. AWS IoT Faça o seguinte:

1. Obtenha o endpoint de AWS IoT dados para você Conta da AWS.

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

   A resposta será semelhante ao exemplo a seguir, se a solicitação for bem-sucedida.

   ```
   {
     "endpointAddress": "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
   }
   ```

2. Obtenha o endpoint AWS IoT de credenciais para seu. Conta da AWS

   ```
   aws iot describe-endpoint --endpoint-type iot:CredentialProvider
   ```

   A resposta será semelhante ao exemplo a seguir, se a solicitação for bem-sucedida.

   ```
   {
     "endpointAddress": "device-credentials-prefix.credentials.iot.us-
   west-2.amazonaws.com"
   }
   ```

Baixe certificados para o dispositivo

O dispositivo usa um certificado de solicitação e uma chave privada para autenticar sua solicitação para provisionar AWS recursos e adquirir um certificado de dispositivo X.509. Você pode incorporar o certificado de solicitação e a chave privada no dispositivo durante a fabricação ou copiar o certificado e a chave no dispositivo durante a instalação. Nesta seção, você copia o certificado de solicitação e a chave privada para o dispositivo. Você também baixa o certificado da Amazon Root Certificate Authority (CA) para o dispositivo.

> **Important**
>
> O provisionamento de chaves privadas de solicitação deve ser protegido em todos os momentos, inclusive nos dispositivos principais do Greengrass. Recomendamos que você use CloudWatch métricas e registros da Amazon para monitorar indícios de uso indevido, como o uso não autorizado do certificado de solicitação para provisionar dispositivos. Se você detectar uso indevido, desative o certificado de solicitação de aprovisionamento para que ele não possa ser usado para provisionamento de dispositivos. Para obter mais informações, consulte Monitorar AWS IoT no Guia do Desenvolvedor AWS IoT Core . Para ajudá-lo a gerenciar melhor o número de dispositivos e quais dispositivos se registram no seu Conta da AWS, você pode especificar um gancho de pré-provisionamento ao criar um modelo de aprovisionamento de frota. Um gancho de pré-provisionamento é uma AWS Lambda função que valida os parâmetros do modelo que os dispositivos fornecem durante o registro. Por exemplo, você pode criar um gancho de pré-provisionamento que compara a ID do dispositivo a um banco de dados para verificar se o dispositivo tem permissão para provisionar. Para obter mais informações, consulte Ganchos de pré-provisionamento no Guia do desenvolvedor.AWS IoT Core

Para baixar certificados de solicitação para o dispositivo

1. Copie o certificado de solicitação e a chave privada para o dispositivo. Se o SSH e o SCP estiverem habilitados no computador de desenvolvimento e no dispositivo, você poderá usar o `scp` comando no computador de desenvolvimento para transferir o certificado de solicitação e a chave privada. O comando de exemplo a seguir transfere esses arquivos de uma pasta chamada `claim-certs` em seu computador de desenvolvimento para o dispositivo. Substitua o endereço *`IP do dispositivo pelo endereço`* IP do seu dispositivo.

```
scp -r claim-certs/ device-ip-address:~
```

2. Crie a pasta raiz do Greengrass no dispositivo. Posteriormente, você instalará o software AWS IoT Greengrass Core nessa pasta.

   Linux or Unix

   - */greengrass/v2*Substitua pela pasta a ser usada.

   ```
   sudo mkdir -p /greengrass/v2
   ```

   Windows Command Prompt

   - Substitua *C:\greengrass\v2* pela pasta a ser usada.

   ```
   mkdir C:\greengrass\v2
   ```

   PowerShell

   - Substitua *C:\greengrass\v2* pela pasta a ser usada.

   ```
   mkdir C:\greengrass\v2
   ```

3. (Somente Linux) Defina as permissões do pai da pasta raiz do Greengrass.

   - Substitua */greengrass* pelo pai da pasta raiz.

   ```
   sudo chmod 755 /greengrass
   ```

4. Mova os certificados de solicitação para a pasta raiz do Greengrass.

   - Substitua */greengrass/v2* ou *C:\greengrass\v2* pela pasta raiz do Greengrass.

Linux or Unix

```
sudo mv ~/claim-certs /greengrass/v2
```

Windows Command Prompt (CMD)

```
move %USERPROFILE%\claim-certs C:\greengrass\v2
```

PowerShell

```
mv -Path ~\claim-certs -Destination C:\greengrass\v2
```

5. Baixe o certificado da autoridade de certificação raiz (CA) da Amazon. AWS IoT os certificados são associados ao certificado CA raiz da Amazon por padrão.

   Linux or Unix

   ```
   sudo curl -o /greengrass/v2/AmazonRootCA1.pem https://www.amazontrust.com/
   repository/AmazonRootCA1.pem
   ```

   Windows Command Prompt (CMD)

   ```
   curl -o C:\greengrass\v2\\AmazonRootCA1.pem https://www.amazontrust.com/
   repository/AmazonRootCA1.pem
   ```

   PowerShell

   ```
   iwr -Uri https://www.amazontrust.com/repository/AmazonRootCA1.pem -OutFile C:
   \greengrass\v2\\AmazonRootCA1.pem
   ```

Configurar o ambiente do dispositivo

Siga as etapas nesta seção para configurar um dispositivo Linux ou Windows para ser usado como seu dispositivo AWS IoT Greengrass principal.

Configurar um dispositivo Linux

Para configurar um dispositivo Linux para AWS IoT Greengrass V2

1. Instale o Java Runtime, que o software AWS IoT Greengrass Core exige para ser executado. Recomendamos que você use as versões de suporte de longo prazo do Amazon Corretto ou do OpenJDK. É necessária a versão 8 ou superior. Os comandos a seguir mostram como instalar o OpenJDK no seu dispositivo.

   - Para distribuições com base em Debian ou em Ubuntu:

     ```
     sudo apt install default-jdk
     ```

   - Para distribuições com base em Red Hat:

     ```
     sudo yum install java-11-openjdk-devel
     ```

   - Para Amazon Linux 2:

     ```
     sudo amazon-linux-extras install java-openjdk11
     ```

   - Para Amazon Linux 2023:

     ```
     sudo dnf install java-11-amazon-corretto -y
     ```

   Quando a instalação for concluída, execute o comando a seguir para verificar se o Java é executado no seu dispositivo Linux.

   ```
   java -version
   ```

   O comando imprime a versão do Java que é executada no dispositivo. Por exemplo, em uma distribuição baseada em Debian, o resultado pode ser semelhante ao exemplo a seguir.

   ```
   openjdk version "11.0.9.1" 2020-11-04
   OpenJDK Runtime Environment (build 11.0.9.1+1-post-Debian-1deb10u2)
   OpenJDK 64-Bit Server VM (build 11.0.9.1+1-post-Debian-1deb10u2, mixed mode)
   ```

2. (Opcional) Crie o usuário e o grupo padrão do sistema que executa componentes no dispositivo. Você também pode optar por permitir que o instalador do software AWS IoT Greengrass Core

crie esse usuário e grupo durante a instalação com o argumento do `--component-default-user` instalador. Para ter mais informações, consulte Argumentos de instalação.

```
sudo useradd --system --create-home ggc_user
sudo groupadd --system ggc_group
```

3. Verifique se o usuário que executa o software AWS IoT Greengrass Core (normalmente`root`) tem permissão para executar `sudo` com qualquer usuário e qualquer grupo.

   a. Execute o comando a seguir para abrir o `/etc/sudoers` arquivo.

   ```
   sudo visudo
   ```

   b. Verifique se a permissão para o usuário se parece com o exemplo a seguir.

   ```
   root    ALL=(ALL:ALL) ALL
   ```

4. (Opcional) Para executar funções Lambda em contêineres, você deve habilitar cgroups v1 e habilitar e montar os cgroups de memória e dispositivos. Se você não planeja executar funções Lambda em contêineres, pode pular esta etapa.

   Para habilitar essas opções de cgroups, inicialize o dispositivo com os seguintes parâmetros do kernel Linux.

   ```
   cgroup_enable=memory cgroup_memory=1 systemd.unified_cgroup_hierarchy=0
   ```

   Para obter informações sobre como visualizar e definir os parâmetros do kernel para seu dispositivo, consulte a documentação do sistema operacional e do carregador de inicialização. Siga as instruções para definir permanentemente os parâmetros do kernel.

5. Instale todas as outras dependências necessárias em seu dispositivo, conforme indicado na lista de requisitos emRequisitos do dispositivo.

Configurar um dispositivo Windows

> **Note**
>
> Esse recurso está disponível para a versão 2.5.0 e posterior do componente de núcleo do Greengrass.

Para configurar um dispositivo Windows para AWS IoT Greengrass V2

1. Instale o Java Runtime, que o software AWS IoT Greengrass Core exige para ser executado. Recomendamos que você use as versões de suporte de longo prazo do Amazon Corretto ou do OpenJDK. É necessária a versão 8 ou superior.
2. Verifique se o Java está disponível na variável de sistema PATH e, caso contrário, adicione-a. A LocalSystem conta executa o software AWS IoT Greengrass Core, então você deve adicionar Java à variável de sistema PATH em vez da variável de usuário PATH para seu usuário. Faça o seguinte:

   a. Pressione a tecla Windows para abrir o menu Iniciar.

   b. Digite **`environment variables`** para pesquisar as opções do sistema no menu Iniciar.

   c. Nos resultados da pesquisa do menu Iniciar, escolha Editar as variáveis de ambiente do sistema para abrir a janela Propriedades do sistema.

   d. Escolha variáveis de ambiente... para abrir a janela Variáveis de ambiente.

   e. Em Variáveis do sistema, selecione Caminho e, em seguida, escolha Editar. Na janela Editar variável de ambiente, você pode visualizar cada caminho em uma linha separada.

   f. Verifique se o caminho para a `bin` pasta da instalação do Java está presente. O caminho pode ser semelhante ao exemplo a seguir.

   ```
   C:\\Program Files\\Amazon Corretto\\jdk11.0.13_8\\bin
   ```

   g. Se a `bin` pasta da instalação do Java estiver ausente do Path, escolha Novo para adicioná-la e, em seguida, escolha OK.
3. Abra o prompt de comando do Windows (`cmd.exe`) como administrador.
4. Crie o usuário padrão na LocalSystem conta no dispositivo Windows. *`Substitua a senha`* por uma senha segura.

   ```
   net user /add ggc_user password
   ```

   > **Tip**
   >
   > Dependendo da configuração do Windows, a senha do usuário pode ser definida para expirar em uma data futura. Para garantir que seus aplicativos Greengrass continuem operando, monitore quando a senha expira e atualize-a antes que ela expire. Você também pode definir a senha do usuário para nunca expirar.

- Para verificar quando um usuário e sua senha expiram, execute o comando a seguir.

  ```
  net user ggc_user | findstr /C:expires
  ```

- Para definir que a senha de um usuário nunca expire, execute o comando a seguir.

  ```
  wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
  ```

- Se você estiver usando o Windows 10 ou posterior, onde o `wmic`comando está obsoleto, execute o comando a seguir. PowerShell

  ```
  Get-CimInstance -Query "SELECT * from Win32_UserAccount WHERE name =
   'ggc_user'" | Set-CimInstance -Property @{PasswordExpires="False"}
  ```

5. Baixe e instale o PsExecutilitário da Microsoft no dispositivo.
6. Use o PsExec utilitário para armazenar o nome de usuário e a senha do usuário padrão na instância do Credential Manager da LocalSystem conta. Substitua a *senha* pela senha do usuário que você definiu anteriormente.

   ```
   psexec -s cmd /c cmdkey /generic:ggc_user /user:ggc_user /pass:password
   ```

   Se for PsExec License Agreementaberto, opte por Acceptconcordar com a licença e execute o comando.

   > **Note**
   >
   > Em dispositivos Windows, a LocalSystem conta executa o núcleo Greengrass, e você deve usar o PsExec utilitário para armazenar as informações padrão do usuário na conta. LocalSystem O uso do aplicativo Credential Manager armazena essas informações na conta do Windows do usuário atualmente conectado, em vez da LocalSystem conta.

Baixe o software AWS IoT Greengrass Core

Você pode baixar a versão mais recente do software AWS IoT Greengrass Core no seguinte local:

- https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip

> **Note**
>
> Você pode baixar uma versão específica do software AWS IoT Greengrass Core no seguinte local. Substitua a *versão* pela versão a ser baixada.
>
> ```
> https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip
> ```

Para baixar o software AWS IoT Greengrass Core

1. Em seu dispositivo principal, baixe o software AWS IoT Greengrass Core para um arquivo chamado`greengrass-nucleus-latest.zip`.

   Linux or Unix

   ```
   curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-
   latest.zip > greengrass-nucleus-latest.zip
   ```

   Windows Command Prompt (CMD)

   ```
   curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-
   latest.zip > greengrass-nucleus-latest.zip
   ```

   PowerShell

   ```
   iwr -Uri https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-
   latest.zip -OutFile greengrass-nucleus-latest.zip
   ```

   Ao fazer download desse software, você concorda com o Contrato de licença do software do Greengrass Core.

2. (Opcional) Para verificar a assinatura do software Greengrass nucleus

   > **Note**
   >
   > Esse recurso está disponível com o Greengrass nucleus versão 2.9.5 e posterior.

a. Use o comando a seguir para verificar a assinatura do artefato do núcleo Greengrass:

Linux or Unix

```
jarsigner -verify -certs -verbose greengrass-nucleus-latest.zip
```

Windows Command Prompt (CMD)

O nome do arquivo pode parecer diferente dependendo da versão do JDK que você instala. *jdk17.0.6_10*Substitua pela versão do JDK que você instalou.

```
"C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe" -
verify -certs -verbose greengrass-nucleus-latest.zip
```

PowerShell

O nome do arquivo pode parecer diferente dependendo da versão do JDK que você instala. *jdk17.0.6_10*Substitua pela versão do JDK que você instalou.

```
'C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe' -
verify -certs -verbose greengrass-nucleus-latest.zip
```

b. A `jarsigner` invocação produz uma saída que indica os resultados da verificação.

i. Se o arquivo zip do Greengrass nucleus estiver assinado, a saída conterá a seguinte declaração:

```
jar verified.
```

ii. Se o arquivo zip do Greengrass nucleus não estiver assinado, a saída conterá a seguinte declaração:

```
jar is unsigned.
```

c. Se você forneceu a `-certs` opção Jarsigner junto com `-verbose` as opções `-verify` e, a saída também incluirá informações detalhadas do certificado do assinante.

3. Descompacte o software AWS IoT Greengrass Core em uma pasta no seu dispositivo. *GreengrassInstaller*Substitua pela pasta que você deseja usar.

Linux or Unix

```
unzip greengrass-nucleus-latest.zip -d GreengrassInstaller && rm greengrass-
nucleus-latest.zip
```

Windows Command Prompt (CMD)

```
mkdir GreengrassInstaller && tar -xf greengrass-nucleus-latest.zip -
C GreengrassInstaller && del greengrass-nucleus-latest.zip
```

PowerShell

```
Expand-Archive -Path greengrass-nucleus-latest.zip -DestinationPath .\
\GreengrassInstaller
rm greengrass-nucleus-latest.zip
```

4. (Opcional) Execute o comando a seguir para ver a versão do software AWS IoT Greengrass Core.

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

> **Important**
>
> Se você instalar uma versão do núcleo do Greengrass anterior à v2.4.0, não remova essa pasta depois de instalar o software Core. AWS IoT Greengrass O software AWS IoT Greengrass Core usa os arquivos dessa pasta para ser executado.
> Se você baixou a versão mais recente do software, instale a versão 2.4.0 ou posterior e poderá remover essa pasta depois de instalar o software AWS IoT Greengrass Core.

Baixe o plug-in de aprovisionamento de AWS IoT frotas

Você pode baixar a versão mais recente do plug-in de aprovisionamento de AWS IoT frotas no seguinte local:

- https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass - FleetProvisioning ByClaim / fleetprovisioningbyclaim-latest.jar

> **Note**
>
> Você pode baixar uma versão específica do plug-in de aprovisionamento de AWS IoT frotas no seguinte local. Substitua a *versão* pela versão a ser baixada. Para obter mais informações sobre cada versão do plug-in de aprovisionamento de frotas, consulte. AWS IoT registro de alterações do plug-in de aprovisionamento de frotas
>
> ```
> https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass-
> FleetProvisioningByClaim/fleetprovisioningbyclaim-version.jar
> ```

O plug-in de provisionamento de frota é de código aberto. Para ver seu código-fonte, consulte o plug-in de provisionamento de AWS IoT frota ativado. GitHub

Para baixar o plug-in de aprovisionamento de AWS IoT frotas

- Em seu dispositivo, baixe o plug-in de aprovisionamento de AWS IoT frota em um arquivo chamado. `aws.greengrass.FleetProvisioningByClaim.jar` *GreengrassInstaller*Substitua pela pasta que você deseja usar.

  Linux or Unix

  ```
  curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass-
  FleetProvisioningByClaim/fleetprovisioningbyclaim-latest.jar
   > GreengrassInstaller/aws.greengrass.FleetProvisioningByClaim.jar
  ```

  Windows Command Prompt (CMD)

  ```
  curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass-
  FleetProvisioningByClaim/fleetprovisioningbyclaim-latest.jar
   > GreengrassInstaller/aws.greengrass.FleetProvisioningByClaim.jar
  ```

  PowerShell

  ```
  iwr -Uri https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass-
  FleetProvisioningByClaim/fleetprovisioningbyclaim-latest.jar -
  OutFile GreengrassInstaller/aws.greengrass.FleetProvisioningByClaim.jar
  ```

Ao fazer download desse software, você concorda com o Contrato de licença do software do Greengrass Core.

Instale o software AWS IoT Greengrass Core

Execute o instalador com argumentos que especificam as seguintes ações:

- Instale a partir de um arquivo de configuração parcial que especifica o uso do plug-in de provisionamento de frota para provisionar recursos. AWS O software AWS IoT Greengrass Core usa um arquivo de configuração que especifica a configuração de cada componente do Greengrass no dispositivo. O instalador cria um arquivo de configuração completo a partir do arquivo de configuração parcial que você fornece e dos AWS recursos que o plug-in de aprovisionamento de frota cria.
- Especifique o uso do usuário do `ggc_user` sistema para executar componentes de software no dispositivo principal. Em dispositivos Linux, esse comando também especifica o uso do grupo do `ggc_group` sistema, e o instalador cria o usuário e o grupo do sistema para você.
- Configure o software AWS IoT Greengrass Core como um serviço do sistema que é executado na inicialização. Em dispositivos Linux, isso requer o sistema init Systemd.

> **Important**
>
> Nos dispositivos principais do Windows, você deve configurar o software AWS IoT Greengrass Core como um serviço do sistema.

Para obter mais informações sobre os argumentos que você pode especificar, consulteArgumentos de instalação.

> **Note**
>
> Se você estiver executando AWS IoT Greengrass em um dispositivo com memória limitada, poderá controlar a quantidade de memória que o software AWS IoT Greengrass Core usa. Para controlar a alocação de memória, você pode definir as opções de tamanho da pilha da JVM no parâmetro de `jvmOptions` configuração em seu componente de núcleo. Para ter mais informações, consulte Controle a alocação de memória com opções de JVM.

Para instalar o software AWS IoT Greengrass Core

1. Verifique a versão do software AWS IoT Greengrass Core.

   - *GreengrassInstaller*Substitua pelo caminho para a pasta que contém o software.

   ```
   java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
   ```

2. Use um editor de texto para criar um arquivo de configuração chamado `config.yaml` para fornecer ao instalador.

   Por exemplo, em um sistema baseado em Linux, você pode executar o seguinte comando para usar o GNU nano para criar o arquivo.

   ```
   nano GreengrassInstaller/config.yaml
   ```

   Copie o seguinte conteúdo YAML para o arquivo. Esse arquivo de configuração parcial especifica os parâmetros para o plug-in de provisionamento de frotas. Para obter mais informações sobre as opções que você pode especificar, consulte[Configurar o plug-in de aprovisionamento de AWS IoT frotas](#).

   Linux or Unix

   ```
   ---
   services:
     aws.greengrass.Nucleus:
       version: "2.12.6"
     aws.greengrass.FleetProvisioningByClaim:
       configuration:
         rootPath: "/greengrass/v2"
         awsRegion: "us-west-2"
         iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
         iotCredentialEndpoint: "device-credentials-prefix.credentials.iot.us-
   west-2.amazonaws.com"
         iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
         provisioningTemplate: "GreengrassFleetProvisioningTemplate"
         claimCertificatePath: "/greengrass/v2/claim-certs/claim.pem.crt"
         claimCertificatePrivateKeyPath: "/greengrass/v2/claim-certs/
   claim.private.pem.key"
         rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
         templateParameters:
   ```

```
        ThingName: "MyGreengrassCore"
        ThingGroupName: "MyGreengrassCoreGroup"
```

Windows

```
---
services:
  aws.greengrass.Nucleus:
    version: "2.12.6"
  aws.greengrass.FleetProvisioningByClaim:
    configuration:
      rootPath: "C:\\greengrass\\v2"
      awsRegion: "us-west-2"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
      iotCredentialEndpoint: "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      provisioningTemplate: "GreengrassFleetProvisioningTemplate"
      claimCertificatePath: "C:\\greengrass\\v2\\claim-certs\\claim.pem.crt"
      claimCertificatePrivateKeyPath: "C:\\greengrass\\v2\\claim-certs\
\claim.private.pem.key"
      rootCaPath: "C:\\greengrass\\v2\\AmazonRootCA1.pem"
      templateParameters:
        ThingName: "MyGreengrassCore"
        ThingGroupName: "MyGreengrassCoreGroup"
```

Então, faça o seguinte:

- Substitua *2.12.6* pela versão do software AWS IoT Greengrass Core.
- Substitua cada instância de */greengrass/v2* ou *C:\greengrass\v2* pela pasta raiz do Greengrass.

> **Note**
>
> Em dispositivos Windows, você deve especificar os separadores de caminho como barras invertidas duplas (\\), como. `C:\\greengrass\\v2`

- Substitua *us-west-2* pela região em que você criou AWS o modelo de provisionamento e outros recursos.
- Substitua o `iotDataEndpoint` por seu endpoint de AWS IoT dados.

- Substitua o `iotCredentialEndpoint` pelo endpoint de suas AWS IoT credenciais.
- *`GreengrassCoreTokenExchangeRoleAlias`*Substitua pelo nome do alias da função de troca de tokens.
- *`GreengrassFleetProvisioningTemplate`*Substitua pelo nome do modelo de aprovisionamento da frota.
- `claimCertificatePath`Substitua o pelo caminho para o certificado de solicitação no dispositivo.
- `claimCertificatePrivateKeyPath`Substitua o pelo caminho para a chave privada do certificado de solicitação no dispositivo.
- Substitua os parâmetros do modelo (`templateParameters`) pelos valores a serem usados para provisionar o dispositivo. Este exemplo se refere ao modelo de exemplo que define `ThingName` `ThingGroupName` parâmetros.

> **Note**
>
> Nesse arquivo de configuração, você pode personalizar outras opções de configuração, como as portas e o proxy de rede a serem usados, conforme mostrado no exemplo a seguir. Para obter mais informações, consulte Configuração do núcleo do Greengrass.
>
> Linux or Unix

```
---
services:
  aws.greengrass.Nucleus:
    version: "2.12.6"
    configuration:
      mqtt:
        port: 443
      greengrassDataPlanePort: 443
      networkProxy:
        noProxyAddresses: "http://192.168.0.1,www.example.com"
        proxy:
          url: "http://my-proxy-server:1100"
          username: "Mary_Major"
          password: "pass@word1357"
  aws.greengrass.FleetProvisioningByClaim:
    configuration:
      rootPath: "/greengrass/v2"
```

```
      awsRegion: "us-west-2"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-
west-2.amazonaws.com"
      iotCredentialEndpoint: "device-credentials-
prefix.credentials.iot.us-west-2.amazonaws.com"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      provisioningTemplate: "GreengrassFleetProvisioningTemplate"
      claimCertificatePath: "/greengrass/v2/claim-certs/claim.pem.crt"
      claimCertificatePrivateKeyPath: "/greengrass/v2/claim-certs/
claim.private.pem.key"
      rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
      templateParameters:
        ThingName: "MyGreengrassCore"
        ThingGroupName: "MyGreengrassCoreGroup"
      mqttPort: 443
      proxyUrl: "http://my-proxy-server:1100"
      proxyUserName: "Mary_Major"
      proxyPassword: "pass@word1357"
```

Windows

```
---
services:
  aws.greengrass.Nucleus:
    version: "2.12.6"
    configuration:
      mqtt:
        port: 443
      greengrassDataPlanePort: 443
      networkProxy:
        noProxyAddresses: "http://192.168.0.1,www.example.com"
        proxy:
          url: "http://my-proxy-server:1100"
          username: "Mary_Major"
          password: "pass@word1357"
  aws.greengrass.FleetProvisioningByClaim:
    configuration:
      rootPath: "C:\\greengrass\\v2"
      awsRegion: "us-west-2"
      iotDataEndpoint: "device-data-prefix-ats.iot.us-
west-2.amazonaws.com"
      iotCredentialEndpoint: "device-credentials-
prefix.credentials.iot.us-west-2.amazonaws.com"
```

```
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      provisioningTemplate: "GreengrassFleetProvisioningTemplate"
      claimCertificatePath: "C:\\greengrass\\v2\\claim-certs\
\claim.pem.crt"
      claimCertificatePrivateKeyPath: "C:\\greengrass\\v2\\claim-certs\
\claim.private.pem.key"
      rootCaPath: "C:\\greengrass\\v2\\AmazonRootCA1.pem"
      templateParameters:
        ThingName: "MyGreengrassCore"
        ThingGroupName: "MyGreengrassCoreGroup"
      mqttPort: 443
      proxyUrl: "http://my-proxy-server:1100"
      proxyUserName: "Mary_Major"
      proxyPassword: "pass@word1357"
```

Para usar um proxy HTTPS, você deve usar a versão 1.1.0 ou posterior do plug-in de provisionamento de frota. Além disso, você deve especificar o `system` item `rootCaPath` abaixo, conforme mostrado no exemplo a seguir.

Linux or Unix

```
---
system:
  rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
services:
  ...
```

Windows

```
---
system:
  rootCaPath: "C:\\greengrass\\v2\\AmazonRootCA1.pem"
services:
  ...
```

3. Execute o instalador. Especifique `--trusted-plugin` para fornecer o plug-in de provisionamento de frota e especifique `--init-config` para fornecer o arquivo de configuração.

   - */greengrass/v2*Substitua pela pasta raiz do Greengrass.

- Substitua cada instância do *GreengrassInstaller* pela pasta em que você descompactou o instalador.

Linux or Unix

```
sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE \
  -jar ./GreengrassInstaller/lib/Greengrass.jar \
  --trusted-plugin ./GreengrassInstaller/
aws.greengrass.FleetProvisioningByClaim.jar \
  --init-config ./GreengrassInstaller/config.yaml \
  --component-default-user ggc_user:ggc_group \
  --setup-system-service true
```

Windows Command Prompt (CMD)

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" ^
  -jar ./GreengrassInstaller/lib/Greengrass.jar ^
  --trusted-plugin ./GreengrassInstaller/
aws.greengrass.FleetProvisioningByClaim.jar ^
  --init-config ./GreengrassInstaller/config.yaml ^
  --component-default-user ggc_user ^
  --setup-system-service true
```

PowerShell

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" `
  -jar ./GreengrassInstaller/lib/Greengrass.jar `
  --trusted-plugin ./GreengrassInstaller/
aws.greengrass.FleetProvisioningByClaim.jar `
  --init-config ./GreengrassInstaller/config.yaml `
  --component-default-user ggc_user `
  --setup-system-service true
```

Important

Nos dispositivos principais do Windows, você deve especificar `--setup-system-service true` a configuração do software AWS IoT Greengrass Core como um serviço do sistema.

Se você especificar`--setup-system-service true`, o instalador `Successfully set up Nucleus as a system service` imprimirá se configurou e executou o software como um serviço do sistema. Caso contrário, o instalador não emitirá nenhuma mensagem se instalar o software com êxito.

> **Note**
>
> Você não pode usar o `deploy-dev-tools` argumento para implantar ferramentas de desenvolvimento local ao executar o instalador sem o `--provision true` argumento. Para obter informações sobre a implantação da CLI do Greengrass diretamente em seu dispositivo, consulte. Interface de linha de comando do Greengrass

4. Verifique a instalação visualizando os arquivos na pasta raiz.

   Linux or Unix

   ```
   ls /greengrass/v2
   ```

   Windows Command Prompt (CMD)

   ```
   dir C:\greengrass\v2
   ```

   PowerShell

   ```
   ls C:\greengrass\v2
   ```

   Se a instalação for bem-sucedida, a pasta raiz conterá várias pastas`config`, como`packages`, e. `logs`

Se você instalou o software AWS IoT Greengrass Core como um serviço do sistema, o instalador executa o software para você. Caso contrário, você deverá executar o software manualmente. Para ter mais informações, consulte Execute o software AWS IoT Greengrass Core.

Para obter mais informações sobre como configurar e usar o software AWS IoT Greengrass, consulte o seguinte:

- Configurar o software AWS IoT Greengrass principal
- Desenvolva AWS IoT Greengrass componentes
- Implemente AWS IoT Greengrass componentes em dispositivos
- Interface de linha de comando do Greengrass

Configure o provisionamento AWS IoT da frota para os principais dispositivos do Greengrass

Para instalar o software AWS IoT Greengrass Core com provisionamento de frota, você deve primeiro configurar os seguintes recursos em seu. Conta da AWS Esses recursos permitem que os dispositivos se registrem AWS IoT e operem como dispositivos principais do Greengrass. Siga as etapas desta seção uma vez para criar e configurar esses recursos no seuConta da AWS.

- Uma função do IAM de troca de tokens, que os dispositivos principais usam para autorizar chamadas para AWS serviços.
- Um alias de AWS IoT função que aponta para a função de troca de tokens.
- (Opcional) Uma AWS IoT política que os dispositivos principais usam para autorizar chamadas para os AWS IoT Greengrass serviços AWS IoT e. Essa AWS IoT política deve permitir a `iot:AssumeRoleWithCertificate` permissão para o alias de AWS IoT função que aponta para a função de troca de tokens.

  Você pode usar uma única AWS IoT política para todos os dispositivos principais da sua frota ou pode configurar seu modelo de aprovisionamento da frota para criar uma AWS IoT política para cada dispositivo principal.
- Um modelo de aprovisionamento de AWS IoT frota. Esse modelo deve especificar o seguinte:
  - Qualquer AWS IoT coisa, recurso. Você pode especificar uma lista de grupos de coisas existentes para implantar componentes em cada dispositivo quando ele estiver on-line.
  - Um recurso AWS IoT político. Esse recurso pode definir uma das seguintes propriedades:
    - O nome de uma AWS IoT política existente. Se você escolher essa opção, os dispositivos principais criados a partir desse modelo usarão a mesma AWS IoT política e você poderá gerenciar suas permissões como uma frota.
    - Um documento AWS IoT de política. Se você escolher essa opção, cada dispositivo principal criado a partir desse modelo usará uma AWS IoT política exclusiva e você poderá gerenciar permissões para cada dispositivo principal individual.

  - Um recurso de AWS IoT certificado. Esse recurso de certificado deve usar o `AWS::IoT::Certificate::Id` parâmetro para anexar o certificado ao dispositivo principal. Para obter mais informações, consulte J ust-in-time provisioning no Guia do AWS IoTdesenvolvedor.
- Um certificado de solicitação AWS IoT de aprovisionamento e uma chave privada para o modelo de aprovisionamento da frota. Você pode incorporar esse certificado e a chave privada nos dispositivos durante a fabricação, para que os dispositivos possam se registrar e se provisionar quando estiverem on-line.

  > **Important**
  >
  > O provisionamento de chaves privadas de solicitação deve ser protegido em todos os momentos, inclusive nos dispositivos principais do Greengrass. Recomendamos que você use CloudWatch métricas e registros da Amazon para monitorar indícios de uso indevido, como o uso não autorizado do certificado de solicitação para provisionar dispositivos. Se você detectar uso indevido, desative o certificado de solicitação de aprovisionamento para que ele não possa ser usado para provisionamento de dispositivos. Para obter mais informações, consulte Monitorar AWS IoT no Guia do Desenvolvedor AWS IoT Core. Para ajudá-lo a gerenciar melhor o número de dispositivos e quais dispositivos se registram no seuConta da AWS, você pode especificar um gancho de pré-provisionamento ao criar um modelo de aprovisionamento de frota. Um gancho de pré-provisionamento é uma AWS Lambda função que valida os parâmetros do modelo que os dispositivos fornecem durante o registro. Por exemplo, você pode criar um gancho de pré-provisionamento que compara a ID do dispositivo a um banco de dados para verificar se o dispositivo tem permissão para provisionar. Para obter mais informações, consulte Ganchos de pré-provisionamento no Guia do desenvolvedor. AWS IoT Core
- Uma AWS IoT política que você anexa ao certificado de solicitação de aprovisionamento para permitir que os dispositivos se registrem e usem o modelo de aprovisionamento da frota.

Tópicos

- Crie uma função de troca de tokens
- Criar uma política do AWS IoT
- Crie um modelo de aprovisionamento de frota
- Crie um certificado de solicitação de aprovisionamento e uma chave privada

Crie uma função de troca de tokens

Os dispositivos principais do Greengrass usam uma função de serviço do IAM, chamada função de troca de tokens, para autorizar chamadas para serviços. AWS O dispositivo usa o provedor de AWS IoT credenciais para obter AWS credenciais temporárias para essa função, o que permite que o dispositivo interajaAWS IoT, envie registros para o Amazon CloudWatch Logs e baixe artefatos de componentes personalizados do Amazon S3. Para ter mais informações, consulte Autorize os dispositivos principais a interagir com os serviços AWS.

Você usa um alias de AWS IoT função para configurar a função de troca de tokens para os dispositivos principais do Greengrass. Os aliases de função permitem que você altere a função de troca de tokens de um dispositivo, mas mantenha a mesma configuração do dispositivo. Para obter mais informações, consulte Autorização de chamadas diretas para AWS serviços no Guia do AWS IoT Core desenvolvedor.

Nesta seção, você cria uma função do IAM de troca de tokens e um alias de AWS IoT função que aponta para a função. Se você já configurou um dispositivo principal do Greengrass, pode usar sua função de troca de tokens e seu alias de função em vez de criar novos.

Para criar uma função do IAM de troca de tokens

1. Crie uma função do IAM que seu dispositivo possa usar como função de troca de tokens. Faça o seguinte:

   a. Crie um arquivo que contenha o documento de política de confiança exigido pela função de troca de tokens.

      Por exemplo, em um sistema baseado em Linux, você pode executar o seguinte comando para usar o GNU nano para criar o arquivo.

      ```
      nano device-role-trust-policy.json
      ```

      Copie o seguinte JSON no arquivo.

      ```
      {
        "Version": "2012-10-17",
        "Statement": [
          {
            "Effect": "Allow",
            "Principal": {
      ```

```
        "Service": "credentials.iot.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

b. Crie a função de troca de tokens com o documento de política de confiança.

- Substitua *GreengrassV2 TokenExchangeRole* pelo nome da função do IAM a ser criada.

```
aws iam create-role --role-name GreengrassV2TokenExchangeRole --assume-role-
policy-document file://device-role-trust-policy.json
```

A resposta será semelhante ao exemplo a seguir, se a solicitação for bem-sucedida.

```
{
  "Role": {
    "Path": "/",
    "RoleName": "GreengrassV2TokenExchangeRole",
    "RoleId": "AROAZ2YMUHYHK5OKM77FB",
    "Arn": "arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole",
    "CreateDate": "2021-02-06T00:13:29+00:00",
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": "credentials.iot.amazonaws.com"
          },
          "Action": "sts:AssumeRole"
        }
      ]
    }
  }
```

c. Crie um arquivo que contenha o documento de política de acesso exigido pela função de troca de tokens.

Por exemplo, em um sistema baseado em Linux, você pode executar o seguinte comando para usar o GNU nano para criar o arquivo.

```
nano device-role-access-policy.json
```

Copie o seguinte JSON no arquivo.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",
        "s3:GetBucketLocation"
      ],
      "Resource": "*"
    }
  ]
}
```

> **Note**
>
> Essa política de acesso não permite acesso a artefatos de componentes em buckets do S3. Para implantar componentes personalizados que definem artefatos no Amazon S3, você deve adicionar permissões à função para permitir que seu dispositivo principal recupere artefatos de componentes. Para ter mais informações, consulte Permitir acesso aos buckets do S3 para artefatos de componentes.
> Se você ainda não tem um bucket do S3 para artefatos de componentes, você pode adicionar essas permissões depois de criar um bucket.

d. Crie a política do IAM a partir do documento de política.

- Substitua *GreengrassV2TokenExchangeRoleAccess* pelo nome da política do IAM a ser criada.

```
aws iam create-policy --policy-name GreengrassV2TokenExchangeRoleAccess --
policy-document file://device-role-access-policy.json
```

A resposta será semelhante ao exemplo a seguir, se a solicitação for bem-sucedida.

```
{
  "Policy": {
    "PolicyName": "GreengrassV2TokenExchangeRoleAccess",
    "PolicyId": "ANPAZ2YMUHYHACI7C5Z66",
    "Arn": "arn:aws:iam::123456789012:policy/
GreengrassV2TokenExchangeRoleAccess",
    "Path": "/",
    "DefaultVersionId": "v1",
    "AttachmentCount": 0,
    "PermissionsBoundaryUsageCount": 0,
    "IsAttachable": true,
    "CreateDate": "2021-02-06T00:37:17+00:00",
    "UpdateDate": "2021-02-06T00:37:17+00:00"
  }
}
```

e. Anexe a política do IAM à função de troca de tokens.

- Substitua *GreengrassV2 TokenExchangeRole* pelo nome da função do IAM.
- Substitua o ARN da política pelo ARN da política do IAM que você criou na etapa anterior.

```
aws iam attach-role-policy --role-name GreengrassV2TokenExchangeRole --policy-
arn arn:aws:iam::123456789012:policy/GreengrassV2TokenExchangeRoleAccess
```

O comando não terá nenhuma saída se a solicitação for bem-sucedida.

2. Crie um alias de AWS IoT função que aponte para a função de troca de tokens.

- *GreengrassCoreTokenExchangeRoleAlias*Substitua pelo nome do alias de função a ser criado.
- Substitua o ARN da função pelo ARN da função do IAM que você criou na etapa anterior.

```
aws iot create-role-alias --role-alias GreengrassCoreTokenExchangeRoleAlias --role-
arn arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole
```

A resposta será semelhante ao exemplo a seguir, se a solicitação for bem-sucedida.

```
{
  "roleAlias": "GreengrassCoreTokenExchangeRoleAlias",
  "roleAliasArn": "arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias"
}
```

> **Note**
>
> Para criar um alias de função, você precisa ter permissão para passar a função do IAM de troca de tokens paraAWS IoT. Se você receber uma mensagem de erro ao tentar criar um alias de função, verifique se o AWS usuário tem essa permissão. Para obter mais informações, consulte Conceder permissões a um usuário para passar uma função para um AWS serviço no Guia do AWS Identity and Access Management usuário.

Criar uma política do AWS IoT

Depois de registrar um dispositivo como uma AWS IoT coisa, esse dispositivo pode usar um certificado digital para se autenticarAWS. Esse certificado inclui uma ou mais AWS IoT políticas que definem as permissões que um dispositivo pode usar com o certificado. Essas políticas permitem que o dispositivo se comunique com AWS IoT AWS IoT Greengrass e.

Com o provisionamento AWS IoT da frota, os dispositivos se conectam AWS IoT para criar e baixar um certificado de dispositivo. No modelo de aprovisionamento de frota que você cria na próxima seção, você pode especificar se AWS IoT anexa a mesma AWS IoT política aos certificados de todos os dispositivos ou cria uma nova política para cada dispositivo.

Nesta seção, você cria uma AWS IoT política AWS IoT anexada aos certificados de todos os dispositivos. Com essa abordagem, você pode gerenciar as permissões para todos os dispositivos como uma frota. Se preferir criar uma nova AWS IoT política para cada dispositivo, você pode pular esta seção e consultar a política nela ao definir seu modelo de frota.

Para criar uma política do AWS IoT

- Crie uma AWS IoT política que defina as AWS IoT permissões para sua frota de dispositivos principais do Greengrass. A política a seguir permite acesso a todos os tópicos do MQTT e operações do Greengrass, para que seu dispositivo funcione com aplicativos personalizados e futuras alterações que exijam novas operações do Greengrass. Essa política também permite a `iot:AssumeRoleWithCertificate` permissão, que permite que seus dispositivos usem a função de troca de tokens que você criou na seção anterior. Você pode restringir essa política com base no seu caso de uso. Para ter mais informações, consulte AWS IoTPolítica mínima para dispositivos AWS IoT Greengrass V2 principais.

  Faça o seguinte:

  a. Crie um arquivo que contenha o documento AWS IoT de política exigido pelos dispositivos principais do Greengrass.

     Por exemplo, em um sistema baseado em Linux, você pode executar o seguinte comando para usar o GNU nano para criar o arquivo.

     ```
     nano greengrass-v2-iot-policy.json
     ```

     Copie o seguinte JSON no arquivo.

     - Substitua o `iot:AssumeRoleWithCertificate` recurso pelo ARN do alias de AWS IoT função que você criou na seção anterior.

     ```
     {
       "Version": "2012-10-17",
       "Statement": [
         {
           "Effect": "Allow",
           "Action": [
             "iot:Publish",
             "iot:Subscribe",
             "iot:Receive",
             "iot:Connect",
             "greengrass:*"
           ],
           "Resource": [
             "*"
     ```

```
      ]
    },
    {
      "Effect": "Allow",
      "Action": "iot:AssumeRoleWithCertificate",
      "Resource": "arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias"
    }
  ]
}
```

b. Crie uma AWS IoT política a partir do documento de política.

- Substitua *GreengrassV2IoT* pelo nome da ThingPolicy política a ser criada.

```
aws iot create-policy --policy-name GreengrassV2IoTThingPolicy --policy-
document file://greengrass-v2-iot-policy.json
```

A resposta será semelhante ao exemplo a seguir, se a solicitação for bem-sucedida.

```
{
  "policyName": "GreengrassV2IoTThingPolicy",
  "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassV2IoTThingPolicy",
  "policyDocument": "{
    \"Version\": \"2012-10-17\",
    \"Statement\": [
      {
        \"Effect\": \"Allow\",
        \"Action\": [
          \"iot:Publish\",
          \"iot:Subscribe\",
          \"iot:Receive\",
          \"iot:Connect\",
          \"greengrass:*\"
        ],
        \"Resource\": [
          \"*\"
        ]
      },
      {
        \"Effect\": \"Allow\",
```

```
        \"Action\": \"iot:AssumeRoleWithCertificate\",
        \"Resource\": \"arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias\"
      }
    ]
  }",
  "policyVersionId": "1"
}
```

Crie um modelo de aprovisionamento de frota

AWS IoTos modelos de aprovisionamento de frotas definem como provisionar AWS IoT itens, políticas e certificados. Para provisionar os dispositivos principais do Greengrass com o plug-in de provisionamento de frota, você deve criar um modelo que especifique o seguinte:

- Qualquer AWS IoT coisa, recurso. Você pode especificar uma lista de grupos de coisas existentes para implantar componentes em cada dispositivo quando ele estiver on-line.
- Um recurso AWS IoT político. Esse recurso pode definir uma das seguintes propriedades:
  - O nome de uma AWS IoT política existente. Se você escolher essa opção, os dispositivos principais criados a partir desse modelo usarão a mesma AWS IoT política e você poderá gerenciar suas permissões como uma frota.
  - Um documento AWS IoT de política. Se você escolher essa opção, cada dispositivo principal criado a partir desse modelo usará uma AWS IoT política exclusiva e você poderá gerenciar permissões para cada dispositivo principal individual.
- Um recurso de AWS IoT certificado. Esse recurso de certificado deve usar o `AWS::IoT::Certificate::Id` parâmetro para anexar o certificado ao dispositivo principal. Para obter mais informações, consulte J ust-in-time provisioning no Guia do AWS IoTdesenvolvedor.

No modelo, você pode especificar a adição da AWS IoT coisa a uma lista de grupos de coisas existentes. Quando o dispositivo principal se conecta AWS IoT Greengrass pela primeira vez, ele recebe implantações do Greengrass para cada grupo do qual é membro. Você pode usar grupos de coisas para implantar o software mais recente em cada dispositivo assim que ele estiver online. Para ter mais informações, consulte Implemente AWS IoT Greengrass componentes em dispositivos.

O AWS IoT serviço exige permissões para criar e atualizar AWS IoT recursos em seu dispositivo Conta da AWS ao provisionar dispositivos. Para dar acesso ao AWS IoT serviço, você cria

uma função do IAM e a fornece ao criar o modelo. AWS IoTfornece uma política gerenciada, AWSIoTThingsRegistration, que permite acesso a todas as permissões que AWS IoT podem ser usadas ao provisionar dispositivos. Você pode usar essa política gerenciada ou criar uma política personalizada que defina as permissões na política gerenciada para seu caso de uso.

Nesta seção, você cria uma função do IAM que permite AWS IoT provisionar recursos para dispositivos e cria um modelo de aprovisionamento de frota que usa essa função do IAM.

Para criar um modelo de aprovisionamento de frota

1. Crie uma função do IAM que AWS IoT possa assumir o provisionamento de recursos em seuConta da AWS. Faça o seguinte:

   a. Crie um arquivo que contenha o documento de política de confiança que AWS IoT permite assumir a função.

      Por exemplo, em um sistema baseado em Linux, você pode executar o seguinte comando para usar o GNU nano para criar o arquivo.

      ```
      nano aws-iot-trust-policy.json
      ```

      Copie o seguinte JSON no arquivo.

      ```
      {
        "Version": "2012-10-17",
        "Statement": [
          {
            "Effect": "Allow",
            "Principal": {
              "Service": "iot.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
          }
        ]
      }
      ```

   b. Crie uma função do IAM com o documento de política de confiança.

      - *GreengrassFleetProvisioningRole*Substitua pelo nome da função do IAM a ser criada.

```
aws iam create-role --role-name GreengrassFleetProvisioningRole --assume-role-
policy-document file://aws-iot-trust-policy.json
```

A resposta será semelhante ao exemplo a seguir, se a solicitação for bem-sucedida.

```
{
  "Role": {
    "Path": "/",
    "RoleName": "GreengrassFleetProvisioningRole",
    "RoleId": "AROAZ2YMUHYHK5OKM77FB",
    "Arn": "arn:aws:iam::123456789012:role/GreengrassFleetProvisioningRole",
    "CreateDate": "2021-07-26T00:15:12+00:00",
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": "iot.amazonaws.com"
          },
          "Action": "sts:AssumeRole"
        }
      ]
    }
  }
}
```

c. Revise a AWSIoTThingsRegistrationpolítica, que permite acesso a todas as permissões que AWS IoT podem ser usadas ao provisionar dispositivos. Você pode usar essa política gerenciada ou criar uma política personalizada que defina permissões com escopo reduzido para seu caso de uso. Se você optar por criar uma política personalizada, faça isso agora.

d. Anexe a política do IAM à função de aprovisionamento da frota.

- Substitua *GreengrassFleetProvisioningRole* pelo nome da função do IAM.
- Se você criou uma política personalizada na etapa anterior, substitua o ARN da política pelo ARN da política do IAM a ser usada.

```
aws iam attach-role-policy --role-name GreengrassFleetProvisioningRole --
policy-arn arn:aws:iam::aws:policy/service-role/AWSIoTThingsRegistration
```

O comando não terá nenhuma saída se a solicitação for bem-sucedida.

2. (Opcional) Crie um gancho de pré-provisionamento, que é uma AWS Lambda função que valida os parâmetros do modelo que os dispositivos fornecem durante o registro. Você pode usar um gancho de pré-provisionamento para obter mais controle sobre quais e quantos dispositivos estão integrados no seu. Conta da AWS Para obter mais informações, consulte Ganchos de pré-provisionamento no Guia do desenvolvedor. AWS IoT Core
3. Crie um modelo de aprovisionamento de frota. Faça o seguinte:

   a. Crie um arquivo para conter o documento modelo de aprovisionamento.

      Por exemplo, em um sistema baseado em Linux, você pode executar o seguinte comando para usar o GNU nano para criar o arquivo.

      ```
      nano greengrass-fleet-provisioning-template.json
      ```

      Escreva o documento modelo de aprovisionamento. Você pode começar com o seguinte exemplo de modelo de aprovisionamento, que especifica a criação de AWS IoT algo com as seguintes propriedades:

      - O nome da coisa é o valor que você especifica no parâmetro do `ThingName` modelo.
      - A coisa é um membro do grupo de coisas que você especifica no parâmetro do `ThingGroupName` modelo. O grupo de coisas deve existir em seuConta da AWS.
      - O certificado da coisa tem a AWS IoT política nomeada `GreengrassV2IoTThingPolicy` anexada a ele.

      Para obter mais informações, consulte Modelos de provisionamento no Guia do AWS IoT Coredesenvolvedor.

      ```
      {
        "Parameters": {
          "ThingName": {
            "Type": "String"
          },
      ```

```
    "ThingGroupName": {
      "Type": "String"
    },
    "AWS::IoT::Certificate::Id": {
      "Type": "String"
    }
  },
  "Resources": {
    "MyThing": {
      "OverrideSettings": {
        "AttributePayload": "REPLACE",
        "ThingGroups": "REPLACE",
        "ThingTypeName": "REPLACE"
      },
      "Properties": {
        "AttributePayload": {},
        "ThingGroups": [
          {
            "Ref": "ThingGroupName"
          }
        ],
        "ThingName": {
          "Ref": "ThingName"
        }
      },
      "Type": "AWS::IoT::Thing"
    },
    "MyPolicy": {
      "Properties": {
        "PolicyName": "GreengrassV2IoTThingPolicy"
      },
      "Type": "AWS::IoT::Policy"
    },
    "MyCertificate": {
      "Properties": {
        "CertificateId": {
          "Ref": "AWS::IoT::Certificate::Id"
        },
        "Status": "Active"
      },
      "Type": "AWS::IoT::Certificate"
    }
  }
```

```
}
```

> **Note**
>
> *MyThing*, *MyPolicy*, e *MyCertificate*são nomes arbitrários que identificam cada especificação de recurso no modelo de aprovisionamento da frota. AWS IoTnão usa esses nomes nos recursos que ele cria a partir do modelo. Você pode usar esses nomes ou substituí-los por valores que ajudam a identificar cada recurso no modelo.

b. Crie o modelo de aprovisionamento da frota a partir do documento do modelo de aprovisionamento.

- *GreengrassFleetProvisioningTemplate*Substitua pelo nome do modelo a ser criado.
- Substitua a descrição do modelo por uma descrição para o seu modelo.
- Substitua o ARN da função de aprovisionamento pelo ARN da função que você criou anteriormente.

Linux or Unix

```
aws iot create-provisioning-template \
  --template-name GreengrassFleetProvisioningTemplate \
  --description "A provisioning template for Greengrass core devices." \
  --provisioning-role-arn "arn:aws:iam::123456789012:role/
GreengrassFleetProvisioningRole" \
  --template-body file://greengrass-fleet-provisioning-template.json \
  --enabled
```

Windows Command Prompt (CMD)

```
aws iot create-provisioning-template ^
  --template-name GreengrassFleetProvisioningTemplate ^
  --description "A provisioning template for Greengrass core devices." ^
  --provisioning-role-arn "arn:aws:iam::123456789012:role/
GreengrassFleetProvisioningRole" ^
  --template-body file://greengrass-fleet-provisioning-template.json ^
  --enabled
```

PowerShell

```
aws iot create-provisioning-template `
  --template-name GreengrassFleetProvisioningTemplate `
  --description "A provisioning template for Greengrass core devices." `
  --provisioning-role-arn "arn:aws:iam::123456789012:role/
GreengrassFleetProvisioningRole" `
  --template-body file://greengrass-fleet-provisioning-template.json `
  --enabled
```

> **Note**
>
> Se você criou um gancho de pré-provisionamento, especifique o ARN da função Lambda do gancho de pré-provisionamento com o argumento. `--pre-provisioning-hook`
>
> ```
> --pre-provisioning-hook targetArn=arn:aws:lambda:us-
> west-2:123456789012:function:GreengrassPreProvisioningHook
> ```

A resposta será semelhante ao exemplo a seguir, se a solicitação for bem-sucedida.

```
{
    "templateArn": "arn:aws:iot:us-west-2:123456789012:provisioningtemplate/
GreengrassFleetProvisioningTemplate",
    "templateName": "GreengrassFleetProvisioningTemplate",
    "defaultVersionId": 1
}
```

Crie um certificado de solicitação de aprovisionamento e uma chave privada

Os certificados de declaração são certificados X.509 que permitem que os dispositivos se registrem como AWS IoT itens e recuperem um certificado de dispositivo X.509 exclusivo para uso em operações regulares. Depois de criar um certificado de solicitação, você anexa uma AWS IoT política que permite que os dispositivos o usem para criar certificados de dispositivo exclusivos e provisionar com um modelo de aprovisionamento de frota. Os dispositivos com o certificado de solicitação podem provisionar usando somente o modelo de provisionamento permitido na AWS IoT política.

Nesta seção, você cria o certificado de solicitação e o configura para dispositivos usarem com o modelo de aprovisionamento de frota que você criou na seção anterior.

> **Important**
>
> O provisionamento de chaves privadas de solicitação deve ser protegido em todos os momentos, inclusive nos dispositivos principais do Greengrass. Recomendamos que você use CloudWatch métricas e registros da Amazon para monitorar indícios de uso indevido, como o uso não autorizado do certificado de solicitação para provisionar dispositivos. Se você detectar uso indevido, desative o certificado de solicitação de aprovisionamento para que ele não possa ser usado para provisionamento de dispositivos. Para obter mais informações, consulte Monitorar AWS IoT no Guia do Desenvolvedor AWS IoT Core. Para ajudá-lo a gerenciar melhor o número de dispositivos e quais dispositivos se registram no seuConta da AWS, você pode especificar um gancho de pré-provisionamento ao criar um modelo de aprovisionamento de frota. Um gancho de pré-provisionamento é uma AWS Lambda função que valida os parâmetros do modelo que os dispositivos fornecem durante o registro. Por exemplo, você pode criar um gancho de pré-provisionamento que compara a ID do dispositivo a um banco de dados para verificar se o dispositivo tem permissão para provisionar. Para obter mais informações, consulte Ganchos de pré-provisionamento no Guia do desenvolvedor. AWS IoT Core

Para criar um certificado de solicitação de aprovisionamento e uma chave privada

1. Crie uma pasta na qual você baixa o certificado de solicitação e a chave privada.

   ```
   mkdir claim-certs
   ```

2. Crie e salve um certificado e uma chave privada para usar no provisionamento. AWS IoTfornece certificados de cliente assinados pela autoridade de certificação (CA) raiz da Amazon.

   Linux or Unix

   ```
   aws iot create-keys-and-certificate \
     --certificate-pem-outfile "claim-certs/claim.pem.crt" \
     --public-key-outfile "claim-certs/claim.public.pem.key" \
     --private-key-outfile "claim-certs/claim.private.pem.key" \
     --set-as-active
   ```

Windows Command Prompt (CMD)

```
aws iot create-keys-and-certificate ^
  --certificate-pem-outfile "claim-certs/claim.pem.crt" ^
  --public-key-outfile "claim-certs/claim.public.pem.key" ^
  --private-key-outfile "claim-certs/claim.private.pem.key" ^
  --set-as-active
```

PowerShell

```
aws iot create-keys-and-certificate `
  --certificate-pem-outfile "claim-certs/claim.pem.crt" `
  --public-key-outfile "claim-certs/claim.public.pem.key" `
  --private-key-outfile "claim-certs/claim.private.pem.key" `
  --set-as-active
```

A resposta contém informações sobre o certificado, se a solicitação for bem-sucedida. Salve o ARN do certificado para usar mais tarde.

3. Crie e anexe uma AWS IoT política que permita que os dispositivos usem o certificado para criar certificados de dispositivos exclusivos e provisionar com o modelo de aprovisionamento da frota. A política a seguir permite o acesso à API MQTT de provisionamento de dispositivos. Para obter mais informações, consulte a API MQTT de provisionamento de dispositivos no Guia do AWS IoT Core desenvolvedor.

   Faça o seguinte:

   a. Crie um arquivo que contenha o documento AWS IoT de política exigido pelos dispositivos principais do Greengrass.

      Por exemplo, em um sistema baseado em Linux, você pode executar o seguinte comando para usar o GNU nano para criar o arquivo.

      ```
      nano greengrass-provisioning-claim-iot-policy.json
      ```

      Copie o seguinte JSON no arquivo.

      - Substitua cada instância da *região* pelo Região da AWS local em que você configurou o provisionamento da frota.

- Substitua cada instância de *account-id* pelo seu Conta da AWS ID.
- Substitua cada instância do *GreengrassFleetProvisioningTemplate*pelo nome do modelo de aprovisionamento de frota que você criou na seção anterior.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:Connect",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish",
        "iot:Receive"
      ],
      "Resource": [
        "arn:aws:iot:region:account-id:topic/$aws/certificates/create/*",
        "arn:aws:iot:region:account-id:topic/$aws/provisioning-
templates/GreengrassFleetProvisioningTemplate/provision/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "iot:Subscribe",
      "Resource": [
        "arn:aws:iot:region:account-id:topicfilter/$aws/certificates/create/*",
        "arn:aws:iot:region:account-id:topicfilter/$aws/provisioning-
templates/GreengrassFleetProvisioningTemplate/provision/*"
      ]
    }
  ]
}
```

b. Crie uma AWS IoT política a partir do documento de política.

- *GreengrassProvisioningClaimPolicy*Substitua pelo nome da política a ser criada.

```
aws iot create-policy --policy-name GreengrassProvisioningClaimPolicy --policy-
document file://greengrass-provisioning-claim-iot-policy.json
```

A resposta será semelhante ao exemplo a seguir, se a solicitação for bem-sucedida.

```
{
  "policyName": "GreengrassProvisioningClaimPolicy",
  "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassProvisioningClaimPolicy",
  "policyDocument": "{
    \"Version\": \"2012-10-17\",
    \"Statement\": [
      {
        \"Effect\": \"Allow\",
        \"Action\": \"iot:Connect\",
        \"Resource\": \"*\"
      },
      {
        \"Effect\": \"Allow\",
        \"Action\": [
          \"iot:Publish\",
          \"iot:Receive\"
        ],
        \"Resource\": [
          \"arn:aws:iot:region:account-id:topic/$aws/certificates/create/*\",
          \"arn:aws:iot:region:account-id:topic/$aws/provisioning-
templates/GreengrassFleetProvisioningTemplate/provision/*\"
        ]
      },
      {
        \"Effect\": \"Allow\",
        \"Action\": \"iot:Subscribe\",
        \"Resource\": [
          \"arn:aws:iot:region:account-id:topicfilter/$aws/certificates/create/
*\",
          \"arn:aws:iot:region:account-id:topicfilter/$aws/provisioning-
templates/GreengrassFleetProvisioningTemplate/provision/*\"
        ]
      }
    ]
  }",
```

```
  "policyVersionId": "1"
}
```

4. Anexe a AWS IoT política ao certificado de solicitação de aprovisionamento.

   - *GreengrassProvisioningClaimPolicy*Substitua pelo nome da política a ser anexada.
   - Substitua o ARN de destino pelo ARN do certificado de solicitação de aprovisionamento.

```
aws iot attach-policy --policy-name GreengrassProvisioningClaimPolicy --
target arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
```

   O comando não terá nenhuma saída se a solicitação for bem-sucedida.

Agora você tem um certificado de solicitação de provisionamento e uma chave privada que os dispositivos podem usar para se registrar AWS IoT e se provisionar como dispositivos principais do Greengrass. Você pode incorporar o certificado de solicitação e a chave privada nos dispositivos durante a fabricação ou copiar o certificado e a chave nos dispositivos antes de instalar o software AWS IoT Greengrass Core. Para ter mais informações, consulte Instale o software AWS IoT Greengrass principal com provisionamento de AWS IoT frota.

Configurar o plug-in de aprovisionamento de AWS IoT frotas

O plug-in de provisionamento de AWS IoT frota fornece os seguintes parâmetros de configuração que você pode personalizar ao instalar o software AWS IoT Greengrass Core com provisionamento de frota.

`rootPath`

O caminho para a pasta a ser usada como raiz do software AWS IoT Greengrass Core.

`awsRegion`

O Região da AWS que o plug-in de provisionamento de frota usa para AWS provisionar recursos.

`iotDataEndpoint`

O endpoint de AWS IoT dados para seu. Conta da AWS

`iotCredentialEndpoint`

O endpoint AWS IoT de credenciais para seu. Conta da AWS

`iotRoleAlias`

O alias de AWS IoT função que aponta para uma função do IAM de troca de tokens. O provedor de AWS IoT credenciais assume essa função para permitir que o dispositivo principal do Greengrass interaja com os serviços. AWS Para ter mais informações, consulte Autorize os dispositivos principais a interagir com os serviços AWS.

`provisioningTemplate`

O modelo de provisionamento de AWS IoT frota a ser usado para AWS provisionar recursos. Esse modelo deve especificar o seguinte:

- Qualquer AWS IoT coisa, recurso. Você pode especificar uma lista de grupos de coisas existentes para implantar componentes em cada dispositivo quando ele estiver on-line.
- Um recurso AWS IoT político. Esse recurso pode definir uma das seguintes propriedades:
  - O nome de uma AWS IoT política existente. Se você escolher essa opção, os dispositivos principais criados a partir desse modelo usarão a mesma AWS IoT política e você poderá gerenciar suas permissões como uma frota.
  - Um documento AWS IoT de política. Se você escolher essa opção, cada dispositivo principal criado a partir desse modelo usará uma AWS IoT política exclusiva e você poderá gerenciar permissões para cada dispositivo principal individual.
- Um recurso de AWS IoT certificado. Esse recurso de certificado deve usar o `AWS::IoT::Certificate::Id` parâmetro para anexar o certificado ao dispositivo principal. Para obter mais informações, consulte J ust-in-time provisioning no Guia do AWS IoTdesenvolvedor.

Para obter mais informações, consulte Modelos de provisionamento no Guia do AWS IoT Coredesenvolvedor.

`claimCertificatePath`

O caminho para o certificado de declaração de aprovisionamento para o modelo de aprovisionamento que você especifica em. `provisioningTemplate` Para obter mais informações, consulte CreateProvisioningClaim na Referência da API do AWS IoT Core.

`claimCertificatePrivateKeyPath`

O caminho para a chave privada do certificado de declaração de aprovisionamento para o modelo de aprovisionamento que você especifica em. `provisioningTemplate` Para obter mais informações, consulte CreateProvisioningClaim na Referência da API do AWS IoT Core.

> **Important**
>
> O provisionamento de chaves privadas de solicitação deve ser protegido em todos os momentos, inclusive nos dispositivos principais do Greengrass. Recomendamos que você use CloudWatch métricas e registros da Amazon para monitorar indícios de uso indevido, como o uso não autorizado do certificado de solicitação para provisionar dispositivos. Se você detectar uso indevido, desative o certificado de solicitação de aprovisionamento para que ele não possa ser usado para provisionamento de dispositivos. Para obter mais informações, consulte Monitorar AWS IoT no Guia do Desenvolvedor AWS IoT Core. Para ajudá-lo a gerenciar melhor o número de dispositivos e quais dispositivos se registram no seuConta da AWS, você pode especificar um gancho de pré-provisionamento ao criar um modelo de aprovisionamento de frota. Um gancho de pré-provisionamento é uma AWS Lambda função que valida os parâmetros do modelo que os dispositivos fornecem durante o registro. Por exemplo, você pode criar um gancho de pré-provisionamento que compara a ID do dispositivo a um banco de dados para verificar se o dispositivo tem permissão para provisionar. Para obter mais informações, consulte Ganchos de pré-provisionamento no Guia do desenvolvedor. AWS IoT Core

`rootCaPath`

O caminho para o certificado da autoridade de certificação raiz (CA) da Amazon.

`templateParameters`

(Opcional) O mapa dos parâmetros a serem fornecidos ao modelo de aprovisionamento da frota. Para obter mais informações, consulte a seção Parâmetros dos modelos de aprovisionamento no Guia do desenvolvedor. AWS IoT Core

`deviceId`

(Opcional) O identificador do dispositivo a ser usado como ID do cliente quando o plug-in de provisionamento de frota cria uma conexão MQTT com. AWS IoT

Padrão: um UUID aleatório.

`mqttPort`

(Opcional) A porta a ser usada para conexões MQTT.

Padrão: `8883`

`proxyUrl`

(Opcional) A URL do servidor proxy no formato`scheme://userinfo@host:port`. Para usar um proxy HTTPS, você deve usar a versão 1.1.0 ou posterior do plug-in de provisionamento de frota.

- `scheme`— O esquema, que deve ser `http` ou`https`.

  > **Important**
  >
  > Os dispositivos principais do Greengrass devem executar o Greengrass nucleus v2.5.0 ou posterior para usar proxies HTTPS.
  > Se você configurar um proxy HTTPS, deverá adicionar o certificado CA do servidor proxy ao certificado de CA raiz da Amazon do dispositivo principal. Para ter mais informações, consulte Permita que o dispositivo principal confie em um proxy HTTPS.

- `userinfo`— (Opcional) As informações de nome de usuário e senha. Se você especificar essas informações no`url`, o dispositivo principal do Greengrass ignorará os campos e. `username` `password`
- `host`— O nome do host ou endereço IP do servidor proxy.
- `port`— (Opcional) O número da porta. Se você não especificar a porta, o dispositivo principal do Greengrass usará os seguintes valores padrão:
  - `http`— 80
  - `https`— 443

`proxyUserName`

(Opcional) O nome de usuário que autentica o servidor proxy.

`proxyPassword`

(Opcional) O nome de usuário que autentica o servidor proxy.

Caminho CSR

(Opcional) O caminho para o arquivo de solicitação de assinatura de certificado (CSR) a ser usado para criar o certificado do dispositivo a partir de um CSR. Para obter mais informações, consulte Provisionamento por solicitação no guia do AWS IoT Coredesenvolvedor.

csrPrivateKeyCaminho

(Opcional, obrigatório se `csrPath` for declarado) O caminho para a chave privada usada para gerar a CSR. A chave privada deve ter sido usada para gerar a CSR. Para obter mais informações, consulte Provisionamento por solicitação no guia do AWS IoT Coredesenvolvedor.

AWS IoT registro de alterações do plug-in de aprovisionamento de frotas

A tabela a seguir descreve as mudanças em cada versão do aprovisionamento da AWS IoT frota pelo plugin claim (`aws.greengrass.FleetProvisioningByClaim`).

Version (Versão)	Alterações
1.2.1	Correções de erros e melhorias • Corrige um problema em que o plug-in de provisionamento de frota fica off-line durante a inicialização do Greengrass Nucleus. O plug-in de provisionamento de frota agora repete indefinidamente as chamadas de conexão MQTT.
1.2.0	Correções de erros e melhorias • Adiciona suporte para provisionamento de dispositivos por meio de solicitação de assinatura de certificado com caminho de chave privada configurável. • Pequenas correções e melhorias.
1.1.0	Correções de erros e melhorias • Adiciona suporte para formatos adicionais de caminho de arquivo quando você configura o plug-in em dispositivos Windows. • Adiciona suporte para configurações de proxy de rede HTTPS. Para obter mais informações, consulte Conectar-se à porta 443 ou por meio de um proxy de rede e Permita que o dispositivo principal confie em um proxy HTTPS.
1.0.0	Versão inicial.

Instale o software AWS IoT Greengrass principal com provisionamento personalizado de recursos

Esse recurso está disponível para a versão 2.4.0 e posterior do componente núcleo do Greengrass.

O instalador do software AWS IoT Greengrass Core fornece uma interface Java que você pode implementar em um plug-in personalizado que provisiona AWS os recursos necessários. Você pode desenvolver um plug-in de provisionamento para usar certificados de cliente X.509 personalizados ou para executar etapas complexas de provisionamento que outros processos de instalação não suportam. Para obter mais informações, consulte Criar seus próprios certificados de cliente no Guia do AWS IoT Core desenvolvedor.

Para executar um plug-in de provisionamento personalizado ao instalar o software AWS IoT Greengrass Core, você cria um arquivo JAR que fornece ao instalador. O instalador executa o plug-in, e o plug-in retorna uma configuração de provisionamento que define os AWS recursos para o dispositivo principal do Greengrass. O instalador usa essas informações para configurar o software AWS IoT Greengrass Core no dispositivo. Para ter mais informações, consulte Desenvolva plugins de provisionamento personalizados.

> **Important**
>
> Antes de baixar o software AWS IoT Greengrass Core, verifique se seu dispositivo principal atende aos requisitos para instalar e executar o software AWS IoT Greengrass Core v2.0.

Tópicos

Pré-requisitos

Para instalar o software AWS IoT Greengrass Core com provisionamento personalizado, você deve ter o seguinte:

- Um arquivo JAR para um plug-in de provisionamento personalizado que implementa o `DeviceIdentityInterface` O plug-in de provisionamento personalizado deve retornar valores para cada parâmetro de configuração do sistema e do núcleo. Caso contrário, você deverá fornecer esses valores no arquivo de configuração durante a instalação. Para ter mais informações, consulte Desenvolva plugins de provisionamento personalizados.

Configurar o ambiente do dispositivo

Siga as etapas nesta seção para configurar um dispositivo Linux ou Windows para ser usado como seu dispositivo AWS IoT Greengrass principal.

Configurar um dispositivo Linux

Para configurar um dispositivo Linux para AWS IoT Greengrass V2

1. Instale o Java Runtime, que o software AWS IoT Greengrass Core exige para ser executado. Recomendamos que você use as versões de suporte de longo prazo do Amazon Corretto ou do OpenJDK. É necessária a versão 8 ou superior. Os comandos a seguir mostram como instalar o OpenJDK no seu dispositivo.

   - Para distribuições com base em Debian ou em Ubuntu:

     ```
     sudo apt install default-jdk
     ```

   - Para distribuições com base em Red Hat:

     ```
     sudo yum install java-11-openjdk-devel
     ```

   - Para Amazon Linux 2:

     ```
     sudo amazon-linux-extras install java-openjdk11
     ```

   - Para Amazon Linux 2023:

     ```
     sudo dnf install java-11-amazon-corretto -y
     ```

   Quando a instalação for concluída, execute o comando a seguir para verificar se o Java é executado no seu dispositivo Linux.

```
java -version
```

O comando imprime a versão do Java que é executada no dispositivo. Por exemplo, em uma distribuição baseada em Debian, o resultado pode ser semelhante ao exemplo a seguir.

```
openjdk version "11.0.9.1" 2020-11-04
OpenJDK Runtime Environment (build 11.0.9.1+1-post-Debian-1deb10u2)
OpenJDK 64-Bit Server VM (build 11.0.9.1+1-post-Debian-1deb10u2, mixed mode)
```

2. (Opcional) Crie o usuário e o grupo padrão do sistema que executa componentes no dispositivo. Você também pode optar por permitir que o instalador do software AWS IoT Greengrass Core crie esse usuário e grupo durante a instalação com o argumento do `--component-default-user` instalador. Para ter mais informações, consulte Argumentos de instalação.

```
sudo useradd --system --create-home ggc_user
sudo groupadd --system ggc_group
```

3. Verifique se o usuário que executa o software AWS IoT Greengrass Core (normalmente`root`) tem permissão para executar `sudo` com qualquer usuário e qualquer grupo.

   a. Execute o comando a seguir para abrir o `/etc/sudoers` arquivo.

   ```
   sudo visudo
   ```

   b. Verifique se a permissão para o usuário se parece com o exemplo a seguir.

   ```
   root    ALL=(ALL:ALL) ALL
   ```

4. (Opcional) Para executar funções Lambda em contêineres, você deve habilitar cgroups v1 e habilitar e montar os cgroups de memória e dispositivos. Se você não planeja executar funções Lambda em contêineres, pode pular esta etapa.

   Para habilitar essas opções de cgroups, inicialize o dispositivo com os seguintes parâmetros do kernel Linux.

   ```
   cgroup_enable=memory cgroup_memory=1 systemd.unified_cgroup_hierarchy=0
   ```

Para obter informações sobre como visualizar e definir os parâmetros do kernel para seu dispositivo, consulte a documentação do sistema operacional e do carregador de inicialização. Siga as instruções para definir permanentemente os parâmetros do kernel.

5. Instale todas as outras dependências necessárias em seu dispositivo, conforme indicado na lista de requisitos emRequisitos do dispositivo.

Configurar um dispositivo Windows

> **Note**
>
> Esse recurso está disponível para a versão 2.5.0 e posterior do componente de núcleo do Greengrass.

Para configurar um dispositivo Windows para AWS IoT Greengrass V2

1. Instale o Java Runtime, que o software AWS IoT Greengrass Core exige para ser executado. Recomendamos que você use as versões de suporte de longo prazo do Amazon Corretto ou do OpenJDK. É necessária a versão 8 ou superior.
2. Verifique se o Java está disponível na variável de sistema PATH e, caso contrário, adicione-o. A LocalSystem conta executa o software AWS IoT Greengrass Core, então você deve adicionar Java à variável de sistema PATH em vez da variável de usuário PATH para seu usuário. Faça o seguinte:

   a. Pressione a tecla Windows para abrir o menu Iniciar.

   b. Digite **`environment variables`** para pesquisar as opções do sistema no menu Iniciar.

   c. Nos resultados da pesquisa do menu Iniciar, escolha Editar as variáveis de ambiente do sistema para abrir a janela Propriedades do sistema.

   d. Escolha variáveis de ambiente... para abrir a janela Variáveis de ambiente.

   e. Em Variáveis do sistema, selecione Caminho e, em seguida, escolha Editar. Na janela Editar variável de ambiente, você pode visualizar cada caminho em uma linha separada.

   f. Verifique se o caminho para a `bin` pasta da instalação do Java está presente. O caminho pode ser semelhante ao exemplo a seguir.

   ```
   C:\\Program Files\\Amazon Corretto\\jdk11.0.13_8\\bin
   ```

g. Se a `bin` pasta da instalação do Java estiver ausente do Path, escolha Novo para adicioná-la e, em seguida, escolha OK.

3. Abra o prompt de comando do Windows (`cmd.exe`) como administrador.
4. Crie o usuário padrão na LocalSystem conta no dispositivo Windows. *`Substitua a senha`* por uma senha segura.

```
net user /add ggc_user password
```

> **Tip**
>
> Dependendo da configuração do Windows, a senha do usuário pode ser definida para expirar em uma data futura. Para garantir que seus aplicativos Greengrass continuem operando, monitore quando a senha expira e atualize-a antes que ela expire. Você também pode definir a senha do usuário para nunca expirar.
>
> - Para verificar quando um usuário e sua senha expiram, execute o comando a seguir.
>
>   ```
>   net user ggc_user | findstr /C:expires
>   ```
>
> - Para definir que a senha de um usuário nunca expire, execute o comando a seguir.
>
>   ```
>   wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
>   ```
>
> - Se você estiver usando o Windows 10 ou posterior, onde o wmiccomando está obsoleto, execute o comando a seguir. PowerShell
>
>   ```
>   Get-CimInstance -Query "SELECT * from Win32_UserAccount WHERE name =
>    'ggc_user'" | Set-CimInstance -Property @{PasswordExpires="False"}
>   ```

5. Baixe e instale o PsExecutilitário da Microsoft no dispositivo.
6. Use o PsExec utilitário para armazenar o nome de usuário e a senha do usuário padrão na instância do Credential Manager da LocalSystem conta. Substitua a *`senha`* pela senha do usuário que você definiu anteriormente.

```
psexec -s cmd /c cmdkey /generic:ggc_user /user:ggc_user /pass:password
```

Se for PsExec License Agreementaberto, escolha Acceptconcordar com a licença e execute o comando.

> **Note**
>
> Em dispositivos Windows, a LocalSystem conta executa o núcleo Greengrass, e você deve usar o PsExec utilitário para armazenar as informações padrão do usuário na conta. LocalSystem O uso do aplicativo Credential Manager armazena essas informações na conta do Windows do usuário atualmente conectado, em vez da LocalSystem conta.

Baixe o software AWS IoT Greengrass Core

Você pode baixar a versão mais recente do software AWS IoT Greengrass Core no seguinte local:

- https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip

> **Note**
>
> Você pode baixar uma versão específica do software AWS IoT Greengrass Core no seguinte local. Substitua a *versão* pela versão a ser baixada.
>
> ```
> https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip
> ```

Para baixar o software AWS IoT Greengrass Core

1. Em seu dispositivo principal, baixe o software AWS IoT Greengrass Core para um arquivo chamado`greengrass-nucleus-latest.zip`.

   Linux or Unix

   ```
   curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-
   latest.zip > greengrass-nucleus-latest.zip
   ```

Windows Command Prompt (CMD)

```
curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-
latest.zip > greengrass-nucleus-latest.zip
```

PowerShell

```
iwr -Uri https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-
latest.zip -OutFile greengrass-nucleus-latest.zip
```

Ao fazer download desse software, você concorda com o Contrato de licença do software do Greengrass Core.

2. (Opcional) Para verificar a assinatura do software Greengrass nucleus

> **Note**
>
> Esse recurso está disponível com o Greengrass nucleus versão 2.9.5 e posterior.

a. Use o comando a seguir para verificar a assinatura do artefato do núcleo Greengrass:

Linux or Unix

```
jarsigner -verify -certs -verbose greengrass-nucleus-latest.zip
```

Windows Command Prompt (CMD)

O nome do arquivo pode parecer diferente dependendo da versão do JDK que você instala. *jdk17.0.6_10*Substitua pela versão do JDK que você instalou.

```
"C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe" -
verify -certs -verbose greengrass-nucleus-latest.zip
```

PowerShell

O nome do arquivo pode parecer diferente dependendo da versão do JDK que você instala. *jdk17.0.6_10*Substitua pela versão do JDK que você instalou.

```
'C:\\Program Files\\Amazon Corretto\\jdk17.0.6_10\\bin\\jarsigner.exe' -
verify -certs -verbose greengrass-nucleus-latest.zip
```

b. A `jarsigner` invocação produz uma saída que indica os resultados da verificação.

   i. Se o arquivo zip do Greengrass nucleus estiver assinado, a saída conterá a seguinte declaração:

   ```
   jar verified.
   ```

   ii. Se o arquivo zip do Greengrass nucleus não estiver assinado, a saída conterá a seguinte declaração:

   ```
   jar is unsigned.
   ```

c. Se você forneceu a `-certs` opção Jarsigner junto com `-verbose` as opções `-verify` e, a saída também incluirá informações detalhadas do certificado do assinante.

3. Descompacte o software AWS IoT Greengrass Core em uma pasta no seu dispositivo. *GreengrassInstaller*Substitua pela pasta que você deseja usar.

   Linux or Unix

   ```
   unzip greengrass-nucleus-latest.zip -d GreengrassInstaller && rm greengrass-
   nucleus-latest.zip
   ```

   Windows Command Prompt (CMD)

   ```
   mkdir GreengrassInstaller && tar -xf greengrass-nucleus-latest.zip -
   C GreengrassInstaller && del greengrass-nucleus-latest.zip
   ```

   PowerShell

   ```
   Expand-Archive -Path greengrass-nucleus-latest.zip -DestinationPath .\
   \GreengrassInstaller
   rm greengrass-nucleus-latest.zip
   ```

4. (Opcional) Execute o comando a seguir para ver a versão do software AWS IoT Greengrass Core.

```
java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
```

> **Important**
>
> Se você instalar uma versão do núcleo do Greengrass anterior à v2.4.0, não remova essa pasta depois de instalar o software Core. AWS IoT Greengrass O software AWS IoT Greengrass Core usa os arquivos dessa pasta para ser executado.
> Se você baixou a versão mais recente do software, instale a versão 2.4.0 ou posterior e poderá remover essa pasta depois de instalar o software AWS IoT Greengrass Core.

Instale o software AWS IoT Greengrass Core

Execute o instalador com argumentos que especificam as seguintes ações:

- Instale a partir de um arquivo de configuração parcial que especifica o uso do plug-in de provisionamento personalizado para provisionar recursos. AWS O software AWS IoT Greengrass Core usa um arquivo de configuração que especifica a configuração de cada componente do Greengrass no dispositivo. O instalador cria um arquivo de configuração completo a partir do arquivo de configuração parcial que você fornece e dos AWS recursos que o plug-in de aprovisionamento personalizado cria.
- Especifique o uso do usuário do `ggc_user` sistema para executar componentes de software no dispositivo principal. Em dispositivos Linux, esse comando também especifica o uso do grupo do `ggc_group` sistema, e o instalador cria o usuário e o grupo do sistema para você.
- Configure o software AWS IoT Greengrass Core como um serviço do sistema que é executado na inicialização. Em dispositivos Linux, isso requer o sistema init Systemd.

> **Important**
>
> Nos dispositivos principais do Windows, você deve configurar o software AWS IoT Greengrass Core como um serviço do sistema.

Para obter mais informações sobre os argumentos que você pode especificar, consulteArgumentos de instalação.

> **Note**
>
> Se você estiver executando AWS IoT Greengrass em um dispositivo com memória limitada, poderá controlar a quantidade de memória que o software AWS IoT Greengrass Core usa. Para controlar a alocação de memória, você pode definir as opções de tamanho da pilha da JVM no parâmetro de `jvmOptions` configuração do componente do núcleo. Para ter mais informações, consulte Controle a alocação de memória com opções de JVM.

Para instalar o software AWS IoT Greengrass Core (Linux)

1. Verifique a versão do software AWS IoT Greengrass Core.

   - *`GreengrassInstaller`*Substitua pelo caminho para a pasta que contém o software.

   ```
   java -jar ./GreengrassInstaller/lib/Greengrass.jar --version
   ```

2. Use um editor de texto para criar um arquivo de configuração chamado `config.yaml` para fornecer ao instalador.

   Por exemplo, em um sistema baseado em Linux, você pode executar o seguinte comando para usar o GNU nano para criar o arquivo.

   ```
   nano GreengrassInstaller/config.yaml
   ```

   Copie o seguinte conteúdo YAML para o arquivo.

   ```
   ---
   system:
     rootpath: "/greengrass/v2"
     # The following values are optional. Return them from the provisioning plugin or
    set them here.
     # certificateFilePath: ""
     # privateKeyPath: ""
     # rootCaPath: ""
     # thingName: ""
   services:
     aws.greengrass.Nucleus:
       version: "2.12.6"
   ```

```
    configuration:
      # The following values are optional. Return them from the provisioning plugin
or set them here.
      # awsRegion: ""
      # iotRoleAlias: ""
      # iotDataEndpoint: ""
      # iotCredEndpoint: ""
  com.example.CustomProvisioning:
    configuration:
      # You can specify configuration parameters to provide to your plugin.
      # pluginParameter: ""
```

Então, faça o seguinte:

- Substitua *2.12.6* pela versão do software AWS IoT Greengrass Core.
- Substitua cada instância do */greengrass/v2* pela pasta raiz do Greengrass.
- (Opcional) Especifique os valores de configuração do sistema e do núcleo. Você deve definir esses valores se seu plug-in de provisionamento não os fornecer.
- (Opcional) Especifique os parâmetros de configuração a serem fornecidos ao seu plug-in de provisionamento.

> **Note**
>
> Nesse arquivo de configuração, você pode personalizar outras opções de configuração, como as portas e o proxy de rede a serem usados, conforme mostrado no exemplo a seguir. Para obter mais informações, consulte Configuração do núcleo do Greengrass.
>
> ```
> ---
> system:
>   rootpath: "/greengrass/v2"
>   # The following values are optional. Return them from the provisioning
>  plugin or set them here.
>   # certificateFilePath: ""
>   # privateKeyPath: ""
>   # rootCaPath: ""
>   # thingName: ""
> services:
>   aws.greengrass.Nucleus:
>     version: "2.12.6"
>     configuration:
> ```

```
      mqtt:
        port: 443
      greengrassDataPlanePort: 443
      networkProxy:
        noProxyAddresses: "http://192.168.0.1,www.example.com"
        proxy:
          url: "http://my-proxy-server:1100"
          username: "Mary_Major"
          password: "pass@word1357"
      # The following values are optional. Return them from the provisioning
 plugin or set them here.
      # awsRegion: ""
      # iotRoleAlias: ""
      # iotDataEndpoint: ""
      # iotCredEndpoint: ""
  com.example.CustomProvisioning:
    configuration:
      # You can specify configuration parameters to provide to your plugin.
      # pluginParameter: ""
```

3. Execute o instalador. Especifique `--trusted-plugin` para fornecer seu plug-in de provisionamento personalizado e especifique `--init-config` para fornecer o arquivo de configuração.

   - Substitua */greengrass/v2* ou *C:\greengrass\v2* pela pasta raiz do Greengrass.
   - Substitua cada instância do *GreengrassInstaller*pela pasta em que você descompactou o instalador.
   - Substitua o caminho para o arquivo JAR do plug-in de provisionamento personalizado pelo caminho para o arquivo JAR do seu plug-in.

   Linux or Unix

   ```
   sudo -E java -Droot="/greengrass/v2" -Dlog.store=FILE \
     -jar ./GreengrassInstaller/lib/Greengrass.jar \
     --trusted-plugin /path/to/com.example.CustomProvisioning.jar \
     --init-config ./GreengrassInstaller/config.yaml \
     --component-default-user ggc_user:ggc_group \
     --setup-system-service true
   ```

Windows Command Prompt (CMD)

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" ^
  -jar ./GreengrassInstaller/lib/Greengrass.jar ^
  --trusted-plugin /path/to/com.example.CustomProvisioning.jar ^
  --init-config ./GreengrassInstaller/config.yaml ^
  --component-default-user ggc_user ^
  --setup-system-service true
```

PowerShell

```
java -Droot="C:\greengrass\v2" "-Dlog.store=FILE" `
  -jar ./GreengrassInstaller/lib/Greengrass.jar `
  --trusted-plugin /path/to/com.example.CustomProvisioning.jar `
  --init-config ./GreengrassInstaller/config.yaml `
  --component-default-user ggc_user `
  --setup-system-service true
```

Important

Nos dispositivos principais do Windows, você deve especificar `--setup-system-service true` a configuração do software AWS IoT Greengrass Core como um serviço do sistema.

Se você especificar`--setup-system-service true`, o instalador `Successfully set up Nucleus as a system service` imprimirá se configurou e executou o software como um serviço do sistema. Caso contrário, o instalador não emitirá nenhuma mensagem se instalar o software com êxito.

Note

Você não pode usar o `deploy-dev-tools` argumento para implantar ferramentas de desenvolvimento local ao executar o instalador sem o `--provision true` argumento. Para obter informações sobre a implantação da CLI do Greengrass diretamente em seu dispositivo, consulte. Interface de linha de comando do Greengrass

4. Verifique a instalação visualizando os arquivos na pasta raiz.

   Linux or Unix

   ```
   ls /greengrass/v2
   ```

   Windows Command Prompt (CMD)

   ```
   dir C:\greengrass\v2
   ```

   PowerShell

   ```
   ls C:\greengrass\v2
   ```

   Se a instalação for bem-sucedida, a pasta raiz conterá várias pastas`config`, como`packages`, e. `logs`

Se você instalou o software AWS IoT Greengrass Core como um serviço do sistema, o instalador executa o software para você. Caso contrário, você deverá executar o software manualmente. Para ter mais informações, consulte Execute o software AWS IoT Greengrass Core.

Para obter mais informações sobre como configurar e usar o software AWS IoT Greengrass, consulte o seguinte:

- Configurar o software AWS IoT Greengrass principal
- Desenvolva AWS IoT Greengrass componentes
- Implemente AWS IoT Greengrass componentes em dispositivos
- Interface de linha de comando do Greengrass

Desenvolva plugins de provisionamento personalizados

Para desenvolver um plugin de provisionamento personalizado, crie uma classe Java que implemente o`com.aws.greengrass.provisioning.DeviceIdentityInterface`Interface. Você pode incluir o arquivo JAR do núcleo Greengrass em seu projeto para acessar essa interface e suas classes. Essa interface define um método que insere uma configuração de plug-in e gera uma configuração de provisionamento. A configuração de provisionamento define configurações para o

sistema e oComponente do núcleo do Greengrass. OAWS IoT GreengrassO instalador de software principal usa essa configuração de provisionamento para configurar oAWS IoT GreengrassSoftware principal em um dispositivo.

Depois de desenvolver um plugin de provisionamento personalizado, crie-o como um arquivo JAR que você pode fornecer aoAWS IoT GreengrassInstalador de software principal para executar seu plugin durante a instalação. O instalador executa seu plug-in de provisionamento personalizado na mesma JVM que o instalador usa, para que você possa criar um JAR que contenha apenas o código do plug-in.

> **Note**
>
> OAWS IoTplugin de provisionamento de frotaImplementa o`DeviceIdentityInterface`Para usar o provisionamento de frota durante a instalação. O plugin de provisionamento de frota é de código aberto, para que você possa explorar seu código-fonte para ver um exemplo de como usar a interface do plugin de provisionamento. Para obter mais informações, consulte o .AWS IoTplugin de provisionamento de frotano GitHub

Tópicos

- Requisitos
- Implemente o DeviceIdentityInterface interface

Requisitos

Para desenvolver um plugin de provisionamento personalizado, você deve criar uma classe Java que atenda aos seguintes requisitos:

- Usa o`com.aws.greengrass`pacote ou um pacote dentro do`com.aws.greengrass`pacote.
- Tem um construtor sem argumentos.
- Implementa o`DeviceIdentityInterface`Interface. Para obter mais informações, consulteImplemente o DeviceIdentityInterface interface

Implemente o DeviceIdentityInterface interface

Para usar a `com.aws.greengrass.provisioning.DeviceIdentityInterface` interface em seu plugin personalizado, adicione o núcleo Greengrass como uma dependência ao seu projeto.

Para usar a DeviceIdentityInterface em um projeto de plug-in de provisionamento personalizado

- Você pode adicionar o arquivo JAR do núcleo Greengrass como uma biblioteca ou adicionar o núcleo Greengrass como uma dependência do Maven. Execute um destes procedimentos:
  - Para adicionar o arquivo JAR do núcleo Greengrass como uma biblioteca, baixe o arquivo JARAWS IoT GreengrassSoftware principal, que contém o núcleo Greengrass JAR. Você pode fazer download da versão mais recente doAWS IoT GreengrassSoftware do núcleo do seguinte local:

    - https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip

    Você pode encontrar o arquivo JAR do Greengrass nucleus (`Greengrass.jar`) no `lib` pasta no arquivo ZIP. Adicione esse arquivo JAR ao projeto.
  - Para consumir o núcleo Greengrass em um projeto Maven, adicione uma dependência no `nucleus` artefato no `com.aws.greengrass` grupo. Você também deve adicionar o `greengrass-common` repositório, porque o núcleo Greengrass não está disponível no Repositório Central do Maven.

```
<project ...>
    ...
    <repositories>
        <repository>
            <id>greengrass-common</id>
            <name>greengrass common</name>
            <url>https://d2jrmugq4soldf.cloudfront.net/snapshots</url>
        </repository>
    </repositories>
    ...
    <dependencies>
        <dependency>
            <groupId>com.aws.greengrass</groupId>
            <artifactId>nucleus</artifactId>
            <version>2.5.0-SNAPSHOT</version>
            <scope>provided</scope>
        </dependency>
```

```
    </dependencies>
</project>
```

A interface DeviceIdentityInterface

O`com.aws.greengrass.provisioning.DeviceIdentityInterface`A interface tem a seguinte forma.

> **Note**
>
> Você também pode explorar essas aulas nopacote com.aws.greengrass.provisioningdoCódigo fonte do núcleo Greengrassno GitHub

```
public interface com.aws.greengrass.provisioning.DeviceIdentityInterface {
    ProvisionConfiguration updateIdentityConfiguration(ProvisionContext context)
            throws RetryableProvisioningException, InterruptedException;

    // Return the name of the plugin.
    String name();
}

com.aws.greengrass.provisioning.ProvisionConfiguration {
    SystemConfiguration systemConfiguration;
    NucleusConfiguration nucleusConfiguration
}

com.aws.greengrass.provisioning.ProvisionConfiguration.SystemConfiguration {
    String certificateFilePath;
    String privateKeyPath;
    String rootCAPath;
    String thingName;
}

com.aws.greengrass.provisioning.ProvisionConfiguration.NucleusConfiguration {
    String awsRegion;
    String iotCredentialsEndpoint;
    String iotDataEndpoint;
    String iotRoleAlias;
}
```

```
com.aws.greengrass.provisioning.ProvisioningContext {
    Map<String, Object> parameterMap;
    String provisioningPolicy;  // The policy is always "PROVISION_IF_NOT_PROVISIONED".
}

com.aws.greengrass.provisioning.exceptions.RetryableProvisioningException {}
```

Cada valor de configuração no`SystemConfiguration`e`NucleusConfiguration`é necessário para instalar oAWS IoT GreengrassSoftware principal, mas você pode retornar`null`. Se o plug-in de provisionamento personalizado retorna`null`para qualquer valor de configuração, você deve fornecer esse valor na configuração do sistema ou núcleo ao criar o`config.yaml`arquivo para fornecer aoAWS IoT GreengrassInstalador do software do núcleo. Se o plug-in de provisionamento personalizado retornar um valor não nulo para uma opção que você também define em`config.yaml`, então o instalador substitui o valor em`config.yaml`com o valor retornado pelo plugin.

Argumentos de instalação

O software AWS IoT Greengrass principal inclui um instalador que configura o software e provisiona os AWS recursos necessários para a execução do dispositivo principal do Greengrass. O instalador inclui os seguintes argumentos que você pode especificar para configurar a instalação:

`-h`, `--help`

(Opcional) Mostre as informações de ajuda do instalador.

`--version`

(Opcional) Mostre a versão do software AWS IoT Greengrass Core.

`-Droot`

(Opcional) O caminho para a pasta a ser usada como raiz do software AWS IoT Greengrass Core.

> **Note**
>
> Esse argumento define uma propriedade da JVM, portanto, você deve especificá-la antes de `-jar` executar o instalador. Por exemplo, especifique `java -Droot="/greengrass/v2" -jar /path/to/Greengrass.jar`.

Padrão:

- Linux: `~/.greengrass`
- Windows: `%USERPROFILE%/.greengrass`

`-ar`, `--aws-region`

O Região da AWS que o software AWS IoT Greengrass Core usa para recuperar ou criar AWS os recursos necessários.

`-p`, `--provision`

(Opcional) Você pode registrar esse dispositivo como uma AWS IoT coisa e provisionar os AWS recursos que o dispositivo principal exige. Se você especificar`true`, o software AWS IoT Greengrass Core provisiona qualquer AWS IoT coisa, (opcional) qualquer grupo de AWS IoT coisas, uma função do IAM e um alias de AWS IoT função.

Padrão: `false`

`-tn`, `--thing-name`

(Opcional) O nome da AWS IoT coisa que você registra como esse dispositivo principal. Se a coisa com o nome não existir no seuConta da AWS, o software AWS IoT Greengrass Core a cria.

> **Note**
>
> O nome da coisa não pode conter caracteres de dois pontos (`:`).

Você deve especificar `--provision true` para aplicar esse argumento.

Padrão: `GreengrassV2IotThing_` mais um UUID aleatório.

`-tgn`, `--thing-group-name`

(Opcional) O nome do grupo de AWS IoT coisas em que você adiciona o item desse dispositivo AWS IoT principal. Se uma implantação tem como alvo esse grupo de coisas, esse dispositivo principal recebe essa implantação quando se conecta AWS IoT Greengrass a. Se o grupo de coisas com esse nome não existir no seuConta da AWS, o software AWS IoT Greengrass Core o criará.

> **Note**
>
> O nome do grupo de coisas não pode conter caracteres de dois pontos (`:`).

Você deve especificar `--provision true` para aplicar esse argumento.

`-tpn`, `--thing-policy-name`

Esse recurso está disponível para a versão 2.4.0 e posterior do componente núcleo do Greengrass.

(Opcional) O nome da AWS IoT política a ser anexada ao certificado de AWS IoT coisas desse dispositivo principal. Se a AWS IoT política com esse nome não existir na suaConta da AWS, o software AWS IoT Greengrass Core a criará.

O software AWS IoT Greengrass Core cria uma AWS IoT política permissiva por padrão. Você pode definir o escopo dessa política ou criar uma política personalizada na qual restrinja as permissões para seu caso de uso. Para ter mais informações, consulte AWS IoTPolítica mínima para dispositivos AWS IoT Greengrass V2 principais.

Você deve especificar `--provision true` para aplicar esse argumento.

Padrão: `GreengrassV2IoTThingPolicy`

`-trn`, `--tes-role-name`

(Opcional) O nome da função do IAM a ser usada para adquirir AWS credenciais que permitem que o dispositivo principal interaja com AWS os serviços. Se a função com esse nome não existir na suaConta da AWS, o software AWS IoT Greengrass Core a criará com a `GreengrassV2TokenExchangeRoleAccess` política. Essa função não tem acesso aos buckets do S3 nos quais você hospeda artefatos de componentes. Portanto, você deve adicionar permissões aos buckets e objetos do S3 dos seus artefatos ao criar um componente. Para ter mais informações, consulte Autorize os dispositivos principais a interagir com os serviços AWS.

Você deve especificar `--provision true` para aplicar esse argumento.

Padrão: `GreengrassV2TokenExchangeRole`

`-tra`, `--tes-role-alias-name`

(Opcional) O nome do alias da AWS IoT função que aponta para a função do IAM que fornece AWS credenciais para esse dispositivo principal. Se o alias de função com esse nome não existir

no seuConta da AWS, o software AWS IoT Greengrass Core o criará e o direcionará para a função do IAM que você especificar.

Você deve especificar `--provision true` para aplicar esse argumento.

Padrão: `GreengrassV2TokenExchangeRoleAlias`

`-ss`, `--setup-system-service`

(Opcional) Você pode configurar o software AWS IoT Greengrass Core como um serviço do sistema que é executado quando esse dispositivo é inicializado. O nome do serviço do sistema é `greengrass`. Para ter mais informações, consulte Configurar o núcleo do Greengrass como um serviço do sistema.

Em sistemas operacionais Linux, esse argumento exige que o sistema init systemd esteja disponível no dispositivo.

> **Important**
>
> Nos dispositivos principais do Windows, você deve configurar o software AWS IoT Greengrass Core como um serviço do sistema.

Padrão: `false`

`-u`, `--component-default-user`

O nome ou ID do usuário que o software AWS IoT Greengrass Core usa para executar componentes. Por exemplo, é possível especificar **`ggc_user`**. Esse valor é necessário quando você executa o instalador nos sistemas operacionais Windows.

Em sistemas operacionais Linux, você também pode especificar opcionalmente o grupo. Especifique o usuário e o grupo separados por dois pontos. Por exemplo, **`ggc_user:ggc_group`**.

As seguintes considerações adicionais se aplicam aos sistemas operacionais Linux:

- Se você executar como root, o usuário padrão do componente é o usuário definido no arquivo de configuração. Se o arquivo de configuração não definir um usuário, o padrão será. `ggc_user:ggc_group` Se existirem `ggc_user` ou `ggc_group` não, o software os cria.
- Se você executa como usuário não root, o software AWS IoT Greengrass Core usa esse usuário para executar componentes.

- Se você não especificar um grupo, o software AWS IoT Greengrass Core usa o grupo primário do usuário do sistema.

Para ter mais informações, consulte Configurar o usuário que executa os componentes.

`-d, --deploy-dev-tools`

(Opcional) Você pode baixar e implantar o componente CLI do Greengrass nesse dispositivo principal. Você pode usar essa ferramenta para desenvolver e depurar componentes nesse dispositivo principal.

> **Important**
>
> Recomendamos que você use esse componente somente em ambientes de desenvolvimento, não em ambientes de produção. Esse componente fornece acesso a informações e operações que você normalmente não precisará em um ambiente de produção. Siga o princípio do menor privilégio implantando esse componente somente nos dispositivos principais onde você precisar dele.

Você deve especificar `--provision true` para aplicar esse argumento.

Padrão: `false`

`-init, --init-config`

(Opcional) O caminho para o arquivo de configuração a ser usado para instalar o software AWS IoT Greengrass Core. Você pode usar essa opção para configurar novos dispositivos principais com uma configuração de núcleo específica, por exemplo.

> **Important**
>
> O arquivo de configuração que você especifica se mescla com o arquivo de configuração existente no dispositivo principal. Isso inclui os componentes e as configurações dos componentes no dispositivo principal. Recomendamos que o arquivo de configuração liste somente as configurações que você está tentando alterar.

`-tp, --trusted-plugin`

(Opcional) O caminho para um arquivo JAR a ser carregado como um plug-in confiável. Use essa opção para fornecer arquivos JAR do plug-in de provisionamento, como para instalar com

provisionamento de frota ou provisionamentopersonalizado, ou para instalar com a chave privada e o certificado em um módulo de segurança de hardware.

`-s`, `--start`

(Opcional) Você pode iniciar o software AWS IoT Greengrass principal após a instalação e, opcionalmente, provisionar recursos.

Padrão: `true`

Execute o software AWS IoT Greengrass Core

Depois de instalar o software AWS IoT Greengrass Core, execute-o para conectar seu dispositivo AWS IoT Greengrass a.

Ao instalar o software AWS IoT Greengrass Core, você pode especificar se deseja instalá-lo como um serviço do sistema com o systemd. Se você escolher essa opção, o instalador executará o software para você e o configurará para ser executado quando o dispositivo for inicializado.

> **Important**
>
> Nos dispositivos principais do Windows, você deve configurar o software AWS IoT Greengrass Core como um serviço do sistema.

Tópicos

- Verifique se o software AWS IoT Greengrass Core funciona como um serviço do sistema
- Execute o software AWS IoT Greengrass principal como um serviço do sistema
- Execute o software AWS IoT Greengrass Core sem um serviço de sistema

Verifique se o software AWS IoT Greengrass Core funciona como um serviço do sistema

Ao instalar o software AWS IoT Greengrass Core, você pode especificar o `--setup-system-service true` argumento para instalar o software AWS IoT Greengrass Core como um serviço do sistema. Os dispositivos Linux exigem que o sistema systemd init configure o software AWS IoT Greengrass Core como um serviço do sistema. Se você usar essa opção, o instalador executará

o software para você e o configurará para ser executado quando o dispositivo for inicializado. O instalador exibirá a seguinte mensagem se instalar com êxito o software AWS IoT Greengrass Core como um serviço do sistema.

```
Successfully set up Nucleus as a system service
```

Se você instalou anteriormente o software AWS IoT Greengrass Core e não tem a saída do instalador, você pode verificar se o software foi instalado como um serviço do sistema.

Para verificar se o software AWS IoT Greengrass Core está instalado como um serviço do sistema

- Execute o comando a seguir para verificar o status do serviço do sistema Greengrass.

  Linux or Unix (systemd)

  ```
  sudo systemctl status greengrass.service
  ```

  A resposta será semelhante ao exemplo a seguir se o software AWS IoT Greengrass Core estiver instalado como um serviço do sistema e ativo.

  ```
  # greengrass.service - Greengrass Core
     Loaded: loaded (/etc/systemd/system/greengrass.service; enabled; vendor
   preset: disabled)
     Active: active (running) since Thu 2021-02-11 01:33:44 UTC; 4 days ago
   Main PID: 16107 (sh)
     CGroup: /system.slice/greengrass.service
             ##16107 /bin/sh /greengrass/v2/alts/current/distro/bin/loader
             ##16111 java -Dlog.store=FILE -Droot=/greengrass/v2 -jar /greengrass/
  v2/alts/current/distro/lib/Greengrass...
  ```

  Se `systemctl` ou `greengrass.service` não for encontrado, o software AWS IoT Greengrass Core não está instalado como um serviço do sistema. Para executar o software, consulteExecute o software AWS IoT Greengrass Core sem um serviço de sistema.

  Windows Command Prompt (CMD)

  ```
  sc query greengrass
  ```

  A resposta será semelhante ao exemplo a seguir se o software AWS IoT Greengrass Core estiver instalado como um serviço do Windows e ativo.

```
SERVICE_NAME: greengrass
        TYPE               : 10  WIN32_OWN_PROCESS
        STATE              : 4  RUNNING
                                (STOPPABLE, NOT_PAUSABLE, ACCEPTS_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x0
```

PowerShell

```
Get-Service greengrass
```

A resposta será semelhante ao exemplo a seguir se o software AWS IoT Greengrass Core estiver instalado como um serviço do Windows e ativo.

```
Status   Name               DisplayName
------   ----               -----------
Running  greengrass         greengrass
```

Execute o software AWS IoT Greengrass principal como um serviço do sistema

Se o software AWS IoT Greengrass Core estiver instalado como um serviço do sistema, você poderá usar o gerenciador de serviços do sistema para iniciar, parar e gerenciar o software. Para ter mais informações, consulte Configurar o núcleo do Greengrass como um serviço do sistema.

Para executar o software AWS IoT Greengrass Core

- Execute o comando a seguir para iniciar o software AWS IoT Greengrass Core.

  Linux or Unix (systemd)

  ```
  sudo systemctl start greengrass.service
  ```

Windows Command Prompt (CMD)

```
sc start greengrass
```

PowerShell

```
Start-Service greengrass
```

Execute o software AWS IoT Greengrass Core sem um serviço de sistema

Nos dispositivos principais do Linux, se o software AWS IoT Greengrass Core não estiver instalado como um serviço do sistema, você poderá executar o script do carregador do software para executar o software.

Para executar o software AWS IoT Greengrass Core sem um serviço de sistema

- Execute o comando a seguir para iniciar o software AWS IoT Greengrass Core. Se você executar esse comando em um terminal, deverá manter a sessão do terminal aberta para manter o software AWS IoT Greengrass Core em execução.
  - Substitua */greengrass/v2* ou *C:\greengrass\v2* pela pasta raiz do Greengrass que você usa.

```
sudo /greengrass/v2/alts/current/distro/bin/loader
```

O software imprime a seguinte mensagem se for iniciado com êxito.

```
Launched Nucleus successfully.
```

Execute AWS IoT Greengrass o software Core em um contêiner Docker

AWS IoT Greengrass pode ser configurado para ser executado em um contêiner Docker. O Docker é uma plataforma que fornece as ferramentas para você criar, executar, testar e implantar aplicativos baseados em contêineres Linux. Ao executar uma imagem do AWS IoT Greengrass Docker, você

pode escolher se deseja fornecer suas AWS credenciais ao contêiner do Docker e permitir que o instalador do software AWS IoT Greengrass Core provisione automaticamente os AWS recursos que um dispositivo principal do Greengrass requer para operar. Se você não quiser fornecer AWS credenciais, poderá provisionar AWS recursos manualmente e executar o software AWS IoT Greengrass Core no contêiner do Docker.

Tópicos

Plataformas compatíveis e requisitos

Os computadores host devem atender aos seguintes requisitos mínimos para instalar e executar o software AWS IoT Greengrass Core em um contêiner Docker:

- Um sistema operacional baseado em Linux com conexão à Internet.
- Docker Engine versão 18.09 ou posterior.
- (Opcional) Docker Compose versão 1.22 ou posterior. O Docker Compose é necessário somente se você quiser usar a CLI do Docker Compose para executar suas imagens do Docker.

Para executar componentes da função Lambda dentro do contêiner Docker, você deve configurar o contêiner para atender aos requisitos adicionais. Para ter mais informações, consulte Requisitos da função do Lambda.

Execute componentes no modo de processo

AWS IoT Greengrass não suporta a execução de funções Lambda ou componentes AWS fornecidos pelo Lambda em um ambiente de tempo de execução isolado dentro do AWS IoT Greengrass contêiner Docker. Você deve executar esses componentes no modo de processo sem nenhum isolamento.

Ao configurar um componente da função Lambda, defina o modo de isolamento como Sem contêiner. Para ter mais informações, consulte Executar AWS Lambda funções.

Ao implantar qualquer um dos componentes AWS fornecidos a seguir, atualize a configuração de cada componente para definir o `containerMode` parâmetro como. `NoContainer` Para obter mais informações sobre atualizações de configuração, consulteAtualizar configurações de componentes.

- CloudWatch métricas
- Device Defender
- Firehose
- Adaptador de protocolo Modbus-RTU
- Amazon SNS

AWS IoT Greengrass Downloads do software Docker

AWS IoT Greengrass fornece um Dockerfile para criar uma imagem de contêiner que tenha o software AWS IoT Greengrass principal e as dependências instaladas em uma imagem base do Amazon Linux 2 (x86_64). Você pode modificar a imagem base no Dockerfile para ser executada AWS IoT Greengrass em uma arquitetura de plataforma diferente.

Baixe o pacote Dockerfile em. GitHub

O Dockerfile usa uma versão mais antiga do Greengrass. Você deve atualizar o arquivo para usar a versão do Greengrass que você deseja. Para obter informações sobre como criar a imagem do AWS IoT Greengrass contêiner a partir do Dockerfile, consulte. Crie a imagem do AWS IoT Greengrass contêiner a partir de um Dockerfile

Escolha como provisionar AWS recursos

Ao instalar o software AWS IoT Greengrass Core em um contêiner Docker, você pode escolher se deseja provisionar automaticamente os AWS recursos que um dispositivo principal do Greengrass requer para operar ou usar os recursos que você provisiona manualmente.

- Provisionamento automático de recursos — o instalador provisiona a AWS IoT AWS IoT coisa, o grupo de coisas, a função do IAM e o alias da AWS IoT função quando você executa a imagem do AWS IoT Greengrass contêiner pela primeira vez. O instalador também pode implantar as ferramentas de desenvolvimento local no dispositivo principal, para que você possa usar o dispositivo para desenvolver e testar componentes de software personalizados. Para provisionar

automaticamente esses recursos, você deve fornecer AWS credenciais como variáveis de ambiente para a imagem do Docker.

Para usar o provisionamento automático, você deve definir a variável de ambiente do Docker `PROVISION=true` e montar um arquivo de credencial para fornecer suas AWS credenciais ao contêiner.

- Provisionamento manual de recursos — se você não quiser fornecer AWS credenciais ao contêiner, poderá provisionar manualmente os AWS recursos antes de executar a imagem do contêiner. AWS IoT Greengrass Você deve criar um arquivo de configuração para fornecer informações sobre esses recursos ao instalador do software AWS IoT Greengrass Core dentro do contêiner Docker.

  Para usar o provisionamento manual, você deve definir a variável de ambiente Docker. `PROVISION=false` O provisionamento manual é a opção padrão.

Para ter mais informações, consulte Crie a imagem do AWS IoT Greengrass contêiner a partir de um Dockerfile.

Crie a imagem do AWS IoT Greengrass contêiner a partir de um Dockerfile

AWSfornece um Dockerfile que você pode baixar e usar para executar o software AWS IoT Greengrass Core em um contêiner Docker. Os Dockerfiles contêm código-fonte para criar imagens de AWS IoT Greengrass contêineres.

Antes de criar uma imagem de AWS IoT Greengrass contêiner, você deve configurar seu Dockerfile para selecionar a versão do software AWS IoT Greengrass Core que você deseja instalar. Você também pode configurar variáveis de ambiente para escolher como provisionar recursos durante a instalação e personalizar outras opções de instalação. Esta seção descreve como configurar e criar uma imagem do AWS IoT Greengrass Docker a partir de um Dockerfile.

Baixe o pacote Dockerfile

Você pode baixar o pacote AWS IoT Greengrass Dockerfile em: GitHub

Repositório Docker do AWS Greengrass

Depois de baixar o pacote, extraia o conteúdo para a *`download-directory/`*`aws-greengrass-docker-`*`nucleus-version`* pasta no seu computador. O Dockerfile usa uma versão mais antiga do Greengrass. Você deve atualizar o arquivo para usar a versão do Greengrass desejada.

Especifique a versão AWS IoT Greengrass do software principal

Use o seguinte argumento de compilação no Dockerfile para especificar a versão do software AWS IoT Greengrass Core que você deseja usar na imagem do AWS IoT Greengrass Docker. Por padrão, o Dockerfile usa a versão mais recente do software AWS IoT Greengrass Core.

`GREENGRASS_RELEASE_VERSION`

A versão do software AWS IoT Greengrass Core. Por padrão, o Dockerfile baixa a versão mais recente disponível do núcleo do Greengrass. Defina o valor para a versão do núcleo que você deseja baixar.

Definição de variáveis de ambiente

As variáveis de ambiente permitem que você personalize como o software AWS IoT Greengrass Core é instalado no contêiner Docker. Você pode definir variáveis de ambiente para sua imagem do AWS IoT Greengrass Docker de várias maneiras.

- Para usar as mesmas variáveis de ambiente para criar várias imagens, defina as variáveis de ambiente diretamente no Dockerfile.
- Se você usa `docker run` para iniciar seu contêiner, passe variáveis de ambiente como argumentos no comando ou defina variáveis de ambiente em um arquivo de variáveis de ambiente e, em seguida, passe o arquivo como argumento. Para obter mais informações sobre a configuração de variáveis de ambiente no Docker, consulte as variáveis de ambiente na documentação do Docker.
- Se você usa `docker-compose up` para iniciar seu contêiner, defina as variáveis de ambiente em um arquivo de variáveis de ambiente e, em seguida, passe o arquivo como argumento. Para obter mais informações sobre a configuração de variáveis de ambiente no Compose, consulte a documentação do Docker.

Você pode configurar as seguintes variáveis de ambiente para a imagem do AWS IoT Greengrass Docker.

> **Note**
>
> Não modifique a `TINI_KILL_PROCESS_GROUP` variável no Dockerfile. Essa variável permite o encaminhamento `SIGTERM` para todos os PIDs no grupo PID para que o software AWS IoT

Greengrass principal possa ser desligado corretamente quando o contêiner do Docker for interrompido.

`GGC_ROOT_PATH`

(Opcional) O caminho para a pasta dentro do contêiner a ser usada como raiz do software AWS IoT Greengrass Core.

Padrão: `/greengrass/v2`

`PROVISION`

(Opcional) Determina se o AWS IoT Greengrass Core provisiona AWS recursos.

- Se você especificar`true`, o software AWS IoT Greengrass Core registra a imagem do contêiner como uma AWS IoT coisa e provisiona os AWS recursos que o dispositivo principal do Greengrass exige. O software AWS IoT Greengrass Core provisiona AWS IoT qualquer coisa, (opcional) qualquer grupo de AWS IoT coisas, uma função do IAM e um alias de AWS IoT função. Para ter mais informações, consulte Execute AWS IoT Greengrass em um contêiner Docker com provisionamento automático de recursos.
- Se você especificar`false`, deverá criar um arquivo de configuração para fornecer ao instalador AWS IoT Greengrass principal que especifique o uso dos AWS recursos e certificados que você criou manualmente. Para ter mais informações, consulte Execute AWS IoT Greengrass em um contêiner Docker com provisionamento manual de recursos.

Padrão: `false`

`AWS_REGION`

(Opcional) O Região da AWS que o software AWS IoT Greengrass Core usa para recuperar ou criar AWS os recursos necessários.

Padrão: `us-east-1`.

`THING_NAME`

(Opcional) O nome da AWS IoT coisa que você registra como esse dispositivo principal. Se a coisa com esse nome não existir no seuConta da AWS, o software AWS IoT Greengrass Core a cria.

Você deve especificar `PROVISION=true` para aplicar esse argumento.

Padrão: `GreengrassV2IotThing_` mais um UUID aleatório.

`THING_GROUP_NAME`

(Opcional) O nome do grupo ao AWS IoT qual você adiciona esse dispositivo principal é AWS IoT Se uma implantação for direcionada a esse grupo, esse e outros dispositivos principais desse grupo receberão essa implantação quando se conectaremAWS IoT Greengrass. Se o grupo de coisas com esse nome não existir no seuConta da AWS, o software AWS IoT Greengrass Core o criará.

Você deve especificar `PROVISION=true` para aplicar esse argumento.

`TES_ROLE_NAME`

(Opcional) O nome da função do IAM a ser usada para adquirir AWS credenciais que permitem que o dispositivo principal do Greengrass interaja com os AWS serviços. Se a função com esse nome não existir na suaConta da AWS, o software AWS IoT Greengrass Core a criará com a `GreengrassV2TokenExchangeRoleAccess` política. Essa função não tem acesso aos buckets do S3 nos quais você hospeda artefatos de componentes. Portanto, você deve adicionar permissões aos buckets e objetos do S3 dos seus artefatos ao criar um componente. Para ter mais informações, consulte Autorize os dispositivos principais a interagir com os serviços AWS.

Padrão: `GreengrassV2TokenExchangeRole`

`TES_ROLE_ALIAS_NAME`

(Opcional) O nome do alias da AWS IoT função que aponta para a função do IAM que fornece AWS credenciais para o dispositivo principal do Greengrass. Se o alias de função com esse nome não existir no seuConta da AWS, o software AWS IoT Greengrass Core o criará e o direcionará para a função do IAM que você especificar.

Padrão: `GreengrassV2TokenExchangeRoleAlias`

`COMPONENT_DEFAULT_USER`

(Opcional) O nome ou ID do usuário e do grupo do sistema que o software AWS IoT Greengrass Core usa para executar componentes. Especifique o usuário e o grupo, separados por dois pontos. O grupo é opcional. Por exemplo, é possível especificar **`ggc_user:ggc_group`** ou **`ggc_user`**.

- Se você executar como root, o padrão é o usuário e o grupo definidos pelo arquivo de configuração. Se o arquivo de configuração não definir um usuário e um grupo, o padrão é. `ggc_user:ggc_group` Se existirem `ggc_user` ou `ggc_group` não, o software os cria.

- Se você executa como usuário não root, o software AWS IoT Greengrass Core usa esse usuário para executar componentes.
- Se você não especificar um grupo, o software AWS IoT Greengrass Core usa o grupo primário do usuário do sistema.

Para ter mais informações, consulte Configurar o usuário que executa os componentes.

`DEPLOY_DEV_TOOLS`

Define se o componente CLI do Greengrass deve ser baixado e implantado na imagem do contêiner. Você pode usar a CLI do Greengrass para desenvolver e depurar componentes localmente.

> **Important**
>
> Recomendamos que você use esse componente somente em ambientes de desenvolvimento, não em ambientes de produção. Esse componente fornece acesso a informações e operações que você normalmente não precisará em um ambiente de produção. Siga o princípio do menor privilégio implantando esse componente somente nos dispositivos principais onde você precisar dele.

Padrão: `false`

`INIT_CONFIG`

(Opcional) O caminho para o arquivo de configuração a ser usado para instalar o software AWS IoT Greengrass Core. Você pode usar essa opção para configurar novos dispositivos principais do Greengrass com uma configuração de núcleo específica ou para especificar recursos provisionados manualmente, por exemplo. Você deve montar seu arquivo de configuração no caminho especificado nesse argumento.

`TRUSTED_PLUGIN`

Esse recurso está disponível para a versão 2.4.0 e posterior do componente núcleo do Greengrass.

(Opcional) O caminho para um arquivo JAR a ser carregado como um plug-in confiável. Use essa opção para fornecer arquivos JAR do plug-in de provisionamento, como para instalação com provisionamento de frota ou provisionamento personalizado.

`THING_POLICY_NAME`

Esse recurso está disponível para a versão 2.4.0 e posterior do componente núcleo do Greengrass.

(Opcional) O nome da AWS IoT política a ser anexada ao certificado de AWS IoT coisas desse dispositivo principal. Se a AWS IoT política com esse nome não existir em sua, Conta da AWS o software AWS IoT Greengrass Core a criará.

Você deve especificar `PROVISION=true` para aplicar esse argumento.

> **Note**
>
> O software AWS IoT Greengrass Core cria uma AWS IoT política permissiva por padrão. Você pode definir o escopo dessa política ou criar uma política personalizada na qual restrinja as permissões para seu caso de uso. Para ter mais informações, consulte AWS IoTPolítica mínima para dispositivos AWS IoT Greengrass V2 principais.

Especifique as dependências a serem instaladas

A instrução RUN no AWS IoT Greengrass Dockerfile prepara o ambiente do contêiner para executar o AWS IoT Greengrass instalador do software Core. Você pode personalizar as dependências que são instaladas antes da execução do instalador do software AWS IoT Greengrass Core no contêiner do Docker.

Crie a AWS IoT Greengrass imagem

Use o AWS IoT Greengrass Dockerfile para criar uma imagem de AWS IoT Greengrass contêiner. Você pode usar a CLI do Docker ou a CLI do Docker Compose para criar a imagem e iniciar o contêiner. Você também pode usar a CLI do Docker para criar a imagem e, em seguida, usar o Docker Compose para iniciar seu contêiner a partir dessa imagem.

Docker

1. Na máquina host, execute o comando a seguir para alternar para o diretório que contém o Dockerfile configurado.

   ```
   cd download-directory/aws-greengrass-docker-nucleus-version
   ```

2. Execute o comando a seguir para criar a imagem do AWS IoT Greengrass contêiner a partir do Dockerfile.

```
sudo docker build -t "platform/aws-iot-greengrass:nucleus-version" ./
```

Docker Compose

1. Na máquina host, execute o comando a seguir para alternar para o diretório que contém o Dockerfile e o arquivo Compose.

```
cd download-directory/aws-greengrass-docker-nucleus-version
```

2. Execute o comando a seguir para usar o arquivo Compose para criar a imagem do AWS IoT Greengrass contêiner.

```
docker-compose -f docker-compose.yml build
```

Você criou com sucesso a imagem do AWS IoT Greengrass contêiner. A imagem do Docker tem o software AWS IoT Greengrass Core instalado. Agora você pode executar o software AWS IoT Greengrass Core em um contêiner Docker.

Execute AWS IoT Greengrass em um contêiner Docker com provisionamento automático de recursos

Este tutorial mostra como instalar e executar o software AWS IoT Greengrass Core em um contêiner Docker com AWS recursos provisionados automaticamente e ferramentas de desenvolvimento local. Você pode usar esse ambiente de desenvolvimento para explorar os AWS IoT Greengrass recursos em um contêiner do Docker. O software requer AWS credenciais para provisionar esses recursos e implantar as ferramentas de desenvolvimento local.

Se você não puder fornecer AWS credenciais para o contêiner, poderá provisionar os AWS recursos que o dispositivo principal precisa para operar. Você também pode implantar as ferramentas de desenvolvimento em um dispositivo principal para usar como dispositivo de desenvolvimento. Isso permite que você forneça menos permissões ao dispositivo ao executar o contêiner. Para ter mais informações, consulte Execute AWS IoT Greengrass em um contêiner Docker com provisionamento manual de recursos.

Pré-requisitos

Para concluir este tutorial, você precisa do seguinte.

- Uma Conta da AWS. Se você não tiver uma, consulte Configurar um Conta da AWS.
- Um usuário AWS do IAM com permissões para provisionar os recursos do IAM AWS IoT e do IAM para um dispositivo principal do Greengrass. O instalador do software AWS IoT Greengrass Core usa suas AWS credenciais para provisionar automaticamente esses recursos. Para obter informações sobre a política mínima do IAM para provisionar recursos automaticamente, consultePolítica mínima de IAM para o instalador provisionar recursos.
- Uma imagem AWS IoT Greengrass do Docker. Você pode criar uma imagem a partir do AWS IoT Greengrass Dockerfile.
- O computador host em que você executa o contêiner Docker deve atender aos seguintes requisitos:
  - Um sistema operacional baseado em Linux com conexão à Internet.
  - Docker Engine versão 18.09 ou posterior.
  - (Opcional) Docker Compose versão 1.22 ou posterior. O Docker Compose é necessário somente se você quiser usar a CLI do Docker Compose para executar suas imagens do Docker.

Configurar as credenciais da AWS

Nesta etapa, você cria um arquivo de credencial no computador host que contém suas credenciais AWS de segurança. Ao executar a imagem do AWS IoT Greengrass Docker, você deve montar a pasta que contém esse arquivo de credencial `/root/.aws/` no contêiner do Docker. O AWS IoT Greengrass instalador usa essas credenciais para provisionar recursos no seuConta da AWS. Para obter informações sobre a política mínima do IAM que o instalador exige para provisionar recursos automaticamente, consultePolítica mínima de IAM para o instalador provisionar recursos.

1. Recupere um dos seguintes.
   - Credenciais de longo prazo para um usuário do IAM. Para obter informações sobre como recuperar credenciais de longo prazo, consulte Gerenciamento de chaves de acesso para usuários do IAM no Guia do usuário do IAM.
   - (Recomendado) Credenciais temporárias para uma função do IAM. Para obter informações sobre como recuperar credenciais temporárias, consulte Como usar credenciais de segurança temporárias com o AWS CLI no Guia do usuário do IAM.

2. Crie uma pasta onde você coloca seu arquivo de credencial.

```
mkdir ./greengrass-v2-credentials
```

3. Use um editor de texto para criar um arquivo de configuração nomeado `credentials` na `./greengrass-v2-credentials` pasta.

   Por exemplo, você pode executar o comando a seguir para usar o GNU nano para criar o `credentials` arquivo.

```
nano ./greengrass-v2-credentials/credentials
```

4. Adicione suas AWS credenciais ao `credentials` arquivo no formato a seguir.

```
[default]
aws_access_key_id = AKIAIOSFODNN7EXAMPLE
aws_secret_access_key = wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
aws_session_token
 = AQoEXAMPLEH4aoAH0gNCAPy...truncated...zrkuWJOgQs8IZZaIv2BXIa2R4Olgk
```

   `aws_session_token`Inclua somente para credenciais temporárias.

> **Important**
>
> Remova o arquivo de credencial do computador host depois de iniciar o AWS IoT Greengrass contêiner. Se você não remover o arquivo de credenciais, suas AWS credenciais permanecerão montadas dentro do contêiner. Para ter mais informações, consulte Execute o software AWS IoT Greengrass principal em um contêiner.

Criar um arquivo de ambiente

Este tutorial usa um arquivo de ambiente para definir as variáveis de ambiente que serão passadas para o instalador do software AWS IoT Greengrass Core dentro do contêiner Docker. Você também pode usar o `--env` argumento `-e` ou em seu `docker run` comando para definir variáveis de ambiente no contêiner do Docker ou definir as variáveis em um `environment` bloco no `docker-compose.yml` arquivo.

1. Use um editor de texto para criar um arquivo de ambiente chamado `.env`.

Por exemplo, em um sistema baseado em Linux, você pode executar o seguinte comando para usar o GNU nano para criar o no `.env` diretório atual.

```
nano .env
```

2. Copie o conteúdo a seguir no arquivo.

```
GGC_ROOT_PATH=/greengrass/v2
AWS_REGION=region
PROVISION=true
THING_NAME=MyGreengrassCore
THING_GROUP_NAME=MyGreengrassCoreGroup
TES_ROLE_NAME=GreengrassV2TokenExchangeRole
TES_ROLE_ALIAS_NAME=GreengrassCoreTokenExchangeRoleAlias
COMPONENT_DEFAULT_USER=ggc_user:ggc_group
```

Em seguida, substitua os valores a seguir.

- */greengrass/v2*. A pasta raiz do Greengrass que você deseja usar para instalação. Você usa a variável de `GGC_ROOT` ambiente para definir esse valor.
- *região*. O Região da AWS local onde você criou os recursos.
- *MyGreengrassCore*. O nome da coisa da AWS IoT. Se a coisa não existir, o instalador a cria. O instalador baixa os certificados para autenticar a AWS IoT coisa.
- *MyGreengrassCoreGroup*. O nome do grupo de AWS IoT coisas. Se o grupo de coisas não existir, o instalador o cria e adiciona a coisa a ele. Se o grupo de coisas existir e tiver uma implantação ativa, o dispositivo principal baixará e executará o software especificado pela implantação.
- *Greengrass TokenExchangeRole* V2. Substitua pelo nome da função de troca de tokens do IAM que permite que o dispositivo principal do Greengrass obtenha credenciais temporáriasAWS. Se a função não existir, o instalador a cria, cria e anexa uma política chamada *TokenExchangeRoleGreengrassV2* Access. Para ter mais informações, consulte Autorize os dispositivos principais a interagir com os serviços AWS.
- *GreengrassCoreTokenExchangeRoleAlias*. O alias da função de troca de tokens. Se o alias da função não existir, o instalador o cria e o direciona para a função de troca de tokens do IAM que você especifica. Para obter mais informações, consulte

> **Note**
>
> Você pode definir a variável de `DEPLOY_DEV_TOOLS` ambiente `true` para implantar o componente CLI do Greengrass, que permite desenvolver componentes personalizados dentro do contêiner do Docker. Recomendamos que você use esse componente somente em ambientes de desenvolvimento, não em ambientes de produção. Esse componente fornece acesso a informações e operações que você normalmente não precisará em um ambiente de produção. Siga o princípio do menor privilégio implantando esse componente somente nos dispositivos principais onde você precisar dele.

Execute o software AWS IoT Greengrass principal em um contêiner

Este tutorial mostra como iniciar a imagem do Docker que você criou em um contêiner do Docker. Você pode usar a CLI do Docker ou a CLI do Docker Compose para AWS IoT Greengrass executar a imagem do software Core em um contêiner do Docker.

Docker

1. Execute o comando a seguir para iniciar o contêiner Docker.

   ```
   docker run --rm --init -it --name docker-image \
    -v path/to/greengrass-v2-credentials:/root/.aws/:ro \
    --env-file .env \
    -p 8883 \
    your-container-image:version
   ```

   Esse exemplo de comando usa os seguintes argumentos para docker run:

   - `--rm`. Limpa o contêiner quando ele sai.
   - `--init`. Usa um processo de inicialização no contêiner.

     > **Note**
     >
     > O `--init` argumento é necessário para desligar o software AWS IoT Greengrass Core quando você interrompe o contêiner Docker.

- `-it`. (Opcional) Executa o contêiner Docker em primeiro plano como um processo interativo. Em vez disso, você pode substituir isso pelo `-d` argumento para executar o contêiner do Docker no modo desanexado. Para obter mais informações, consulte Desanexado versus primeiro plano na documentação do Docker.
- `--name`. Executa um contêiner chamado `aws-iot-greengrass`
- `-v`. Monta um volume no contêiner do Docker para disponibilizar o arquivo de configuração e os arquivos de certificado para AWS IoT Greengrass execução dentro do contêiner.
- `--env-file`. (Opcional) Especifica o arquivo de ambiente para definir as variáveis de ambiente que serão passadas para o instalador do software AWS IoT Greengrass Core dentro do contêiner Docker. Esse argumento é necessário somente se você criou um arquivo de ambiente para definir variáveis de ambiente. Se você não criou um arquivo de ambiente, pode usar `--env` argumentos para definir variáveis de ambiente diretamente no comando de execução do Docker.
- `-p`. (Opcional) Publica a porta do contêiner 8883 na máquina host. Esse argumento é necessário se você quiser se conectar e se comunicar pelo MQTT porque AWS IoT Greengrass usa a porta 8883 para tráfego MQTT. Para abrir outras portas, use `-p` argumentos adicionais.

> **Note**
>
> Para executar seu contêiner Docker com maior segurança, você pode usar os `--cap-add` argumentos `--cap-drop` e para habilitar seletivamente os recursos do Linux para seu contêiner. Para obter mais informações, consulte Privilégio de tempo de execução e recursos do Linux na documentação do Docker.

2. Remova as credenciais `./greengrass-v2-credentials` do dispositivo host.

```
rm -rf ./greengrass-v2-credentials
```

> **Important**
>
> Você está removendo essas credenciais porque elas fornecem amplas permissões que o dispositivo principal precisa somente durante a configuração. Se você não remover essas credenciais, os componentes do Greengrass e outros processos em execução no contêiner poderão acessá-las. Se você precisar fornecer AWS

credenciais para um componente do Greengrass, use o serviço de troca de tokens. Para ter mais informações, consulte Interaja com AWS os serviços.

Docker Compose

1. Use um editor de texto para criar um arquivo Docker Compose chamado. `docker-compose.yml`

   Por exemplo, em um sistema baseado em Linux, você pode executar o seguinte comando para usar o GNU nano para criar o no `docker-compose.yml` diretório atual.

   ```
   nano docker-compose.yml
   ```

   > **Note**
   >
   > Você também pode baixar e usar a versão mais recente do arquivo Compose AWS fornecido em. GitHub

2. Adicione o conteúdo a seguir ao arquivo Compose. O arquivo deve ser semelhante ao exemplo a seguir. Substitua *`docker-image pelo`* nome da sua imagem Docker.

   ```
   version: '3.7'

   services:
     greengrass:
       init: true
       container_name: aws-iot-greengrass
       image: docker-image
       volumes:
         - ./greengrass-v2-credentials:/root/.aws/:ro
       env_file: .env
       ports:
         - "8883:8883"
   ```

   Os seguintes parâmetros neste exemplo de arquivo Compose são opcionais:

- `ports`—Publica as portas do contêiner 8883 na máquina host. Esse parâmetro é necessário se você quiser se conectar e se comunicar pelo MQTT porque AWS IoT Greengrass usa a porta 8883 para tráfego MQTT.
- `env_file`—Especifica o arquivo de ambiente para definir as variáveis de ambiente que serão passadas para o instalador do software AWS IoT Greengrass Core dentro do contêiner Docker. Esse parâmetro é necessário somente se você criou um arquivo de ambiente para definir variáveis de ambiente. Se você não criou um arquivo de ambiente, pode usar o parâmetro de ambiente para definir as variáveis diretamente no seu arquivo Compose.

> **Note**
>
> Para executar seu contêiner Docker com maior segurança, você pode usar `cap_drop` e `cap_add` em seu arquivo Compose para habilitar seletivamente os recursos do Linux para seu contêiner. Para obter mais informações, consulte Privilégio de tempo de execução e recursos do Linux na documentação do Docker.

3. Execute o comando a seguir para iniciar o contêiner Docker.

```
docker-compose -f docker-compose.yml up
```

4. Remova as credenciais `./greengrass-v2-credentials` do dispositivo host.

```
rm -rf ./greengrass-v2-credentials
```

> **Important**
>
> Você está removendo essas credenciais porque elas fornecem amplas permissões que o dispositivo principal precisa somente durante a configuração. Se você não remover essas credenciais, os componentes do Greengrass e outros processos em execução no contêiner poderão acessá-las. Se você precisar fornecer AWS credenciais para um componente do Greengrass, use o serviço de troca de tokens. Para ter mais informações, consulte Interaja com AWS os serviços.

Próximas etapas

AWS IoT GreengrassO software principal agora está sendo executado em um contêiner Docker. Execute o comando a seguir para recuperar o ID do contêiner em execução no momento.

```
docker ps
```

Em seguida, você pode executar o comando a seguir para acessar o contêiner e explorar o software AWS IoT Greengrass principal executado dentro do contêiner.

```
docker exec -it container-id /bin/bash
```

Para obter informações sobre como criar um componente simples, consulte Etapa 4: desenvolver e testar um componente em seu dispositivo em Tutorial: Conceitos básicos do AWS IoT Greengrass V2

> **Note**
>
> Quando você usa `docker exec` para executar comandos dentro do contêiner do Docker, esses comandos não são registrados nos registros do Docker. Para registrar seus comandos nos registros do Docker, anexe um shell interativo ao contêiner do Docker. Para ter mais informações, consulte Anexe um shell interativo ao contêiner Docker.

O arquivo de log AWS IoT Greengrass principal é chamado `greengrass.log` e está localizado em*/greengrass/v2*`/logs`. Os arquivos de log de componentes também estão localizados no mesmo diretório. Para copiar os registros do Greengrass para um diretório temporário no host, execute o seguinte comando:

```
docker cp container-id:/greengrass/v2/logs /tmp/logs
```

Se você quiser manter os registros após a saída ou remoção de um contêiner, recomendamos que você vincule e monte somente o */greengrass/v2*`/logs` diretório no diretório de registros temporários no host, em vez de montar todo o diretório Greengrass. Para ter mais informações, consulte Persiste os registros do Greengrass fora do contêiner Docker.

Para interromper a execução de um contêiner AWS IoT Greengrass Docker, execute `docker stop` o`docker-compose -f docker-compose.yml stop`. Essa ação é enviada `SIGTERM`

para o processo do Greengrass e encerra todos os processos associados que foram iniciados no contêiner. O contêiner Docker é inicializado com o `docker-init` executável como processo PID 1, o que ajuda a remover qualquer processo zumbi restante. Para obter mais informações, consulte Especificar um processo de inicialização na documentação do Docker.

Para obter informações sobre como solucionar problemas com a execução AWS IoT Greengrass em um contêiner Docker, consulteSolução de problemas do AWS IoT Greengrass em um contêiner do Docker.

Execute AWS IoT Greengrass em um contêiner Docker com provisionamento manual de recursos

Este tutorial mostra como instalar e executar o software AWS IoT Greengrass Core no contêiner Docker com recursos provisionados AWS manualmente.

Tópicos

Pré-requisitos

Para concluir este tutorial, você precisará do seguinte:

- Uma Conta da AWS. Se você não tiver uma, consulte Configurar um Conta da AWS.
- Uma imagem AWS IoT Greengrass do Docker. Você pode criar uma imagem a partir do AWS IoT Greengrass Dockerfile.

- O computador host em que você executa o contêiner Docker deve atender aos seguintes requisitos:
  - Um sistema operacional baseado em Linux com conexão à Internet.
  - Docker Engine versão 18.09 ou posterior.
  - (Opcional) Docker Compose versão 1.22 ou posterior. O Docker Compose é necessário somente se você quiser usar a CLI do Docker Compose para executar suas imagens do Docker.

Recupere endpoints AWS IoT

Obtenha os AWS IoT endpoints para você Conta da AWS e salve-os para usar mais tarde. Seu dispositivo usa esses endpoints para se conectar a. AWS IoT Faça o seguinte:

1. Obtenha o endpoint de AWS IoT dados para vocêConta da AWS.

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

   A resposta será semelhante ao exemplo a seguir, se a solicitação for bem-sucedida.

   ```
   {
     "endpointAddress": "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
   }
   ```

2. Obtenha o endpoint AWS IoT de credenciais para seu. Conta da AWS

   ```
   aws iot describe-endpoint --endpoint-type iot:CredentialProvider
   ```

   A resposta será semelhante ao exemplo a seguir, se a solicitação for bem-sucedida.

   ```
   {
     "endpointAddress": "device-credentials-prefix.credentials.iot.us-
   west-2.amazonaws.com"
   }
   ```

Criação de uma coisa do AWS IoT

AWS IoTas coisas representam dispositivos e entidades lógicas que se conectam AWS IoT a. Os principais dispositivos do Greengrass são AWS IoT coisas. Quando você registra um dispositivo como uma AWS IoT coisa, esse dispositivo pode usar um certificado digital para se autenticarAWS.

Nesta seção, você cria AWS IoT algo que representa seu dispositivo.

Como criar uma coisa AWS IoT

1. Crie qualquer AWS IoT coisa para o seu dispositivo. No seu computador de desenvolvimento, execute o comando a seguir.

   - *MyGreengrassCore*Substitua pelo nome da coisa a ser usada. Esse nome também é o nome do seu dispositivo principal do Greengrass.

     > **Note**
     >
     > O nome da coisa não pode conter caracteres de dois pontos (`:`).

   ```
   aws iot create-thing --thing-name MyGreengrassCore
   ```

   A resposta será semelhante ao exemplo a seguir, se a solicitação for bem-sucedida.

   ```
   {
     "thingName": "MyGreengrassCore",
     "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
     "thingId": "8cb4b6cd-268e-495d-b5b9-1713d71dbf42"
   }
   ```

2. (Opcional) Adicione a AWS IoT coisa a um grupo de coisas novo ou existente. Você usa grupos de coisas para gerenciar frotas de dispositivos principais do Greengrass. Ao implantar componentes de software em seus dispositivos, você pode segmentar dispositivos individuais ou grupos de dispositivos. Você pode adicionar um dispositivo a um grupo de coisas com uma implantação ativa do Greengrass para implantar os componentes de software desse grupo de coisas no dispositivo. Faça o seguinte:

   a. (Opcional) Crie um grupo de AWS IoT coisas.

- *MyGreengrassCoreGroup*Substitua pelo nome do grupo de coisas a ser criado.

> **Note**
>
> O nome do grupo de coisas não pode conter caracteres de dois pontos (`:`).

```
aws iot create-thing-group --thing-group-name MyGreengrassCoreGroup
```

A resposta será semelhante ao exemplo a seguir, se a solicitação for bem-sucedida.

```
{
  "thingGroupName": "MyGreengrassCoreGroup",
  "thingGroupArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/
MyGreengrassCoreGroup",
  "thingGroupId": "4df721e1-ff9f-4f97-92dd-02db4e3f03aa"
}
```

b. Adicione a AWS IoT coisa a um grupo de coisas.

- *MyGreengrassCore*Substitua pelo nome da sua AWS IoT coisa.
- *MyGreengrassCoreGroup*Substitua pelo nome do grupo de coisas.

```
aws iot add-thing-to-thing-group --thing-name MyGreengrassCore --thing-group-
name MyGreengrassCoreGroup
```

O comando não terá nenhuma saída se a solicitação for bem-sucedida.

Crie o certificado da coisa

Quando você registra um dispositivo como uma AWS IoT coisa, esse dispositivo pode usar um certificado digital para se autenticarAWS. Esse certificado permite que o dispositivo se comunique com AWS IoT AWS IoT Greengrass e.

Nesta seção, você cria e baixa certificados que seu dispositivo pode usar para se conectarAWS.

Para criar o certificado da coisa

1. Crie uma pasta na qual você baixa os certificados da AWS IoT coisa.

```
mkdir greengrass-v2-certs
```

2. Crie e baixe os certificados da AWS IoT coisa.

```
aws iot create-keys-and-certificate --set-as-active --certificate-pem-outfile
 greengrass-v2-certs/device.pem.crt --public-key-outfile greengrass-v2-certs/
public.pem.key --private-key-outfile greengrass-v2-certs/private.pem.key
```

A resposta será semelhante ao exemplo a seguir, se a solicitação for bem-sucedida.

```
{
  "certificateArn": "arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
  "certificateId":
 "aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4",
  "certificatePem": "-----BEGIN CERTIFICATE-----
MIICiTCCAfICCQD6m7oRw0uXOjANBgkqhkiG9w
 0BAQUFADCBiDELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZ
 WF0dGxlMQ8wDQYDVQQKEwZBbWF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIw
 EAYDVQQDEwlUZXN0Q2lsYWMxHzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5
 jb20wHhcNMTEwNDI1MjA0NTIxWhcNMTIwNDI0MjA0NTIxWjCBiDELMAkGA1UEBh
 MCVVMxCzAJBgNVBAgTAldBMRAwDgYDVQQHEwdTZWF0dGxlMQ8wDQYDVQQKEwZBb
 WF6b24xFDASBgNVBAsTC0lBTSBDb25zb2xlMRIwEAYDVQQDEwlUZXN0Q2lsYWMx
 HzAdBgkqhkiG9w0BCQEWEG5vb25lQGFtYXpvbi5jb20wgZ8wDQYJKoZIhvcNAQE
 BBQADgY0AMIGJAoGBAMaK0dn+a4GmWIWJ21uUSfwfEvySWtC2XADZ4nB+BLYgVI
 k60CpiwsZ3G93vUEIO3IyNoH/f0wYK8m9TrDHudUZg3qX4waLG5M43q7Wgc/MbQ
 ITxOUSQv7c7ugFFDzQGBzZswY6786m86gpEIbb3OhjZnzcvQAaRHhdlQWIMm2nr
 AgMBAAEwDQYJKoZIhvcNAQEFBQADgYEAtCu4nUhVVxYUntneD9+h8Mg9q6q+auN
 KyExzyLwaxlAoo7TJHidbtS4J5iNmZgXL0FkbFFBjvSfpJIlJ00zbhNYS5f6Guo
 EDmFJl0ZxBHjJnyp378OD8uTs7fLvjx79LjSTbNYiytVbZPQUQ5Yaxu2jXnimvw
 3rrszlaEXAMPLE=
-----END CERTIFICATE-----",
  "keyPair": {
    "PublicKey": "-----BEGIN PUBLIC KEY-----\
MIIBIjANBgkqhkEXAMPLEQEFAAOCAQ8AMIIBCgKCAQEAEXAMPLE1nnyJwKSMHw4h\
MMEXAMPLEuuN/dMAS3fyce8DW/4+EXAMPLEyjmoF/YVF/gHr99VEEXAMPLE5VF13\
```

```
59VK7cEXAMPLE67GK+y+jikqXOgHh/xJTwo
+sGpWEXAMPLEDz18xOd2ka4tCzuWEXAMPLEahJbYkCPUBSU8opVkR7qkEXAMPLE1DR6sx2HocliOOLtu6Fkw91swQWE
\\GB3ZPrNh0PzQYvjUStZeccyNCx2EXAMPLEvp9mQOUXP6plfgxwKRX2fEXAMPLEDa\
hJLXkX3rHU2xbxJSq7D+XEXAMPLEcw+LyFhI5mgFRl88eGdsAEXAMPLElnI9EesG\
FQIDAQAB\
-----END PUBLIC KEY-----\
",
    "PrivateKey": "-----BEGIN RSA PRIVATE KEY-----\
key omitted for security reasons\
-----END RSA PRIVATE KEY-----\
"
  }
}
```

Salve o Amazon Resource Name (ARN) do certificado para usar na configuração posterior do certificado.

Configure o certificado da coisa

Anexe o certificado do AWS IoT item ao que você criou anteriormente e adicione uma AWS IoT política ao certificado para definir as AWS IoT permissões para o dispositivo principal.

Para configurar o certificado da coisa

1. Anexe o certificado à AWS IoT coisa.

   - *MyGreengrassCore*Substitua pelo nome da sua AWS IoT coisa.
   - Substitua o certificado Amazon Resource Name (ARN) pelo ARN do certificado que você criou na etapa anterior.

   ```
   aws iot attach-thing-principal --thing-name MyGreengrassCore
    --principal arn:aws:iot:us-west-2:123456789012:cert/
   aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
   ```

   O comando não terá nenhuma saída se a solicitação for bem-sucedida.
2. Crie e anexe uma AWS IoT política que defina as AWS IoT permissões para seu dispositivo principal do Greengrass. A política a seguir permite acesso a todos os tópicos do MQTT e operações do Greengrass, para que seu dispositivo funcione com aplicativos personalizados e futuras alterações que exijam novas operações do Greengrass. Você pode restringir essa

política com base no seu caso de uso. Para ter mais informações, consulte AWS IoTPolítica mínima para dispositivos AWS IoT Greengrass V2 principais.

Se você já configurou um dispositivo principal do Greengrass, pode anexar sua AWS IoT política em vez de criar uma nova.

Faça o seguinte:

a. Crie um arquivo que contenha o documento AWS IoT de política exigido pelos dispositivos principais do Greengrass.

   Por exemplo, em um sistema baseado em Linux, você pode executar o seguinte comando para usar o GNU nano para criar o arquivo.

   ```
   nano greengrass-v2-iot-policy.json
   ```

   Copie o seguinte JSON no arquivo.

   ```
   {
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Action": [
           "iot:Publish",
           "iot:Subscribe",
           "iot:Receive",
           "iot:Connect",
           "greengrass:*"
         ],
         "Resource": [
           "*"
         ]
       }
     ]
   }
   ```

b. Crie uma AWS IoT política a partir do documento de política.

   - Substitua *GreengrassV2IoT* pelo nome da ThingPolicy política a ser criada.

```
aws iot create-policy --policy-name GreengrassV2IoTThingPolicy --policy-
document file://greengrass-v2-iot-policy.json
```

A resposta será semelhante ao exemplo a seguir, se a solicitação for bem-sucedida.

```
{
  "policyName": "GreengrassV2IoTThingPolicy",
  "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassV2IoTThingPolicy",
  "policyDocument": "{
    \\"Version\\": \\"2012-10-17\\",
    \\"Statement\\": [
      {
        \\"Effect\\": \\"Allow\\",
        \\"Action\\": [
          \\"iot:Publish\\",
          \\"iot:Subscribe\\",
          \\"iot:Receive\\",
          \\"iot:Connect\\",
          \\"greengrass:*\\"
        ],
        \\"Resource\\": [
          \\"*\\"
        ]
      }
    ]
  }",
  "policyVersionId": "1"
}
```

c. Anexe a AWS IoT política ao certificado da AWS IoT coisa.

- Substitua *GreengrassV2IoT pelo nome da política ThingPolicy a ser* anexada.
- Substitua o ARN de destino pelo ARN do certificado de sua coisa. AWS IoT

```
aws iot attach-policy --policy-name GreengrassV2IoTThingPolicy
 --target arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
```

O comando não terá nenhuma saída se a solicitação for bem-sucedida.

Crie uma função de troca de tokens

Os dispositivos principais do Greengrass usam uma função de serviço do IAM, chamada função de troca de tokens, para autorizar chamadas para serviços. AWS O dispositivo usa o provedor de AWS IoT credenciais para obter AWS credenciais temporárias para essa função, o que permite que o dispositivo interajaAWS IoT, envie registros para o Amazon CloudWatch Logs e baixe artefatos de componentes personalizados do Amazon S3. Para ter mais informações, consulte Autorize os dispositivos principais a interagir com os serviços AWS.

Você usa um alias de AWS IoT função para configurar a função de troca de tokens para os dispositivos principais do Greengrass. Os aliases de função permitem que você altere a função de troca de tokens de um dispositivo, mas mantenha a mesma configuração do dispositivo. Para obter mais informações, consulte Autorização de chamadas diretas para AWS serviços no Guia do AWS IoT Core desenvolvedor.

Nesta seção, você cria uma função do IAM de troca de tokens e um alias de AWS IoT função que aponta para a função. Se você já configurou um dispositivo principal do Greengrass, pode usar sua função de troca de tokens e seu alias de função em vez de criar novos. Em seguida, você configura o dispositivo AWS IoT para usar essa função e alias.

Para criar uma função do IAM de troca de tokens

1. Crie uma função do IAM que seu dispositivo possa usar como função de troca de tokens. Faça o seguinte:

   a. Crie um arquivo que contenha o documento de política de confiança exigido pela função de troca de tokens.

      Por exemplo, em um sistema baseado em Linux, você pode executar o seguinte comando para usar o GNU nano para criar o arquivo.

      ```
      nano device-role-trust-policy.json
      ```

      Copie o seguinte JSON no arquivo.

      ```
      {
      ```

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.iot.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

b. Crie a função de troca de tokens com o documento de política de confiança.

- Substitua *GreengrassV2 TokenExchangeRole* pelo nome da função do IAM a ser criada.

```
aws iam create-role --role-name GreengrassV2TokenExchangeRole --assume-role-
policy-document file://device-role-trust-policy.json
```

A resposta será semelhante ao exemplo a seguir, se a solicitação for bem-sucedida.

```
{
  "Role": {
    "Path": "/",
    "RoleName": "GreengrassV2TokenExchangeRole",
    "RoleId": "AROAZ2YMUHYHK5OKM77FB",
    "Arn": "arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole",
    "CreateDate": "2021-02-06T00:13:29+00:00",
    "AssumeRolePolicyDocument": {
      "Version": "2012-10-17",
      "Statement": [
        {
          "Effect": "Allow",
          "Principal": {
            "Service": "credentials.iot.amazonaws.com"
          },
          "Action": "sts:AssumeRole"
        }
      ]
    }
```

```
  }
```

c. Crie um arquivo que contenha o documento de política de acesso exigido pela função de troca de tokens.

   Por exemplo, em um sistema baseado em Linux, você pode executar o seguinte comando para usar o GNU nano para criar o arquivo.

```
nano device-role-access-policy.json
```

   Copie o seguinte JSON no arquivo.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",
        "s3:GetBucketLocation"
      ],
      "Resource": "*"
    }
  ]
}
```

> **Note**
>
> Essa política de acesso não permite acesso a artefatos de componentes em buckets do S3. Para implantar componentes personalizados que definem artefatos no Amazon S3, você deve adicionar permissões à função para permitir que seu dispositivo principal recupere artefatos de componentes. Para ter mais informações, consulte Permitir acesso aos buckets do S3 para artefatos de componentes.
> Se você ainda não tem um bucket do S3 para artefatos de componentes, você pode adicionar essas permissões depois de criar um bucket.

d. Crie a política do IAM a partir do documento de política.

- Substitua *GreengrassV2 TokenExchangeRoleAccess* pelo nome da política do IAM a ser criada.

```
aws iam create-policy --policy-name GreengrassV2TokenExchangeRoleAccess --
policy-document file://device-role-access-policy.json
```

A resposta será semelhante ao exemplo a seguir, se a solicitação for bem-sucedida.

```
{
  "Policy": {
    "PolicyName": "GreengrassV2TokenExchangeRoleAccess",
    "PolicyId": "ANPAZ2YMUHYHACI7C5Z66",
    "Arn": "arn:aws:iam::123456789012:policy/
GreengrassV2TokenExchangeRoleAccess",
    "Path": "/",
    "DefaultVersionId": "v1",
    "AttachmentCount": 0,
    "PermissionsBoundaryUsageCount": 0,
    "IsAttachable": true,
    "CreateDate": "2021-02-06T00:37:17+00:00",
    "UpdateDate": "2021-02-06T00:37:17+00:00"
  }
}
```

e. Anexe a política do IAM à função de troca de tokens.

- Substitua *GreengrassV2 TokenExchangeRole* pelo nome da função do IAM.
- Substitua o ARN da política pelo ARN da política do IAM que você criou na etapa anterior.

```
aws iam attach-role-policy --role-name GreengrassV2TokenExchangeRole --policy-
arn arn:aws:iam::123456789012:policy/GreengrassV2TokenExchangeRoleAccess
```

O comando não terá nenhuma saída se a solicitação for bem-sucedida.

2. Crie um alias de AWS IoT função que aponte para a função de troca de tokens.

- *GreengrassCoreTokenExchangeRoleAlias*Substitua pelo nome do alias de função a ser criado.
- Substitua o ARN da função pelo ARN da função do IAM que você criou na etapa anterior.

```
aws iot create-role-alias --role-alias GreengrassCoreTokenExchangeRoleAlias --role-
arn arn:aws:iam::123456789012:role/GreengrassV2TokenExchangeRole
```

A resposta será semelhante ao exemplo a seguir, se a solicitação for bem-sucedida.

```
{
  "roleAlias": "GreengrassCoreTokenExchangeRoleAlias",
  "roleAliasArn": "arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias"
}
```

> **Note**
>
> Para criar um alias de função, você precisa ter permissão para passar a função do IAM de troca de tokens paraAWS IoT. Se você receber uma mensagem de erro ao tentar criar um alias de função, verifique se o AWS usuário tem essa permissão. Para obter mais informações, consulte Conceder permissões a um usuário para passar uma função para um AWS serviço no Guia do AWS Identity and Access Management usuário.

3. Crie e anexe uma AWS IoT política que permita que seu dispositivo principal do Greengrass use o alias de função para assumir a função de troca de tokens. Se você já configurou um dispositivo principal do Greengrass, pode anexar sua AWS IoT política de alias de função em vez de criar uma nova. Faça o seguinte:

   a. (Opcional) Crie um arquivo que contenha o documento AWS IoT de política exigido pelo alias da função.

      Por exemplo, em um sistema baseado em Linux, você pode executar o seguinte comando para usar o GNU nano para criar o arquivo.

      ```
      nano greengrass-v2-iot-role-alias-policy.json
      ```

      Copie o seguinte JSON no arquivo.

      - Substitua o ARN do recurso pelo ARN do seu alias de função.

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:AssumeRoleWithCertificate",
      "Resource": "arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias"
    }
  ]
}
```

b. Crie uma AWS IoT política a partir do documento de política.

- *GreengrassCoreTokenExchangeRoleAliasPolicy*Substitua pelo nome da AWS IoT política a ser criada.

```
aws iot create-policy --policy-name GreengrassCoreTokenExchangeRoleAliasPolicy
 --policy-document file://greengrass-v2-iot-role-alias-policy.json
```

A resposta será semelhante ao exemplo a seguir, se a solicitação for bem-sucedida.

```
{
  "policyName": "GreengrassCoreTokenExchangeRoleAliasPolicy",
  "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassCoreTokenExchangeRoleAliasPolicy",
  "policyDocument": "{
    \\"Version\\":\\"2012-10-17\\",
    \\"Statement\\": [
      {
        \\"Effect\\": \\"Allow\\",
        \\"Action\\": \\"iot:AssumeRoleWithCertificate\\",
        \\"Resource\\": \\"arn:aws:iot:us-west-2:123456789012:rolealias/
GreengrassCoreTokenExchangeRoleAlias\\"
      }
    ]
  }",
  "policyVersionId": "1"
}
```

c. Anexe a AWS IoT política ao certificado da AWS IoT coisa.

- *GreengrassCoreTokenExchangeRoleAliasPolicy*Substitua pelo nome da AWS IoT política de alias de função.
- Substitua o ARN de destino pelo ARN do certificado de sua coisa. AWS IoT

```
aws iot attach-policy --policy-name GreengrassCoreTokenExchangeRoleAliasPolicy
 --target arn:aws:iot:us-west-2:123456789012:cert/
aa0b7958770878eabe251d8a7ddd547f4889c524c9b574ab9fbf65f32248b1d4
```

O comando não terá nenhuma saída se a solicitação for bem-sucedida.

Baixe certificados para o dispositivo

Anteriormente, você baixou o certificado do seu dispositivo para o seu computador de desenvolvimento. Nesta seção, você baixa o certificado da autoridade de certificação raiz (CA) da Amazon. Em seguida, se você planeja executar o software AWS IoT Greengrass Core no Docker em um computador diferente do seu computador de desenvolvimento, copie os certificados para esse computador host. O software AWS IoT Greengrass Core usa esses certificados para se conectar ao serviço de AWS IoT nuvem.

Para baixar certificados para o dispositivo

1. No seu computador de desenvolvimento, baixe o certificado da autoridade de certificação raiz (CA) da Amazon. AWS IoTos certificados são associados ao certificado CA raiz da Amazon por padrão.

   Linux or Unix

   ```
   sudo curl -o ./greengrass-v2-certs/AmazonRootCA1.pem https://
   www.amazontrust.com/repository/AmazonRootCA1.pem
   ```

   Windows Command Prompt (CMD)

   ```
   curl -o .\greengrass-v2-certs\AmazonRootCA1.pem https://www.amazontrust.com/
   repository/AmazonRootCA1.pem
   ```

PowerShell

```
iwr -Uri https://www.amazontrust.com/repository/AmazonRootCA1.pem -OutFile .
\greengrass-v2-certs\AmazonRootCA1.pem
```

2. Se você planeja executar o software AWS IoT Greengrass Core no Docker em um dispositivo diferente do seu computador de desenvolvimento, copie os certificados para o computador host. Se o SSH e o SCP estiverem habilitados no computador de desenvolvimento e no computador host, você poderá usar o `scp` comando no computador de desenvolvimento para transferir os certificados. *`device-ip-address`*Substitua pelo endereço IP do seu computador host.

```
scp -r greengrass-v2-certs/ device-ip-address:~
```

Criar um arquivo de configuração

1. No computador host, crie uma pasta na qual você coloca seu arquivo de configuração.

```
mkdir ./greengrass-v2-config
```

2. Use um editor de texto para criar um arquivo de configuração nomeado `config.yaml` na `./greengrass-v2-config` pasta.

   Por exemplo, você pode executar o seguinte comando para usar o GNU nano para criar o. `config.yaml`

```
nano ./greengrass-v2-config/config.yaml
```

3. Copie o seguinte conteúdo YAML para o arquivo. Esse arquivo de configuração parcial especifica os parâmetros do sistema e os parâmetros do núcleo do Greengrass.

```
---
system:
  certificateFilePath: "/tmp/certs/device.pem.crt"
  privateKeyPath: "/tmp/certs/private.pem.key"
  rootCaPath: "/tmp/certs/AmazonRootCA1.pem"
  rootpath: "/greengrass/v2"
  thingName: "MyGreengrassCore"
services:
  aws.greengrass.Nucleus:
```

```
    componentType: "NUCLEUS"
    version: "nucleus-version"
    configuration:
      awsRegion: "region"
      iotRoleAlias: "GreengrassCoreTokenExchangeRoleAlias"
      iotDataEndpoint: "device-data-prefix-ats.iot.region.amazonaws.com"
      iotCredEndpoint: "device-credentials-prefix.credentials.region.amazonaws.com"
```

Em seguida, substitua os seguintes valores:

- */tmp/certs*. O diretório no contêiner do Docker no qual você monta os certificados baixados ao iniciar o contêiner.
- */greengrass/v2*. A pasta raiz do Greengrass que você deseja usar para instalação. Você usa a variável de `GGC_ROOT` ambiente para definir esse valor.
- *MyGreengrassCore*. O nome da coisa da AWS IoT.
- *versão núcleo.* A versão do software AWS IoT Greengrass Core a ser instalada. Esse valor deve corresponder à versão da imagem do Docker ou do Dockerfile que você baixou. Se você baixou a imagem do Greengrass Docker com a `latest` tag, use **`docker inspect image-id`** para ver a versão da imagem.
- *região*. O Região da AWS local onde você criou seus AWS IoT recursos. Você também deve especificar o mesmo valor para a variável de `AWS_REGION` ambiente em seu arquivo de ambiente.
- *GreengrassCoreTokenExchangeRoleAlias*. O alias da função de troca de tokens.
- *device-data-prefix*. O prefixo do seu endpoint AWS IoT de dados.
- *device-credentials-prefix*. O prefixo do seu endpoint de AWS IoT credenciais.

Criar um arquivo de ambiente

Este tutorial usa um arquivo de ambiente para definir as variáveis de ambiente que serão passadas para o instalador do software AWS IoT Greengrass Core dentro do contêiner Docker. Você também pode usar o `--env` argumento `-e` ou em seu `docker run` comando para definir variáveis de ambiente no contêiner do Docker ou definir as variáveis em um `environment` bloco no `docker-compose.yml` arquivo.

1. Use um editor de texto para criar um arquivo de ambiente chamado `.env`.

Por exemplo, em um sistema baseado em Linux, você pode executar o seguinte comando para usar o GNU nano para criar o no `.env` diretório atual.

```
nano .env
```

2. Copie o conteúdo a seguir no arquivo.

```
GGC_ROOT_PATH=/greengrass/v2
AWS_REGION=region
PROVISION=false
COMPONENT_DEFAULT_USER=ggc_user:ggc_group
INIT_CONFIG=/tmp/config/config.yaml
```

Em seguida, substitua os valores a seguir.

- */greengrass/v2*. O caminho para a pasta raiz a ser usada para instalar o software AWS IoT Greengrass Core.
- *região*. O Região da AWS local onde você criou seus AWS IoT recursos. Você deve especificar o mesmo valor para o parâmetro `awsRegion` de configuração em seu arquivo de configuração.
- */tmp/config/*. A pasta na qual você monta o arquivo de configuração ao iniciar o contêiner do Docker.

> **Note**
>
> Você pode definir a variável de `DEPLOY_DEV_TOOLS` ambiente `true` para implantar o componente CLI do Greengrass, que permite desenvolver componentes personalizados dentro do contêiner do Docker. Recomendamos que você use esse componente somente em ambientes de desenvolvimento, não em ambientes de produção. Esse componente fornece acesso a informações e operações que você normalmente não precisará em um ambiente de produção. Siga o princípio do menor privilégio implantando esse componente somente nos dispositivos principais onde você precisar dele.

Execute o software AWS IoT Greengrass principal em um contêiner

Este tutorial mostra como iniciar a imagem do Docker que você criou em um contêiner do Docker. Você pode usar a CLI do Docker ou a CLI do Docker Compose para AWS IoT Greengrass executar a imagem do software Core em um contêiner do Docker.

Docker

- Este tutorial mostra como iniciar a imagem do Docker que você criou em um contêiner do Docker.

  ```
  docker run --rm --init -it --name docker-image \
   -v path/to/greengrass-v2-config:/tmp/config/:ro \
   -v path/to/greengrass-v2-certs:/tmp/certs:ro \
   --env-file .env \
   -p 8883 \
   your-container-image:version
  ```

  Esse exemplo de comando usa os seguintes argumentos para docker run:

  - `--rm`. Limpa o contêiner quando ele sai.
  - `--init`. Usa um processo de inicialização no contêiner.

    > **Note**
    >
    > O `--init` argumento é necessário para desligar o software AWS IoT Greengrass Core quando você interrompe o contêiner Docker.

  - `-it`. (Opcional) Executa o contêiner Docker em primeiro plano como um processo interativo. Em vez disso, você pode substituir isso pelo `-d` argumento para executar o contêiner do Docker no modo desanexado. Para obter mais informações, consulte Desanexado versus primeiro plano na documentação do Docker.
  - `--name`. Executa um contêiner chamado `aws-iot-greengrass`
  - `-v`. Monta um volume no contêiner do Docker para disponibilizar o arquivo de configuração e os arquivos de certificado para AWS IoT Greengrass execução dentro do contêiner.
  - `--env-file`. (Opcional) Especifica o arquivo de ambiente para definir as variáveis de ambiente que serão passadas para o instalador do software AWS IoT Greengrass Core dentro do contêiner Docker. Esse argumento é necessário somente se você criou um

arquivo de ambiente para definir variáveis de ambiente. Se você não criou um arquivo de ambiente, pode usar `--env` argumentos para definir variáveis de ambiente diretamente no comando de execução do Docker.

- `-p`. (Opcional) Publica a porta do contêiner 8883 na máquina host. Esse argumento é necessário se você quiser se conectar e se comunicar pelo MQTT porque AWS IoT Greengrass usa a porta 8883 para tráfego MQTT. Para abrir outras portas, use `-p` argumentos adicionais.

> **Note**
>
> Para executar seu contêiner Docker com maior segurança, você pode usar os `--cap-add` argumentos `--cap-drop` e para habilitar seletivamente os recursos do Linux para seu contêiner. Para obter mais informações, consulte Privilégio de tempo de execução e recursos do Linux na documentação do Docker.

Docker Compose

1. Use um editor de texto para criar um arquivo Docker Compose chamado. `docker-compose.yml`

   Por exemplo, em um sistema baseado em Linux, você pode executar o seguinte comando para usar o GNU nano para criar o no `docker-compose.yml` diretório atual.

   ```
   nano docker-compose.yml
   ```

   > **Note**
   >
   > Você também pode baixar e usar a versão mais recente do arquivo Compose AWS fornecido em. GitHub

2. Adicione o conteúdo a seguir ao arquivo Compose. O arquivo deve ser semelhante ao exemplo a seguir. *`Substitua:version your-container-name pelo nome da sua imagem do Docker.`*

   ```
   version: '3.7'
   ```

```
services:
  greengrass:
    init: true
    build:
      context: .
    container_name: aws-iot-greengrass
    image: your-container-name:version
    volumes:
      - /path/to/greengrass-v2-config:/tmp/config/:ro
      - /path/to/greengrass-v2-certs:/tmp/certs:ro
    env_file: .env
    ports:
      - "8883:8883"
```

Os parâmetros a seguir neste exemplo de arquivo Compose são opcionais:

- `ports`—Publica as portas do contêiner 8883 na máquina host. Esse parâmetro é necessário se você quiser se conectar e se comunicar pelo MQTT porque AWS IoT Greengrass usa a porta 8883 para tráfego MQTT.
- `env_file`—Especifica o arquivo de ambiente para definir as variáveis de ambiente que serão passadas para o instalador do software AWS IoT Greengrass Core dentro do contêiner Docker. Esse parâmetro é necessário somente se você criou um arquivo de ambiente para definir variáveis de ambiente. Se você não criou um arquivo de ambiente, pode usar o parâmetro de ambiente para definir as variáveis diretamente no seu arquivo Compose.

> **Note**
>
> Para executar seu contêiner Docker com maior segurança, você pode usar `cap_drop` e `cap_add` em seu arquivo Compose para habilitar seletivamente os recursos do Linux para seu contêiner. Para obter mais informações, consulte Privilégio de tempo de execução e recursos do Linux na documentação do Docker.

3. Execute o comando a seguir para iniciar o contêiner.

```
docker-compose -f docker-compose.yml up
```

Próximas etapas

AWS IoT GreengrassO software principal agora está sendo executado em um contêiner Docker. Execute o comando a seguir para recuperar o ID do contêiner em execução no momento.

```
docker ps
```

Em seguida, você pode executar o comando a seguir para acessar o contêiner e explorar o software AWS IoT Greengrass principal executado dentro do contêiner.

```
docker exec -it container-id /bin/bash
```

Para obter informações sobre a criação de um componente simples, consulte Etapa 4: desenvolver e testar um componente em seu dispositivo em Tutorial: Conceitos básicos do AWS IoT Greengrass V2

> **Note**
>
> Quando você usa `docker exec` para executar comandos dentro do contêiner do Docker, esses comandos não são registrados nos registros do Docker. Para registrar seus comandos nos registros do Docker, anexe um shell interativo ao contêiner do Docker. Para ter mais informações, consulte Anexe um shell interativo ao contêiner Docker.

O arquivo de log AWS IoT Greengrass principal é chamado `greengrass.log` e está localizado em*/greengrass/v2*`/logs`. Os arquivos de log de componentes também estão localizados no mesmo diretório. Para copiar os registros do Greengrass para um diretório temporário no host, execute o seguinte comando:

```
docker cp container-id:/greengrass/v2/logs /tmp/logs
```

Se você quiser manter os registros após a saída ou remoção de um contêiner, recomendamos que você vincule e monte somente o */greengrass/v2*`/logs` diretório no diretório de registros temporários no host, em vez de montar todo o diretório Greengrass. Para ter mais informações, consulte Persiste os registros do Greengrass fora do contêiner Docker.

Para interromper a execução de um contêiner AWS IoT Greengrass Docker, execute `docker stop` o`docker-compose -f docker-compose.yml stop`. Essa ação é enviada `SIGTERM`

para o processo do Greengrass e encerra todos os processos associados que foram iniciados no contêiner. O contêiner Docker é inicializado com o `docker-init` executável como processo PID 1, o que ajuda a remover qualquer processo zumbi restante. Para obter mais informações, consulte Especificar um processo de inicialização na documentação do Docker.

Para obter informações sobre como solucionar problemas com a execução AWS IoT Greengrass em um contêiner Docker, consulteSolução de problemas do AWS IoT Greengrass em um contêiner do Docker.

Solução de problemas do AWS IoT Greengrass em um contêiner do Docker

Use as informações a seguir para ajudá-lo a solucionar problemas com a execução AWS IoT Greengrass em um contêiner do Docker e para depurar problemas AWS IoT Greengrass no contêiner do Docker.

Tópicos

- Solução de problemas com a execução do contêiner Docker
- Depurar o AWS IoT Greengrass em um contêiner do Docker

Solução de problemas com a execução do contêiner Docker

Use as informações a seguir para ajudá-lo a solucionar problemas comuns com a execução do AWS IoT Greengrass em um contêiner do Docker.

Tópicos

- Erro: não é possível realizar um login interativo em um dispositivo não TTY
- Erro: Opções desconhecidas: - no-include-email
- Erro: Um firewall está bloqueando o compartilhamento de arquivos entre janelas e os contêineres.
- Erro: ocorreu um erro (AccessDeniedException) ao chamar a GetAuthorizationToken operação: Usuário: arn:aws:iam:: account-id:user/ <user-name>não está autorizado a realizar: ecr: on resource: * GetAuthorizationToken
- Erro: Você atingiu seu limite de taxa de pull

Erro: não é possível realizar um login interativo em um dispositivo não TTY

Esse erro pode ocorrer quando você executa o `aws ecr get-login-password` comando. Verifique se você instalou a versão 2 ou a versão 1 mais recente do AWS CLI. É recomendável usar

a versão 2 mais recente do AWS CLI. Para obter mais informações, consulte Instalar a AWS CLI no Guia do usuário da AWS Command Line Interface.

Erro: Opções desconhecidas: - no-include-email

Esse erro pode ocorrer quando você executa o `aws ecr get-login` comando. Verifique se você tem a versão mais recente da AWS CLI instalada (por exemplo, execute: `pip install awscli --upgrade --user`). Para obter informações, consulte Instalar a AWS Command Line Interface no Microsoft Windows no Guia do usuário do AWS Command Line Interface.

Erro: Um firewall está bloqueando o compartilhamento de arquivos entre janelas e os contêineres.

Você pode receber esse erro ou uma `Firewall Detected` mensagem ao executar o Docker em um computador Windows. Esse erro também poderá ocorrer se você estiver conectado em uma rede privada virtual (VPN), e as configurações de rede estiverem impedindo a montagem da unidade compartilhada. Nesse caso, desative a VPN e execute novamente o contêiner do Docker.

Erro: ocorreu um erro (AccessDeniedException) ao chamar a GetAuthorizationToken operação: Usuário: arn:aws:iam:: ***account-id:user/*** <user-name>não está autorizado a realizar: ecr: on resource: * GetAuthorizationToken

É possível que você receba esse erro ao executar o comando `aws ecr get-login-password` se não tiver permissões suficientes para acessar um repositório do Amazon ECR. Para obter mais informações, consulte Exemplos de políticas de repositório do Amazon ECR e Como acessar um repositório do Amazon ECR no Guia do usuário do Amazon ECR.

Erro: Você atingiu seu limite de taxa de pull

O Docker Hub limita o número de pull requests que usuários anônimos e gratuitos do Docker Hub podem fazer. Se você exceder os limites de taxa para pull requests de usuários anônimos ou gratuitos, receberá um dos seguintes erros:

```
ERROR: toomanyrequests: Too Many Requests.
```

```
You have reached your pull rate limit.
```

Para resolver esses erros, você pode esperar algumas horas antes de tentar outra pull request. Se você planeja enviar consistentemente um grande número de pull requests, consulte o site do Docker

Hub para obter informações sobre limites de taxa e opções para autenticar e atualizar sua conta do Docker.

Depurar o AWS IoT Greengrass em um contêiner do Docker

Para depurar problemas com um contêiner do Docker, você pode manter os logs de runtime do Greengrass ou anexar um shell interativo ao contêiner do Docker.

Persiste os registros do Greengrass fora do contêiner Docker

Depois de parar um AWS IoT Greengrass contêiner, você pode usar o `docker cp` comando a seguir para copiar os registros do Greengrass do contêiner Docker para um diretório de registros temporário.

```
docker cp container-id:/greengrass/v2/logs /tmp/logs
```

Para manter os registros mesmo após a saída ou remoção de um contêiner, você deve executar o contêiner do AWS IoT Greengrass Docker após a montagem do diretório. */greengrass/v2*`/logs`

Para vincular e montar o */greengrass/v2*`/logs` diretório, faça o seguinte ao executar um novo AWS IoT Greengrass contêiner do Docker.

- Inclua `-v` */tmp/logs*`:`*/greengrass/v2*`/logs:ro` em seu `docker run` comando.

  Modifique o `volumes` bloco no arquivo Compose para incluir a seguinte linha antes de executar o `docker-compose up` comando.

  ```
  volumes:
   - /tmp/logs:/greengrass/v2/logs:ro
  ```

Em seguida, você pode verificar seus registros */tmp/logs* em seu host para ver os registros do Greengrass enquanto AWS IoT Greengrass está sendo executado dentro do contêiner do Docker.

Para obter informações sobre a execução de contêineres do Greengrass Docker, consulte e Execute AWS IoT Greengrass no Docker com provisionamento manual Execute AWS IoT Greengrass no Docker com provisionamento automático

Anexe um shell interativo ao contêiner Docker

Quando você usa `docker exec` para executar comandos dentro do contêiner do Docker, esses comandos não são capturados nos registros do Docker. Registrar seus comandos nos registros do Docker pode ajudá-lo a investigar o estado do contêiner Greengrass Docker. Execute um destes procedimentos:

- Execute o comando a seguir em um terminal separado para conectar a entrada, a saída e o erro padrão do seu terminal ao contêiner em execução. Isso permite que você visualize e controle o contêiner Docker a partir do seu terminal atual.

  ```
  docker attach container-id
  ```

- Execute o comando a seguir em um terminal separado. Isso permite que você execute seus comandos no modo interativo, mesmo que o contêiner não esteja conectado.

  ```
  docker exec -it container-id sh -c "command > /proc/1/fd/1"
  ```

Para AWS IoT Greengrass solução geral de problemas, consulteSolução de problemas.

Configurar o software AWS IoT Greengrass principal

O software AWS IoT Greengrass Core fornece opções que você pode usar para configurar o software. Você pode criar implantações para configurar o software AWS IoT Greengrass Core em cada dispositivo principal.

Tópicos

- Implemente o componente central do Greengrass
- Configurar o núcleo do Greengrass como um serviço do sistema
- Controle a alocação de memória com opções de JVM
- Configurar o usuário que executa os componentes
- Configurar limites de recursos do sistema para componentes
- Conectar-se à porta 443 ou por meio de um proxy de rede
- Use um certificado de dispositivo assinado por uma CA privada
- Defina os tempos limite do MQTT e as configurações de cache

Implemente o componente central do Greengrass

AWS IoT Greengrass fornece o software AWS IoT Greengrass Core como um componente que você pode implantar em seus dispositivos principais do Greengrass. Você pode criar uma implantação para aplicar a mesma configuração a vários dispositivos principais do Greengrass. Para obter mais informações, consulte Núcleo Greengrass e Atualize o software AWS IoT Greengrass principal (OTA).

Configurar o núcleo do Greengrass como um serviço do sistema

Você deve configurar o software AWS IoT Greengrass Core como um serviço do sistema no sistema inicial do seu dispositivo para fazer o seguinte:

- Inicie o software AWS IoT Greengrass Core quando o dispositivo for inicializado. Essa é uma boa prática se você gerencia grandes frotas de dispositivos.
- Instale e execute os componentes do plug-in. Vários componentes AWS fornecidos são componentes de plug-ins, o que permite que eles interajam diretamente com o núcleo do Greengrass. Para obter mais informações sobre tipos de componentes, consulte Tipos de componentes.
- Aplique atualizações over-the-air (OTA) ao software principal do dispositivo AWS IoT Greengrass principal. Para ter mais informações, consulte Atualize o software AWS IoT Greengrass principal (OTA).
- Permita que os componentes reiniciem o software AWS IoT Greengrass principal ou o dispositivo principal quando uma implantação atualiza o componente para uma nova versão ou atualiza determinados parâmetros de configuração. Para obter mais informações, consulte a etapa do ciclo de vida do bootstrap.

> **Important**
>
> Nos dispositivos principais do Windows, você deve configurar o software AWS IoT Greengrass Core como um serviço do sistema.

Tópicos

- Configurar o núcleo como um serviço do sistema (Linux)
- Configurar o núcleo como um serviço do sistema (Windows)

Configurar o núcleo como um serviço do sistema (Linux)

Os dispositivos Linux oferecem suporte a diferentes sistemas de inicialização, como initd, systemd e systemV. Você usa o `--setup-system-service true` argumento ao instalar o software AWS IoT Greengrass Core para iniciar o núcleo como um serviço do sistema e configurá-lo para ser iniciado quando o dispositivo for inicializado. O instalador configura o software AWS IoT Greengrass Core como um serviço do sistema com o systemd.

Você também pode configurar manualmente o núcleo para ser executado como um serviço do sistema. O exemplo a seguir é um arquivo de serviço systemd.

```
[Unit]
Description=Greengrass Core

[Service]
Type=simple
PIDFile=/greengrass/v2/alts/loader.pid
RemainAfterExit=no
Restart=on-failure
RestartSec=10
ExecStart=/bin/sh /greengrass/v2/alts/current/distro/bin/loader

[Install]
WantedBy=multi-user.target
```

Depois de configurar o serviço do sistema, você pode executar os seguintes comandos para configurar a inicialização do dispositivo na inicialização e para iniciar ou parar o software AWS IoT Greengrass Core.

- Para verificar o status do serviço (systemd)

  ```
  sudo systemctl status greengrass.service
  ```

- Para permitir que o núcleo inicie quando o dispositivo for inicializado.

  ```
  sudo systemctl enable greengrass.service
  ```

- Para impedir que o núcleo inicie quando o dispositivo é inicializado.

  ```
  sudo systemctl disable greengrass.service
  ```

- Para iniciar o software AWS IoT Greengrass Core.

```
sudo systemctl start greengrass.service
```

- Para parar o software AWS IoT Greengrass Core.

```
sudo systemctl stop greengrass.service
```

Configurar o núcleo como um serviço do sistema (Windows)

Você usa o `--setup-system-service true` argumento ao instalar o software AWS IoT Greengrass Core para iniciar o núcleo como um serviço do Windows e configurá-lo para ser iniciado quando o dispositivo for inicializado.

Depois de configurar o serviço, você pode executar os seguintes comandos para configurar a inicialização do dispositivo na inicialização e para iniciar ou parar o software AWS IoT Greengrass Core. Você deve executar o prompt de comando ou PowerShell como administrador para executar esses comandos.

Windows Command Prompt (CMD)

- Para verificar o status do serviço

```
sc query "greengrass"
```

- Para permitir que o núcleo inicie quando o dispositivo for inicializado.

```
sc config "greengrass" start=auto
```

- Para impedir que o núcleo inicie quando o dispositivo é inicializado.

```
sc config "greengrass" start=disabled
```

- Para iniciar o software AWS IoT Greengrass Core.

```
sc start "greengrass"
```

- Para parar o software AWS IoT Greengrass Core.

```
sc stop "greengrass"
```

> **Note**
>
> Em dispositivos Windows, o software AWS IoT Greengrass Core ignora esse sinal de desligamento enquanto desliga os processos dos componentes do Greengrass. Se o software AWS IoT Greengrass Core ignorar o sinal de desligamento ao executar esse comando, aguarde alguns segundos e tente novamente.

PowerShell

- Para verificar o status do serviço

```
Get-Service -Name "greengrass"
```

- Para permitir que o núcleo inicie quando o dispositivo for inicializado.

```
Set-Service -Name "greengrass" -Status stopped -StartupType automatic
```

- Para impedir que o núcleo inicie quando o dispositivo é inicializado.

```
Set-Service -Name "greengrass" -Status stopped -StartupType disabled
```

- Para iniciar o software AWS IoT Greengrass Core.

```
Start-Service -Name "greengrass"
```

- Para parar o software AWS IoT Greengrass Core.

```
Stop-Service -Name "greengrass"
```

> **Note**
>
> Em dispositivos Windows, o software AWS IoT Greengrass Core ignora esse sinal de desligamento enquanto desliga os processos dos componentes do Greengrass. Se o

software AWS IoT Greengrass Core ignorar o sinal de desligamento ao executar esse comando, aguarde alguns segundos e tente novamente.

Controle a alocação de memória com opções de JVM

Se você estiver executando AWS IoT Greengrass em um dispositivo com memória limitada, poderá usar as opções da máquina virtual Java (JVM) para controlar o tamanho máximo da pilha, os modos de coleta de lixo e as opções do compilador, que controlam a quantidade de memória usada pelo software Core. AWS IoT Greengrass O tamanho do heap na JVM determina a quantidade de memória que um aplicativo pode usar antes que a coleta de lixo ocorra ou antes que o aplicativo fique sem memória. O tamanho máximo do heap especifica a quantidade máxima de memória que a JVM pode alocar ao expandir o heap durante uma atividade intensa.

Para controlar a alocação de memória, crie uma nova implantação ou revise uma implantação existente que inclua o componente do núcleo e especifique suas opções de JVM no parâmetro de configuração na `jvmOptions` configuração do componente do núcleo.

Dependendo dos seus requisitos, você pode executar o software AWS IoT Greengrass Core com alocação de memória reduzida ou com alocação mínima de memória.

Alocação de memória reduzida

Para executar o software AWS IoT Greengrass Core com alocação de memória reduzida, recomendamos que você use o seguinte exemplo de atualização de mesclagem de configuração para definir as opções de JVM na configuração do núcleo:

```
{
  "jvmOptions": "-Xmx64m -XX:+UseSerialGC -XX:TieredStopAtLevel=1"
}
```

Alocação mínima de memória

Para executar o software AWS IoT Greengrass Core com alocação mínima de memória, recomendamos que você use o seguinte exemplo de atualização de mesclagem de configuração para definir as opções de JVM na configuração do núcleo:

```
{
  "jvmOptions": "-Xmx32m -XX:+UseSerialGC -Xint"
```

```
}
```

Esses exemplos de atualizações de mesclagem de configurações usam as seguintes opções de JVM:

`-Xmx`*NN*`m`

Define o tamanho máximo da pilha da JVM.

Para reduzir a alocação de memória, use `-Xmx64m` como valor inicial para limitar o tamanho da pilha a 64 MB. Para alocação mínima de memória, use `-Xmx32m` como valor inicial para limitar o tamanho da pilha a 32 MB.

Você pode aumentar ou diminuir o `-Xmx` valor dependendo dos seus requisitos reais; no entanto, é altamente recomendável que você não defina o tamanho máximo da pilha abaixo de 16 MB. Se o tamanho máximo da pilha for muito baixo para seu ambiente, o software AWS IoT Greengrass Core poderá encontrar erros inesperados devido à memória insuficiente.

`-XX:+UseSerialGC`

Especifica o uso da coleta de lixo serial para o espaço de pilha da JVM. O coletor de lixo serial é mais lento, mas usa menos memória do que outras implementações de coleta de lixo da JVM.

`-XX:TieredStopAtLevel=1`

Instrui a JVM a usar o compilador Java just-in-time (JIT) uma vez. Como o código compilado pelo JIT usa espaço na memória do dispositivo, usar o compilador JIT mais de uma vez consome mais memória do que uma única compilação.

`-Xint`

Instrui a JVM a não usar o compilador just-in-time (JIT). Em vez disso, a JVM é executada no modo somente interpretado. Esse modo é mais lento do que a execução de código compilado JIT; no entanto, o código compilado não usa espaço na memória.

Para obter informações sobre como criar atualizações de mesclagem de configurações, consulteAtualizar configurações de componentes.

Configurar o usuário que executa os componentes

O software AWS IoT Greengrass principal pode executar processos de componentes como usuário e grupo do sistema diferente daquele que executa o software. Isso aumenta a segurança, porque você

pode executar o software AWS IoT Greengrass Core como root ou como usuário administrador, sem conceder essas permissões aos componentes que são executados no dispositivo principal.

A tabela a seguir indica quais tipos de componentes o software AWS IoT Greengrass Core pode ser executado como usuário especificado por você. Para ter mais informações, consulte Tipos de componentes.

Tipo de componente	Configurar usuário do componente
Núcleo	 Nº
Plug-in	 Nº
Genérico	 Sim
Lambda (sem contêineres)	 Sim
Lambda (em contêineres)	 Sim

Você deve criar o usuário do componente antes de poder especificá-lo em uma configuração de implantação. Em dispositivos baseados em Windows, você também deve armazenar o nome de

usuário e a senha do usuário na instância do gerenciador de credenciais da conta. LocalSystem Para ter mais informações, consulte Configurar um usuário componente em dispositivos Windows.

Ao configurar o usuário do componente em um dispositivo baseado em Linux, você também pode especificar um grupo. Você especifica o usuário e o grupo separados por dois pontos (`:`) no seguinte formato:*`user`*`:`*`group`*. Se você não especificar um grupo, o software AWS IoT Greengrass Core assumirá como padrão o grupo primário do usuário. Você pode usar o nome ou o ID para identificar o usuário e o grupo.

Em dispositivos baseados em Linux, você também pode executar componentes como um usuário do sistema que não existe, também chamado de usuário desconhecido, para aumentar a segurança. Um processo Linux pode sinalizar qualquer outro processo executado pelo mesmo usuário. Um usuário desconhecido não executa outros processos, então você pode executar componentes como um usuário desconhecido para evitar que os componentes sinalizem outros componentes no dispositivo principal. Para executar componentes como um usuário desconhecido, especifique uma ID de usuário que não exista no dispositivo principal. Você também pode especificar um ID de grupo que não existe para ser executado como um grupo desconhecido.

Você pode configurar o usuário para cada componente e para cada dispositivo principal.

- Configurar para um componente

  Você pode configurar cada componente para ser executado com um usuário específico desse componente. Ao criar uma implantação, você pode especificar o usuário para cada componente na `runWith` configuração desse componente. O software AWS IoT Greengrass Core executa componentes como o usuário especificado se você os configurar. Caso contrário, o padrão é executar componentes como o usuário padrão que você configura para o dispositivo principal. Para obter mais informações sobre como especificar o usuário do componente na configuração de implantação, consulte o parâmetro `runWith`de configuração emCriar implantações.

- Configurar o usuário padrão para um dispositivo principal

  Você pode configurar um usuário padrão que o software AWS IoT Greengrass Core usa para executar componentes. Quando o software AWS IoT Greengrass Core executa um componente, ele verifica se você especificou um usuário para esse componente e o usa para executar o componente. Se o componente não especificar um usuário, o software AWS IoT Greengrass Core executará o componente como o usuário padrão que você configurou para o dispositivo principal. Para ter mais informações, consulte Configurar o usuário padrão do componente.

> **Note**
>
> Em dispositivos baseados em Windows, você deve especificar pelo menos um usuário padrão para executar componentes.
>
> Em dispositivos baseados em Linux, as seguintes considerações se aplicam se você não configurar um usuário para executar componentes:
>
> - Se você executar o software AWS IoT Greengrass Core como root, o software não executará componentes. Você deve especificar um usuário padrão para executar componentes se você executar como root.
> - Se você executar o software AWS IoT Greengrass Core como usuário não root, o software executará componentes como esse usuário.

Tópicos

Configurar um usuário componente em dispositivos Windows

Para configurar um usuário do componente em um dispositivo baseado em Windows

1. Crie o usuário do componente na LocalSystem conta do dispositivo.

   ```
   net user /add component-user password
   ```

2. Use o PsExec utilitário da Microsoft para armazenar o nome de usuário e a senha do usuário do componente na instância do Credential Manager da LocalSystem conta.

   ```
   psexec -s cmd /c cmdkey /generic:component-user /user:component-user /pass:password
   ```

   > **Note**
   >
   > Em dispositivos baseados em Windows, a LocalSystem conta executa o núcleo Greengrass, e você deve usar o PsExec utilitário para armazenar as informações do usuário do componente na conta. LocalSystem O uso do aplicativo Credential Manager

armazena essas informações na conta do Windows do usuário atualmente conectado, em vez da LocalSystem conta.

Configurar o usuário padrão do componente

Você pode usar uma implantação para configurar o usuário padrão em um dispositivo principal. Nessa implantação, você atualiza a configuração do componente do núcleo.

> **Note**
>
> Você também pode definir o usuário padrão ao instalar o software AWS IoT Greengrass Core com a `--component-default-user` opção. Para ter mais informações, consulte Instalar o software do AWS IoT Greengrass Core.

Crie uma implantação que especifique a seguinte atualização de configuração para o `aws.greengrass.Nucleus` componente.

Linux

```
{
  "runWithDefault": {
    "posixUser": "ggc_user:ggc_group"
  }
}
```

Windows

```
{
  "runWithDefault": {
    "windowsUser": "ggc_user"
  }
}
```

> **Note**
>
> O usuário que você especificar deve existir, e o nome de usuário e a senha desse usuário devem ser armazenados na instância do gerenciador de credenciais da LocalSystem

conta em seu dispositivo Windows. Para ter mais informações, consulte Configurar um usuário componente em dispositivos Windows.

O exemplo a seguir define uma implantação para um dispositivo baseado em Linux que é configurado `ggc_user` como usuário padrão e `ggc_group` como grupo padrão. A atualização da `merge` configuração requer um objeto JSON serializado.

```
{
  "components": {
    "aws.greengrass.Nucleus": {
      "version": "2.12.6",
      "configurationUpdate": {
        "merge": "{\"runWithDefault\":{\"posixUser\":\"ggc_user:ggc_group\"}}"
      }
    }
  }
}
```

Configurar limites de recursos do sistema para componentes

> **Note**
>
> Esse recurso está disponível para a versão 2.4.0 e posterior do componente núcleo do Greengrass. AWS IoT Greengrass atualmente não oferece suporte a esse recurso nos dispositivos principais do Windows.

Você pode configurar a quantidade máxima de uso de CPU e RAM que os processos de cada componente podem usar no dispositivo principal.

A tabela a seguir mostra os tipos de componentes que suportam os limites de recursos do sistema. Para ter mais informações, consulte Tipos de componentes.

Tipo de componente	Configurar limites de recursos do sistema
Núcleo	 Nº
Plug-in	 Nº
Genérico	 Sim
Lambda (sem contêineres)	 Sim
Lambda (em contêineres)	 Nº

> **Important**
>
> Os limites de recursos do sistema não são suportados quando você executa o software AWS IoT Greengrass Core em um contêiner Docker.

Você pode configurar os limites de recursos do sistema para cada componente e para cada dispositivo principal.

- Configurar para um componente

  Você pode configurar cada componente com limites de recursos do sistema específicos para esse componente. Ao criar uma implantação, você pode especificar os limites de recursos do sistema para cada componente na implantação. Se o componente suportar os limites de recursos do sistema, o software AWS IoT Greengrass principal aplicará os limites aos processos do componente. Se você não especificar os limites de recursos do sistema para um componente, o software AWS IoT Greengrass Core usará todos os padrões que você configurou para o dispositivo principal. Para ter mais informações, consulte Criar implantações.

- Configurar padrões para um dispositivo principal

  Você pode configurar os limites padrão de recursos do sistema que o software AWS IoT Greengrass Core aplica aos componentes que suportam esses limites. Quando o software AWS IoT Greengrass Core executa um componente, ele aplica os limites de recursos do sistema que você especifica para esse componente. Se esse componente não especificar limites de recursos do sistema, o software AWS IoT Greengrass Core aplicará os limites padrão de recursos do sistema que você configura para o dispositivo principal. Se você não especificar limites padrão de recursos do sistema, o software AWS IoT Greengrass Core não aplicará nenhum limite de recursos do sistema por padrão. Para ter mais informações, consulte Configurar limites padrão de recursos do sistema.

Configurar limites padrão de recursos do sistema

Você pode implantar o componente Greengrass nucleus para configurar os limites padrão de recursos do sistema para um dispositivo principal. Para configurar os limites padrão de recursos do sistema, crie uma implantação que especifique a seguinte atualização de configuração para o `aws.greengrass.Nucleus` componente.

```
{
  "runWithDefault": {
    "systemResourceLimits": {
      "cpu": cpuTimeLimit,
      "memory": memoryLimitInKb
    }
  }
}
```

O exemplo a seguir define uma implantação que configura o limite de tempo da CPU em2, o que equivale a 50% de uso em um dispositivo com 4 núcleos de CPU. Este exemplo também configura o uso da memória para 100 MB.

```
{
  "components": {
    "aws.greengrass.Nucleus": {
      "version": "2.12.6",
      "configurationUpdate": {
        "merge": "{\"runWithDefault\":{\"systemResourceLimits\":\"cpus\":2,\"memory
\":102400}}}"
      }
    }
  }
}
```

Conectar-se à porta 443 ou por meio de um proxy de rede

AWS IoT Greengrass os dispositivos principais se comunicam AWS IoT Core usando o protocolo de mensagens MQTT com autenticação de cliente TLS. Por convenção, o MQTT sobre TLS usa a porta 8883. No entanto, como uma medida de segurança, ambientes restritivos podem limitar o tráfego de entrada e saída a um pequeno intervalo de portas TCP. Por exemplo, um firewall corporativo pode abrir a porta 443 para o tráfego HTTPS, mas fechar outras portas que são usadas por protocolos menos comuns, como a porta 8883 para tráfego MQTT. Outros ambientes restritivos podem exigir que todo o tráfego passe por um proxy antes de se conectar à Internet.

> **Note**
>
> Os principais dispositivos do Greengrass que executam o componente Greengrass nucleus v2.0.3 e versões anteriores usam a porta 8443 para se conectar ao endpoint do plano de dados. AWS IoT Greengrass Esses dispositivos devem ser capazes de se conectar a esse endpoint na porta 8443. Para ter mais informações, consulte Permitir o tráfego de dispositivos por meio de um proxy ou firewall.

Para habilitar a comunicação nesses cenários, AWS IoT Greengrass fornece as seguintes opções de configuração:

- Comunicação MQTT pela porta 443. Se sua rede permitir conexões com a porta 443, você poderá configurar o dispositivo principal do Greengrass para usar a porta 443 para tráfego MQTT em vez da porta padrão 8883. Isso pode ser uma conexão direta com a porta 443 ou uma conexão por meio de um servidor de proxy de rede. Diferentemente da configuração padrão, que usa autenticação de cliente baseada em certificado, o MQTT na porta 443 usa a função de serviço do dispositivo para autenticação.

  Para ter mais informações, consulte Configurar o MQTT pela porta 443.

- Comunicação HTTPS pela porta 443. O software AWS IoT Greengrass Core envia tráfego HTTPS pela porta 8443 por padrão, mas você pode configurá-lo para usar a porta 443. AWS IoT Greengrass usa a extensão TLS da Application Layer Protocol Network (ALPN) para habilitar essa conexão. Assim como na configuração padrão, o HTTPS na porta 443 usa autenticação de cliente baseada em certificado.

  > **Important**
  >
  > Para usar o ALPN e habilitar a comunicação HTTPS pela porta 443, seu dispositivo principal deve executar a atualização 252 ou posterior do Java 8. Todas as atualizações do Java versão 9 e posteriores também oferecem suporte ao ALPN.

  Para ter mais informações, consulte Configurar HTTPS pela porta 443.

- Conexão por meio de um proxy de rede. Você pode configurar um servidor proxy de rede para atuar como intermediário na conexão com o dispositivo principal do Greengrass. AWS IoT Greengrass oferece suporte à autenticação básica para proxies HTTP e HTTPS.

  Os dispositivos principais do Greengrass devem executar o Greengrass nucleus v2.5.0 ou posterior para usar proxies HTTPS.

  O software AWS IoT Greengrass principal passa a configuração do proxy para os componentes por meio das variáveis de `NO_PROXY` ambiente `ALL_PROXY` `HTTP_PROXY``HTTPS_PROXY`,,, e. Os componentes devem usar essas configurações para se conectar por meio do proxy. Os componentes usam bibliotecas comuns (como boto3, cURL e o `requests` pacote python) que normalmente usam essas variáveis de ambiente por padrão para fazer conexões. Se um componente também especificar essas variáveis de ambiente, AWS IoT Greengrass não as substituirá.

  Para ter mais informações, consulte Configurar um proxy de rede.

Configurar o MQTT pela porta 443

Você pode configurar o MQTT pela porta 443 em dispositivos principais existentes ou ao instalar o software AWS IoT Greengrass Core em um novo dispositivo principal.

Tópicos

Configure o MQTT pela porta 443 em dispositivos principais existentes

Você pode usar uma implantação para configurar o MQTT pela porta 443 em um dispositivo de núcleo único ou em um grupo de dispositivos principais. Nessa implantação, você atualiza a configuração do componente do núcleo. O núcleo reinicia quando você atualiza sua `mqtt` configuração.

Para configurar o MQTT pela porta 443, crie uma implantação que especifique a seguinte atualização de configuração para o componente. `aws.greengrass.Nucleus`

```
{
  "mqtt": {
    "port": 443
  }
}
```

O exemplo a seguir define uma implantação que configura o MQTT na porta 443. A atualização da `merge` configuração requer um objeto JSON serializado.

```
{
  "components": {
    "aws.greengrass.Nucleus": {
      "version": "2.12.6",
      "configurationUpdate": {
        "merge": "{\"mqtt\":{\"port\":443}}"
      }
    }
  }
}
```

Configure o MQTT pela porta 443 durante a instalação

Você pode configurar o MQTT pela porta 443 ao instalar o software AWS IoT Greengrass Core em um dispositivo principal. Use o argumento do `--init-config` instalador para configurar o MQTT pela porta 443. Você pode especificar esse argumento ao instalar com provisionamento manual, provisionamento defrota ou provisionamento personalizado.

Configurar HTTPS pela porta 443

Esse recurso requer a Núcleo Greengrass versão v2.0.4 ou posterior.

Você pode configurar HTTPS pela porta 443 em dispositivos principais existentes ou ao instalar o software AWS IoT Greengrass Core em um novo dispositivo principal.

Tópicos

- Configure HTTPS pela porta 443 em dispositivos principais existentes
- Configure HTTPS pela porta 443 durante a instalação

Configure HTTPS pela porta 443 em dispositivos principais existentes

Você pode usar uma implantação para configurar HTTPS pela porta 443 em um dispositivo de núcleo único ou em um grupo de dispositivos principais. Nessa implantação, você atualiza a configuração do componente do núcleo.

Para configurar HTTPS pela porta 443, crie uma implantação que especifique a seguinte atualização de configuração para o `aws.greengrass.Nucleus` componente.

```
{
  "greengrassDataPlanePort": 443
}
```

O exemplo a seguir define uma implantação que configura HTTPS pela porta 443. A atualização da `merge` configuração requer um objeto JSON serializado.

```
{
  "components": {
    "aws.greengrass.Nucleus": {
      "version": "2.12.6",
      "configurationUpdate": {
        "merge": "{\"greengrassDataPlanePort\":443}"
```

```
      }
    }
  }
}
```

Configure HTTPS pela porta 443 durante a instalação

Você pode configurar HTTPS pela porta 443 ao instalar o software AWS IoT Greengrass Core em um dispositivo principal. Use o argumento `--init-config` do instalador para configurar HTTPS pela porta 443. Você pode especificar esse argumento ao instalar com provisionamento manual, provisionamento defrota ou provisionamento personalizado.

Configurar um proxy de rede

Siga o procedimento nesta seção para configurar os dispositivos principais do Greengrass para se conectarem à Internet por meio de um proxy de rede HTTP ou HTTPS. Para obter mais informações sobre os endpoints e portas que os dispositivos principais usam, consultePermitir o tráfego de dispositivos por meio de um proxy ou firewall.

Important

Se o seu dispositivo principal executa uma versão do núcleo do Greengrass anterior à v2.4.0, a função do seu dispositivo deve permitir as seguintes permissões para usar um proxy de rede:

- `iot:Connect`
- `iot:Publish`
- `iot:Receive`
- `iot:Subscribe`

Isso é necessário porque o dispositivo usa AWS credenciais do serviço de troca de tokens para autenticar conexões MQTT com. AWS IoT O dispositivo usa o MQTT para receber e instalar implantações do Nuvem AWS, portanto, seu dispositivo não funcionará a menos que você defina essas permissões em sua função. Os dispositivos normalmente usam certificados X.509 para autenticar conexões MQTT, mas os dispositivos não podem fazer isso para se autenticar quando usam um proxy.
Para obter mais informações sobre como configurar a função do dispositivo, consulteAutorize os dispositivos principais a interagir com os serviços AWS.

Tópicos

Configurar um proxy de rede nos dispositivos principais existentes

Você pode usar uma implantação para configurar um proxy de rede em um dispositivo de núcleo único ou em um grupo de dispositivos principais. Nessa implantação, você atualiza a configuração do componente do núcleo. O núcleo reinicia quando você atualiza sua `networkProxy` configuração.

Para configurar um proxy de rede, crie uma implantação para o `aws.greengrass.Nucleus` componente que mescla a seguinte atualização de configuração. Essa atualização de configuração contém o objeto NetworkProxy.

```
{
  "networkProxy": {
    "noProxyAddresses": "http://192.168.0.1,www.example.com",
    "proxy": {
      "url": "https://my-proxy-server:1100"
    }
  }
}
```

O exemplo a seguir define uma implantação que configura um proxy de rede. A atualização da `merge` configuração requer um objeto JSON serializado.

```
{
  "components": {
    "aws.greengrass.Nucleus": {
      "version": "2.12.6",
      "configurationUpdate": {
        "merge": "{\"networkProxy\":{\"noProxyAddresses\":
\"http://192.168.0.1,www.example.com\",\"proxy\":{\"url\":\"https://my-proxy-
server:1100\",\"username\":\"Mary_Major\",\"password\":\"pass@word1357\"}}}"
      }
    }
  }
```

```
}
```

Configurar um proxy de rede durante a instalação

Você pode configurar um proxy de rede ao instalar o software AWS IoT Greengrass Core em um dispositivo principal. Use o argumento `--init-config` do instalador para configurar o proxy de rede. Você pode especificar esse argumento ao instalar com provisionamento manual, provisionamento defrota ou provisionamento personalizado.

Permita que o dispositivo principal confie em um proxy HTTPS

Ao configurar um dispositivo principal para usar um proxy HTTPS, você deve adicionar a cadeia de certificados do servidor proxy à do dispositivo principal para permitir que ele confie no proxy HTTPS. Caso contrário, o dispositivo principal poderá encontrar erros ao tentar rotear o tráfego por meio do proxy. Adicione o certificado CA do servidor proxy ao arquivo de certificado de CA raiz da Amazon do dispositivo principal.

Para permitir que o dispositivo principal confie no proxy HTTPS

1. Encontre o arquivo de certificado CA raiz da Amazon no dispositivo principal.

   - Se você instalou o software AWS IoT Greengrass Core com provisionamento automático, o arquivo de certificado CA raiz da Amazon existe em. */greengrass/v2*`/rootCA.pem`
   - Se você instalou o software AWS IoT Greengrass Core com provisionamento manual ou de frota, o arquivo de certificado CA raiz da Amazon pode existir em. */greengrass/v2*`/AmazonRootCA1.pem`

   Se o certificado CA raiz da Amazon não existir nesses locais, verifique a `system.rootCaPath` propriedade */greengrass/v2*`/config/effectiveConfig.yaml` para encontrar sua localização.
2. Adicione o conteúdo do arquivo de certificado CA do servidor proxy ao arquivo de certificado CA raiz da Amazon.

   O exemplo a seguir mostra um certificado CA do servidor proxy adicionado ao arquivo de certificado CA raiz da Amazon.

   ```
   -----BEGIN CERTIFICATE-----
   MIIEFTCCAv2gAwIQWgIVAMHSAzWG/5YVRYtRQOxXUTEpHuEmApzGCSqGSIb3DQEK
   \nCwUAhuL9MQswCQwJVUzEPMAVUzEYMBYGA1UECgwP1hem9uLmNvbSBJbmMuMRww
   ```

```
... content of proxy CA certificate ...
+vHIRlt0e5JAm5\noTIZGoFbK82A0/nO7f/t5PSIDAim9V3Gc3pSXxCCAQoFYnui
GaPUlGk1gCE84a0X\n7Rp/lND/PuMZ/s8YjlkY2NmYmNjMCAXDTE5MTEyN2cM216
gJMIADggEPADf2/m45hzEXAMPLE=
-----END CERTIFICATE-----

-----BEGIN CERTIFICATE-----
MIIDQTCCAimgF6AwIBAgITBmyfz/5mjAo54vB4ikPmljZKyjANJmApzyMZFo6qBg
ADA5MQswCQYDVQQGEwJVUzEPMA0tMVT8QtPHRh8jrdkGA1UEChMGDV3QQDExBBKW
... content of root CA certificate ...
o/ufQJQWUCyziar1hem9uMRkwFwYVPSHCb2XV4cdFyQzR1KldZwgJcIQ6XUDgHaa
5MsI+yMRQ+hDaXJiobldXgjUka642M4UwtBV8oK2xJNDd2ZhwLnoQdeXeGADKkpy
rqXRfKoQnoZsG4q5WTP46EXAMPLE
-----END CERTIFICATE-----
```

O objeto NetworkProxy

Use o objeto `networkProxy` para especificar informações sobre o proxy de rede. Esse objeto contém as seguintes informações:

`noProxyAddresses`

(Opcional) Uma lista separada por vírgulas de endereços IP ou nomes de host que estão isentos do proxy.

`proxy`

O proxy ao qual se conectar. Esse objeto contém as seguintes informações:

`url`

O URL do servidor proxy no formato`scheme://userinfo@host:port`.

- `scheme`— O esquema, que deve ser `http` ou`https`.

> **Important**
>
> Os dispositivos principais do Greengrass devem executar o Greengrass nucleus v2.5.0 ou posterior para usar proxies HTTPS.
> Se você configurar um proxy HTTPS, deverá adicionar o certificado CA do servidor proxy ao certificado de CA raiz da Amazon do dispositivo principal. Para ter mais

informações, consulte Permita que o dispositivo principal confie em um proxy HTTPS.

- `userinfo`— (Opcional) As informações de nome de usuário e senha. Se você especificar essas informações no`url`, o dispositivo principal do Greengrass ignorará os campos e. `username` `password`
- `host`— O nome do host ou endereço IP do servidor proxy.
- `port`— (Opcional) O número da porta. Se você não especificar a porta, o dispositivo principal do Greengrass usará os seguintes valores padrão:
  - `http`— 80
  - `https`— 443

`username`

(Opcional) O nome de usuário que autentica o servidor proxy.

`password`

(Opcional) A senha que autentica o servidor proxy.

Use um certificado de dispositivo assinado por uma CA privada

Se você estiver usando uma autoridade de certificação (CA) privada personalizada, você deve definir o núcleo **`greengrassDataPlaneEndpoint`** do Greengrass como. **`iotdata`** Você pode definir essa opção durante a implantação ou instalação usando o argumento do **`--init-config`** instalador.

Você pode personalizar o endpoint do plano de dados do Greengrass ao qual o dispositivo se conecta. Você pode definir essa opção de configuração **`iotdata`** para definir o endpoint do plano de dados do Greengrass como o mesmo endpoint de dados de IoT, que você pode especificar com o. **`iotDataEndpoint`**

Defina os tempos limite do MQTT e as configurações de cache

No AWS IoT Greengrass ambiente, os componentes podem usar o MQTT para se comunicar com AWS IoT Core eles. O software AWS IoT Greengrass Core gerencia mensagens MQTT para componentes. Quando o dispositivo principal perde a conexão com o Nuvem AWS, o software armazena em cache as mensagens MQTT para tentar novamente mais tarde, quando a conexão for restaurada. Você pode definir configurações como o tempo limite das mensagens e o tamanho

do cache. Para obter mais informações, consulte os parâmetros `mqtt` e os parâmetros de `mqtt.spooler` configuração do componente do núcleo do Greengrass.

AWS IoT Core impõe cotas de serviço em seu agente de mensagens MQTT. Essas cotas podem se aplicar às mensagens que você envia entre os dispositivos principais e. AWS IoT Core Para obter mais informações, consulte cotas do serviço de agente de AWS IoT Core mensagens no Referência geral da AWS.

Atualize o software AWS IoT Greengrass principal (OTA)

O software AWS IoT Greengrass principal compreende o componente do núcleo Greengrass e outros componentes opcionais que você pode implantar em seus dispositivos para realizar atualizações over-the-air (OTA) do software. Esse recurso é incorporado ao software AWS IoT Greengrass Core.

As atualizações OTA são mais eficientes para:

- Corrigir vulnerabilidades de segurança.
- Resolver problemas de estabilidade do software.
- Implantar atributos novos ou melhorados.

Tópicos

Requisitos

Os seguintes requisitos se aplicam à implantação de atualizações OTA do software AWS IoT Greengrass Core:

- O dispositivo principal do Greengrass deve ter uma conexão com o para receber Nuvem AWS a implantação.
- O dispositivo principal do Greengrass deve estar corretamente configurado e provisionado com certificados e chaves para autenticação com e. AWS IoT Core AWS IoT Greengrass

- O software AWS IoT Greengrass principal deve ser configurado e executado como um serviço do sistema. As atualizações OTA não funcionam se você executar o núcleo a partir do arquivo JAR,`Greengrass.jar`. Para ter mais informações, consulte Configurar o núcleo do Greengrass como um serviço do sistema.

Considerações sobre dispositivos principais

Antes de realizar uma atualização OTA, esteja ciente do impacto nos dispositivos principais que você atualiza e nos dispositivos clientes conectados:

- O núcleo do Greengrass é desligado.
- Todos os componentes em execução no dispositivo principal também são desligados. Se esses componentes forem gravados em recursos locais, eles poderão deixar esses recursos em um estado incorreto, a menos que sejam desligados corretamente. Os componentes podem usar a comunicação entre processos para pedir ao componente do núcleo que adie a atualização até que limpe os recursos que usam.
- Enquanto o componente do núcleo é desligado, o dispositivo principal perde suas conexões com Nuvem AWS os dispositivos locais. O dispositivo principal não roteará mensagens dos dispositivos clientes enquanto estiver desligado.
- Funções Lambda de longa duração que são executadas como componentes perdem suas informações de estado dinâmico e eliminam todo o trabalho pendente.

Comportamento de atualização do núcleo Greengrass

Quando você implanta um componente, AWS IoT Greengrass instala as versões mais recentes suportadas de todas as dependências desse componente. Por esse motivo, novas versões AWS de patch dos componentes públicos fornecidos podem ser implantadas automaticamente em seus dispositivos principais se você adicionar novos dispositivos a um grupo de coisas ou atualizar a implantação que visa esses dispositivos. Algumas atualizações automáticas, como a atualização do nucleus, podem fazer com que seus dispositivos reiniciem inesperadamente.

Quando a versão do componente do núcleo do Greengrass muda, o software AWS IoT Greengrass Core — que inclui o núcleo e todos os outros componentes do seu dispositivo — reinicia para aplicar as alterações. Devido ao impacto nos dispositivos principais quando o componente nucleus é atualizado, talvez você queira controlar quando uma nova versão do patch nucleus é implantada em seus dispositivos. Para fazer isso, você deve incluir diretamente o componente do núcleo do

Greengrass em sua implantação. Incluir diretamente um componente significa que você inclui uma versão específica desse componente em sua configuração de implantação e não depende de dependências de componentes para implantar esse componente em seus dispositivos. Para obter mais informações sobre como definir dependências em suas receitas de componentes, consulteFormato da receita.

Analise a tabela a seguir para entender o comportamento de atualização do componente nucleus do Greengrass com base em suas ações e configurações de implantação.

Ação	Configuração de implantação	Comportamento de atualização do Nucleus
Adicione novos dispositivos a um grupo de coisas visado por uma implantação existente sem revisar a implantação.	A implantação não inclui diretamente o núcleo Greengrass. A implantação inclui diretamente pelo menos um componente AWS fornecido ou inclui um componente personalizado que depende de um componente AWS fornecido ou do núcleo do Greengrass.	Em novos dispositivos, instala a versão de patch mais recente do nucleus que atende a todos os requisitos de dependência de componentes. Em dispositivos existentes, não atualiza a versão instalada do núcleo.
Adicione novos dispositivos a um grupo de coisas visado por uma implantação existente sem revisar a implantação.	A implantação inclui diretamente uma versão específica do núcleo do Greengrass.	Em novos dispositivos, instala a versão do núcleo especificada. Em dispositivos existentes, não atualiza a versão instalada do núcleo.
Crie uma nova implantação ou revise uma implantação existente.	A implantação não inclui diretamente o núcleo Greengrass.	Em todos os dispositivos de destino, instala a versão de patch mais recente do núcleo que atende a todos os requisitos de dependênc

Ação	Configuração de implantação	Comportamento de atualizaç ão do Nucleus
	A implantação inclui diretamen te pelo menos um component e AWS fornecido ou inclui um componente personali zado que depende de um componente AWS fornecido ou do núcleo do Greengrass.	ia de componentes, inclusive em qualquer novo dispositivo que você adicione ao grupo de itens de destino.
Crie uma nova implantação ou revise uma implantação existente.	A implantação inclui diretamen te uma versão específica do núcleo do Greengrass.	Em todos os dispositivos de destino, instala a versão do núcleo especificada, incluindo todos os novos dispositivos adicionados ao grupo de itens de destino.

Execute uma atualização OTA

Para realizar uma atualização OTA, crie uma implantação que inclua o componente nucleus e a versão a ser instalada.

Desinstale o software AWS IoT Greengrass principal

Você pode desinstalar o software AWS IoT Greengrass Core para removê-lo de um dispositivo que você não deseja usar como dispositivo principal do Greengrass. Você também pode usar essas etapas para limpar uma instalação que falhou.

Para desinstalar o software AWS IoT Greengrass Core

1. Se você executar o software como um serviço do sistema, deverá interromper, desativar e remover o serviço. Execute os comandos a seguir conforme apropriado para seu sistema operacional.

Linux

1. Interrompa o serviço .

```
sudo systemctl stop greengrass.service
```

2. Desative o serviço.

```
sudo systemctl disable greengrass.service
```

3. Remova o serviço.

```
sudo rm /etc/systemd/system/greengrass.service
```

4. Verifique se o serviço foi excluído.

```
sudo systemctl daemon-reload && sudo systemctl reset-failed
```

Windows (Command Prompt)

> **Note**
>
> Você deve executar o prompt de comando como administrador para executar esses comandos.

1. Interrompa o serviço .

```
sc stop "greengrass"
```

2. Desative o serviço.

```
sc config "greengrass" start=disabled
```

3. Remova o serviço.

```
sc delete "greengrass"
```

4. Reinicie o dispositivo.

Windows (PowerShell)

> **Note**
>
> Você deve executar PowerShell como administrador para executar esses comandos.

1. Interrompa o serviço .

```
Stop-Service -Name "greengrass"
```

2. Desative o serviço.

```
Set-Service -Name "greengrass" -Status stopped -StartupType disabled
```

3. Remova o serviço.

   - Para PowerShell 6.0 e versões posteriores:

```
Remove-Service -Name "greengrass" -Confirm:$false -Verbose
```

   - Para PowerShell versões anteriores à 6.0:

```
Get-Item HKLM:\SYSTEM\CurrentControlSet\Services\greengrass | Remove-Item
 -Force -Verbose
```

4. Reinicie o dispositivo.

2. Remova a pasta raiz do dispositivo. Substitua */greengrass/v2* ou *C:\greengrass\v2* pelo caminho para a pasta raiz.

   Linux

```
sudo rm -rf /greengrass/v2
```

   Windows (Command Prompt)

```
rmdir /s /q C:\greengrass\v2
```

Windows (PowerShell)

```
cmd.exe /c "rmdir /s /q C:\greengrass\v2"
```

3. Exclua o dispositivo principal do AWS IoT Greengrass serviço. Essa etapa remove as informações de status do dispositivo principal doNuvem AWS. Certifique-se de concluir esta etapa se você planeja reinstalar o software AWS IoT Greengrass Core em um dispositivo principal com o mesmo nome.
   - Para excluir um dispositivo principal do AWS IoT Greengrass console, faça o seguinte:
     a. Navegue até o console do AWS IoT Greengrass.
     b. Escolha dispositivos principais.
     c. Escolha o dispositivo principal a ser excluído.
     d. Escolha Excluir.
     e. No modal de confirmação, escolha Excluir.
   - Para excluir um dispositivo principal com oAWS Command Line Interface, use a DeleteCoreDeviceoperação. Execute o comando a seguir e *MyGreengrassCore*substitua pelo nome do dispositivo principal.

     ```
     aws greengrassv2 delete-core-device --core-device-thing-name MyGreengrassCore
     ```

Tutoriais do AWS IoT Greengrass V2

Você pode concluir os seguintes tutoriais para conhecer AWS IoT Greengrass V2 e conhecer seus recursos.

Tópicos

Tutorial: Desenvolva um componente do Greengrass que adia as atualizações de componentes

Você pode concluir este tutorial para desenvolver um componente que adia as atualizações de over-the-air implantação. Ao implantar atualizações em seus dispositivos, talvez você queira atrasar as atualizações com base em condições, como as seguintes:

- O dispositivo tem um nível de bateria baixo.
- O dispositivo está executando um processo ou trabalho que não pode ser interrompido.
- O dispositivo tem uma conexão de internet limitada ou cara.

> **Note**
>
> Um componente é um módulo de software executado em dispositivos AWS IoT Greengrass principais. Os componentes permitem que você crie e gerencie aplicativos complexos como blocos de construção discretos que você pode reutilizar de um dispositivo principal do Greengrass para outro.

Neste tutorial, você faz o seguinte:

1. Instale o Greengrass Development Kit CLI (GDK CLI) em seu computador de desenvolvimento. A CLI do GDK fornece recursos que ajudam você a desenvolver componentes personalizados do Greengrass.
2. Desenvolva um componente Hello World que adie as atualizações de componentes quando o nível da bateria do dispositivo principal estiver abaixo de um limite. Esse componente se inscreve para atualizar as notificações usando a operação SubscribeToComponentUpdatesIPC. Ao receber a notificação, ele verifica se o nível da bateria está abaixo de um limite personalizável. Se o nível da bateria estiver abaixo do limite, a atualização será adiada por 30 segundos usando a operação DeferComponentUpdateIPC. Você desenvolve esse componente em seu computador de desenvolvimento usando a CLI do GDK.

   > **Note**
   >
   > Esse componente lê o nível da bateria de um arquivo que você cria no dispositivo principal para imitar uma bateria real, para que você possa concluir este tutorial em um dispositivo principal sem bateria.

3. Publique esse componente no AWS IoT Greengrass serviço.
4. Implante esse componente do Nuvem AWS em um dispositivo principal do Greengrass para testá-lo. Em seguida, você modifica o nível da bateria virtual no dispositivo principal e cria implantações adicionais para ver como o dispositivo principal adia as atualizações quando o nível da bateria está baixo.

Você pode esperar passar de 20 a 30 minutos neste tutorial.

Pré-requisitos

Para concluir este tutorial, você precisará do seguinte:

- Uma Conta da AWS. Se você não tiver uma, consulte Configurar um Conta da AWS.
- Um usuário AWS Identity and Access Management (IAM) com permissões de administrador.
- Um dispositivo principal do Greengrass com conexão à internet. Para obter mais informações sobre como configurar um dispositivo principal, consulteConfigurando dispositivos AWS IoT Greengrass principais.
  - Python 3.6 ou posterior instalado para todos os usuários no dispositivo principal e adicionado à variável de ambiente. `PATH` No Windows, você também deve ter o Python Launcher para Windows instalado para todos os usuários.

> **Important**
>
> No Windows, o Python não é instalado para todos os usuários por padrão. Ao instalar o Python, você deve personalizar a instalação para configurá-la para que o software AWS IoT Greengrass Core execute scripts Python. Por exemplo, se você usa o instalador gráfico do Python, faça o seguinte:
>
> 1. Selecione Instalar lançador para todos os usuários (recomendado).
> 2. Selecione Customize installation.
> 3. Selecione Next.
> 4. Selecione Install for all users.
> 5. Selecione Add Python to environment variables.
> 6. Escolha Instalar.
>
> Para obter mais informações, consulte Usando o Python no Windows na documentação do Python 3.

- Um computador de desenvolvimento semelhante ao Windows, macOS ou Unix com conexão à Internet.
  - Python 3.6 ou posterior instalado em seu computador de desenvolvimento.
  - Git instalado no seu computador de desenvolvimento.
  - AWS Command Line Interface(AWS CLI) instalado e configurado com credenciais em seu computador de desenvolvimento. Para obter mais informações, consulte Instalação, atualização e desinstalação do AWS CLI e Configuração do AWS CLI no Guia do AWS Command Line Interface Usuário.

> **Note**
>
> Se você usa um Raspberry Pi ou outro dispositivo ARM de 32 bits, instale AWS CLI a V1. AWS CLI A V2 não está disponível para dispositivos ARM de 32 bits. Para obter mais informações, consulte Instalando, atualizando e desinstalando a AWS CLI versão 1.

Etapa 1: Instalar o Greengrass Development Kit CLI

O Greengrass Development Kit CLI (GDK CLI) fornece recursos que ajudam você a desenvolver componentes personalizados do Greengrass. Você pode usar a CLI do GDK para criar, criar e publicar componentes personalizados.

Se você não instalou a CLI do GDK em seu computador de desenvolvimento, conclua as etapas a seguir para instalá-la.

Para instalar a versão mais recente da CLI do GDK

1. No seu computador de desenvolvimento, execute o comando a seguir para instalar a versão mais recente da CLI do GDK a partir de GitHub seu repositório.

   ```
   python3 -m pip install -U git+https://github.com/aws-greengrass/aws-greengrass-gdk-cli.git@v1.6.2
   ```

2. Execute o comando a seguir para verificar se a CLI do GDK foi instalada com êxito.

   ```
   gdk --help
   ```

   Se o `gdk` comando não for encontrado, adicione sua pasta ao PATH.

   - Em dispositivos Linux, adicione `/home/`*`MyUser`*`/.local/bin` ao PATH e *`MyUser`*substitua pelo nome do seu usuário.
   - Em dispositivos Windows, adicione *`PythonPath`*`\\Scripts` ao PATH e *`PythonPath`*substitua pelo caminho para a pasta Python no seu dispositivo.

Etapa 2: desenvolver um componente que adia as atualizações

Nesta seção, você desenvolve um componente Hello World em Python que adia as atualizações do componente quando o nível da bateria do dispositivo principal está abaixo do limite que você configura ao implantar o componente. Neste componente, você usa a interface de comunicação entre processos (IPC) na AWS IoT Device SDK v2 para Python. Você usa a operação SubscribeToComponentUpdatesIPC para receber notificações quando o dispositivo principal recebe uma implantação. Em seguida, você usa a operação DeferComponentUpdateIPC para adiar ou confirmar a atualização com base no nível da bateria do dispositivo.

Para desenvolver um componente Hello World que adia atualizações

1. No seu computador de desenvolvimento, crie uma pasta para o código-fonte do componente.

```
mkdir com.example.BatteryAwareHelloWorld
cd com.example.BatteryAwareHelloWorld
```

2. Use um editor de texto para criar um arquivo chamado`gdk-config.json`. A CLI do GDK lê o arquivo de configuração da CLI do GDK, `gdk-config.json` chamado, para criar e publicar componentes. Esse arquivo de configuração existe na raiz da pasta do componente.

   Por exemplo, em um sistema baseado em Linux, você pode executar o seguinte comando para usar o GNU nano para criar o arquivo.

```
nano gdk-config.json
```

   Copie o seguinte JSON no arquivo.

   - Substitua *Amazon* pelo seu nome.
   - Substitua *us-west-2* pelo Região da AWS local em que seu dispositivo principal opera. A CLI do GDK publica o componente neste. Região da AWS
   - *greengrass-component-artifacts*Substitua pelo prefixo do bucket S3 a ser usado. Quando você usa a CLI do GDK para publicar o componente, a CLI do GDK carrega os artefatos do componente para o bucket do S3 cujo nome é formado por esse valor, o, e seu ID usando Região da AWS o seguinte formato:. Conta da AWS *bucketPrefix*-*region*-*accountId*

     Por exemplo, se você especificar **greengrass-component-artifacts** e**us-west-2**, e seu Conta da AWS ID for**123456789012**, a CLI do GDK usará o bucket do S3 chamado. `greengrass-component-artifacts-us-west-2-123456789012`

```
{
  "component": {
    "com.example.BatteryAwareHelloWorld": {
      "author": "Amazon",
      "version": "NEXT_PATCH",
      "build": {
        "build_system" : "zip"
      },
```

```
      "publish": {
        "region": "us-west-2",
        "bucket": "greengrass-component-artifacts"
      }
    }
  },
  "gdk_version": "1.0.0"
}
```

O arquivo de configuração especifica o seguinte:

- A versão a ser usada quando a CLI do GDK publica o componente Greengrass no serviço de nuvem. AWS IoT Greengrass `NEXT_PATCH`especifica a escolha da próxima versão do patch após a versão mais recente disponível no serviço de AWS IoT Greengrass nuvem. Se o componente ainda não tiver uma versão no serviço de AWS IoT Greengrass nuvem, a `1.0.0` CLI do GDK usará.
- O sistema de construção do componente. Quando você usa o sistema de `zip` compilação, a CLI do GDK empacota a fonte do componente em um arquivo ZIP que se torna o único artefato do componente.
- O Região da AWS local onde a CLI do GDK publica o componente Greengrass.
- O prefixo do bucket do S3 em que a CLI do GDK carrega os artefatos do componente.

3. Use um editor de texto para criar o código-fonte do componente em um arquivo chamado`main.py`.

   Por exemplo, em um sistema baseado em Linux, você pode executar o seguinte comando para usar o GNU nano para criar o arquivo.

   ```
   nano main.py
   ```

   Copie o código Python a seguir no arquivo.

   ```
   import json
   import os
   import sys
   import time
   import traceback

   from pathlib import Path
   ```

```
from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2

HELLO_WORLD_PRINT_INTERVAL = 15  # Seconds
DEFER_COMPONENT_UPDATE_INTERVAL = 30 * 1000  # Milliseconds


class BatteryAwareHelloWorldPrinter():
    def __init__(self, ipc_client: GreengrassCoreIPCClientV2, battery_file_path:
 Path, battery_threshold: float):
        self.battery_file_path = battery_file_path
        self.battery_threshold = battery_threshold
        self.ipc_client = ipc_client
        self.subscription_operation = None

    def on_component_update_event(self, event):
        try:
            if event.pre_update_event is not None:
                if self.is_battery_below_threshold():
                    self.defer_update(event.pre_update_event.deployment_id)
                    print('Deferred update for deployment %s' %
                          event.pre_update_event.deployment_id)
                else:
                    self.acknowledge_update(
                        event.pre_update_event.deployment_id)
                    print('Acknowledged update for deployment %s' %
                          event.pre_update_event.deployment_id)
            elif event.post_update_event is not None:
                print('Applied update for deployment')
        except:
            traceback.print_exc()

    def subscribe_to_component_updates(self):
        if self.subscription_operation == None:
            # SubscribeToComponentUpdates returns a tuple with the response and the
 operation.
            _, self.subscription_operation =
 self.ipc_client.subscribe_to_component_updates(
                on_stream_event=self.on_component_update_event)

    def close_subscription(self):
        if self.subscription_operation is not None:
            self.subscription_operation.close()
            self.subscription_operation = None
```

```
    def defer_update(self, deployment_id):
        self.ipc_client.defer_component_update(
            deployment_id=deployment_id,
 recheck_after_ms=DEFER_COMPONENT_UPDATE_INTERVAL)

    def acknowledge_update(self, deployment_id):
        # Specify recheck_after_ms=0 to acknowledge a component update.
        self.ipc_client.defer_component_update(
            deployment_id=deployment_id, recheck_after_ms=0)

    def is_battery_below_threshold(self):
        return self.get_battery_level() < self.battery_threshold

    def get_battery_level(self):
        # Read the battery level from the virtual battery level file.
        with self.battery_file_path.open('r') as f:
            data = json.load(f)
            return float(data['battery_level'])

    def print_message(self):
        message = 'Hello, World!'
        if self.is_battery_below_threshold():
            message += ' Battery level (%d) is below threshold (%d), so the
 component will defer updates' % (
                self.get_battery_level(), self.battery_threshold)
        else:
            message += ' Battery level (%d) is above threshold (%d), so the
 component will acknowledge updates' % (
                self.get_battery_level(), self.battery_threshold)
        print(message)


def main():
    # Read the battery threshold and virtual battery file path from command-line
 args.
    args = sys.argv[1:]
    battery_threshold = float(args[0])
    battery_file_path = Path(args[1])
    print('Reading battery level from %s and deferring updates when below %d' % (
        str(battery_file_path), battery_threshold))

    try:
        # Create an IPC client and a Hello World printer that defers component
 updates.
```

```
        ipc_client = GreengrassCoreIPCClientV2()
        hello_world_printer = BatteryAwareHelloWorldPrinter(
            ipc_client, battery_file_path, battery_threshold)
        hello_world_printer.subscribe_to_component_updates()
        try:
            # Keep the main thread alive, or the process will exit.
            while True:
                hello_world_printer.print_message()
                time.sleep(HELLO_WORLD_PRINT_INTERVAL)
        except InterruptedError:
            print('Subscription interrupted')
        hello_world_printer.close_subscription()
    except Exception:
        print('Exception occurred', file=sys.stderr)
        traceback.print_exc()
        exit(1)


if __name__ == '__main__':
    main()
```

Esse aplicativo Python faz o seguinte:

- Lê o nível da bateria do dispositivo principal a partir de um arquivo virtual de nível de bateria que você criará posteriormente no dispositivo principal. Esse arquivo virtual de nível de bateria imita uma bateria real, então você pode concluir este tutorial em dispositivos principais que não têm bateria.
- Lê os argumentos da linha de comando para o limite da bateria e o caminho para o arquivo virtual do nível da bateria. A receita do componente define esses argumentos de linha de comando com base nos parâmetros de configuração, para que você possa personalizar esses valores ao implantar o componente.
- Usa o cliente IPC V2 na AWS IoT Device SDKv2 para Python para se comunicar com o software Core. AWS IoT Greengrass Em comparação com o cliente IPC original, o cliente IPC V2 reduz a quantidade de código que você precisa escrever para usar o IPC em componentes personalizados.
- Se inscreve para atualizar as notificações usando a operação SubscribeToComponentUpdatesIPC. O software AWS IoT Greengrass Core envia notificações antes e depois de cada implantação. O componente chama a função a seguir sempre que recebe uma notificação. Se a notificação for para uma implantação futura, o

componente verificará se o nível da bateria está abaixo do limite. Se o nível da bateria estiver abaixo do limite, o componente adia a atualização por 30 segundos usando a operação DeferComponentUpdateIPC. Caso contrário, se o nível da bateria não estiver abaixo do limite, o componente confirma a atualização, para que a atualização possa continuar.

```
def on_component_update_event(self, event):
    try:
        if event.pre_update_event is not None:
            if self.is_battery_below_threshold():
                self.defer_update(event.pre_update_event.deployment_id)
                print('Deferred update for deployment %s' %
                      event.pre_update_event.deployment_id)
            else:
                self.acknowledge_update(
                    event.pre_update_event.deployment_id)
                print('Acknowledged update for deployment %s' %
                      event.pre_update_event.deployment_id)
        elif event.post_update_event is not None:
            print('Applied update for deployment')
    except:
        traceback.print_exc()
```

> **Note**
>
> O software AWS IoT Greengrass Core não envia notificações de atualização para implantações locais, então você implanta esse componente usando o serviço de AWS IoT Greengrass nuvem para testá-lo.

4. Use um editor de texto para criar a receita do componente em um arquivo chamado `recipe.json` ou`recipe.yaml`. A receita do componente define os metadados do componente, os parâmetros de configuração padrão e os scripts de ciclo de vida específicos da plataforma.

   JSON

   Por exemplo, em um sistema baseado em Linux, você pode executar o seguinte comando para usar o GNU nano para criar o arquivo.

   ```
   nano recipe.json
   ```

Copie o seguinte JSON no arquivo.

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "COMPONENT_NAME",
  "ComponentVersion": "COMPONENT_VERSION",
  "ComponentDescription": "This Hello World component defers updates when the
 battery level is below a threshold.",
  "ComponentPublisher": "COMPONENT_AUTHOR",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "BatteryThreshold": 50,
      "LinuxBatteryFilePath": "/home/ggc_user/virtual_battery.json",
      "WindowsBatteryFilePath": "C:\\Users\\ggc_user\\virtual_battery.json"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "python3 -m pip install --user awsiotsdk --upgrade",
        "run": "python3 -u {artifacts:decompressedPath}/
com.example.BatteryAwareHelloWorld/main.py \"{configuration:/BatteryThreshold}\"
 \"{configuration:/LinuxBatteryFilePath}\""
      },
      "Artifacts": [
        {
          "Uri": "s3://BUCKET_NAME/COMPONENT_NAME/COMPONENT_VERSION/
com.example.BatteryAwareHelloWorld.zip",
          "Unarchive": "ZIP"
        }
      ]
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "install": "py -3 -m pip install --user awsiotsdk --upgrade",
```

```
        "run": "py -3 -u {artifacts:decompressedPath}/
com.example.BatteryAwareHelloWorld/main.py \"{configuration:/BatteryThreshold}\"
 \"{configuration:/WindowsBatteryFilePath}\""
      },
      "Artifacts": [
        {
          "Uri": "s3://BUCKET_NAME/COMPONENT_NAME/COMPONENT_VERSION/
com.example.BatteryAwareHelloWorld.zip",
          "Unarchive": "ZIP"
        }
      ]
    }
  ]
}
```

YAML

Por exemplo, em um sistema baseado em Linux, você pode executar o seguinte comando para usar o GNU nano para criar o arquivo.

```
nano recipe.yaml
```

Copie o seguinte YAML no arquivo.

```
---
RecipeFormatVersion: "2020-01-25"
ComponentName: "COMPONENT_NAME"
ComponentVersion: "COMPONENT_VERSION"
ComponentDescription: "This Hello World component defers updates when the
 battery level is below a threshold."
ComponentPublisher: "COMPONENT_AUTHOR"
ComponentConfiguration:
  DefaultConfiguration:
    BatteryThreshold: 50
    LinuxBatteryFilePath: "/home/ggc_user/virtual_battery.json"
    WindowsBatteryFilePath: "C:\\Users\\ggc_user\\virtual_battery.json"
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: python3 -m pip install --user awsiotsdk --upgrade
```

```
      run: python3 -u {artifacts:decompressedPath}/
com.example.BatteryAwareHelloWorld/main.py "{configuration:/BatteryThreshold}"
 "{configuration:/LinuxBatteryFilePath}"
    Artifacts:
      - Uri: "s3://BUCKET_NAME/COMPONENT_NAME/COMPONENT_VERSION/
com.example.BatteryAwareHelloWorld.zip"
        Unarchive: ZIP
  - Platform:
      os: windows
    Lifecycle:
      install: py -3 -m pip install --user awsiotsdk --upgrade
      run: py -3 -u {artifacts:decompressedPath}/
com.example.BatteryAwareHelloWorld/main.py "{configuration:/BatteryThreshold}"
 "{configuration:/WindowsBatteryFilePath}"
    Artifacts:
      - Uri: "s3://BUCKET_NAME/COMPONENT_NAME/COMPONENT_VERSION/
com.example.BatteryAwareHelloWorld.zip"
        Unarchive: ZIP
```

Esta receita especifica o seguinte:

- Parâmetros de configuração padrão para o limite da bateria, o caminho do arquivo da bateria virtual nos dispositivos principais do Linux e o caminho do arquivo da bateria virtual nos dispositivos principais do Windows.
- Um `install` ciclo de vida que instala a versão mais recente da AWS IoT Device SDK v2 para Python.
- Um `run` ciclo de vida que executa o aplicativo Python em. `main.py`
- Espaços reservados, como `COMPONENT_NAME` e`COMPONENT_VERSION`, em que a CLI do GDK substitui as informações ao criar a receita do componente.

Para obter mais informações sobre receitas de componentes, consulteAWS IoT Greengrass referência da receita do componente.

Etapa 3: publicar o componente no AWS IoT Greengrass serviço

Nesta seção, você publica o componente Hello World no serviço de AWS IoT Greengrass nuvem. Depois que um componente estiver disponível no serviço de AWS IoT Greengrass nuvem, você poderá implantá-lo nos dispositivos principais. Você usa a CLI do GDK para publicar o componente

do seu computador de desenvolvimento no serviço de nuvemAWS IoT Greengrass. A CLI do GDK carrega a receita e os artefatos do componente para você.

Para publicar o componente Hello World no AWS IoT Greengrass serviço

1. Execute o comando a seguir para criar o componente usando a CLI do GDK. O comando de construção do componente cria uma receita e artefatos com base no arquivo de configuração da CLI do GDK. Nesse processo, a CLI do GDK cria um arquivo ZIP que contém o código-fonte do componente.

   ```
   gdk component build
   ```

   Você deve ver mensagens semelhantes ao exemplo a seguir.

   ```
   [2022-04-28 11:20:16] INFO - Getting project configuration from gdk-config.json
   [2022-04-28 11:20:16] INFO - Found component recipe file 'recipe.yaml' in the
    project directory.
   [2022-04-28 11:20:16] INFO - Building the component
    'com.example.BatteryAwareHelloWorld' with the given project configuration.
   [2022-04-28 11:20:16] INFO - Using 'zip' build system to build the component.
   [2022-04-28 11:20:16] WARNING - This component is identified as using 'zip' build
    system. If this is incorrect, please exit and specify custom build command in the
    'gdk-config.json'.
   [2022-04-28 11:20:16] INFO - Zipping source code files of the component.
   [2022-04-28 11:20:16] INFO - Copying over the build artifacts to the greengrass
    component artifacts build folder.
   [2022-04-28 11:20:16] INFO - Updating artifact URIs in the recipe.
   [2022-04-28 11:20:16] INFO - Creating component recipe in 'C:\Users\finthomp
   \greengrassv2\com.example.BatteryAwareHelloWorld\greengrass-build\recipes'.
   ```

2. Execute o comando a seguir para publicar o componente no serviço de AWS IoT Greengrass nuvem. O comando de publicação do componente carrega o artefato do arquivo ZIP do componente em um bucket do S3. Em seguida, ele atualiza o URI S3 do arquivo ZIP na receita do componente e carrega a receita no serviço. AWS IoT Greengrass Nesse processo, a CLI do GDK verifica qual versão do componente Hello World já está disponível AWS IoT Greengrass no serviço de nuvem, para que possa escolher a próxima versão do patch após essa versão. Se o componente ainda não existir, a CLI do GDK usa a versão. `1.0.0`

   ```
   gdk component publish
   ```

Você deve ver mensagens semelhantes ao exemplo a seguir. A saída informa a versão do componente que a CLI do GDK criou.

```
[2022-04-28 11:20:29] INFO - Getting project configuration from gdk-config.json
[2022-04-28 11:20:29] INFO - Found component recipe file 'recipe.yaml' in the
 project directory.
[2022-04-28 11:20:29] INFO - Found credentials in shared credentials file: ~/.aws/
credentials
[2022-04-28 11:20:30] INFO - No private version of the component
 'com.example.BatteryAwareHelloWorld' exist in the account. Using '1.0.0' as the
 next version to create.
[2022-04-28 11:20:30] INFO - Publishing the component
 'com.example.BatteryAwareHelloWorld' with the given project configuration.
[2022-04-28 11:20:30] INFO - Uploading the component built artifacts to s3 bucket.
[2022-04-28 11:20:30] INFO - Uploading component artifacts to S3
 bucket: greengrass-component-artifacts-us-west-2-123456789012. If this is your
 first time using this bucket, add the 's3:GetObject' permission to each core
 device's token exchange role to allow it to download the component artifacts. For
 more information, see https://docs.aws.amazon.com/greengrass/v2/developerguide/
device-service-role.html.
[2022-04-28 11:20:30] INFO - Not creating an artifacts bucket as it already exists.
[2022-04-28 11:20:30] INFO - Updating the component recipe
 com.example.BatteryAwareHelloWorld-1.0.0.
[2022-04-28 11:20:31] INFO - Creating a new greengrass component
 com.example.BatteryAwareHelloWorld-1.0.0
[2022-04-28 11:20:31] INFO - Created private version '1.0.0' of the component in
 the account.'com.example.BatteryAwareHelloWorld'.
```

3. Copie o nome do bucket S3 da saída. Você usa o nome do bucket posteriormente para permitir que o dispositivo principal baixe artefatos de componentes desse bucket.
4. (Opcional) Visualize o componente no AWS IoT Greengrass console para verificar se ele foi carregado com êxito. Faça o seguinte:

   a. No menu de navegação AWS IoT Greengrassdo console, escolha Componentes.

   b. Na página Componentes, escolha a guia Meus componentes e, em seguida, escolha com.example.BatteryAwareHelloWorld.

      Nesta página, você pode ver a receita do componente e outras informações sobre o componente.
5. Permita que o dispositivo principal acesse artefatos de componentes no bucket do S3.

Cada dispositivo principal tem uma função de IAM do dispositivo principal que permite interagir AWS IoT e enviar registros para a AWS nuvem. Essa função de dispositivo não permite acesso aos buckets do S3 por padrão, então você deve criar e anexar uma política que permita que o dispositivo principal recupere artefatos do componente do bucket do S3.

Se a função do seu dispositivo já permitir o acesso ao bucket do S3, você pode pular esta etapa. Caso contrário, crie uma política do IAM que permita acesso e anexe-a à função, da seguinte forma:

a. No menu de navegação do console do IAM, escolha Políticas e, em seguida, escolha Criar política.

b. Na guia JSON, substitua o conteúdo do espaço reservado pela política a seguir. Substitua *greengrass-component-artifacts-us-west-2-123456789012* pelo nome do bucket do S3 em que a CLI do GDK fez o upload dos artefatos do componente.

   Por exemplo, se você especificou **greengrass-component-artifacts** e está **us-west-2** no arquivo de configuração da CLI do GDK e Conta da AWS sua ID **123456789012** é, a CLI do GDK usa o bucket do S3 chamado. `greengrass-component-artifacts-us-west-2-123456789012`

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": "arn:aws:s3:::greengrass-component-artifacts-us-
west-2-123456789012/*"
    }
  ]
}
```

c. Escolha Próximo.

d. Na seção Detalhes da política, em Nome, insira**MyGreengrassV2ComponentArtifactPolicy**.

e. Escolha Criar política.

f. No menu de navegação do console do IAM, escolha Role e, em seguida, escolha o nome da função para o dispositivo principal. Você especificou esse nome de função ao instalar o software AWS IoT Greengrass Core. Se você não especificou um nome, o padrão será`GreengrassV2TokenExchangeRole`.

g. Em Permissões, escolha Adicionar permissões e, em seguida, escolha Anexar políticas.

h. Na página Adicionar permissões, marque a caixa de seleção ao lado da `MyGreengrassV2ComponentArtifactPolicy` política que você criou e escolha Adicionar permissões.

Etapa 4: implantar e testar o componente em um dispositivo principal

Nesta seção, você implanta o componente no dispositivo principal para testar sua funcionalidade. No dispositivo principal, você cria o arquivo virtual do nível da bateria para imitar uma bateria real. Em seguida, você cria implantações adicionais e observa os arquivos de log do componente no dispositivo principal para ver o componente adiar e confirmar as atualizações.

Para implantar e testar o componente Hello World que adia as atualizações

1. Use um editor de texto para criar um arquivo virtual de nível de bateria. Este arquivo imita uma bateria real.

   - Nos dispositivos principais do Linux, crie um arquivo chamado`/home/ggc_user/virtual_battery.json`. Execute o editor de texto com `sudo` permissões.
   - Nos dispositivos principais do Windows, crie um arquivo chamado`C:\Users\ggc_user\virtual_battery.json`. Execute o editor de texto como administrador.

   Por exemplo, em um sistema baseado em Linux, você pode executar o seguinte comando para usar o GNU nano para criar o arquivo.

   ```
   sudo nano /home/ggc_user/virtual_battery.json
   ```

   Copie o seguinte JSON no arquivo.

   ```
   {
     "battery_level": 50
   }
   ```

2. Implante o componente Hello World no dispositivo principal. Faça o seguinte:

   a. No menu de navegação AWS IoT Greengrassdo console, escolha Componentes.

   b. Na página Componentes, escolha a guia Meus componentes e, em seguida, escolha com.example.BatteryAwareHelloWorld.

   c. Na página com.example.BatteryAwareHelloWorld, escolha Implantar.

   d. Em Adicionar à implantação, escolha uma implantação existente para revisar ou opte por criar uma nova implantação e, em seguida, escolha Avançar.

   e. Se você optar por criar uma nova implantação, escolha o dispositivo principal ou grupo de itens de destino para a implantação. Na página Especificar destino, em Destino de implantação, escolha um dispositivo principal ou grupo de itens e, em seguida, escolha Avançar.

   f. Na página Selecionar componentes, verifique se o com.example.BatteryAwareHelloWorldcomponente está selecionado e escolha Avançar.

   g. Na página Configurar componentes com.example.BatteryAwareHelloWorld, selecione e faça o seguinte:

      i. Escolha Configurar componente.

      ii. No com.example.BatteryAwareHelloWorld modal Configurar, em Atualização de configuração, em Configuração para mesclar, insira a seguinte atualização de configuração.

      ```
      {
        "BatteryThreshold": 70
      }
      ```

      iii. Escolha Confirmar para fechar o modal e, em seguida, escolha Avançar.

   h. Na página Confirmar configurações avançadas, na seção Políticas de implantação, em Política de atualização de componentes, confirme se a opção Notificar componentes está selecionada. Notificar componentes é selecionado por padrão quando você cria uma nova implantação.

   i. Na página Review, escolha Deploy.

      A implantação pode levar até um minuto para ser concluída.

3. O software AWS IoT Greengrass Core salva o stdout dos processos dos componentes em arquivos de log na `logs` pasta. Execute o comando a seguir para verificar se o componente Hello World é executado e imprime mensagens de status.

   Linux or Unix

   ```
   sudo tail -f /greengrass/v2/logs/com.example.BatteryAwareHelloWorld.log
   ```

   Windows Command Prompt (CMD)

   ```
   type C:\greengrass\v2\logs\com.example.BatteryAwareHelloWorld.log
   ```

   PowerShell

   ```
   gc C:\greengrass\v2\logs\com.example.BatteryAwareHelloWorld.log -Tail 10 -Wait
   ```

   Você deve ver mensagens semelhantes ao exemplo a seguir.

   ```
   Hello, World! Battery level (50) is below threshold (70), so the component will
    defer updates.
   ```

   > **Note**
   >
   > Se o arquivo não existir, a implantação talvez ainda não tenha sido concluída. Se o arquivo não existir em 30 segundos, a implantação provavelmente falhou. Isso pode ocorrer se o dispositivo principal não tiver permissão para baixar os artefatos do componente do bucket do S3, por exemplo. Execute o comando a seguir para visualizar o arquivo de log do software AWS IoT Greengrass principal. Esse arquivo inclui registros do serviço de implantação do dispositivo principal do Greengrass.
   >
   > Linux or Unix
   >
   > ```
   > sudo tail -f /greengrass/v2/logs/greengrass.log
   > ```
   >
   > Windows Command Prompt (CMD)
   >
   > ```
   > type C:\greengrass\v2\logs\greengrass.log
   > ```

O `type` comando grava o conteúdo do arquivo no terminal. Execute esse comando várias vezes para observar as alterações no arquivo.

PowerShell

```
gc C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
```

4. Crie uma nova implantação no dispositivo principal para verificar se o componente adia a atualização. Faça o seguinte:

   a. No menu de navegação AWS IoT Greengrassdo console, escolha Implantações.

   b. Escolha a implantação que você criou ou revisou anteriormente.

   c. Na página de implantação, escolha Revisar.

   d. No modal Revise deployment, escolha Revise deployment.

   e. Escolha Avançar em cada etapa e, em seguida, escolha Implantar.

5. Execute o comando a seguir para visualizar novamente os registros do componente e verificar se ele adia a atualização.

   Linux or Unix

   ```
   sudo tail -f /greengrass/v2/logs/com.example.BatteryAwareHelloWorld.log
   ```

   Windows Command Prompt (CMD)

   ```
   type C:\greengrass\v2\logs\com.example.BatteryAwareHelloWorld.log
   ```

   PowerShell

   ```
   gc C:\greengrass\v2\logs\com.example.BatteryAwareHelloWorld.log -Tail 10 -Wait
   ```

   Você deve ver mensagens semelhantes ao exemplo a seguir. O componente adia a atualização por 30 segundos, então o componente imprime essa mensagem repetidamente.

   ```
   Deferred update for deployment 50722a95-a05f-4e2a-9414-da80103269aa.
   ```

6. Use um editor de texto para editar o arquivo virtual do nível da bateria e alterar o nível da bateria para um valor acima do limite, para que a implantação possa continuar.

   - Nos dispositivos principais do Linux, edite o arquivo chamado`/home/ggc_user/virtual_battery.json`. Execute o editor de texto com `sudo` permissões.
   - Nos dispositivos principais do Windows, edite o arquivo chamado`C:\Users\ggc_user\virtual_battery.json`. Execute o editor de texto como administrador.

   Por exemplo, em um sistema baseado em Linux, você pode executar o seguinte comando para usar o GNU nano para criar o arquivo.

   ```
   sudo nano /home/ggc_user/virtual_battery.json
   ```

   Altere o nível da bateria para`80`.

   ```
   {
     "battery_level": 80
   }
   ```

7. Execute o comando a seguir para visualizar novamente os registros do componente e verificar se ele reconhece a atualização.

   Linux or Unix

   ```
   sudo tail -f /greengrass/v2/logs/com.example.BatteryAwareHelloWorld.log
   ```

   Windows Command Prompt (CMD)

   ```
   type C:\greengrass\v2\logs\com.example.BatteryAwareHelloWorld.log
   ```

   PowerShell

   ```
   gc C:\greengrass\v2\logs\com.example.BatteryAwareHelloWorld.log -Tail 10 -Wait
   ```

   Você deve ver mensagens semelhantes aos exemplos a seguir.

```
Hello, World! Battery level (80) is above threshold (70), so the component will
 acknowledge updates.
Acknowledged update for deployment f9499eb2-4a40-40a7-86c1-c89887d859f1.
```

Você concluiu este tutorial. O componente Hello World adia ou confirma as atualizações com base no nível da bateria do dispositivo principal. Para obter mais informações sobre os tópicos que este tutorial explora, consulte o seguinte:

- Desenvolva AWS IoT Greengrass componentes
- Implemente AWS IoT Greengrass componentes em dispositivos
- Use o AWS IoT Device SDK para se comunicar com o núcleo do Greengrass, outros componentes e AWS IoT Core
- AWS IoT GreengrassInterface de linha de comando do kit de desenvolvimento

Tutorial: Interaja com dispositivos locais de IoT por meio do MQTT

Você pode concluir este tutorial para configurar um dispositivo principal para interagir com dispositivos IoT locais, chamados de dispositivos cliente, que se conectam ao dispositivo principal por meio do MQTT. Neste tutorial, você configura AWS IoT as coisas para usar a descoberta na nuvem para se conectar ao dispositivo principal como dispositivos clientes. Quando você configura a descoberta na nuvem, um dispositivo cliente pode enviar uma solicitação ao serviço de AWS IoT Greengrass nuvem para descobrir os dispositivos principais. A resposta de AWS IoT Greengrass inclui informações de conectividade e certificados para os dispositivos principais que você configura o dispositivo cliente para descobrir. Em seguida, o dispositivo cliente pode usar essas informações para se conectar a um dispositivo principal disponível, onde ele pode se comunicar pelo MQTT.

Neste tutorial, você faz o seguinte:

1. Revise e atualize as permissões do dispositivo principal, se necessário.
2. Associe dispositivos cliente ao dispositivo principal para que eles possam descobrir o dispositivo principal usando a descoberta na nuvem.
3. Implante componentes do Greengrass no dispositivo principal para permitir o suporte ao dispositivo cliente.
4. Conecte dispositivos cliente ao dispositivo principal e teste a comunicação com o serviço de AWS IoT Core nuvem.

5. Desenvolva um componente personalizado do Greengrass que se comunique com os dispositivos do cliente.
6. Desenvolva um componente personalizado que interaja com as sombras do dispositivo cliente. AWS IoT

Este tutorial usa um único dispositivo central e um único dispositivo cliente. Você também pode seguir o tutorial para conectar e testar vários dispositivos clientes.

Você pode esperar passar de 30 a 60 minutos neste tutorial.

Pré-requisitos

Para concluir este tutorial, você precisará do seguinte:

- Uma Conta da AWS. Se você não tiver uma, consulte Configurar um Conta da AWS.
- Um usuário AWS Identity and Access Management (IAM) com permissões de administrador.
- Um dispositivo principal do Greengrass. Para obter mais informações sobre como configurar um dispositivo principal, consulteConfigurando dispositivos AWS IoT Greengrass principais.
  - O dispositivo principal deve executar o Greengrass nucleus v2.6.0 ou posterior. Essa versão inclui suporte para curingas na comunicação local de publicação/assinatura e suporte para sombras de dispositivos clientes.

    > **Note**
    >
    > O suporte ao dispositivo cliente requer o Greengrass nucleus v2.2.0 ou posterior. No entanto, este tutorial explora recursos mais novos, como suporte para curingas MQTT na publicação/assinatura local e suporte para sombras de dispositivos clientes. Esses recursos exigem o Greengrass nucleus v2.6.0 ou posterior.

  - O dispositivo principal deve estar na mesma rede que os dispositivos cliente para se conectar.
  - (Opcional) Para concluir os módulos nos quais você desenvolve componentes personalizados do Greengrass, o dispositivo principal deve executar a CLI do Greengrass. Para ter mais informações, consulte Instale a CLI do Greengrass.
- Qualquer AWS IoT coisa para se conectar como um dispositivo cliente neste tutorial. Para obter mais informações, consulte Criar AWS IoT recursos no Guia do AWS IoT Core desenvolvedor.
  - A AWS IoT política do dispositivo cliente deve permitir a `greengrass:Discover` permissão. Para ter mais informações, consulte AWS IoTPolítica mínima para dispositivos clientes.

- O dispositivo cliente deve estar na mesma rede que o dispositivo principal.
- O dispositivo cliente deve executar o Python 3.
- O dispositivo cliente deve executar o Git.

Etapa 1: revisar e atualizar a AWS IoT política do dispositivo principal

Para oferecer suporte a dispositivos cliente, a AWS IoT política de um dispositivo principal deve permitir as seguintes permissões:

- `greengrass:PutCertificateAuthorities`
- `greengrass:VerifyClientDeviceIdentity`
- `greengrass:VerifyClientDeviceIoTCertificateAssociation`
- `greengrass:GetConnectivityInfo`
- `greengrass:UpdateConnectivityInfo`— (Opcional) Essa permissão é necessária para usar o componente detector de IP, que reporta as informações de conectividade de rede do dispositivo principal ao serviço de AWS IoT Greengrass nuvem.

Para obter mais informações sobre essas permissões e AWS IoT políticas para dispositivos principais, consulte Políticas do AWS IoT para operações de plano de dados AWS IoTPolítica mínima para oferecer suporte aos dispositivos do cliente e.

Nesta seção, você revisa as AWS IoT políticas do seu dispositivo principal e adiciona as permissões necessárias que estão faltando. Se você usou o instalador de software AWS IoT Greengrass Core para provisionar recursos, seu dispositivo principal tem uma AWS IoT política que permite acesso a todas as AWS IoT Greengrass ações (`greengrass:*`). Nesse caso, você deve atualizar a AWS IoT política somente se planeja configurar o componente do gerenciador de sombras para sincronizar as sombras do dispositivo. AWS IoT Core Caso contrário, você pode pular esta seção.

Para revisar e atualizar a AWS IoT política de um dispositivo principal

1. No menu de navegação AWS IoT Greengrassdo console, escolha Dispositivos principais.
2. Na página Dispositivos principais, escolha o dispositivo principal a ser atualizado.
3. Na página de detalhes do dispositivo principal, escolha o link para o item do dispositivo principal. Esse link abre a página de detalhes do item no AWS IoT console.
4. Na página de detalhes do item, escolha Certificados.

5. Na guia Certificados, escolha o certificado ativo do item.
6. Na página de detalhes do certificado, escolha Políticas.
7. Na guia Políticas, escolha a AWS IoT política a ser revisada e atualizada. Você pode adicionar as permissões necessárias a qualquer política anexada ao certificado ativo do dispositivo principal.

   > **Note**
   >
   > Se você usou o instalador de software AWS IoT Greengrass Core para provisionar recursos, você tem duas AWS IoT políticas. Recomendamos que você escolha a política nomeada GreengrassV2IoTThingPolicy, se ela existir. Os dispositivos principais que você cria com o instalador rápido usam esse nome de política por padrão. Se você adicionar permissões a essa política, também estará concedendo essas permissões a outros dispositivos principais que usam essa política.

8. Na visão geral da política, escolha Editar versão ativa.
9. Revise a política para obter as permissões necessárias e adicione as permissões necessárias que estejam faltando.
10. Para definir uma nova versão da política como a versão ativa, em Status da versão da política, selecione Definir a versão editada como a versão ativa desta política.
11. Selecione Salvar como nova versão.

Etapa 2: ativar o suporte ao dispositivo cliente

Para que um dispositivo cliente use a descoberta na nuvem para se conectar a um dispositivo principal, você deve associar os dispositivos. Ao associar um dispositivo cliente a um dispositivo principal, você permite que esse dispositivo cliente recupere os endereços IP e certificados do dispositivo principal para usar na conexão.

Para permitir que os dispositivos cliente se conectem com segurança a um dispositivo principal e se comuniquem com os componentes do GreengrassAWS IoT Core, você implanta os seguintes componentes do Greengrass no dispositivo principal:

- Autenticação do dispositivo cliente (`aws.greengrass.clientdevices.Auth`)

Implante o componente de autenticação do dispositivo cliente para autenticar dispositivos cliente e autorizar ações do dispositivo cliente. Esse componente permite que suas AWS IoT coisas se conectem a um dispositivo principal.

Esse componente requer alguma configuração para ser usado. Você deve especificar grupos de dispositivos clientes e as operações que cada grupo está autorizado a realizar, como se conectar e se comunicar pelo MQTT. Para obter mais informações, consulte Configuração do componente de autenticação do dispositivo cliente.

- Corretor MQTT 3.1.1 (Moquette) (`aws.greengrass.clientdevices.mqtt.Moquette`)

  Implante o componente Moquette MQTT broker para executar um broker MQTT leve. O agente Moquette MQTT é compatível com o MQTT 3.1.1 e inclui suporte local para QoS 0, QoS 1, QoS 2, mensagens retidas, mensagens de última vontade e assinaturas persistentes.

  Você não precisa configurar esse componente para usá-lo. No entanto, você pode configurar a porta em que esse componente opera o agente MQTT. Por padrão, ele usa a porta 8883.

- Ponte MQTT (`aws.greengrass.clientdevices.mqtt.Bridge`)

  (Opcional) Implante o componente de ponte MQTT para retransmitir mensagens entre dispositivos cliente (MQTT local), publicação/assinatura local e MQTT. AWS IoT Core Configure esse componente para sincronizar dispositivos cliente AWS IoT Core e interagir com dispositivos clientes a partir dos componentes do Greengrass.

  Esse componente requer configuração para ser usado. Você deve especificar os mapeamentos de tópicos em que esse componente retransmite mensagens. Para obter mais informações, consulte Configuração do componente de ponte MQTT.

- Detector IP (`aws.greengrass.clientdevices.IPDetector`)

  (Opcional) Implante o componente detector de IP para reportar automaticamente os endpoints do broker MQTT do dispositivo principal ao serviço de AWS IoT Greengrass nuvem. Você não pode usar esse componente se tiver uma configuração de rede complexa, como aquela em que um roteador encaminha a porta do agente MQTT para o dispositivo principal.

  Você não precisa configurar esse componente para usá-lo.

Nesta seção, você usa o AWS IoT Greengrass console para associar dispositivos cliente e implantar componentes do dispositivo cliente em um dispositivo principal.

Para habilitar o suporte ao dispositivo cliente

1. Navegue até o console do AWS IoT Greengrass.
2. No menu de navegação à esquerda, escolha Dispositivos principais.
3. Na página Dispositivos principais, escolha o dispositivo principal em que você deseja ativar o suporte ao dispositivo cliente.
4. Na página de detalhes do dispositivo principal, escolha a guia Dispositivos clientes.
5. Na guia Dispositivos do cliente, escolha Configurar descoberta na nuvem.

   A página Configurar descoberta do dispositivo principal é aberta. Nesta página, você pode associar dispositivos cliente a um dispositivo principal e implantar componentes do dispositivo cliente. Esta página seleciona o dispositivo principal para você na Etapa 1: Selecione os dispositivos principais de destino.

   > **Note**
   >
   > Você também pode usar esta página para configurar a descoberta de dispositivos principais para um grupo de coisas. Se você escolher essa opção, poderá implantar componentes do dispositivo cliente em todos os dispositivos principais em um grupo de coisas. No entanto, se você escolher essa opção, deverá associar manualmente os dispositivos cliente a cada dispositivo principal depois de criar a implantação. Neste tutorial, você configura um dispositivo de núcleo único.

6. Na Etapa 2: Associar dispositivos cliente, associe a AWS IoT coisa do dispositivo cliente ao dispositivo principal. Isso permite que o dispositivo cliente use a descoberta na nuvem para recuperar as informações e os certificados de conectividade do dispositivo principal. Faça o seguinte:

   a. Escolha Associar dispositivos cliente.
   b. No modal Associar dispositivos cliente ao dispositivo principal, insira o nome da AWS IoT coisa a ser associada.
   c. Escolha Adicionar.
   d. Selecione Associar.
7. Na Etapa 3: Configurar e implantar componentes do Greengrass, implante componentes para habilitar o suporte ao dispositivo cliente. Se o dispositivo principal de destino tiver uma implantação anterior, esta página revisará essa implantação. Caso contrário, essa página criará

uma nova implantação para o dispositivo principal. Faça o seguinte para configurar e implantar os componentes do dispositivo cliente:

a. O dispositivo principal deve executar o Greengrass nucleus v2.6.0 ou posterior para concluir este tutorial. Se o dispositivo principal executar uma versão anterior, faça o seguinte:

   i. Selecione a caixa para implantar o aws.greengrass.Nucleuscomponente.

   ii. Para o aws.greengrass.Nucleuscomponente, escolha Editar configuração.

   iii. Para a versão do componente, escolha a versão 2.6.0 ou posterior.

   iv. Selecione a opção Confirmar.

> **Note**
>
> Se você atualizar o núcleo do Greengrass de uma versão secundária anterior e o dispositivo principal executar componentes AWS fornecidos que dependem do núcleo, você também deverá atualizar os AWS componentes fornecidos para versões mais recentes. Você pode configurar a versão desses componentes ao analisar a implantação posteriormente neste tutorial. Para ter mais informações, consulte Atualize o software AWS IoT Greengrass principal (OTA).

b. Para o aws.greengrass.clientdevices.Authcomponente, escolha Editar configuração.

c. No modal Editar configuração para o componente de autenticação do dispositivo cliente, configure uma política de autorização que permita que os dispositivos cliente publiquem e assinem o agente MQTT no dispositivo principal. Faça o seguinte:

   i. Em Configuração, no bloco de código Configuração para mesclar, insira a configuração a seguir, que contém uma política de autorização de dispositivo cliente. Cada política de autorização de grupo de dispositivos especifica um conjunto de ações e os recursos nos quais um dispositivo cliente pode realizar essas ações.

   - Essa política permite que dispositivos clientes cujos nomes comecem com se conectem e `MyClientDevice` se comuniquem em todos os tópicos do MQTT. Substitua *`MyClientDevice*`* pelo nome da AWS IoT coisa a ser conectada como dispositivo cliente. Você também pode especificar um nome com o * caractere curinga que corresponda ao nome do dispositivo cliente. O * caractere curinga deve estar no final do nome.

Se você tiver um segundo dispositivo cliente para se conectar, substitua *MyOtherClientDevice** pelo nome desse dispositivo cliente ou por um padrão curinga que corresponda ao nome desse dispositivo cliente. Caso contrário, você pode remover ou manter essa seção da regra de seleção que permite que dispositivos clientes com nomes correspondentes `MyOtherClientDevice*` se conectem e se comuniquem.

- Essa política usa um `OR` operador para também permitir que dispositivos clientes cujos nomes comecem com `MyOtherClientDevice` se conectem e se comuniquem em todos os tópicos do MQTT. Você pode remover essa cláusula na regra de seleção ou modificá-la para corresponder aos dispositivos cliente a serem conectados.
- Essa política permite que os dispositivos do cliente publiquem e se inscrevam em todos os tópicos do MQTT. Para seguir as melhores práticas de segurança, restrinja as `mqtt:subscribe` operações `mqtt:publish` e ao conjunto mínimo de tópicos que os dispositivos cliente usam para se comunicar.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice* OR
 thingName: MyOtherClientDevice*",
        "policyName": "MyClientDevicePolicy"
      }
    },
    "policies": {
      "MyClientDevicePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
```

```
          "statementDescription": "Allow client devices to publish to all
 topics.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to all
 topics.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
}
```

Para obter mais informações, consulte Configuração do componente de autenticação do dispositivo cliente.

ii. Selecione a opção Confirmar.

d. Para o aws.greengrass.clientdevices.mqtt.Bridgecomponente, escolha Editar configuração.

e. No modal Editar configuração para o componente de ponte MQTT, configure um mapeamento de tópicos que retransmita mensagens MQTT dos dispositivos do cliente para o. AWS IoT Core Faça o seguinte:

i. Em Configuração, no bloco Configuração para mesclar código, insira a configuração a seguir. Essa configuração especifica a retransmissão de mensagens MQTT no filtro de `clients/+/hello/world` tópicos dos dispositivos do cliente para o AWS IoT Core serviço de nuvem. Por exemplo, esse filtro de tópico corresponde ao `clients/MyClientDevice1/hello/world` tópico.

```
{
  "mqttTopicMapping": {
```

```
    "HelloWorldIotCoreMapping": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

Para obter mais informações, consulte Configuração do componente de ponte MQTT.

ii. Selecione a opção Confirmar.

8. Escolha Revisar e implantar para revisar a implantação que esta página cria para você.
9. Se você ainda não configurou a função de serviço do Greengrass nessa região, o console abre um modal para configurar a função de serviço para você. O componente de autenticação do dispositivo cliente usa essa função de serviço para verificar a identidade dos dispositivos cliente, e o componente detector de IP usa essa função de serviço para gerenciar as principais informações de conectividade do dispositivo. Escolha Grant permissions (Conceder permissões).
10. Na página Revisar, escolha Implantar para iniciar a implantação no dispositivo principal.
11. Para verificar se a implantação foi bem-sucedida, verifique o status da implantação e verifique os registros no dispositivo principal. Para verificar o status da implantação no dispositivo principal, você pode escolher Target na Visão geral da implantação. Para ver mais informações, consulte:

    - Verificar status da implantação
    - Monitore AWS IoT Greengrass os registros

Etapa 3: Conectar dispositivos cliente

Os dispositivos cliente podem usar o AWS IoT Device SDK para descobrir, conectar e se comunicar com um dispositivo principal. O dispositivo cliente deve ser AWS IoT alguma coisa. Para obter mais informações, consulte Criar um objeto no Guia do AWS IoT Core desenvolvedor.

Nesta seção, você instala a AWS IoT Device SDKv2 para Python e executa o aplicativo de amostra Greengrass discovery a partir do. AWS IoT Device SDK

> **Note**
>
> Também AWS IoT Device SDK está disponível em outras linguagens de programação. Este tutorial usa a AWS IoT Device SDK v2 para Python, mas você pode explorar os outros SDKs

para seu caso de uso. Para obter mais informações, consulte Dispositivos SDK AWS IoT no Guia do desenvolvedor do AWS IoT Core.

Para conectar um dispositivo cliente a um dispositivo principal

1. Baixe e instale a AWS IoT Device SDKv2 para Python AWS IoT na coisa para se conectar como um dispositivo cliente.

   No dispositivo cliente, faça o seguinte:

   a. Clone o repositório AWS IoT Device SDK v2 for Python para baixá-lo.

   ```
   git clone https://github.com/aws/aws-iot-device-sdk-python-v2.git
   ```

   b. Instale a AWS IoT Device SDK v2 para Python.

   ```
   python3 -m pip install --user ./aws-iot-device-sdk-python-v2
   ```

2. Mude para a pasta de amostras na AWS IoT Device SDK v2 para Python.

   ```
   cd aws-iot-device-sdk-python-v2/samples
   ```

3. Execute o aplicativo de descoberta Greengrass de amostra. Esse aplicativo espera argumentos que especifiquem o nome do dispositivo cliente, o tópico e a mensagem do MQTT a serem usados e os certificados que autenticam e protegem a conexão. O exemplo a seguir envia uma mensagem Hello World para o `clients/`*`MyClientDevice1`*`/hello/world` tópico.

   > **Note**
   >
   > Este tópico corresponde ao tópico para o qual você configurou a ponte MQTT para retransmitir mensagens anteriores. AWS IoT Core

   - Substitua *`MyClientDevice1`* pelo nome do item do dispositivo cliente.
   - Substitua *`~/certs/ AmazonRoot CA1.pem`* pelo caminho para o certificado CA raiz da Amazon no dispositivo cliente.
   - Substitua *`~/certs/device.pem.crt`* pelo caminho para o certificado do dispositivo no dispositivo cliente.

- Substitua *~/certs/private.pem.key pelo caminho para o arquivo de chave* privada no dispositivo cliente.
- Substitua *us-east-1* pela AWS região em que seu dispositivo cliente e dispositivo principal operam.

```
python3 basic_discovery.py \\
  --thing_name MyClientDevice1 \\
  --topic 'clients/MyClientDevice1/hello/world' \\
  --message 'Hello World!' \\
  --ca_file ~/certs/AmazonRootCA1.pem \\
  --cert ~/certs/device.pem.crt \\
  --key ~/certs/private.pem.key \\
  --region us-east-1 \\
  --verbosity Warn
```

O aplicativo de amostra de descoberta envia a mensagem 10 vezes e se desconecta. Ele também se inscreve no mesmo tópico em que publica mensagens. Se a saída indicar que o aplicativo recebeu mensagens MQTT sobre o tópico, o dispositivo cliente poderá se comunicar com êxito com o dispositivo principal.

```
Performing greengrass discovery...
awsiot.greengrass_discovery.DiscoverResponse(gg_groups=[awsiot.greengrass_discovery.GGGroup
coreDevice-MyGreengrassCore',
 cores=[awsiot.greengrass_discovery.GGCore(thing_arn='arn:aws:iot:us-
east-1:123456789012:thing/MyGreengrassCore',
 connectivity=[awsiot.greengrass_discovery.ConnectivityInfo(id='203.0.113.0',
 host_address='203.0.113.0', metadata='', port=8883)])],
 certificate_authorities=['-----BEGIN CERTIFICATE-----\
MIICiT...EXAMPLE=\
-----END CERTIFICATE-----\
'])])
Trying core arn:aws:iot:us-east-1:123456789012:thing/MyGreengrassCore at host
 203.0.113.0 port 8883
Connected!
Published topic clients/MyClientDevice1/hello/world: {"message": "Hello World!",
 "sequence": 0}

Publish received on topic clients/MyClientDevice1/hello/world
b'{"message": "Hello World!", "sequence": 0}'
```

```
Published topic clients/MyClientDevice1/hello/world: {"message": "Hello World!",
 "sequence": 1}

Publish received on topic clients/MyClientDevice1/hello/world
b'{"message": "Hello World!", "sequence": 1}'

...

Published topic clients/MyClientDevice1/hello/world: {"message": "Hello World!",
 "sequence": 9}

Publish received on topic clients/MyClientDevice1/hello/world
b'{"message": "Hello World!", "sequence": 9}'
```

Se, em vez disso, o aplicativo gerar um erro, consulte Solução de problemas de descoberta do Greengrass.

Você também pode ver os registros do Greengrass no dispositivo principal para verificar se o dispositivo cliente se conecta e envia mensagens com sucesso. Para ter mais informações, consulte Monitore AWS IoT Greengrass os registros.

4. Verifique se a ponte MQTT retransmite as mensagens do dispositivo cliente para o. AWS IoT Core Você pode usar o cliente de teste MQTT no AWS IoT Core console para assinar um filtro de tópicos do MQTT. Faça o seguinte:

   a. Navegue até o console do AWS IoT.

   b. No menu de navegação à esquerda, em Teste, escolha Cliente de teste MQTT.

   c. Na guia Inscrever-se em um tópico, em Filtro de tópicos, insira `clients/+/hello/world` para assinar as mensagens do dispositivo cliente do dispositivo principal.

   d. Escolha Assinar.

   e. Execute novamente o aplicativo de publicação/assinatura no dispositivo cliente.

      O cliente de teste MQTT exibe as mensagens que você envia do dispositivo cliente em tópicos que correspondem a esse filtro de tópicos.

Etapa 4: desenvolver um componente que se comunique com os dispositivos do cliente

Você pode desenvolver componentes do Greengrass que se comunicam com os dispositivos do cliente. Os componentes usam comunicação entre processos (IPC) e a interface local de publicação/assinatura para se comunicar em um dispositivo principal. Para interagir com dispositivos clientes, configure o componente de ponte MQTT para retransmitir mensagens entre dispositivos clientes e a interface local de publicação/assinatura.

Nesta seção, você atualiza o componente de ponte MQTT para retransmitir mensagens dos dispositivos cliente para a interface local de publicação/assinatura. Em seguida, você desenvolve um componente que assina essas mensagens e as imprime quando as recebe.

Desenvolver um componente que se comunica com os dispositivos do cliente

1. Revise a implantação no dispositivo principal e configure o componente de ponte MQTT para retransmitir mensagens dos dispositivos do cliente para a publicação/assinatura local. Faça o seguinte:

   a. Navegue até o console do AWS IoT Greengrass.

   b. No menu de navegação à esquerda, escolha Dispositivos principais.

   c. Na página Dispositivos principais, escolha o dispositivo principal que você está usando para este tutorial.

   d. Na página de detalhes do dispositivo principal, escolha a guia Dispositivos clientes.

   e. Na guia Dispositivos do cliente, escolha Configurar descoberta na nuvem.

      A página Configurar descoberta do dispositivo principal é aberta. Nesta página, você pode alterar ou configurar quais componentes do dispositivo cliente são implantados no dispositivo principal.

   f. Na Etapa 3, para o aws.greengrass.clientdevices.mqtt.Bridgecomponente, escolha Editar configuração.

   g. No modal Editar configuração para o componente de ponte MQTT, configure um mapeamento de tópicos que retransmita mensagens MQTT dos dispositivos do cliente para a interface local de publicação/assinatura. Faça o seguinte:

      i. Em Configuração, no bloco Configuração para mesclar código, insira a configuração a seguir. Essa configuração especifica a retransmissão de mensagens MQTT em tópicos

que correspondam ao filtro de tópicos dos `clients/+/hello/world` dispositivos do cliente para o serviço de AWS IoT Core nuvem e para o agente local de publicação/assinatura do Greengrass.

```
{
  "mqttTopicMapping": {
    "HelloWorldIotCoreMapping": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    },
    "HelloWorldPubsubMapping": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "Pubsub"
    }
  }
}
```

Para obter mais informações, consulte Configuração do componente de ponte MQTT.

ii. Selecione a opção Confirmar.

h. Escolha Revisar e implantar para revisar a implantação que esta página cria para você.

i. Na página Revisar, escolha Implantar para iniciar a implantação no dispositivo principal.

j. Para verificar se a implantação foi bem-sucedida, verifique o status da implantação e verifique os registros no dispositivo principal. Para verificar o status da implantação no dispositivo principal, você pode escolher Target na Visão geral da implantação. Para ver mais informações, consulte:

- Verificar status da implantação
- Monitore AWS IoT Greengrass os registros

2. Desenvolva e implante um componente do Greengrass que assine mensagens do Hello World a partir de dispositivos clientes. Faça o seguinte:

a. Crie pastas para receitas e artefatos no dispositivo principal.

Linux or Unix

```
mkdir recipes
```

```
mkdir -p artifacts/com.example.clientdevices.MyHelloWorldSubscriber/1.0.0
```

Windows Command Prompt (CMD)

```
mkdir recipes
mkdir artifacts\com.example.clientdevices.MyHelloWorldSubscriber\1.0.0
```

PowerShell

```
mkdir recipes
mkdir artifacts\com.example.clientdevices.MyHelloWorldSubscriber\1.0.0
```

> **Important**
>
> Você deve usar o seguinte formato para o caminho da pasta de artefatos. Inclua o nome e a versão do componente que você especificar na receita.
>
> ```
> artifacts/componentName/componentVersion/
> ```

b. Use um editor de texto para criar uma receita de componente com o conteúdo a seguir. Esta receita especifica a instalação da AWS IoT Device SDK v2 para Python e a execução de um script que assina o tópico e imprime mensagens.

Por exemplo, em um sistema baseado em Linux, você pode executar o seguinte comando para usar o GNU nano para criar o arquivo.

```
nano recipes/com.example.clientdevices.MyHelloWorldSubscriber-1.0.0.json
```

Copie a seguinte receita para o arquivo.

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.clientdevices.MyHelloWorldSubscriber",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that subscribes to Hello World messages
 from client devices.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
```

```
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.clientdevices.MyHelloWorldSubscriber:pubsub:1": {
            "policyDescription": "Allows access to subscribe to all topics.",
            "operations": [
              "aws.greengrass#SubscribeToTopic"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "python3 -m pip install --user awsiotsdk",
        "run": "python3 -u {artifacts:path}/hello_world_subscriber.py"
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "install": "py -3 -m pip install --user awsiotsdk",
        "run": "py -3 -u {artifacts:path}/hello_world_subscriber.py"
      }
    }
  ]
}
```

c. Use um editor de texto para criar um artefato de script Python chamado `hello_world_subscriber.py` com o conteúdo a seguir. Esse aplicativo usa o serviço IPC de publicação/assinatura para assinar o `clients/+/hello/world` tópico e imprimir as mensagens recebidas.

Por exemplo, em um sistema baseado em Linux, você pode executar o seguinte comando para usar o GNU nano para criar o arquivo.

```
nano artifacts/com.example.clientdevices.MyHelloWorldSubscriber/1.0.0/
hello_world_subscriber.py
```

Copie o código Python a seguir no arquivo.

```
import sys
import time
import traceback

from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2

CLIENT_DEVICE_HELLO_WORLD_TOPIC = 'clients/+/hello/world'
TIMEOUT = 10


def on_hello_world_message(event):
    try:
        message = str(event.binary_message.message, 'utf-8')
        print('Received new message: %s' % message)
    except:
        traceback.print_exc()


try:
    ipc_client = GreengrassCoreIPCClientV2()

    # SubscribeToTopic returns a tuple with the response and the operation.
    _, operation = ipc_client.subscribe_to_topic(
        topic=CLIENT_DEVICE_HELLO_WORLD_TOPIC,
 on_stream_event=on_hello_world_message)
    print('Successfully subscribed to topic: %s' %
          CLIENT_DEVICE_HELLO_WORLD_TOPIC)

    # Keep the main thread alive, or the process will exit.
    try:
        while True:
            time.sleep(10)
    except InterruptedError:
        print('Subscribe interrupted.')
```

```
    operation.close()
except Exception:
    print('Exception occurred when using IPC.', file=sys.stderr)
    traceback.print_exc()
    exit(1)
```

> **Note**
>
> Esse componente usa o cliente IPC V2 na v2 AWS IoT Device SDKpara Python para se comunicar com o software Core. AWS IoT Greengrass Em comparação com o cliente IPC original, o cliente IPC V2 reduz a quantidade de código que você precisa escrever para usar o IPC em componentes personalizados.

d. Use a CLI do Greengrass para implantar o componente.

Linux or Unix

```
sudo /greengrass/v2/bin/greengrass-cli deployment create \
  --recipeDir recipes \
  --artifactDir artifacts \
  --merge "com.example.clientdevices.MyHelloWorldSubscriber=1.0.0"
```

Windows Command Prompt (CMD)

```
C:\greengrass\v2/bin/greengrass-cli deployment create ^
  --recipeDir recipes ^
  --artifactDir artifacts ^
  --merge "com.example.clientdevices.MyHelloWorldSubscriber=1.0.0"
```

PowerShell

```
C:\greengrass\v2/bin/greengrass-cli deployment create `
  --recipeDir recipes `
  --artifactDir artifacts `
  --merge "com.example.clientdevices.MyHelloWorldSubscriber=1.0.0"
```

3. Visualize os registros do componente para verificar se o componente foi instalado com êxito e se inscreveu no tópico.

Linux or Unix

```
sudo tail -f /greengrass/v2/logs/
com.example.clientdevices.MyHelloWorldSubscriber.log
```

PowerShell

```
gc C:\greengrass\v2/logs/com.example.clientdevices.MyHelloWorldSubscriber.log -
Tail 10 -Wait
```

Você pode manter o feed de registro aberto para verificar se o dispositivo principal recebe mensagens.

4. No dispositivo cliente, execute novamente o aplicativo de descoberta Greengrass de amostra para enviar mensagens ao dispositivo principal.

```
python3 basic_discovery.py \\
  --thing_name MyClientDevice1 \\
  --topic 'clients/MyClientDevice1/hello/world' \\
  --message 'Hello World!' \\
  --ca_file ~/certs/AmazonRootCA1.pem \\
  --cert ~/certs/device.pem.crt \\
  --key ~/certs/private.pem.key \\
  --region us-east-1 \\
  --verbosity Warn
```

5. Visualize os registros do componente novamente para verificar se o componente recebe e imprime as mensagens do dispositivo cliente.

Linux or Unix

```
sudo tail -f /greengrass/v2/logs/
com.example.clientdevices.MyHelloWorldSubscriber.log
```

PowerShell

```
gc C:\greengrass\v2/logs/com.example.clientdevices.MyHelloWorldSubscriber.log -
Tail 10 -Wait
```

Etapa 5: desenvolver um componente que interaja com as sombras do dispositivo cliente

Você pode desenvolver componentes do Greengrass que interajam com as sombras do AWS IoTdispositivo cliente. Uma sombra é um documento JSON que armazena as informações de estado atuais ou desejadas de AWS IoT algo, como um dispositivo cliente. Os componentes personalizados podem acessar as sombras dos dispositivos cliente para gerenciar seu estado, mesmo quando o dispositivo cliente não está conectado. AWS IoT Cada AWS IoT coisa tem uma sombra sem nome, e você também pode criar várias sombras nomeadas para cada coisa.

Nesta seção, você implanta o componente do gerenciador de sombras para gerenciar sombras no dispositivo principal. Você também atualiza o componente de ponte MQTT para retransmitir mensagens de sombra entre dispositivos cliente e o componente gerenciador de sombra. Em seguida, você desenvolve um componente que atualiza as sombras dos dispositivos cliente e executa um aplicativo de amostra nos dispositivos cliente que responde às atualizações de sombra do componente. Esse componente representa um aplicativo inteligente de gerenciamento de luz, em que o dispositivo principal gerencia o estado das cores das luzes inteligentes que se conectam a ele como dispositivos clientes.

Para desenvolver um componente que interaja com as sombras do dispositivo cliente

1. Revise a implantação no dispositivo principal para implantar o componente shadow manager e configure o componente bridge MQTT para retransmitir mensagens paralelas entre os dispositivos do cliente e a publicação/assinatura local, onde o shadow manager se comunica. Faça o seguinte:

   a. Navegue até o console do AWS IoT Greengrass.

   b. No menu de navegação à esquerda, escolha Dispositivos principais.

   c. Na página Dispositivos principais, escolha o dispositivo principal que você está usando para este tutorial.

   d. Na página de detalhes do dispositivo principal, escolha a guia Dispositivos clientes.

   e. Na guia Dispositivos do cliente, escolha Configurar descoberta na nuvem.

      A página Configurar descoberta do dispositivo principal é aberta. Nesta página, você pode alterar ou configurar quais componentes do dispositivo cliente são implantados no dispositivo principal.

f. Na Etapa 3, para o aws.greengrass.clientdevices.mqtt.Bridgecomponente, escolha Editar configuração.

g. No modal Editar configuração para o componente de ponte MQTT, configure um mapeamento de tópicos que retransmita mensagens MQTT em tópicos de sombra do dispositivo entre dispositivos cliente e a interface local de publicação/assinatura. Você também confirma que a implantação especifica uma versão compatível do MQTT bridge. O suporte à sombra do dispositivo cliente requer o MQTT bridge v2.2.0 ou posterior. Faça o seguinte:

   i. Para a versão do componente, escolha a versão 2.2.0 ou posterior.

   ii. Em Configuração, no bloco Configuração para mesclar código, insira a configuração a seguir. Essa configuração especifica a retransmissão de mensagens MQTT em tópicos paralelos.

```
{
  "mqttTopicMapping": {
    "HelloWorldIotCoreMapping": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    },
    "HelloWorldPubsubMapping": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ShadowsLocalMqttToPubsub": {
      "topic": "$aws/things/+/shadow/#",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ShadowsPubsubToLocalMqtt": {
      "topic": "$aws/things/+/shadow/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    }
  }
}
```

   Para obter mais informações, consulte Configuração do componente de ponte MQTT.

iii. Selecione a opção Confirmar.

h. Na Etapa 3, selecione o aws.greengrass.ShadowManagercomponente para implantá-lo.

i. Escolha Revisar e implantar para revisar a implantação que esta página cria para você.

j. Na página Revisar, escolha Implantar para iniciar a implantação no dispositivo principal.

k. Para verificar se a implantação foi bem-sucedida, verifique o status da implantação e verifique os registros no dispositivo principal. Para verificar o status da implantação no dispositivo principal, você pode escolher Target na Visão geral da implantação. Para ver mais informações, consulte:

- Verificar status da implantação
- Monitore AWS IoT Greengrass os registros

2. Desenvolva e implante um componente do Greengrass que gerencia dispositivos cliente de iluminação inteligente. Faça o seguinte:

a. Crie uma pasta com os artefatos do componente no dispositivo principal.

Linux or Unix

```
mkdir -p artifacts/com.example.clientdevices.MySmartLightManager/1.0.0
```

Windows Command Prompt (CMD)

```
mkdir artifacts\com.example.clientdevices.MySmartLightManager\1.0.0
```

PowerShell

```
mkdir artifacts\com.example.clientdevices.MySmartLightManager\1.0.0
```

> **Important**
>
> Você deve usar o seguinte formato para o caminho da pasta de artefatos. Inclua o nome e a versão do componente que você especificar na receita.
>
> ```
> artifacts/componentName/componentVersion/
> ```

b. Use um editor de texto para criar uma receita de componente com o conteúdo a seguir. Esta receita especifica a instalação da AWS IoT Device SDK v2 para Python e a execução de um script que interage com as sombras dos dispositivos cliente de luz inteligente para gerenciar suas cores.

   Por exemplo, em um sistema baseado em Linux, você pode executar o seguinte comando para usar o GNU nano para criar o arquivo.

   ```
   nano recipes/com.example.clientdevices.MySmartLightManager-1.0.0.json
   ```

   Copie a seguinte receita para o arquivo.

   ```
   {
     "RecipeFormatVersion": "2020-01-25",
     "ComponentName": "com.example.clientdevices.MySmartLightManager",
     "ComponentVersion": "1.0.0",
     "ComponentDescription": "A component that interacts with smart light client
    devices.",
     "ComponentPublisher": "Amazon",
     "ComponentDependencies": {
       "aws.greengrass.Nucleus": {
         "VersionRequirement": "^2.6.0"
       },
       "aws.greengrass.ShadowManager": {
         "VersionRequirement": "^2.2.0"
       },
       "aws.greengrass.clientdevices.mqtt.Bridge": {
         "VersionRequirement": "^2.2.0"
       }
     },
     "ComponentConfiguration": {
       "DefaultConfiguration": {
         "smartLightDeviceNames": [],
         "accessControl": {
           "aws.greengrass.ShadowManager": {
             "com.example.clientdevices.MySmartLightManager:shadow:1": {
               "policyDescription": "Allows access to client devices' unnamed
    shadows",
               "operations": [
                 "aws.greengrass#GetThingShadow",
                 "aws.greengrass#UpdateThingShadow"
               ],
   ```

```
            "resources": [
              "$aws/things/MyClientDevice*/shadow"
            ]
          }
        },
        "aws.greengrass.ipc.pubsub": {
          "com.example.clientdevices.MySmartLightManager:pubsub:1": {
            "policyDescription": "Allows access to client devices' unnamed
 shadow updates",
            "operations": [
              "aws.greengrass#SubscribeToTopic"
            ],
            "resources": [
              "$aws/things/+/shadow/update/accepted"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "python3 -m pip install --user awsiotsdk",
        "run": "python3 -u {artifacts:path}/smart_light_manager.py"
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "install": "py -3 -m pip install --user awsiotsdk",
        "run": "py -3 -u {artifacts:path}/smart_light_manager.py"
      }
    }
  ]
}
```

c. Use um editor de texto para criar um artefato de script Python chamado `smart_light_manager.py` com o conteúdo a seguir. Esse aplicativo usa o serviço IPC

paralelo para obter e atualizar as sombras do dispositivo cliente e o serviço IPC local de publicação/assinatura para receber atualizações paralelas relatadas.

Por exemplo, em um sistema baseado em Linux, você pode executar o seguinte comando para usar o GNU nano para criar o arquivo.

```
nano artifacts/com.example.clientdevices.MySmartLightManager/1.0.0/
smart_light_manager.py
```

Copie o código Python a seguir no arquivo.

```
import json
import random
import sys
import time
import traceback
from uuid import uuid4

from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2
from awsiot.greengrasscoreipc.model import ResourceNotFoundError

SHADOW_COLOR_PROPERTY = 'color'
CONFIGURATION_CLIENT_DEVICE_NAMES = 'smartLightDeviceNames'
COLORS = ['red', 'orange', 'yellow', 'green', 'blue', 'purple']
SHADOW_UPDATE_TOPIC = '$aws/things/+/shadow/update/accepted'
SET_COLOR_INTERVAL = 15


class SmartLightDevice():
    def __init__(self, client_device_name: str, reported_color: str = None):
        self.name = client_device_name
        self.reported_color = reported_color
        self.desired_color = None


class SmartLightDeviceManager():
    def __init__(self, ipc_client: GreengrassCoreIPCClientV2):
        self.ipc_client = ipc_client
        self.devices = {}
        self.client_tokens = set()
        self.shadow_update_accepted_subscription_operation = None
        self.client_device_names_configuration_subscription_operation = None
```

```
        self.update_smart_light_device_list()

    def update_smart_light_device_list(self):
        # Update the device list from the component configuration.
        response = self.ipc_client.get_configuration(
            key_path=[CONFIGURATION_CLIENT_DEVICE_NAMES])
        # Identify the difference between the configuration and the currently
tracked devices.
        current_device_names = self.devices.keys()
        updated_device_names =
response.value[CONFIGURATION_CLIENT_DEVICE_NAMES]
        added_device_names = set(updated_device_names) -
set(current_device_names)
        removed_device_names = set(current_device_names) -
set(updated_device_names)
        # Stop tracking any smart light devices that are no longer in the
configuration.
        for name in removed_device_names:
            print('Removing %s from smart light device manager' % name)
            self.devices.pop(name)
        # Start tracking any new smart light devices that are in the
configuration.
        for name in added_device_names:
            print('Adding %s to smart light device manager' % name)
            device = SmartLightDevice(name)
            device.reported_color = self.get_device_reported_color(device)
            self.devices[name] = device
            print('Current color for %s is %s' % (name,
device.reported_color))

    def get_device_reported_color(self, smart_light_device):
        try:
            response = self.ipc_client.get_thing_shadow(
                thing_name=smart_light_device.name, shadow_name='')
            shadow = json.loads(str(response.payload, 'utf-8'))
            if 'reported' in shadow['state']:
                return shadow['state']['reported'].get(SHADOW_COLOR_PROPERTY)
            return None
        except ResourceNotFoundError:
            return None

    def request_device_color_change(self, smart_light_device, color):
        # Generate and track a client token for the request.
        client_token = str(uuid4())
```

```
        self.client_tokens.add(client_token)
        # Create a shadow payload, which must be a blob.
        payload_json = {
            'state': {
                'desired': {
                    SHADOW_COLOR_PROPERTY: color
                }
            },
            'clientToken': client_token
        }
        payload = bytes(json.dumps(payload_json), 'utf-8')
        self.ipc_client.update_thing_shadow(
            thing_name=smart_light_device.name, shadow_name='',
payload=payload)
        smart_light_device.desired_color = color

    def subscribe_to_shadow_update_accepted_events(self):
        if self.shadow_update_accepted_subscription_operation == None:
            # SubscribeToTopic returns a tuple with the response and the
operation.
            _, self.shadow_update_accepted_subscription_operation =
self.ipc_client.subscribe_to_topic(
                topic=SHADOW_UPDATE_TOPIC,
on_stream_event=self.on_shadow_update_accepted_event)
            print('Successfully subscribed to shadow update accepted topic')

    def close_shadow_update_accepted_subscription(self):
        if self.shadow_update_accepted_subscription_operation is not None:
            self.shadow_update_accepted_subscription_operation.close()

    def on_shadow_update_accepted_event(self, event):
        try:
            message = str(event.binary_message.message, 'utf-8')
            accepted_payload = json.loads(message)
            # Check for reported states from smart light devices and ignore
desired states from components.
            if 'reported' in accepted_payload['state']:
                # Process this update only if it uses a client token created by
this component.
                client_token = accepted_payload.get('clientToken')
                if client_token is not None and client_token in
self.client_tokens:
                    self.client_tokens.remove(client_token)
                    shadow_state = accepted_payload['state']['reported']
```

```
                    if SHADOW_COLOR_PROPERTY in shadow_state:
                        reported_color = shadow_state[SHADOW_COLOR_PROPERTY]
                        topic = event.binary_message.context.topic
                        client_device_name = topic.split('/')[2]
                        if client_device_name in self.devices:
                            # Set the reported color for the smart light
device.
                            self.devices[client_device_name].reported_color =
reported_color
                            print(
                                'Received shadow update confirmation from
client device: %s' % client_device_name)
                    else:
                        print("Shadow update doesn't specify color")
        except:
            traceback.print_exc()

    def subscribe_to_client_device_name_configuration_updates(self):
        if self.client_device_names_configuration_subscription_operation ==
None:
            # SubscribeToConfigurationUpdate returns a tuple with the response
and the operation.
            _, self.client_device_names_configuration_subscription_operation =
self.ipc_client.subscribe_to_configuration_update(
                key_path=[CONFIGURATION_CLIENT_DEVICE_NAMES],
on_stream_event=self.on_client_device_names_configuration_update_event)
            print(
                'Successfully subscribed to configuration updates for smart
light device names')

    def close_client_device_names_configuration_subscription(self):
        if self.client_device_names_configuration_subscription_operation is not
None:

self.client_device_names_configuration_subscription_operation.close()

    def on_client_device_names_configuration_update_event(self, event):
        try:
            if CONFIGURATION_CLIENT_DEVICE_NAMES in
event.configuration_update_event.key_path:
                print('Received configuration update for list of client
devices')
                self.update_smart_light_device_list()
        except:
```

```
            traceback.print_exc()


def choose_random_color():
    return random.choice(COLORS)

def main():
    try:
        # Create an IPC client and a smart light device manager.
        ipc_client = GreengrassCoreIPCClientV2()
        smart_light_manager = SmartLightDeviceManager(ipc_client)
        smart_light_manager.subscribe_to_shadow_update_accepted_events()

 smart_light_manager.subscribe_to_client_device_name_configuration_updates()
        try:
            # Keep the main thread alive, or the process will exit.
            while True:
                # Set each smart light device to a random color at a regular
 interval.
                for device_name in smart_light_manager.devices:
                    device = smart_light_manager.devices[device_name]
                    desired_color = choose_random_color()
                    print('Chose random color (%s) for %s' %
                          (desired_color, device_name))
                    if desired_color == device.desired_color:
                        print('Desired color for %s is already %s' %
                              (device_name, desired_color))
                    elif desired_color == device.reported_color:
                        print('Reported color for %s is already %s' %
                              (device_name, desired_color))
                    else:
                        smart_light_manager.request_device_color_change(
                            device, desired_color)
                        print('Requested color change for %s to %s' %
                              (device_name, desired_color))
                time.sleep(SET_COLOR_INTERVAL)
        except InterruptedError:
            print('Application interrupted')
        smart_light_manager.close_shadow_update_accepted_subscription()

 smart_light_manager.close_client_device_names_configuration_subscription()
    except Exception:
        print('Exception occurred', file=sys.stderr)
        traceback.print_exc()
```

```
        exit(1)


if __name__ == '__main__':
    main()
```

Esse aplicativo Python faz o seguinte:

- Lê a configuração do componente para obter a lista de dispositivos smart light client para gerenciar.
- Assina as notificações de atualização de configuração usando a operação SubscribeToConfigurationUpdate IPC. O software AWS IoT Greengrass Core envia notificações sempre que a configuração do componente é alterada. Quando o componente recebe uma notificação de atualização de configuração, ele atualiza a lista de dispositivos cliente smart light que ele gerencia.
- Obtém a sombra de cada dispositivo cliente de luz inteligente para obter seu estado inicial de cor.
- Define a cor de cada dispositivo cliente de luz inteligente para uma cor aleatória a cada 15 segundos. O componente atualiza a sombra da coisa do dispositivo cliente para mudar sua cor. Essa operação envia um evento delta de sombra para o dispositivo cliente pelo MQTT.
- Assina as mensagens aceitas de atualização paralela na interface local de publicação/assinatura usando a SubscribeToTopic operação IPC. Esse componente recebe essas mensagens para rastrear a cor de cada dispositivo cliente de luz inteligente. Quando um dispositivo cliente smart light recebe uma atualização paralela, ele envia uma mensagem MQTT para confirmar que recebeu a atualização. A ponte MQTT retransmite essa mensagem para a interface local de publicação/assinatura.

d. Use a CLI do Greengrass para implantar o componente. Ao implantar esse componente, você especifica a lista de dispositivos clientes`smartLightDeviceNames`, cujas sombras ele gerencia. Substitua *`MyClientDevice1`* pelo nome do item do dispositivo cliente.

   Linux or Unix

   ```
   sudo /greengrass/v2/bin/greengrass-cli deployment create \
     --recipeDir recipes \
     --artifactDir artifacts \
     --merge "com.example.clientdevices.MySmartLightManager=1.0.0" \
     --update-config '{
   ```

```
    "com.example.clientdevices.MySmartLightManager": {
      "MERGE": {
        "smartLightDeviceNames": [
          "MyClientDevice1"
        ]
      }
    }
  }'
```

Windows Command Prompt (CMD)

```
C:\greengrass\v2/bin/greengrass-cli deployment create ^
  --recipeDir recipes ^
  --artifactDir artifacts ^
  --merge "com.example.clientdevices.MySmartLightManager=1.0.0" ^
  --update-config '{"com.example.clientdevices.MySmartLightManager":
{"MERGE":{"smartLightDeviceNames":["MyClientDevice1"]}}}'
```

PowerShell

```
C:\greengrass\v2/bin/greengrass-cli deployment create `
  --recipeDir recipes `
  --artifactDir artifacts `
  --merge "com.example.clientdevices.MySmartLightManager=1.0.0" `
  --update-config '{
    "com.example.clientdevices.MySmartLightManager": {
      "MERGE": {
        "smartLightDeviceNames": [
          "MyClientDevice1"
        ]
      }
    }
  }'
```

3. Visualize os registros do componente para verificar se o componente foi instalado e executado com êxito.

   Linux or Unix

   ```
   sudo tail -f /greengrass/v2/logs/
   com.example.clientdevices.MySmartLightManager.log
   ```

PowerShell

```
gc C:\greengrass\v2/logs/com.example.clientdevices.MySmartLightManager.log -Tail
 10 -Wait
```

O componente envia solicitações para alterar a cor do dispositivo cliente de luz inteligente. O gerenciador de sombra recebe a solicitação e define o `desired` estado da sombra. No entanto, o dispositivo cliente de luz inteligente ainda não está funcionando, então o `reported` estado da sombra não muda. Os registros do componente incluem as seguintes mensagens.

```
2022-07-07T03:49:24.908Z [INFO] (Copier)
 com.example.clientdevices.MySmartLightManager: stdout. Chose random color (blue)
 for MyClientDevice1.
 {scriptName=services.com.example.clientdevices.MySmartLightManager.lifecycle.Run,
 serviceName=com.example.clientdevices.MySmartLightManager, currentState=RUNNING}
2022-07-07T03:49:24.912Z [INFO] (Copier)
 com.example.clientdevices.MySmartLightManager: stdout.
 Requested color change for MyClientDevice1 to blue.
 {scriptName=services.com.example.clientdevices.MySmartLightManager.lifecycle.Run,
 serviceName=com.example.clientdevices.MySmartLightManager, currentState=RUNNING}
```

Você pode manter o feed de registro aberto para ver quando o componente imprime mensagens.

4. Baixe e execute um aplicativo de amostra que usa o Greengrass Discovery e assina as atualizações do Device Shadow. No dispositivo cliente, faça o seguinte:

   a. Mude para a pasta de amostras na AWS IoT Device SDK v2 para Python. Esse aplicativo de amostra usa um módulo de análise de linha de comando na pasta de amostras.

   ```
   cd aws-iot-device-sdk-python-v2/samples
   ```

   b. Use um editor de texto para criar um script Python chamado `basic_discovery_shadow.py` com o conteúdo a seguir. Esse aplicativo usa a descoberta e as sombras do Greengrass para manter uma propriedade sincronizada entre o dispositivo cliente e o dispositivo principal.

   Por exemplo, em um sistema baseado em Linux, você pode executar o seguinte comando para usar o GNU nano para criar o arquivo.

```
nano basic_discovery_shadow.py
```

Copie o código Python a seguir no arquivo.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0.

from awscrt import io
from awscrt import mqtt
from awsiot import iotshadow
from awsiot.greengrass_discovery import DiscoveryClient
from awsiot import mqtt_connection_builder
from concurrent.futures import Future
import sys
import threading
import traceback
from uuid import uuid4

# Parse arguments
import utils.command_line_utils;
cmdUtils = utils.command_line_utils.CommandLineUtils("Basic Discovery -
 Greengrass discovery example with device shadows.")
cmdUtils.add_common_mqtt_commands()
cmdUtils.add_common_topic_message_commands()
cmdUtils.add_common_logging_commands()
cmdUtils.register_command("key", "<path>", "Path to your key in PEM format.",
 True, str)
cmdUtils.register_command("cert", "<path>", "Path to your client certificate in
 PEM format.", True, str)
cmdUtils.remove_command("endpoint")
cmdUtils.register_command("thing_name", "<str>", "The name assigned to your IoT
 Thing", required=True)
cmdUtils.register_command("region", "<str>", "The region to connect through.",
 required=True)
cmdUtils.register_command("shadow_property", "<str>", "The name of the shadow
 property you want to change (optional, default='color'", default="color")
# Needs to be called so the command utils parse the commands
cmdUtils.get_args()

# Using globals to simplify sample code
is_sample_done = threading.Event()
mqtt_connection = None
```

```
shadow_thing_name = cmdUtils.get_command_required("thing_name")
shadow_property = cmdUtils.get_command("shadow_property")

SHADOW_VALUE_DEFAULT = "off"

class LockedData:
    def __init__(self):
        self.lock = threading.Lock()
        self.shadow_value = None
        self.disconnect_called = False
        self.request_tokens = set()

locked_data = LockedData()


def on_connection_interupted(connection, error, **kwargs):
    print('connection interrupted with error {}'.format(error))


def on_connection_resumed(connection, return_code, session_present, **kwargs):
    print('connection resumed with return code {}, session present
 {}'.format(return_code, session_present))

# Try IoT endpoints until we find one that works
def try_iot_endpoints():
    for gg_group in discover_response.gg_groups:
        for gg_core in gg_group.cores:
            for connectivity_info in gg_core.connectivity:
                try:
                    print('Trying core {} at host {} port
 {}'.format(gg_core.thing_arn, connectivity_info.host_address,
 connectivity_info.port))
                    mqtt_connection = mqtt_connection_builder.mtls_from_path(
                        endpoint=connectivity_info.host_address,
                        port=connectivity_info.port,
                        cert_filepath=cmdUtils.get_command_required("cert"),
                        pri_key_filepath=cmdUtils.get_command_required("key"),

 ca_bytes=gg_group.certificate_authorities[0].encode('utf-8'),
                        on_connection_interrupted=on_connection_interupted,
                        on_connection_resumed=on_connection_resumed,
                        client_id=cmdUtils.get_command_required("thing_name"),
                        clean_session=False,
                        keep_alive_secs=30)
```

```
                    connect_future = mqtt_connection.connect()
                    connect_future.result()
                    print('Connected!')
                    return mqtt_connection

                except Exception as e:
                    print('Connection failed with exception {}'.format(e))
                    continue

    exit('All connection attempts failed')

# Function for gracefully quitting this sample
def exit(msg_or_exception):
    if isinstance(msg_or_exception, Exception):
        print("Exiting sample due to exception.")
        traceback.print_exception(msg_or_exception.__class__, msg_or_exception,
 sys.exc_info()[2])
    else:
        print("Exiting sample:", msg_or_exception)

    with locked_data.lock:
        if not locked_data.disconnect_called:
            print("Disconnecting...")
            locked_data.disconnect_called = True
            future = mqtt_connection.disconnect()
            future.add_done_callback(on_disconnected)

def on_disconnected(disconnect_future):
    # type: (Future) -> None
    print("Disconnected.")

    # Signal that sample is finished
    is_sample_done.set()

def on_get_shadow_accepted(response):
    # type: (iotshadow.GetShadowResponse) -> None
    try:
        with locked_data.lock:
            # check that this is a response to a request from this session
            try:
                locked_data.request_tokens.remove(response.client_token)
            except KeyError:
                return
```

```
            print("Finished getting initial shadow state.")
            if locked_data.shadow_value is not None:
                print("  Ignoring initial query because a delta event has
 already been received.")
                return

        if response.state:
            if response.state.delta:
                value = response.state.delta.get(shadow_property)
                if value:
                    print("  Shadow contains delta value '{}'.".format(value))
                    change_shadow_value(value)
                    return

            if response.state.reported:
                value = response.state.reported.get(shadow_property)
                if value:
                    print("  Shadow contains reported value
 '{}'.".format(value))

 set_local_value_due_to_initial_query(response.state.reported[shadow_property])
                    return

        print("  Shadow document lacks '{}' property. Setting
 defaults...".format(shadow_property))
        change_shadow_value(SHADOW_VALUE_DEFAULT)
        return

    except Exception as e:
        exit(e)

def on_get_shadow_rejected(error):
    # type: (iotshadow.ErrorResponse) -> None
    try:
        # check that this is a response to a request from this session
        with locked_data.lock:
            try:
                locked_data.request_tokens.remove(error.client_token)
            except KeyError:
                return

        if error.code == 404:
            print("Thing has no shadow document. Creating with defaults...")
```

```
            change_shadow_value(SHADOW_VALUE_DEFAULT)
        else:
            exit("Get request was rejected. code:{} message:'{}'".format(
                error.code, error.message))

    except Exception as e:
        exit(e)

def on_shadow_delta_updated(delta):
    # type: (iotshadow.ShadowDeltaUpdatedEvent) -> None
    try:
        print("Received shadow delta event.")
        if delta.state and (shadow_property in delta.state):
            value = delta.state[shadow_property]
            if value is None:
                print("  Delta reports that '{}' was deleted. Resetting
 defaults...".format(shadow_property))
                change_shadow_value(SHADOW_VALUE_DEFAULT)
                return
            else:
                print("  Delta reports that desired value is '{}'. Changing
 local value...".format(value))
                if (delta.client_token is not None):
                    print ("  ClientToken is: " + delta.client_token)
                change_shadow_value(value, delta.client_token)
        else:
            print("  Delta did not report a change in
 '{}'".format(shadow_property))

    except Exception as e:
        exit(e)

def on_publish_update_shadow(future):
    #type: (Future) -> None
    try:
        future.result()
        print("Update request published.")
    except Exception as e:
        print("Failed to publish update request.")
        exit(e)

def on_update_shadow_accepted(response):
    # type: (iotshadow.UpdateShadowResponse) -> None
    try:
```

```
        # check that this is a response to a request from this session
        with locked_data.lock:
            try:
                locked_data.request_tokens.remove(response.client_token)
            except KeyError:
                return

        try:
            if response.state.reported != None:
                if shadow_property in response.state.reported:
                    print("Finished updating reported shadow value to
 '{}'.".format(response.state.reported[shadow_property])) # type: ignore
                else:
                    print ("Could not find shadow property with name:
 '{}'.".format(shadow_property)) # type: ignore
            else:
                print("Shadow states cleared.") # when the shadow states are
 cleared, reported and desired are set to None
        except:
            exit("Updated shadow is missing the target property")

    except Exception as e:
        exit(e)

def on_update_shadow_rejected(error):
    # type: (iotshadow.ErrorResponse) -> None
    try:
        # check that this is a response to a request from this session
        with locked_data.lock:
            try:
                locked_data.request_tokens.remove(error.client_token)
            except KeyError:
                return

        exit("Update request was rejected. code:{} message:'{}'".format(
            error.code, error.message))

    except Exception as e:
        exit(e)

def set_local_value_due_to_initial_query(reported_value):
    with locked_data.lock:
        locked_data.shadow_value = reported_value
```

```
def change_shadow_value(value, token=None):
    with locked_data.lock:
        if locked_data.shadow_value == value:
            print("Local value is already '{}'.".format(value))
            return

        print("Changed local shadow value to '{}'.".format(value))
        locked_data.shadow_value = value

        print("Updating reported shadow value to '{}'...".format(value))

        reuse_token = token is not None
        # use a unique token so we can correlate this "request" message to
        # any "response" messages received on the /accepted and /rejected
 topics
        if not reuse_token:
            token = str(uuid4())

        # if the value is "clear shadow" then send a UpdateShadowRequest with
 None
        # for both reported and desired to clear the shadow document
 completely.
        if value == "clear_shadow":
            tmp_state = iotshadow.ShadowState(reported=None, desired=None,
 reported_is_nullable=True, desired_is_nullable=True)
            request = iotshadow.UpdateShadowRequest(
                thing_name=shadow_thing_name,
                state=tmp_state,
                client_token=token,
            )
        # Otherwise, send a normal update request
        else:
            # if the value is "none" then set it to a Python none object to
            # clear the individual shadow property
            if value == "none":
                value = None

            request = iotshadow.UpdateShadowRequest(
            thing_name=shadow_thing_name,
            state=iotshadow.ShadowState(
                reported={ shadow_property: value }
                ),
                client_token=token,
            )
```

```
        future = shadow_client.publish_update_shadow(request,
 mqtt.QoS.AT_LEAST_ONCE)

        if not reuse_token:
            locked_data.request_tokens.add(token)

        future.add_done_callback(on_publish_update_shadow)


if __name__ == '__main__':
    tls_options =
 io.TlsContextOptions.create_client_with_mtls_from_path(cmdUtils.get_command_required("
 cmdUtils.get_command_required("key"))
    if cmdUtils.get_command(cmdUtils.m_cmd_ca_file):
        tls_options.override_default_trust_store_from_path(None,
 cmdUtils.get_command(cmdUtils.m_cmd_ca_file))
    tls_context = io.ClientTlsContext(tls_options)

    socket_options = io.SocketOptions()

    print('Performing greengrass discovery...')
    discovery_client =
 DiscoveryClient(io.ClientBootstrap.get_or_create_static_default(),
 socket_options, tls_context, cmdUtils.get_command_required("region"))
    resp_future =
 discovery_client.discover(cmdUtils.get_command_required("thing_name"))
    discover_response = resp_future.result()

    print(discover_response)
    if cmdUtils.get_command("print_discover_resp_only"):
        exit(0)

    mqtt_connection = try_iot_endpoints()
    shadow_client = iotshadow.IotShadowClient(mqtt_connection)

    try:
        # Subscribe to necessary topics.
        # Note that is **is** important to wait for "accepted/rejected"
 subscriptions
        # to succeed before publishing the corresponding "request".
        print("Subscribing to Update responses...")
        update_accepted_subscribed_future, _ =
 shadow_client.subscribe_to_update_shadow_accepted(
```

```
request=iotshadow.UpdateShadowSubscriptionRequest(thing_name=shadow_thing_name),
            qos=mqtt.QoS.AT_LEAST_ONCE,
            callback=on_update_shadow_accepted)

        update_rejected_subscribed_future, _ =
shadow_client.subscribe_to_update_shadow_rejected(

request=iotshadow.UpdateShadowSubscriptionRequest(thing_name=shadow_thing_name),
            qos=mqtt.QoS.AT_LEAST_ONCE,
            callback=on_update_shadow_rejected)

        # Wait for subscriptions to succeed
        update_accepted_subscribed_future.result()
        update_rejected_subscribed_future.result()

        print("Subscribing to Get responses...")
        get_accepted_subscribed_future, _ =
shadow_client.subscribe_to_get_shadow_accepted(

request=iotshadow.GetShadowSubscriptionRequest(thing_name=shadow_thing_name),
            qos=mqtt.QoS.AT_LEAST_ONCE,
            callback=on_get_shadow_accepted)

        get_rejected_subscribed_future, _ =
shadow_client.subscribe_to_get_shadow_rejected(

request=iotshadow.GetShadowSubscriptionRequest(thing_name=shadow_thing_name),
            qos=mqtt.QoS.AT_LEAST_ONCE,
            callback=on_get_shadow_rejected)

        # Wait for subscriptions to succeed
        get_accepted_subscribed_future.result()
        get_rejected_subscribed_future.result()

        print("Subscribing to Delta events...")
        delta_subscribed_future, _ =
shadow_client.subscribe_to_shadow_delta_updated_events(

request=iotshadow.ShadowDeltaUpdatedSubscriptionRequest(thing_name=shadow_thing_name),
            qos=mqtt.QoS.AT_LEAST_ONCE,
            callback=on_shadow_delta_updated)

        # Wait for subscription to succeed
```

```
        delta_subscribed_future.result()

        # The rest of the sample runs asynchronously.

        # Issue request for shadow's current state.
        # The response will be received by the on_get_accepted() callback
        print("Requesting current shadow state...")

        with locked_data.lock:
            # use a unique token so we can correlate this "request" message to
            # any "response" messages received on the /accepted and /rejected
topics
            token = str(uuid4())

            publish_get_future = shadow_client.publish_get_shadow(

request=iotshadow.GetShadowRequest(thing_name=shadow_thing_name,
client_token=token),
                qos=mqtt.QoS.AT_LEAST_ONCE)

            locked_data.request_tokens.add(token)

        # Ensure that publish succeeds
        publish_get_future.result()

    except Exception as e:
        exit(e)

    # Wait for the sample to finish (user types 'quit', or an error occurs)
    is_sample_done.wait()
```

Esse aplicativo Python faz o seguinte:

- Usa o Greengrass Discovery para descobrir e se conectar ao dispositivo principal.
- Solicita o documento paralelo do dispositivo principal para obter o estado inicial da propriedade.
- Assina eventos delta de sombra, que o dispositivo principal envia quando o `desired` valor da propriedade é diferente de seu `reported` valor. Quando o aplicativo recebe um evento delta paralelo, ele altera o valor da propriedade e envia uma atualização para o dispositivo principal para definir o novo valor como seu `reported` valor.

Este aplicativo combina as amostras de descoberta e sombra do Greengrass da AWS IoT Device SDK v2.

c. Execute o aplicativo de amostra. Esse aplicativo espera argumentos que especifiquem o nome do dispositivo cliente, a propriedade shadow a ser usada e os certificados que autenticam e protegem a conexão.

- Substitua *MyClientDevice1* pelo nome do item do dispositivo cliente.
- Substitua *~/certs/ AmazonRoot CA1.pem* pelo caminho para o certificado CA raiz da Amazon no dispositivo cliente.
- Substitua *~/certs/device.pem.crt* pelo caminho para o certificado do dispositivo no dispositivo cliente.
- Substitua *~/certs/private.pem.key pelo caminho para o arquivo de chave* privada no dispositivo cliente.
- Substitua *us-east-1* pela AWS região em que seu dispositivo cliente e dispositivo principal operam.

```
python3 basic_discovery_shadow.py \
  --thing_name MyClientDevice1 \
  --shadow_property color \
  --ca_file ~/certs/AmazonRootCA1.pem \
  --cert ~/certs/device.pem.crt \
  --key ~/certs/private.pem.key \
  --region us-east-1 \
  --verbosity Warn
```

O aplicativo de amostra se inscreve nos tópicos paralelos e espera receber eventos delta de sombra do dispositivo principal. Se a saída indicar que o aplicativo recebe e responde aos eventos delta de sombra, o dispositivo cliente pode interagir com sucesso com sua sombra no dispositivo principal.

```
Performing greengrass discovery...
awsiot.greengrass_discovery.DiscoverResponse(gg_groups=[awsiot.greengrass_discovery.GGG
coreDevice-MyGreengrassCore',
 cores=[awsiot.greengrass_discovery.GGCore(thing_arn='arn:aws:iot:us-
east-1:123456789012:thing/MyGreengrassCore',
 connectivity=[awsiot.greengrass_discovery.ConnectivityInfo(id='203.0.113.0',
```

```
 host_address='203.0.113.0', metadata='', port=8883)])],
 certificate_authorities=['-----BEGIN CERTIFICATE-----
\nMIICiT...EXAMPLE=\n-----END CERTIFICATE-----\n'])])
Trying core arn:aws:iot:us-east-1:123456789012:thing/MyGreengrassCore at host
 203.0.113.0 port 8883
Connected!
Subscribing to Update responses...
Subscribing to Get responses...
Subscribing to Delta events...
Requesting current shadow state...
Received shadow delta event.
  Delta reports that desired value is 'purple'. Changing local value...
  ClientToken is: 3dce4d3f-e336-41ac-aa4f-7882725f0033
Changed local shadow value to 'purple'.
Updating reported shadow value to 'purple'...
Update request published.
```

Se, em vez disso, o aplicativo gerar um erro, consulte Solução de problemas de descoberta do Greengrass.

Você também pode ver os registros do Greengrass no dispositivo principal para verificar se o dispositivo cliente se conecta e envia mensagens com sucesso. Para ter mais informações, consulte Monitore AWS IoT Greengrass os registros.

5. Visualize os registros do componente novamente para verificar se o componente recebe confirmações de atualização de sombra do dispositivo cliente smart light.

   Linux or Unix

   ```
   sudo tail -f /greengrass/v2/logs/
   com.example.clientdevices.MySmartLightManager.log
   ```

   PowerShell

   ```
   gc C:\greengrass\v2/logs/com.example.clientdevices.MySmartLightManager.log -Tail
    10 -Wait
   ```

   O componente registra mensagens para confirmar que o dispositivo cliente Smart Light mudou de cor.

```
2022-07-07T03:49:24.908Z [INFO] (Copier)
 com.example.clientdevices.MySmartLightManager: stdout. Chose random color (blue)
 for MyClientDevice1.
 {scriptName=services.com.example.clientdevices.MySmartLightManager.lifecycle.Run,
 serviceName=com.example.clientdevices.MySmartLightManager, currentState=RUNNING}
2022-07-07T03:49:24.912Z [INFO] (Copier)
 com.example.clientdevices.MySmartLightManager: stdout.
 Requested color change for MyClientDevice1 to blue.
 {scriptName=services.com.example.clientdevices.MySmartLightManager.lifecycle.Run,
 serviceName=com.example.clientdevices.MySmartLightManager, currentState=RUNNING}
2022-07-07T03:49:24.959Z [INFO] (Copier)
 com.example.clientdevices.MySmartLightManager: stdout. Received
 shadow update confirmation from client device: MyClientDevice1.
 {scriptName=services.com.example.clientdevices.MySmartLightManager.lifecycle.Run,
 serviceName=com.example.clientdevices.MySmartLightManager, currentState=RUNNING}
```

> **Note**
>
> A sombra do dispositivo cliente está sincronizada entre o dispositivo principal e o dispositivo cliente. No entanto, o dispositivo principal não sincroniza com a sombra do dispositivo clienteAWS IoT Core. Você pode sincronizar uma sombra com AWS IoT Core para visualizar ou modificar o estado de todos os dispositivos em sua frota, por exemplo. Para obter mais informações sobre como configurar o componente do gerenciador de sombras para sincronizar sombrasAWS IoT Core, consulteSincronize sombras do dispositivo local com AWS IoT Core.

Você concluiu este tutorial. O dispositivo cliente se conecta ao dispositivo principal, envia mensagens MQTT para os componentes do Greengrass AWS IoT Core e recebe atualizações paralelas do dispositivo principal. Para obter mais informações sobre os tópicos abordados neste tutorial, consulte o seguinte:

- Associe dispositivos do cliente
- Gerencie os endpoints principais do dispositivo
- Teste as comunicações entre dispositivos clientes
- API RESTful de descoberta do Greengrass
- Retransmita mensagens MQTT entre dispositivos clientes e AWS IoT Core

- Interaja com dispositivos clientes em componentes
- Interaja com as sombras do dispositivo
- Interaja e sincronize as sombras do dispositivo cliente

Tutorial: Comece a usar o SageMaker Edge Manager

> **Important**
>
> SageMaker O Edge Manager será descontinuado em 26 de abril de 2024. Para obter mais informações sobre como continuar implantando seus modelos em dispositivos de ponta, consulte Fim da vida útil do SageMaker Edge Manager.

O Amazon SageMaker Edge Manager é um agente de software executado em dispositivos periféricos. SageMaker O Edge Manager fornece gerenciamento de modelos para dispositivos de borda para que você possa empacotar e usar modelos SageMaker compilados pelo Amazon Neo diretamente nos dispositivos principais do Greengrass. Ao usar o SageMaker Edge Manager, você também pode amostrar dados de entrada e saída do modelo de seus dispositivos principais e enviar esses dados Nuvem AWS para monitoramento e análise. Para obter mais informações sobre como o SageMaker Edge Manager funciona nos dispositivos principais do Greengrass, consulte. Use o Amazon SageMaker Edge Manager nos dispositivos principais do Greengrass

Este tutorial mostra como começar a usar o SageMaker Edge Manager com componentes AWS de amostra fornecidos em um dispositivo principal existente. Esses componentes de amostra usam o componente SageMaker Edge Manager como uma dependência para implantar o agente do Edge Manager e realizar inferência usando modelos pré-treinados que foram compilados usando o Neo. SageMaker Para obter mais informações sobre o agente do SageMaker Edge Manager, consulte o SageMaker Edge Manager no Amazon SageMaker Developer Guide.

Para configurar e usar o agente do SageMaker Edge Manager em um dispositivo principal existente do Greengrass, AWS fornece um exemplo de código que você pode usar para criar os seguintes exemplos de componentes de inferência e modelo.

- Classificação de imagens
  - `com.greengrass.SageMakerEdgeManager.ImageClassification`
  - `com.greengrass.SageMakerEdgeManager.ImageClassification.Model`

- Detecção de objetos
  - `com.greengrass.SageMakerEdgeManager.ObjectDetection`
  - `com.greengrass.SageMakerEdgeManager.ObjectDetection.Model`

Este tutorial mostra como implantar os componentes de amostra e o agente do SageMaker Edge Manager.

Tópicos

Pré-requisitos

Para concluir este tutorial, você deve atender aos seguintes pré-requisitos:

- Um dispositivo principal do Greengrass executado no Amazon Linux 2, uma plataforma Linux baseada em Debian (x86_64 ou Armv8) ou Windows (x86_64). Se você não tiver uma, consulte Tutorial: Conceitos básicos do AWS IoT Greengrass V2.
- Python 3.6 ou posterior, inclusive `pip` para sua versão do Python, instalada em seu dispositivo principal.
- O runtime `libgl1-mesa-glx` () da API OpenGL GLX instalado em seu dispositivo principal.
- Um usuário AWS Identity and Access Management (IAM) com permissões de administrador.
- Um computador de desenvolvimento compatível com a Internet, Windows, Mac ou Unix que atenda aos seguintes requisitos:
  - Python 3.6 ou posterior instalado.
  - AWS CLI instalado e configurado com suas credenciais de usuário administrador do IAM. Para obter mais informações, consulte Instalando o AWS CLI e configurando o. AWS CLI
- Os seguintes buckets do S3 foram criados no mesmo dispositivo principal do Greengrass Conta da AWS e Região da AWS que são criados no mesmo dispositivo principal:
  - Um bucket do S3 para armazenar os artefatos incluídos na inferência de amostra e nos componentes do modelo. Este tutorial usa DOC-EXAMPLE-BUCKET1 para se referir a esse bucket.

- Um bucket S3 que você associa à sua frota de dispositivos de SageMaker ponta. SageMaker O Edge Manager requer um bucket S3 para criar a frota de dispositivos de ponta e armazenar dados de amostra da execução de inferência em seu dispositivo. Este tutorial usa DOC-EXAMPLE-BUCKET2 para se referir a esse bucket.

Para obter informações sobre a criação de buckets do S3, consulte Introdução ao Amazon S3.

- A função do dispositivo Greengrass foi configurada com o seguinte:
  - Uma relação de confiança que permite `credentials.iot.amazonaws.com` e assume `sagemaker.amazonaws.com` a função, conforme mostrado no exemplo de política do IAM a seguir.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.iot.amazonaws.com"
       },
      "Action": "sts:AssumeRole"
    },
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "sagemaker.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

  - A política gerenciada AmazonSageMakerEdgeDeviceFleetPolicydo IAM.
  - A política gerenciada AmazonSageMakerFullAccessdo IAM.
  - A `s3:GetObject` ação para o bucket do S3 que contém os artefatos do seu componente, conforme mostrado no exemplo de política do IAM a seguir.

```
{
  "Version": "2012-10-17",
  "Statement": [
  {
```

```
    "Action": [
      "s3:GetObject"
    ],
    "Resource": [
      "arn:aws:s3:::DOC-EXAMPLE-BUCKET1/*"
    ],
    "Effect": "Allow"
  }
  ]
}
```

Configure seu dispositivo principal do Greengrass no SageMaker Edge Manager

As frotas de dispositivos SageMaker Edge no Edge Manager são coleções de dispositivos agrupados logicamente. Para usar o SageMaker Edge Manager com AWS IoT Greengrass, você deve criar uma frota de dispositivos de borda que use o mesmo alias de AWS IoT função do dispositivo principal do Greengrass no qual você implanta SageMaker o agente do Edge Manager. Em seguida, você deve registrar o dispositivo principal como parte dessa frota.

Tópicos

Crie uma frota de dispositivos de ponta

Para criar uma frota de dispositivos de ponta (console)

1. No SageMaker console da Amazon, escolha Edge Manager e, em seguida, escolha Frotas de dispositivos Edge.
2. Na página Frotas de dispositivos, escolha Criar frota de dispositivos.
3. Em Propriedades da frota de dispositivos, faça o seguinte:
   - Em Nome da frota de dispositivos, insira um nome para sua frota de dispositivos.
   - Para a função IAM, insira o Amazon Resource Name (ARN) do alias de AWS IoT função que você especificou ao configurar seu dispositivo principal do Greengrass.
   - Desative a opção Criar alias de função do IAM.

4. Escolha Próximo.
5. Em Configuração de saída, para URI do bucket do S3, insira o URI do bucket do S3 que você deseja associar à frota de dispositivos.
6. Selecione Enviar.

Registre seu dispositivo principal do Greengrass

Para registrar seu dispositivo principal do Greengrass como um dispositivo de ponta (console)

1. No SageMaker console da Amazon, escolha Edge Manager e, em seguida, escolha Dispositivos Edge.
2. Na página Dispositivos, escolha Registrar dispositivos.
3. Em Propriedades do dispositivo, em Nome da frota de dispositivos, insira o nome da frota de dispositivos que você criou e escolha Avançar.
4. Escolha Próximo.
5. Em Fonte do dispositivo, em Nome do dispositivo, insira o AWS IoT nome do item do seu dispositivo principal do Greengrass.
6. Selecione Enviar.

Crie os componentes de amostra

Para ajudar você a começar a usar o componente SageMaker Edge Manager, AWS fornece um script Python GitHub que cria a amostra de inferência e os componentes do modelo e os carrega no para você. Nuvem AWS Conclua as etapas a seguir em um computador de desenvolvimento.

Para criar os componentes de amostra

1. Baixe o repositório de exemplos de AWS IoT Greengrass componentes em GitHub seu computador de desenvolvimento.
2. Navegue até a `/machine-learning/sagemaker-edge-manager` pasta baixada.

   ```
   cd download-directory/machine-learning/sagemaker-edge-manager
   ```

3. Execute o comando a seguir para criar e carregar os componentes de amostra no Nuvem AWS.

   ```
   python3 create_components.py -r region -b DOC-EXAMPLE-BUCKET
   ```

Substitua a *`região`* pelo Região da AWS local em que você criou seu dispositivo principal do Greengrass e substitua DOC-EXAMPLE-BUCKET1 pelo nome do bucket do S3 para armazenar seus artefatos de componentes.

> **Note**
>
> Por padrão, o script cria componentes de amostra para classificação de imagens e inferência de detecção de objetos. Para criar componentes somente para um tipo específico de inferência, especifique o `-i` *`ImageClassification | ObjectDetection`* argumento.

Componentes de inferência e modelo de amostra para uso com o SageMaker Edge Manager agora são criados em seu Conta da AWS. Para ver os componentes de amostra no AWS IoT Greengrass console, escolha Componentes e, em Meus componentes, pesquise os seguintes componentes:

- `com.greengrass.SageMakerEdgeManager.ImageClassification`
- `com.greengrass.SageMakerEdgeManager.ImageClassification.Model`
- `com.greengrass.SageMakerEdgeManager.ObjectDetection`
- `com.greengrass.SageMakerEdgeManager.ObjectDetection.Model`

Execute uma amostra de inferência de classificação de imagens

Para executar a inferência de classificação de imagens usando os componentes AWS de amostra fornecidos e o agente do SageMaker Edge Manager, você deve implantar esses componentes em seu dispositivo principal. A implantação desses componentes baixa um modelo Resnet-50 pré-treinado e SageMaker compilado pela NEO e instala o agente Edge Manager em seu dispositivo. SageMaker O agente do SageMaker Edge Manager carrega o modelo e publica os resultados da inferência sobre o `gg/sageMakerEdgeManager/image-classification` tópico. Para visualizar esses resultados de inferência, use o cliente AWS IoT MQTT no AWS IoT console para assinar este tópico.

Tópicos

- Inscreva-se no tópico de notificações
- Implemente os componentes de amostra

- Veja os resultados da inferência

Inscreva-se no tópico de notificações

Nesta etapa, você configura o cliente AWS IoT MQTT no AWS IoT console para assistir às mensagens MQTT publicadas pelo componente de inferência de amostra. Por padrão, o componente publica resultados de inferência sobre o `gg/sageMakerEdgeManager/image-classification` tópico. Inscreva-se neste tópico antes de implantar o componente em seu dispositivo principal do Greengrass para ver os resultados da inferência quando o componente é executado pela primeira vez.

Para se inscrever no tópico de notificações padrão

1. No menu de navegação do AWS IoT console, escolha Testar, cliente de teste MQTT.
2. Em Inscrever-se em um tópico, na caixa Nome do tópico, digite**`gg/sageMakerEdgeManager/image-classification`**.
3. Escolha Assinar.

Implemente os componentes de amostra

Nesta etapa, você configura e implanta os seguintes componentes em seu dispositivo principal:

- `aws.greengrass.SageMakerEdgeManager`
- `com.greengrass.SageMakerEdgeManager.ImageClassification`
- `com.greengrass.SageMakerEdgeManager.ImageClassification.Model`

Para implantar seus componentes (console)

1. No menu de navegação do AWS IoT Greengrass console, escolha Implantações e, em seguida, escolha a implantação do dispositivo de destino que você deseja revisar.
2. Na página de implantação, escolha Revisar e, em seguida, escolha Revisar implantação.
3. Na página Especificar destino, escolha Próximo.
4. Na página Selecionar componentes, faça o seguinte:

    a. Em Meus componentes, selecione os seguintes componentes:

- `com.greengrass.SageMakerEdgeManager.ImageClassification`
- `com.greengrass.SageMakerEdgeManager.ImageClassification.Model`

b. Em Componentes públicos, desative a opção Mostrar somente componentes selecionados e, em seguida, selecione o `aws.greengrass.SageMakerEdgeManager` componente.

c. Escolha Próximo.

5. Na página Configurar componentes, selecione o `aws.greengrass.SageMakerEdgeManager` componente e faça o seguinte.

a. Escolha Configurar componente.

b. Em Atualização de configuração, em Configuração a ser mesclada, insira a configuração a seguir.

```
{
    "DeviceFleetName": "device-fleet-name",
    "BucketName": "DOC-EXAMPLE-BUCKET"
}
```

*`device-fleet-name`*Substitua pelo nome da frota de dispositivos de ponta que você criou e substitua *`DOC-EXAMPLE-BUCKET pelo nome do bucket`* S3 que está associado à sua frota de dispositivos.

c. Escolha Confirmar e, em seguida, Avançar.

6. Na página Definir configurações avançadas, mantenha as configurações padrão e escolha Avançar.

7. Na página Revisar, escolha Implantar

Para implantar seus componentes (AWS CLI)

1. No seu computador de desenvolvimento, crie um `deployment.json` arquivo para definir a configuração de implantação dos componentes do SageMaker Edge Manager. Esse arquivo deve se parecer com o exemplo a seguir.

```
{
  "targetArn":"targetArn",
  "components": {
    "aws.greengrass.SageMakerEdgeManager": {
      "componentVersion": "1.0.x",
```

```
      "configurationUpdate": {
        "merge": "{\"DeviceFleetName\":\"device-fleet-name\",\"BucketName\":\"DOC-
EXAMPLE-BUCKET2\"}"
      }
    },
    "com.greengrass.SageMakerEdgeManager.ImageClassification": {
      "componentVersion": "1.0.x",
      "configurationUpdate": {
      }
    },
    "com.greengrass.SageMakerEdgeManager.ImageClassification.Model": {
      "componentVersion": "1.0.x",
      "configurationUpdate": {
      }
    },
  }
}
```

- No campo `targetArn`, substitua *`targetArn`* pelo nome do recurso da Amazon (ARN) da coisa ou do grupo de coisas do destino da implantação, no seguinte formato:
  - Coisa: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`thingName`*
  - Grupo de coisas: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thinggroup/`*`thingGroupName`*
- No `merge` campo, *`device-fleet-name`*substitua pelo nome da frota de dispositivos de ponta que você criou. Em seguida, substitua *`DOC-EXAMPLE-BUCKET2`* pelo nome do bucket S3 associado à sua frota de dispositivos.
- Substitua as versões dos componentes de cada componente pela versão mais recente disponível.

2. Execute o seguinte comando para implantar os componentes no dispositivo:

```
aws greengrassv2 create-deployment \
    --cli-input-json file://path/to/deployment.json
```

A implantação pode levar vários minutos para ser concluída. Na próxima etapa, verifique o log do componente para verificar se a implantação foi concluída com êxito e para ver os resultados da inferência.

Veja os resultados da inferência

Depois de implantar os componentes, você pode visualizar os resultados da inferência no registro do componente no seu dispositivo principal do Greengrass e no cliente MQTT AWS IoT no console. AWS IoT Para assinar o tópico no qual o componente publica resultados de inferência, consulte. Inscreva-se no tópico de notificações

- AWS IoT Cliente MQTT — Para visualizar os resultados que o componente de inferência publica no tópico de notificações padrão, conclua as seguintes etapas:

  1. No menu de navegação do AWS IoT console, escolha Testar, cliente de teste MQTT.
  2. Em Assinaturas, escolha. **`gg/sageMakerEdgeManager/image-classification`**

- Registro do componente — Para visualizar os resultados da inferência no registro do componente, execute o comando a seguir em seu dispositivo principal do Greengrass.

```
sudo tail -f /greengrass/v2/logs/
com.greengrass.SageMakerEdgeManager.ImageClassification.log
```

Se você não conseguir ver os resultados da inferência no log do componente ou no cliente MQTT, a implantação falhou ou não atingiu o dispositivo principal. Isso pode ocorrer se seu dispositivo principal não estiver conectado à Internet ou não tiver as permissões corretas para executar o componente. Execute o comando a seguir em seu dispositivo principal para visualizar o arquivo de log do software AWS IoT Greengrass principal. Esse arquivo inclui registros do serviço de implantação do dispositivo principal do Greengrass.

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

Para ter mais informações, consulte Solução de problemas de inferência de aprendizado de máquina.

Tutorial: Execute inferência de classificação de imagens de amostra usando o Lite TensorFlow

Este tutorial mostra como usar o componente de inferência de classificação de imagem TensorFlow Lite para realizar exemplos de inferência de classificação de imagens em um dispositivo principal do Greengrass. Esse componente inclui as seguintes dependências de componentes:

- TensorFlow Componente de armazenamento de modelos de classificação de imagem Lite
- TensorFlow Componente Lite Runtime

Quando você implanta esse componente, ele baixa um modelo MobileNet v1 pré-treinado e instala o tempo de execução TensorFlow Lite e suas dependências. Esse componente publica resultados de inferência sobre o `ml/tflite/image-classification` tópico. Para visualizar esses resultados de inferência, use o cliente AWS IoT MQTT no AWS IoT console para assinar este tópico.

Neste tutorial, você implanta o componente de inferência de amostra para realizar a classificação da imagem na imagem de amostra fornecida pelaAWS IoT Greengrass. Depois de concluir este tutorial, você pode concluirTutorial: Execute inferência de classificação de imagens de amostra em imagens de uma câmera usando o TensorFlow Lite, que mostra como modificar o componente de inferência de amostra para realizar a classificação de imagens em imagens de uma câmera localmente em um dispositivo principal do Greengrass.

Para obter mais informações sobre aprendizado de máquina em dispositivos Greengrass, consulte. Executar a inferência de machine learning

Tópicos

Pré-requisitos

Para concluir este tutorial, você precisará do seguinte:

- Um dispositivo principal do Linux Greengrass. Se você não tiver uma, consulte Tutorial: Conceitos básicos do AWS IoT Greengrass V2. O dispositivo principal deve atender aos seguintes requisitos:
  - Nos principais dispositivos do Greengrass que executam o Amazon Linux 2 ou o Ubuntu 18.04, a GNU C Library (glibc) versão 2.27 ou posterior está instalada no dispositivo.
  - Em dispositivos ARMv7L, como o Raspberry Pi, dependências do OpenCV-Python instaladas no dispositivo. Execute o comando a seguir para instalar as dependências.

    ```
    sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
     libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
    ```
  - Os dispositivos Raspberry Pi que executam o Raspberry Pi OS Bullseye devem atender aos seguintes requisitos:
    - NumPy 1.22.4 ou posterior instalado no dispositivo. O Raspberry Pi OS Bullseye inclui uma versão anterior do NumPy, então você pode executar o seguinte comando para atualizar NumPy o dispositivo.

      ```
      pip3 install --upgrade numpy
      ```
    - A pilha de câmeras antiga ativada no dispositivo. O Raspberry Pi OS Bullseye inclui uma nova pilha de câmeras que é ativada por padrão e não é compatível, portanto, você deve habilitar a pilha de câmeras antiga.

      Para habilitar a pilha de câmeras antiga

      1. Execute o comando a seguir para abrir a ferramenta de configuração do Raspberry Pi.

         ```
         sudo raspi-config
         ```
      2. Selecione Opções de interface.
      3. Selecione Câmera antiga para ativar a pilha de câmeras antigas.
      4. Reinicie o Raspberry Pi.

Etapa 1: inscrever-se no tópico de notificações padrão

Nesta etapa, você configura o cliente AWS IoT MQTT no AWS IoT console para assistir às mensagens MQTT publicadas pelo componente de classificação de imagem TensorFlow Lite. Por padrão, o componente publica resultados de inferência sobre o `ml/tflite/image-classification` tópico. Inscreva-se neste tópico antes de implantar o componente em seu

dispositivo principal do Greengrass para ver os resultados da inferência quando o componente é executado pela primeira vez.

Para se inscrever no tópico de notificações padrão

1. No menu de navegação do AWS IoTconsole, escolha Testar, cliente de teste MQTT.
2. Em Inscrever-se em um tópico, na caixa Nome do tópico, digite**`ml/tflite/image-classification`**.
3. Escolha Assinar.

Etapa 2: implantar o componente de classificação de imagem TensorFlow Lite

Nesta etapa, você implanta o componente de classificação de imagem TensorFlow Lite em seu dispositivo principal:

Para implantar o componente de classificação de imagem TensorFlow Lite (console)

1. No menu de navegação AWS IoT Greengrassdo console, escolha Componentes.
2. Na página Componentes, na guia Componentes públicos, escolha `aws.greengrass.TensorFlowLiteImageClassification`.
3. Na página aws.greengrass.TensorFlowLiteImageClassification, escolha Implantar.
4. Em Adicionar à implantação, escolha uma das seguintes opções:
   a. Para mesclar esse componente a uma implantação existente em seu dispositivo de destino, escolha Adicionar à implantação existente e selecione a implantação que você deseja revisar.
   b. Para criar uma nova implantação em seu dispositivo de destino, escolha Criar nova implantação. Se você tiver uma implantação existente em seu dispositivo, escolher essa etapa substituirá a implantação existente.
5. Na página Especificar destino, faça o seguinte:
   a. Em Informações de implantação, insira ou modifique o nome amigável para sua implantação.

   b. Em Destinos de implantação, selecione um alvo para sua implantação e escolha Avançar. Você não pode alterar o destino de implantação se estiver revisando uma implantação existente.

6. Na página Selecionar componentes, em Componentes públicos, verifique se o `aws.greengrass.TensorFlowLiteImageClassification` componente está selecionado e escolha Avançar.
7. Na página Configurar componentes, mantenha as configurações padrão e escolha Avançar.
8. Na página Definir configurações avançadas, mantenha as configurações padrão e escolha Avançar.
9. Na página Revisar, escolha Implantar

Para implantar o componente de classificação de imagem TensorFlow Lite (AWS CLI)

1. Crie um `deployment.json` arquivo para definir a configuração de implantação do componente de classificação de imagem TensorFlow Lite. Esse arquivo deve ter a seguinte aparência:

```
{
  "targetArn":"targetArn",
  "components": {
    "aws.greengrass.TensorFlowLiteImageClassification": {
      "componentVersion": 2.1.0,
      "configurationUpdate": {
      }
    }
  }
}
```

   - No `targetArn` campo, *`targetArn`*substitua pelo nome de recurso da Amazon (ARN) da coisa ou grupo de itens a ser destinado para a implantação, no seguinte formato:
     - Coisa: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`thingName`*
     - Grupo de coisas: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thinggroup/`*`thingGroupName`*
   - Este tutorial usa a versão 2.1.0 do componente. No objeto `aws.greengrass.TensorFlowLiteObjectDetection` componente, substitua *`2.1.0`* para usar uma versão diferente do componente de detecção de objetos TensorFlow Lite.
2. Execute o comando a seguir para implantar o componente de classificação de imagem TensorFlow Lite no dispositivo:

```
aws greengrassv2 create-deployment \
    --cli-input-json file://path/to/deployment.json
```

A implantação pode levar vários minutos para ser concluída. Na próxima etapa, verifique o log do componente para verificar se a implantação foi concluída com êxito e para ver os resultados da inferência.

Etapa 3: ver os resultados da inferência

Depois de implantar o componente, você pode visualizar os resultados da inferência no registro do componente em seu dispositivo principal do Greengrass e no cliente MQTT AWS IoT no console. AWS IoT Para assinar o tópico no qual o componente publica resultados de inferência, consulte. Etapa 1: inscrever-se no tópico de notificações padrão

- AWS IoTCliente MQTT — Para visualizar os resultados que o componente de inferência publica no tópico de notificações padrão, conclua as seguintes etapas:

  1. No menu de navegação do AWS IoTconsole, escolha Testar, cliente de teste MQTT.
  2. Em Assinaturas, escolha. **`ml/tflite/image-classification`**

     Você deve ver mensagens semelhantes ao exemplo a seguir.

```
{
  "timestamp": "2021-01-01 00:00:00.000000",
  "inference-type": "image-classification",
  "inference-description": "Top 5 predictions with score 0.3 or above ",
  "inference-results": [
    {
      "Label": "cougar, puma, catamount, mountain lion, painter, panther, Felis
 concolor",
      "Score": "0.5882352941176471"
    },
    {
      "Label": "Persian cat",
      "Score": "0.5882352941176471"
    },
    {
      "Label": "tiger cat",
      "Score": "0.5882352941176471"
```

```
    },
    {
      "Label": "dalmatian, coach dog, carriage dog",
      "Score": "0.5607843137254902"
    },
    {
      "Label": "malamute, malemute, Alaskan malamute",
      "Score": "0.5450980392156862"
    }
  ]
}
```

- Registro do componente — Para visualizar os resultados da inferência no registro do componente, execute o comando a seguir em seu dispositivo principal do Greengrass.

  ```
  sudo tail -f /greengrass/v2/logs/aws.greengrass.TensorFlowLiteImageClassification.log
  ```

  Você deve ver resultados semelhantes aos do exemplo a seguir.

  ```
  2021-01-01 00:00:00.000000 [INFO] (Copier)
   aws.greengrass.TensorFlowLiteImageClassification: stdout. Publishing results to the
   IoT core....
   {scriptName=services.aws.greengrass.TensorFlowLiteImageClassification.lifecycle.Run.script,
   serviceName=aws.greengrass.TensorFlowLiteImageClassification, currentState=RUNNING}

  2021-01-01 00:00:00.000000 [INFO] (Copier)
   aws.greengrass.TensorFlowLiteImageClassification: stdout. {"timestamp":
   "2021-01-01 00:00:00.000000", "inference-type": "image-classification", "inference-
  description": "Top 5 predictions with score 0.3 or above ", "inference-results":
   [{"Label": "cougar, puma, catamount, mountain lion, painter, panther, Felis
   concolor", "Score": "0.5882352941176471"}, {"Label": "Persian cat", "Score":
   "0.5882352941176471"}, {"Label": "tiger cat", "Score": "0.5882352941176471"},
   {"Label": "dalmatian, coach dog, carriage dog", "Score": "0.5607843137254902"},
   {"Label": "malamute, malemute, Alaskan malamute", "Score": "0.5450980392156862"}]}.
   {scriptName=services.aws.greengrass.TensorFlowLiteImageClassification.lifecycle.Run.script,
   serviceName=aws.greengrass.TensorFlowLiteImageClassification, currentState=RUNNING}
  ```

Se você não conseguir ver os resultados da inferência no log do componente ou no cliente MQTT, a implantação falhou ou não atingiu o dispositivo principal. Isso pode ocorrer se seu dispositivo principal não estiver conectado à Internet ou não tiver as permissões corretas para executar o componente. Execute o comando a seguir em seu dispositivo principal para visualizar o arquivo

de log do software AWS IoT Greengrass principal. Esse arquivo inclui registros do serviço de implantação do dispositivo principal do Greengrass.

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

Para ter mais informações, consulte Solução de problemas de inferência de aprendizado de máquina.

Próximas etapas

Se você tiver um dispositivo principal do Greengrass com uma interface de câmera compatível, você pode concluirTutorial: Execute inferência de classificação de imagens de amostra em imagens de uma câmera usando o TensorFlow Lite, o que mostra como modificar o componente de inferência de amostra para realizar a classificação de imagens em imagens de uma câmera.

Para explorar mais a configuração do componente de inferência de classificação de imagem TensorFlow Lite de amostra, tente o seguinte:

- Modifique o parâmetro `InferenceInterval` de configuração para alterar a frequência com que o código de inferência é executado.
- Modifique os parâmetros `ImageName` e de `ImageDirectory` configuração na configuração do componente de inferência para especificar uma imagem personalizada a ser usada para inferência.

Para obter informações sobre como personalizar a configuração de componentes públicos ou criar componentes personalizados de aprendizado de máquina, consultePersonalize seus componentes de aprendizado de máquina.

Tutorial: Execute inferência de classificação de imagens de amostra em imagens de uma câmera usando o TensorFlow Lite

Este tutorial mostra como usar o componente de inferência de classificação de imagem TensorFlow Lite para realizar exemplos de inferência de classificação de imagens em imagens de uma câmera localmente em um dispositivo principal do Greengrass. Esse componente inclui as seguintes dependências de componentes:

- TensorFlow Componente de armazenamento de modelos de classificação de imagem Lite

- TensorFlow Componente Lite Runtime

> **Note**
>
> Este tutorial acessa o módulo de câmera para dispositivos Raspberry Pi ou NVIDIA Jetson Nano, mas AWS IoT Greengrass oferece suporte a outros dispositivos nas plataformas ARMv7l, Armv8 ou x86_64. Para configurar uma câmera para um dispositivo diferente, consulte a documentação relevante do seu dispositivo.

Para obter mais informações sobre aprendizado de máquina em dispositivos Greengrass, consulte. Executar a inferência de machine learning

Tópicos

Pré-requisitos

Para concluir este tutorial, você deve primeiro concluí-loTutorial: Execute inferência de classificação de imagens de amostra usando o Lite TensorFlow .

Você precisará dos seguintes itens:

- Um dispositivo central Linux Greengrass com uma interface de câmera. Este tutorial acessa o módulo da câmera em um dos seguintes dispositivos compatíveis:
  - Raspberry Pi executando o Raspberry Pi OS (anteriormente chamado de Raspbian)
  - NVIDIA Jetson Nano

  Para obter informações sobre como configurar um dispositivo principal do Greengrass, consulte. Tutorial: Conceitos básicos do AWS IoT Greengrass V2

O dispositivo principal deve atender aos seguintes requisitos:

- Nos principais dispositivos do Greengrass que executam o Amazon Linux 2 ou o Ubuntu 18.04, a GNU C Library (glibc) versão 2.27 ou posterior está instalada no dispositivo.
- Em dispositivos ARMv7L, como o Raspberry Pi, dependências do OpenCV-Python instaladas no dispositivo. Execute o comando a seguir para instalar as dependências.

  ```
  sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
   libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
  ```

- Os dispositivos Raspberry Pi que executam o Raspberry Pi OS Bullseye devem atender aos seguintes requisitos:
  - NumPy 1.22.4 ou posterior instalado no dispositivo. O Raspberry Pi OS Bullseye inclui uma versão anterior do NumPy, então você pode executar o seguinte comando para atualizar NumPy o dispositivo.

    ```
    pip3 install --upgrade numpy
    ```

  - A pilha de câmeras antiga ativada no dispositivo. O Raspberry Pi OS Bullseye inclui uma nova pilha de câmeras que é ativada por padrão e não é compatível, portanto, você deve habilitar a pilha de câmeras antiga.

    Para habilitar a pilha de câmeras antiga

    1. Execute o comando a seguir para abrir a ferramenta de configuração do Raspberry Pi.

       ```
       sudo raspi-config
       ```

    2. Selecione Opções de interface.
    3. Selecione Câmera antiga para ativar a pilha de câmeras antigas.
    4. Reinicie o Raspberry Pi.
- Para dispositivos Raspberry Pi ou NVIDIA Jetson Nano, módulo de câmera Raspberry Pi V2 - 8 megapixels, 1080p. Para aprender a configurar a câmera, consulte Conectar a câmera na documentação do Raspberry Pi.

Etapa 1: configurar o módulo da câmera no seu dispositivo

Nesta etapa, você instala e ativa o módulo da câmera para o seu dispositivo. Execute os seguintes comandos no dispositivo.

Raspberry Pi (Armv7l)

1. Instale a `picamera` interface do módulo da câmera. Execute o comando a seguir para instalar o módulo de câmera e as outras bibliotecas Python necessárias para este tutorial.

   ```
   sudo apt-get install -y python3-picamera
   ```

2. Verifique se o Picamera foi instalado com sucesso.

   ```
   sudo -u ggc_user bash -c 'python3 -c "import picamera"'
   ```

   Se a saída não contiver erros, a validação será bem-sucedida.

   > **Note**
   >
   > Se o arquivo executável do Python instalado em seu dispositivo estiver`python3.7`, use `python3.7` em vez dos `python3` comandos deste tutorial. Verifique se a instalação do pip mapeia para a versão correta do `python3` ou `python3.7` para evitar erros de dependência.

3. Reinicie o dispositivo.

   ```
   sudo reboot
   ```

4. Abra a ferramenta de configuração do Raspberry Pi.

   ```
   sudo raspi-config
   ```

5. Use as setas do teclado para abrir Interfacing Options e habilitar a interface da câmera. Se solicitado, permita que o dispositivo seja reinicializado.
6. Execute o comando a seguir para testar a configuração da câmera.

   ```
   raspistill -v -o test.jpg
   ```

Isso abre uma janela de visualização no Raspberry Pi, salva uma imagem chamada `test.jpg` no seu diretório atual e exibe informações sobre a câmera no terminal do Raspberry Pi.

7. Execute o comando a seguir para criar um link simbólico para permitir que o componente de inferência acesse sua câmera a partir do ambiente virtual criado pelo componente de tempo de execução.

```
sudo ln -s /usr/lib/python3/dist-packages/picamera "MLRootPath/
greengrass_ml_tflite_venv/lib/python3.7/site-packages"
```

O valor padrão RootPath para *ML* neste tutorial é*/greengrass/v2*`/work/variant.TensorFlowLite/greengrass_ml`. A `greengrass_ml_tflite_venv` pasta nesse local é criada quando você implanta o componente de inferência pela primeira vez emTutorial: Execute inferência de classificação de imagens de amostra usando o Lite TensorFlow .

Jetson Nano (Armv8)

1. Execute o comando a seguir para testar a configuração da câmera.

```
gst-launch-1.0 nvarguscamerasrc num-buffers=1 ! "video/x-raw(memory:NVMM),
 width=1920, height=1080, format=NV12, framerate=30/1" ! nvjpegenc ! filesink
 location=test.jpg
```

Isso captura e salva uma imagem nomeada em `test.jpg` seu diretório atual.

2. (Opcional) Reinicie o dispositivo. Se você encontrar problemas ao executar o `gst-launch` comando na etapa anterior, a reinicialização do dispositivo poderá resolver esses problemas.

```
sudo reboot
```

> **Note**
>
> Para dispositivos Armv8 (AArch64), como o Jetson Nano, você não precisa criar um link simbólico para permitir que o componente de inferência acesse a câmera a partir do ambiente virtual criado pelo componente de tempo de execução.

Etapa 2: verifique sua assinatura do tópico de notificações padrão

EmTutorial: Execute inferência de classificação de imagens de amostra usando o Lite TensorFlow , você configurou que o cliente AWS IoT MQTT está configurado no AWS IoT console para assistir às mensagens MQTT publicadas pelo componente de classificação de imagem TensorFlow Lite sobre o `ml/tflite/image-classification` tópico. No AWS IoT console, verifique se essa assinatura existe. Caso contrário, siga as etapas Etapa 1: inscrever-se no tópico de notificações padrão para assinar este tópico antes de implantar o componente em seu dispositivo principal do Greengrass.

Etapa 3: modificar a configuração do componente de classificação de imagem TensorFlow Lite e implantá-la

Nesta etapa, você configura e implanta o componente de classificação de imagem TensorFlow Lite em seu dispositivo principal:

Para configurar e implantar o componente de classificação de imagem TensorFlow Lite (console)

1. No menu de navegação AWS IoT Greengrassdo console, escolha Componentes.
2. Na página Componentes, na guia Componentes públicos, escolha `aws.greengrass.TensorFlowLiteImageClassification`.
3. Na página aws.greengrass.TensorFlowLiteImageClassification, escolha Implantar.
4. Em Adicionar à implantação, escolha uma das seguintes opções:
    a. Para mesclar esse componente a uma implantação existente em seu dispositivo de destino, escolha Adicionar à implantação existente e selecione a implantação que você deseja revisar.
    b. Para criar uma nova implantação em seu dispositivo de destino, escolha Criar nova implantação. Se você tiver uma implantação existente em seu dispositivo, escolher essa etapa substituirá a implantação existente.
5. Na página Especificar destino, faça o seguinte:
    a. Em Informações de implantação, insira ou modifique o nome amigável para sua implantação.
    b. Em Destinos de implantação, selecione um alvo para sua implantação e escolha Avançar. Você não pode alterar o destino de implantação se estiver revisando uma implantação existente.

6. Na página Selecionar componentes, em Componentes públicos, verifique se o `aws.greengrass.TensorFlowLiteImageClassification` componente está selecionado e escolha Avançar.
7. Na página Configurar componentes, faça o seguinte:
   a. Selecione o componente de inferência e escolha Configurar componente.
   b. Em Atualização de configuração, insira a seguinte atualização de configuração na caixa Configuração a ser mesclada.

      ```
      {
        "InferenceInterval": "60",
        "UseCamera": "true"
      }
      ```

      Com essa atualização de configuração, o componente acessa o módulo da câmera em seu dispositivo e realiza inferências nas imagens tiradas pela câmera. O código de inferência é executado a cada 60 segundos.
   c. Escolha Confirmar e, em seguida, Avançar.
8. Na página Definir configurações avançadas, mantenha as configurações padrão e escolha Avançar.
9. Na página Revisar, escolha Implantar

Para configurar e implantar o componente de classificação de imagem TensorFlow Lite (AWS CLI)

1. Crie um `deployment.json` arquivo para definir a configuração de implantação do componente de classificação de imagem TensorFlow Lite. Esse arquivo deve ter a seguinte aparência:

   ```
   {
     "targetArn":"targetArn",
     "components": {
       "aws.greengrass.TensorFlowLiteImageClassification": {
         "componentVersion": 2.1.0,
         "configurationUpdate": {
           "InferenceInterval": "60",
           "UseCamera": "true"
         }
       }
     }
   ```

```
}
```

- No `targetArn` campo, *`targetArn`*substitua pelo nome de recurso da Amazon (ARN) da coisa ou grupo de itens a ser destinado para a implantação, no seguinte formato:
  - Coisa: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`thingName`*
  - Grupo de coisas: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thinggroup/`*`thingGroupName`*
- Este tutorial usa a versão 2.1.0 do componente. No objeto `aws.greengrass.TensorFlowLiteImageClassification` componente, substitua *`2.1.0`* para usar uma versão diferente do componente de classificação de imagem TensorFlow Lite.

Com essa atualização de configuração, o componente acessa o módulo da câmera em seu dispositivo e realiza inferências nas imagens tiradas pela câmera. O código de inferência é executado a cada 60 segundos. Substitua os seguintes valores

2. Execute o comando a seguir para implantar o componente de classificação de imagem TensorFlow Lite no dispositivo:

```
aws greengrassv2 create-deployment \
    --cli-input-json file://path/to/deployment.json
```

A implantação pode levar vários minutos para ser concluída. Na próxima etapa, verifique o log do componente para verificar se a implantação foi concluída com êxito e para ver os resultados da inferência.

Etapa 4: ver os resultados da inferência

Depois de implantar o componente, você pode visualizar os resultados da inferência no registro do componente em seu dispositivo principal do Greengrass e no cliente MQTT AWS IoT no console. AWS IoT Para assinar o tópico no qual o componente publica resultados de inferência, consulte. Etapa 2: verifique sua assinatura do tópico de notificações padrão

- AWS IoTCliente MQTT — Para visualizar os resultados que o componente de inferência publica no tópico de notificações padrão, conclua as seguintes etapas:

  1. No menu de navegação do AWS IoTconsole, escolha Testar, cliente de teste MQTT.
  2. Em Assinaturas, escolha. **`ml/tflite/image-classification`**

- Registro do componente — Para visualizar os resultados da inferência no registro do componente, execute o comando a seguir em seu dispositivo principal do Greengrass.

```
sudo tail -f /greengrass/v2/logs/aws.greengrass.TensorFlowLiteImageClassification.log
```

Se você não conseguir ver os resultados da inferência no log do componente ou no cliente MQTT, a implantação falhou ou não atingiu o dispositivo principal. Isso pode ocorrer se seu dispositivo principal não estiver conectado à Internet ou não tiver as permissões necessárias para executar o componente. Execute o comando a seguir em seu dispositivo principal para visualizar o arquivo de log do software AWS IoT Greengrass principal. Esse arquivo inclui registros do serviço de implantação do dispositivo principal do Greengrass.

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

Para ter mais informações, consulte Solução de problemas de inferência de aprendizado de máquina.

Próximas etapas

Este tutorial mostra como usar o componente de classificação de imagem TensorFlow Lite, com opções de configuração personalizadas para realizar a classificação de imagens de amostra em imagens tiradas por uma câmera.

Para obter mais informações sobre como personalizar a configuração de componentes públicos ou criar componentes personalizados de aprendizado de máquina, consultePersonalize seus componentes de aprendizado de máquina.

Componentes

AWS IoT Greengrasscomponentes são módulos de software que você implanta nos dispositivos principais do Greengrass. Os componentes podem representar aplicativos, instaladores em tempo de execução, bibliotecas ou qualquer código que você executaria em um dispositivo. Você pode definir componentes que dependem de outros componentes. Por exemplo, você pode definir um componente que instala o Python e, em seguida, definir esse componente como uma dependência dos componentes que executam aplicativos Python. Quando você implanta seus componentes em suas frotas de dispositivos, o Greengrass implanta somente os módulos de software que seus dispositivos exigem.

Tópicos

AWS-componentes fornecidos

AWS IoT Greengrass fornece e mantém componentes pré-criados que você pode implantar em seus dispositivos. Esses componentes incluem recursos (como gerenciador de fluxo), conectores AWS IoT Greengrass V1 (como CloudWatch métricas) e ferramentas de desenvolvimento local (como a CLI AWS IoT Greengrass). Você pode implantar esses componentes em seus dispositivos para obter sua funcionalidade independente ou pode usá-los como dependências em seus componentes personalizados do Greengrass.

> **Note**
>
> Vários componentes AWS fornecidos dependem de versões secundárias específicas do núcleo do Greengrass. Por causa dessa dependência, você precisa atualizar esses componentes ao atualizar o núcleo do Greengrass para uma nova versão secundária. Para obter informações sobre as versões específicas do núcleo das quais cada componente depende, consulte o tópico do componente correspondente. Para obter mais informações

sobre a atualização do núcleo, consulteAtualize o software AWS IoT Greengrass principal (OTA).

Quando um componente tem um tipo de componente genérico e Lambda, a versão atual do componente é do tipo genérico e uma versão anterior do componente é do tipo Lambda.

Componente	Descrição	Tipo de componente	SO com suporte	Código aberto
Núcleo Greengrass	O núcleo do software AWS IoT Greengrass Core. Use esse componente para configurar e atualizar o software em seus dispositivos principais.	Núcleo	Linux, Windows	Sim
Autenticação do dispositivo cliente	Permite que dispositivos IoT locais, chamados de dispositivos cliente, se conectem ao dispositivo principal.	Plug-in	Linux, Windows	Sim
CloudWatch métricas	Publica métricas personalizadas	Genérico, Lambda	Linux, Windows	Sim

Componente	Descrição	Tipo de componente	SO com suporte	Código aberto
	na Amazon CloudWatch.			
AWS IoT Device Defender	Notifica os administradores sobre mudanças no estado do dispositivo principal do Greengrass para identificar comportamentos incomuns.	Genérico, Lambda	Linux, Windows	Sim
Spooler de disco	Habilita uma opção de armazenamento persistente para mensagens enviadas dos principais dispositivos do Greengrass para o. AWS IoT Core Esse componente armazenará essas mensagens de saída no disco.	Plug-in	Linux, Windows	Sim

Componente	Descrição	Tipo de componente	SO com suporte	Código aberto
Gerenciador de aplicativos Docker	Permite AWS IoT Greengrass baixar imagens do Docker do Docker Hub e do Amazon Elastic Container Registry (Amazon ECR).	Genérico	Linux, Windows	Não
Conector Edge para Kinesis Video Streams	Lê feeds de vídeo de câmeras locais, publica os streams no Kinesis Video Streams e exibe os streams nos painéis da Grafana com. AWS IoT TwinMaker	Genérico	Linux	Não

Componente	Descrição	Tipo de componente	SO com suporte	Código aberto
CLI do Greengrass	Fornece uma interface de linha de comando que você pode usar para criar implantações locais e interagir com o dispositivo principal do Greengrass e seus componentes.	Plug-in	Linux, Windows	Sim
Detector IP	Relata as informações de conectividade do agente MQTT para AWS IoT Greengrass que os dispositivos do cliente possam descobrir como se conectar.	Plug-in	Linux, Windows	Sim

Componente	Descrição	Tipo de componente	SO com suporte	Código aberto
Firehose	Publica dados por meio de fluxos de entrega do Amazon Data Firehose para destinos no. Nuvem AWS	Lambda	Linux	Não
Lançador Lambda	Lida com processos e configuração de ambiente para funções Lambda.	Genérico	Linux	Não
Gerente do Lambda	Lida com comunicação entre processos e escalabilidade para funções Lambda.	Plug-in	Linux	Não
Runtimes do Lambda	Fornece artefatos para cada tempo de execução do Lambda.	Genérico	Linux	Não

Componente	Descrição	Tipo de componente	SO com suporte	Código aberto
Roteador de assinatura antigo	Gerencia assinaturas para funções Lambda que são executadas na V1. AWS IoT Greengrass	Genérico	Linux	Não
Console de depuração local	Fornece um console local que você pode usar para depurar e gerenciar o dispositivo principal do Greengrass e seus componentes.	Plug-in	Linux, Windows	Sim
Gerenciador de registros	Coleta e carrega registros no dispositivo principal do Greengrass.	Plug-in	Linux, Windows	Sim

Componente	Descrição	Tipo de componente	SO com suporte	Código aberto
Componentes de aprendizado de máquina	Fornece modelos de aprendizado de máquina e exemplos de código de inferência que você pode usar para realizar inferência de aprendizado de máquina nos dispositivos principais do Greengrass.	Consulte Componentes de aprendizado de máquina.		
Adaptador de protocolo Modbus-RTU	Pesquisa informações de dispositivos Modbus RTU locais.	Lambda	Linux	Não

Componente	Descrição	Tipo de componente	SO com suporte	Código aberto
Emissor de telemetria Nucleus	Publica dados de telemetria de integridade do sistema coletados do núcleo para um tópico local ou para um tópico do MQTT. AWS IoT Core	Plug-in	Linux, Windows	Sim
Ponte MQTT	Retransmite mensagens MQTT entre dispositivos clientes, AWS IoT Greengrass publicação/assinatura local e. AWS IoT Core	Plug-in	Linux, Windows	Sim
Corretor MQTT 3.1.1 (Moquette)	Executa um broker MQTT 3.1.1 que manipula mensagens entre dispositivos cliente e o dispositivo principal.	Plug-in	Linux, Windows	Sim

Componente	Descrição	Tipo de componente	SO com suporte	Código aberto
Corretora MQTT 5 (EMQX)	Executa um broker MQTT 5 que manipula mensagens entre dispositivos cliente e o dispositivo principal.	Genérico	Linux, Windows	Não
Fornecedor PKCS #11	Permite que os componentes do Greengrass acessem uma chave privada e um certificado que você armazena com segurança em um módulo de segurança de hardware (HSM).	Plug-in	Linux	Sim

Componente	Descrição	Tipo de componente	SO com suporte	Código aberto
Gerente secreto	Implanta segredos a partir de AWS Secrets Manager segredos para que você possa usar com segurança credenciais, como senhas, em componentes personalizados no dispositivo principal do Greengrass.	Plug-in	Linux, Windows	Sim

Componente	Descrição	Tipo de componente	SO com suporte	Código aberto
Tunelamento seguro	Permite conexões de tunelamento AWS IoT seguras que você pode usar para estabelecer comunicações direcionais com os principais dispositivos do Greengrass que estão protegidos por firewalls restritos.	Genérico	Linux	Não

Componente	Descrição	Tipo de componente	SO com suporte	Código aberto
Gerenciador de sombras	Permite a interação com sombras no dispositivo principal. Ele gerencia o armazenamento de documentos paralelos e também a sincronização dos estados paralelos locais com o serviço AWS IoT Device Shadow.	Plug-in	Linux, Windows	Sim
Amazon SNS	Publica mensagens em tópicos do Amazon SNS.	Lambda	Linux	Não
Gerenciador de fluxo	Transmite dados de alto volume de fontes locais para o. Nuvem AWS	Genérico	Linux, Windows	Não

Componente	Descrição	Tipo de componente	SO com suporte	Código aberto
Agente Systems Manager	Gerencie o dispositivo principal com AWS Systems Manager, o que permite corrigir dispositivos, executar comandos e muito mais.	Genérico	Linux	Não
Serviço de troca de tokens	AWS Fornece credencia is que você pode usar para interagir com os AWS serviços.	Genérico	Linux, Windows	Não
Coletor IoT OPC-UA SiteWise	Coleta dados dos servidores OPC-UA.	Genérico	Linux, Windows	Não
Simulador de fonte de dados IoT SiteWise OPC-UA	Executa um servidor OPC-UA local que gera dados de amostra.	Genérico	Linux, Windows	Não
Editora de IoT SiteWise	Publica dados na AWS nuvem.	Genérico	Linux, Windows	Não

Componente	Descrição	Tipo de componente	SO com suporte	Código aberto
Processador de IoT SiteWise	Processa dados nos dispositivos principais do Greengrass.	Genérico	Linux, Windows	Não

Núcleo Greengrass

O componente do núcleo do Greengrass (`aws.greengrass.Nucleus`) é um componente obrigatório e o requisito mínimo para executar o software AWS IoT Greengrass Core em um dispositivo. Você pode configurar esse componente para personalizar e atualizar seu software AWS IoT Greengrass Core remotamente. Implante esse componente para definir configurações como proxy, função do dispositivo e configuração do AWS IoT item em seus dispositivos principais.

Important

Quando a versão do componente do núcleo muda, ou quando você altera determinados parâmetros de configuração, o software AWS IoT Greengrass Core, que inclui o núcleo e todos os outros componentes do seu dispositivo, reinicia a aplicação das alterações. Quando você implanta um componente, AWS IoT Greengrass instala as versões mais recentes suportadas de todas as dependências desse componente. Por esse motivo, novas versões AWS de patch dos componentes públicos fornecidos podem ser implantadas automaticamente em seus dispositivos principais se você adicionar novos dispositivos a um grupo de coisas ou atualizar a implantação que visa esses dispositivos. Algumas atualizações automáticas, como a atualização do nucleus, podem fazer com que seus dispositivos reiniciem inesperadamente.

Para evitar atualizações não intencionais para um componente que está sendo executado em seu dispositivo, recomendamos que você inclua diretamente sua versão preferida desse componente ao criar uma implantação. Para obter mais informações sobre o comportamento de atualização AWS IoT Greengrass do software Core, consulteAtualize o software AWS IoT Greengrass principal (OTA).

Tópicos

Versões

Esse componente tem as seguintes versões:

- 2.12.x
- 2.11.x
- 2.10.x
- 2.9.x
- 2,8.x
- 2.7.x
- 2.6.x
- 2,5.x
- 2.4.x
- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

Sistema operacional

Esse componente pode ser instalado em dispositivos principais que executam os seguintes sistemas operacionais:

- Linux
- Windows

Para ter mais informações, consulte Plataformas compatíveis.

Requisitos

Os dispositivos devem atender a certos requisitos para instalar e executar o núcleo Greengrass e o AWS IoT Greengrass software Core. Para ter mais informações, consulte Requisitos do dispositivo.

O componente do núcleo do Greengrass tem suporte para ser executado em uma VPC. Para implantar esse componente em uma VPC, é necessário o seguinte.

- O componente do núcleo do Greengrass deve ter conectividade com AWS IoT data, AWS IoT Credentials e Amazon S3.

Dependências

O núcleo do Greengrass não inclui nenhuma dependência de componentes. No entanto, vários componentes AWS fornecidos incluem o núcleo como uma dependência. Para ter mais informações, consulte AWS-componentes fornecidos.

Para obter mais informações sobre dependências de componentes, consulte a referência da receita do componente.

Download e instalação

Você pode baixar um instalador que configura o componente nucleus do Greengrass em seu dispositivo. Esse instalador configura seu dispositivo como um dispositivo principal do Greengrass. Há dois tipos de instalações que você pode realizar: uma instalação rápida que cria AWS os recursos necessários para você ou uma instalação manual em que você mesmo cria os AWS recursos. Para ter mais informações, consulte Instalar o software do AWS IoT Greengrass Core.

Você também pode seguir um tutorial para instalar o núcleo do Greengrass e explorar o desenvolvimento de componentes do Greengrass. Para ter mais informações, consulte Tutorial: Conceitos básicos do AWS IoT Greengrass V2.

Configuração

Esse componente fornece os seguintes parâmetros de configuração que você pode personalizar ao implantar o componente. Alguns parâmetros exigem que o software AWS IoT Greengrass Core seja reiniciado para entrar em vigor. Para obter mais informações sobre por que e como configurar esse componente, consulteConfigurar o software AWS IoT Greengrass principal.

`iotRoleAlias`

O alias de AWS IoT função que aponta para uma função do IAM de troca de tokens. O provedor de AWS IoT credenciais assume essa função para permitir que o dispositivo principal do Greengrass interaja com os serviços. AWS Para ter mais informações, consulte Autorize os dispositivos principais a interagir com os serviços AWS.

Quando você executa o software AWS IoT Greengrass Core com a `--provision true` opção, o software provisiona um alias de função e define seu valor no componente do núcleo.

`interpolateComponentConfiguration`

(Opcional) Você pode habilitar o núcleo do Greengrass para interpolar variáveis de receitas de componentes em configurações de componentes e mesclar atualizações de configuração. Recomendamos que você defina essa opção para `true` que o dispositivo principal possa executar componentes do Greengrass que usam variáveis de receita em suas configurações.

Esse recurso está disponível para a versão 2.6.0 e posterior desse componente.

Padrão: `false`

`networkProxy`

(Opcional) O proxy de rede a ser usado em todas as conexões. Para ter mais informações, consulte Conectar-se à porta 443 ou por meio de um proxy de rede.

> **Important**
>
> Quando você implanta uma alteração nesse parâmetro de configuração, o software AWS IoT Greengrass principal é reiniciado para que a alteração entre em vigor.

Esse objeto contém as seguintes informações:

`noProxyAddresses`

(Opcional) Uma lista separada por vírgulas de endereços IP ou nomes de host que estão isentos do proxy.

`proxy`

O proxy ao qual se conectar. Esse objeto contém as seguintes informações:

`url`

O URL do servidor proxy no formato`scheme://userinfo@host:port`.

- `scheme`— O esquema, que deve ser `http` ou`https`.

  > **Important**
  >
  > Os dispositivos principais do Greengrass devem executar o Greengrass nucleus v2.5.0 ou posterior para usar proxies HTTPS.
  > Se você configurar um proxy HTTPS, deverá adicionar o certificado CA do servidor proxy ao certificado de CA raiz da Amazon do dispositivo principal. Para ter mais informações, consulte Permita que o dispositivo principal confie em um proxy HTTPS.

- `userinfo`— (Opcional) As informações de nome de usuário e senha. Se você especificar essas informações no`url`, o dispositivo principal do Greengrass ignorará os campos e. `username` `password`
- `host`— O nome do host ou endereço IP do servidor proxy.
- `port`— (Opcional) O número da porta. Se você não especificar a porta, o dispositivo principal do Greengrass usará os seguintes valores padrão:
  - `http`— 80
  - `https`— 443

`username`

(Opcional) O nome de usuário que autentica o servidor proxy.

`password`

(Opcional) A senha que autentica o servidor proxy.

`mqtt`

(Opcional) A configuração MQTT para o dispositivo principal do Greengrass. Para ter mais informações, consulte Conectar-se à porta 443 ou por meio de um proxy de rede.

> **Important**
>
> Quando você implanta uma alteração nesse parâmetro de configuração, o software AWS IoT Greengrass principal é reiniciado para que a alteração entre em vigor.

Esse objeto contém as seguintes informações:

`port`

(Opcional) A porta a ser usada para conexões MQTT.

Padrão: `8883`

`keepAliveTimeoutMs`

(Opcional) A quantidade de tempo em milissegundos entre cada `PING` mensagem que o cliente envia para manter a conexão MQTT ativa. Esse valor deve ser maior que`pingTimeoutMs`.

Padrão: `60000` (60 segundos)

`pingTimeoutMs`

(Opcional) A quantidade de tempo em milissegundos que o cliente espera para receber uma `PINGACK` mensagem do servidor. Se a espera exceder o tempo limite, o dispositivo principal fecha e reabre a conexão MQTT. Esse valor deve ser menor que`keepAliveTimeoutMs`.

Padrão: `30000` (30 segundos)

`operationTimeoutMs`

(Opcional) A quantidade de tempo em milissegundos que o cliente espera que as operações do MQTT (como `CONNECT` ou`PUBLISH`) sejam concluídas. Essa opção não se aplica ao MQTT `PING` nem às mensagens de manutenção ativa.

Padrão: `30000` (30 segundos)

`maxInFlightPublishes`

(Opcional) O número máximo de mensagens de QoS 1 não confirmadas do MQTT que podem estar em andamento ao mesmo tempo.

Esse recurso está disponível para a versão 2.1.0 e posterior desse componente.

Padrão: `5`

Intervalo válido: valor máximo de 100

`maxMessageSizeInBytes`

(Opcional) O tamanho máximo de uma mensagem MQTT. Se uma mensagem exceder esse tamanho, o núcleo do Greengrass a rejeitará com um erro.

Esse recurso está disponível para a versão 2.1.0 e posterior desse componente.

Padrão: `131072` (128 KB)

Intervalo válido: valor máximo de `2621440` (2,5 MB)

`maxPublishRetry`

(Opcional) O número máximo de vezes para repetir uma mensagem que não foi publicada. Você pode especificar `-1` para tentar novamente vezes ilimitadas.

Esse recurso está disponível para a versão 2.1.0 e posterior desse componente.

Padrão: `100`

`spooler`

(Opcional) A configuração do spooler MQTT para o dispositivo principal Greengrass. Esse objeto contém as seguintes informações:

`storageType`

O tipo de armazenamento para armazenar mensagens. Se `storageType` estiver definido como`Disk`, o `pluginName` pode ser configurado. Você pode especificar `Memory` ou `Disk`.

Esse recurso está disponível para a versão 2.11.0 e posterior do componente núcleo do Greengrass.

> **Important**
>
> Se o spooler MQTT `storageType` estiver definido como `Disk` e você quiser fazer o downgrade do Greengrass nucleus da versão 2.11.x para uma versão anterior, você deverá alterar a configuração novamente para. `Memory` A única configuração para `storageType` isso é suportada nas versões 2.10.x e anteriores do Greengrass nucleus é. `Memory` Não seguir essas orientações pode resultar na quebra do spooler. Isso faria com que seu dispositivo principal do Greengrass não conseguisse enviar mensagens MQTT para o. Nuvem AWS

Padrão: `Memory`

`pluginName`

(Opcional) O nome do componente do plug-in. Esse componente só será usado se `storageType` estiver definido como`Disk`. Essa opção é padronizada `aws.greengrass.DiskSpooler` e usará o fornecido pelo Greengrass. Spooler de disco

Esse recurso está disponível para a versão 2.11.0 e posterior do componente núcleo do Greengrass.

Padrão: `"aws.greengrass.DiskSpooler"`

`maxSizeInBytes`

(Opcional) O tamanho máximo do cache em que o dispositivo principal armazena mensagens MQTT não processadas na memória. Se o cache estiver cheio, novas mensagens serão rejeitadas.

Padrão: `2621440` (2,5 MB)

`keepQos0WhenOffline`

(Opcional) Você pode agrupar mensagens de QoS 0 do MQTT que o dispositivo principal recebe enquanto está off-line. Se você definir essa opção como`true`, o dispositivo principal armazenará mensagens de QoS 0 que ele não pode enviar enquanto estiver off-line. Se você definir essa opção como`false`, o dispositivo principal descartará essas mensagens. O dispositivo principal sempre armazena mensagens de QoS 1, a menos que o spool esteja cheio.

Padrão: `false`

`version`

(Opcional) A versão do MQTT. Você pode especificar `mqtt3` ou `mqtt5`.

Esse recurso está disponível para a versão 2.10.0 e posterior do componente núcleo do Greengrass.

Padrão: `mqtt5`

`receiveMaximum`

(Opcional) O número máximo de pacotes de QoS1 não reconhecidos que o agente pode enviar.

Esse recurso está disponível para a versão 2.10.0 e posterior do componente núcleo do Greengrass.

Padrão: `100`

`sessionExpirySeconds`

(Opcional) A quantidade de tempo em segundos que você pode solicitar para que uma sessão dure a partir do IoT Core. O padrão é o tempo máximo suportado pelo AWS IoT Core.

Esse recurso está disponível para a versão 2.10.0 e posterior do componente núcleo do Greengrass.

Padrão: `604800 (7 days)`

`minimumReconnectDelaySeconds`

(Opcional) Uma opção para o comportamento de reconexão. O tempo mínimo em segundos para o MQTT se reconectar.

Esse recurso está disponível para a versão 2.10.0 e posterior do componente núcleo do Greengrass.

Padrão: `1`

`maximumReconnectDelaySeconds`

(Opcional) Uma opção para o comportamento de reconexão. O tempo máximo em segundos para o MQTT se reconectar.

Esse recurso está disponível para a versão 2.10.0 e posterior do componente núcleo do Greengrass.

Padrão: `120`

`minimumConnectedTimeBeforeRetryResetSeconds`

(Opcional) Uma opção para o comportamento de reconexão. A quantidade de tempo em segundos em que uma conexão deve estar ativa antes que o atraso da nova tentativa seja redefinido ao mínimo.

Esse recurso está disponível para a versão 2.10.0 e posterior do componente núcleo do Greengrass.

Padrão: `30`

`jvmOptions`

(Opcional) As opções de JVM a serem usadas para executar o software AWS IoT Greengrass Core. Para obter informações sobre as opções recomendadas de JVM para executar o software AWS IoT Greengrass Core, consulte. Controle a alocação de memória com opções de JVM

> **Important**
>
> Quando você implanta uma alteração nesse parâmetro de configuração, o software AWS IoT Greengrass principal é reiniciado para que a alteração entre em vigor.

`iotDataEndpoint`

O endpoint de AWS IoT dados para seu. Conta da AWS

Quando você executa o software AWS IoT Greengrass Core com a `--provision true` opção, o software obtém seus endpoints de dados e credenciais AWS IoT e os define no componente nuclear.

`iotCredEndpoint`

O endpoint AWS IoT de credenciais para seu. Conta da AWS

Quando você executa o software AWS IoT Greengrass Core com a `--provision true` opção, o software obtém seus endpoints de dados e credenciais AWS IoT e os define no componente nuclear.

`greengrassDataPlaneEndpoint`

Esse recurso está disponível na versão 2.7.0 e posterior desse componente.

Para ter mais informações, consulte Use um certificado de dispositivo assinado por uma CA privada.

`greengrassDataPlanePort`

Esse recurso está disponível na versão 2.0.4 e posterior desse componente.

(Opcional) A porta a ser usada para conexões de plano de dados. Para ter mais informações, consulte Conectar-se à porta 443 ou por meio de um proxy de rede.

> **Important**
>
> Você deve especificar uma porta na qual o dispositivo possa fazer conexões de saída. Se você especificar uma porta bloqueada, o dispositivo não conseguirá se conectar AWS IoT Greengrass para receber implantações.

Escolha uma das seguintes opções:

- `443`
- `8443`

Padrão: `8443`

`awsRegion`

O Região da AWS para usar.

`runWithDefault`

O usuário do sistema a ser usado para executar componentes.

> **Important**
>
> Quando você implanta uma alteração nesse parâmetro de configuração, o software AWS IoT Greengrass principal é reiniciado para que a alteração entre em vigor.

Esse objeto contém as seguintes informações:

`posixUser`

O nome ou ID do usuário do sistema e, opcionalmente, do grupo do sistema que o dispositivo principal usa para executar componentes genéricos e Lambda. Especifique o usuário e

o grupo separando-os por dois pontos (`:`), no seguinte formato: `user:group`. O grupo é opcional. Se você não especificar um grupo, o software AWS IoT Greengrass Core usará o grupo primário para o usuário. Por exemplo, é possível especificar `ggc_user` ou `ggc_user:ggc_group`. Para ter mais informações, consulte Configurar o usuário que executa os componentes.

Quando você executa o instalador do software AWS IoT Greengrass Core com a `--component-default-user` *`ggc_user:ggc_group`* opção, o software define esse parâmetro no componente do núcleo.

`windowsUser`

Esse recurso está disponível na versão 2.5.0 e posterior desse componente.

O nome do usuário do Windows a ser usado para executar esse componente nos dispositivos principais do Windows. O usuário deve existir em cada dispositivo principal do Windows, e seu nome e senha devem ser armazenados na instância do Gerenciador de Credenciais da LocalSystem conta. Para ter mais informações, consulte Configurar o usuário que executa os componentes.

Quando você executa o instalador do software AWS IoT Greengrass Core com a `--component-default-user` *`ggc_user`* opção, o software define esse parâmetro no componente do núcleo.

`systemResourceLimits`

Esse recurso está disponível na versão 2.4.0 e posterior desse componente. AWS IoT Greengrass atualmente não oferece suporte a esse recurso nos dispositivos principais do Windows.

Por padrão, os limites de recursos do sistema devem ser aplicados a processos de componentes Lambda genéricos e não conteinerizados. Você pode substituir os limites de recursos do sistema para componentes individuais ao criar uma implantação. Para ter mais informações, consulte Configurar limites de recursos do sistema para componentes.

Esse objeto contém as seguintes informações:

`cpus`

A quantidade máxima de tempo de CPU que os processos de cada componente podem usar no dispositivo principal. O tempo total da CPU de um dispositivo essencial é

equivalente ao número de núcleos da CPU do dispositivo. Por exemplo, em um dispositivo principal com 4 núcleos de CPU, você pode definir esse valor 2 para limitar os processos de cada componente a 50% de uso de cada núcleo da CPU. Em um dispositivo com 1 núcleo de CPU, você pode definir esse valor `0.25` para limitar os processos de cada componente a 25% de uso da CPU. Se você definir esse valor como um número maior que o número de núcleos de CPU, o software AWS IoT Greengrass Core não limitará o uso da CPU dos componentes.

`memory`

A quantidade máxima de RAM (em kilobytes) que os processos de cada componente podem usar no dispositivo principal.

`s3EndpointType`

(Opcional) O tipo de endpoint S3. Esse parâmetro só terá efeito para a região Leste dos EUA (Norte da Virgíniaus-east-1) (). A configuração desse parâmetro em qualquer outra região será ignorada. Escolha uma das seguintes opções:

- `REGIONAL`— O cliente S3 e o URL pré-assinado usam o endpoint regional.
- `GLOBAL`— O cliente S3 e o URL pré-assinado usam o endpoint legado.

Padrão: `GLOBAL`

`logging`

(Opcional) A configuração de registro do dispositivo principal. Para obter mais informações sobre como configurar e usar os registros do Greengrass, consulte. Monitore AWS IoT Greengrass os registros

Esse objeto contém as seguintes informações:

`level`

(Opcional) O nível mínimo de mensagens de log a serem enviadas.

Escolha entre os seguintes níveis de registro, listados aqui em ordem de nível:

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

Padrão: `INFO`

`format`

(Opcional) O formato de dados dos registros. Escolha uma das seguintes opções:

- `TEXT`— Escolha essa opção se quiser visualizar os registros em formato de texto.
- `JSON`— Escolha essa opção se quiser visualizar os registros com o comando de logs do Greengrass CLI ou interagir com os registros de forma programática.

Padrão: `TEXT`

`outputType`

(Opcional) O tipo de saída para registros. Escolha uma das seguintes opções:

- `FILE`— O software AWS IoT Greengrass Core envia registros para arquivos no diretório que você especifica. `outputDirectory`
- `CONSOLE`— O software AWS IoT Greengrass Core imprime registros em`stdout`. Escolha essa opção para ver os registros à medida que o dispositivo principal os imprime.

Padrão: `FILE`

`fileSizeKB`

(Opcional) O tamanho máximo de cada arquivo de log (em kilobytes). Depois que um arquivo de log excede esse tamanho máximo de arquivo, o software AWS IoT Greengrass Core cria um novo arquivo de log.

Esse parâmetro se aplica somente quando você especifica `FILE` para`outputType`.

Padrão: `1024`

`totalLogsSizeKB`

(Opcional) O tamanho total máximo dos arquivos de log (em kilobytes) para cada componente, incluindo o núcleo do Greengrass. Os arquivos de log do Greengrass nucleus também incluem registros de componentes de plug-ins. Depois que o tamanho total dos arquivos de log de um componente excede esse tamanho máximo, o software AWS IoT Greengrass Core exclui os arquivos de log mais antigos desse componente.

Esse parâmetro é equivalente ao parâmetro de limite de espaço em disco do componente gerenciador de registros (`diskSpaceLimit`), que você pode especificar para o núcleo Greengrass (sistema) e cada componente. O software AWS IoT Greengrass Core usa o

mínimo dos dois valores como o tamanho máximo total do log para o núcleo do Greengrass e cada componente.

Esse parâmetro se aplica somente quando você especifica `FILE` para`outputType`.

Padrão: `10240`

`outputDirectory`

(Opcional) O diretório de saída dos arquivos de log.

Esse parâmetro se aplica somente quando você especifica `FILE` para`outputType`.

Padrão:*`/greengrass/v2`*`/logs`, onde *`/greengrass/v2`* está a pasta AWS IoT Greengrass raiz.

`fleetstatus`

Esse parâmetro está disponível na versão 2.1.0 e posterior desse componente.

(Opcional) A configuração do status da frota para o dispositivo principal.

Esse objeto contém as seguintes informações:

`periodicStatusPublishIntervalSeconds`

(Opcional) A quantidade de tempo (em segundos) entre o qual o dispositivo principal publica o status do dispositivo no Nuvem AWS.

Mínimo: `86400` (24 horas)

Padrão: `86400` (24 horas)

`telemetry`

(Opcional) A configuração de telemetria de integridade do sistema para o dispositivo principal. Para obter mais informações sobre métricas de telemetria e como agir com base nos dados de telemetria, consulte. Colete dados de telemetria de integridade do sistema a partir dos dispositivos principais AWS IoT Greengrass

Esse objeto contém as seguintes informações:

`enabled`

(Opcional) Você pode ativar ou desativar a telemetria.

Padrão: `true`

`periodicAggregateMetricsIntervalSeconds`

(Opcional) O intervalo (em segundos) durante o qual o dispositivo principal agrega métricas.

Se você definir esse valor abaixo do valor mínimo suportado, o núcleo usará o valor padrão em vez disso.

Mínimo: `3600`

Padrão: `3600`

`periodicPublishMetricsIntervalSeconds`

(Opcional) A quantidade de tempo (em segundos) entre o qual o dispositivo principal publica métricas de telemetria no. Nuvem AWS

Se você definir esse valor abaixo do valor mínimo suportado, o núcleo usará o valor padrão em vez disso.

Mínimo: `86400`

Padrão: `86400`

`deploymentPollingFrequencySeconds`

(Opcional) O período em segundos para pesquisar as notificações de implantação.

Padrão: `15`

`componentStoreMaxSizeBytes`

(Opcional) O tamanho máximo em disco do armazenamento de componentes, que inclui receitas e artefatos de componentes.

Padrão: `10000000000` (10 GB)

`platformOverride`

(Opcional) Um dicionário de atributos que identifica a plataforma do dispositivo principal. Use isso para definir atributos de plataforma personalizados que as receitas de componentes podem usar para identificar o ciclo de vida e os artefatos corretos do componente. Por exemplo, você pode definir um atributo de capacidade de hardware para implantar somente o conjunto mínimo de

artefatos para execução de um componente. Para obter mais informações, consulte o parâmetro da plataforma de manifesto na receita do componente.

Você também pode usar esse parâmetro para substituir os atributos `os` e da `architecture` plataforma do dispositivo principal.

`httpClient`

Esse parâmetro está disponível na versão 2.5.0 e posterior desse componente.

(Opcional) A configuração do cliente HTTP para o dispositivo principal. Essas opções de configuração se aplicam a todas as solicitações HTTP feitas por esse componente. Se um dispositivo principal for executado em uma rede mais lenta, você poderá aumentar essas durações de tempo limite para evitar que as solicitações HTTP atinjam o tempo limite.

Esse objeto contém as seguintes informações:

`connectionTimeoutMs`

(Opcional) A quantidade de tempo (em milissegundos) de espera pela abertura de uma conexão antes que a solicitação de conexão expire.

Padrão: `2000` (2 segundos)

`socketTimeoutMs`

(Opcional) A quantidade de tempo (em milissegundos) de espera pela transferência dos dados por uma conexão aberta antes que a conexão expire.

Padrão: `30000` (30 segundos)

Example Exemplo: atualização da mesclagem de configurações

```
{
  "iotRoleAlias": "GreengrassCoreTokenExchangeRoleAlias",
  "networkProxy": {
    "noProxyAddresses": "http://192.168.0.1,www.example.com",
    "proxy": {
      "url": "http://my-proxy-server:1100",
      "username": "Mary_Major",
      "password": "pass@word1357"
    }
  },
```

```
  "mqtt": {
    "port": 443
  },
  "greengrassDataPlanePort": 443,
  "jvmOptions": "-Xmx64m",
  "runWithDefault": {
    "posixUser": "ggc_user:ggc_group"
  }
}
```

Arquivo de log local

Esse componente usa o seguinte arquivo de log.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

Para ver os registros desse componente

- Execute o comando a seguir no dispositivo principal para visualizar o arquivo de log desse componente em tempo real. Substitua */greengrass/v2* ou *C:\greengrass\v2* pelo caminho para a pasta AWS IoT Greengrass raiz.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

Changelog

A tabela a seguir descreve as alterações em cada versão do componente.

Version (Versão)	Alterações
2.12.6	Correções de erros e melhorias • Corrige um problema que causa uma falha na inicialização em determinados processadores ARMv8, incluindo o Jetson Nano.
2.12.5	Correções de erros e melhorias • Corrige um problema em que a reversão da implantação ocasional mente trava ao reverter um componente anteriormente quebrado com dependências rígidas. • Corrige um problema em que o núcleo não publica atualizações de status após o provisionamento da frota. • Adiciona novas tentativas para a `GetDeploymentConfiguration` API após receber erros 404.
2.12.4	Correções de erros e melhorias • Corrige um problema em que o núcleo entra em uma condição de impasse durante a inicialização em alguns dispositivos Linux.
2.12.3	**Warning** Essa versão não está mais disponível. As melhorias nesta versão estão disponíveis em versões posteriores desse componente. Correções de erros e melhorias • Corrige um problema em que o núcleo não relata o status correto do componente após a reinicialização do núcleo e durante a recuperação do componente. • Melhorias e correções de erros gerais.
2.12.2	Correções de erros e melhorias • Corrige um problema em que os registros antigos não eram limpos adequadamente. • Melhorias e correções de erros gerais.

Version (Versão)	Alterações
2.12.1	Correções de erros e melhorias • Corrige um problema em que o núcleo pode duplicar as assinaturas do MQTT para tópicos de implantação, levando a registros adicionais e publicações do MQTT.
2.12.0	Novos atributos • Permite que você execute as etapas do ciclo de vida do bootstrap como parte de uma implantação de reversão.
2.11.3	Correções de erros e melhorias • Corrige um problema no núcleo em que ele pode iniciar incorretamente um componente quando suas dependências falham. Novos atributos • Adiciona um tipo de endpoint s3 configurável.
2.11.2	Correções de erros e melhorias • Corrige um problema no cliente nucleus MQTT 5 em que ele pode aparecer offline quando um grande número (> 50) de assinaturas está em uso. • Adiciona uma nova tentativa para a falha do docker dial TCP.
2.11.1	Correções de erros e melhorias • Corrige um problema em que o núcleo não é iniciado se uma tarefa de bootstrap falhar e o arquivo de metadados de implantação estiver corrompido. • Corrige um problema em que os componentes Lambda sob demanda não são relatados nas atualizações de status de implantação. • Adiciona suporte para IDs de política de autorização duplicados.

Version (Versão)	Alterações
2.11.0	Novos atributos • Permite que você cancele uma implantação local. • Permite que você configure uma política de tratamento de falhas para uma implantação local. • Adiciona suporte para um plug-in de spooler de disco.
2.10.3	Correções de erros e melhorias • Corrige um problema em que o Greengrass não se inscreve para receber notificações de implantação ao usar o provedor PKCS #11.
2.10.2	Correções de erros e melhorias • Permite a análise dos ciclos de vida dos componentes sem distinção entre maiúsculas e minúsculas. • Corrige um problema em que a variável PATH do ambiente não foi recriada corretamente. • Corrige a codificação de URI de proxy para componentes, incluindo gerenciador de fluxo para nomes de usuário com caracteres especiais.
2.10.1	Correções de erros e melhorias • Corrige um problema que poderia causar uma falha na inicialização em determinados processadores ARMv8, incluindo o Jetson Nano. • O Greengrass não fecha mais o padrão de um componente, isso reverte o comportamento para o comportamento anterior à 2.10.0

Version (Versão)	Alterações
2.10.0	Novos atributos • Adiciona `interpolateComponentConfiguration` suporte para a expressão regular vazia. O Greengrass agora interpola a partir do objeto de configuração raiz. • Adiciona suporte para MQTT5. • Adiciona um mecanismo para carregar componentes do plug-in rapidamente sem digitalizar. • Permite que o Greengrass economize espaço em disco excluindo imagens não utilizadas do Docker. Correções de erros e melhorias • Corrige um problema em que a reversão deixa determinados valores de configuração em vigor a partir de uma implantação. • Corrige um problema em que o núcleo do Greengrass valida uma sequência de AWS domínio em terminais de dados e não AWS credenciais personalizados. • Atualiza a resolução de dependências de vários grupos para reresolve r todas as dependências do grupo por meio de Nuvem AWS negociaçã o, em vez de se restringir à versão ativa. Essa atualização também remove o código de erro de implantação`INSTALLED_COMPONEN T_NOT_FOUND` . • Atualiza o núcleo do Greengrass para pular o download de imagens do Docker quando elas já existem localmente. • Atualiza o núcleo do Greengrass para reiniciar uma etapa de instalação do componente antes que o tempo limite expire. • Pequenas correções e melhorias adicionais.

Version (Versão)	Alterações
2.9.6	Correções de erros e melhorias • Corrige um problema em que uma implantação do Greengrass falha com o erro LAUNCH_DIRECTORY_CORRUPTED e uma reinicialização subsequente do dispositivo falha ao iniciar o Greengrass. Esse erro pode ocorrer quando você move o dispositivo Greengrass entre vários grupos de coisas com implantações que exigem a reinicialização do Greengrass.
2.9.5	Novos atributos • Adiciona suporte à verificação de assinatura do software Greengrass nucleus. Correções de erros e melhorias • Corrige um problema em que uma implantação falha quando a região de metadados da receita local não corresponde à região de lançamento do núcleo Greengrass. O núcleo do Greengrass agora renegocia com a nuvem quando isso acontece. • Corrige um problema em que o spooler de mensagens do MQTT é preenchido e nunca remove mensagens. • Pequenas correções e melhorias adicionais.
2.9.4	Correções de erros e melhorias • Verifica se há uma mensagem nula antes de descartar mensagens de QOS 0. • Trunca os valores detalhados do status do trabalho se eles excederem o limite de 1024 caracteres. • Atualiza o script de bootstrap para Windows para ler corretamente o caminho raiz do Greengrass se esse caminho incluir espaços. • Atualiza a assinatura para AWS IoT Core que ela elimine as mensagens do cliente se a resposta da assinatura não for enviada. • Garante que o núcleo carregue sua configuração a partir dos arquivos de backup quando o arquivo de configuração principal estiver corrompid o ou ausente.

Version (Versão)	Alterações
2.9.3	Correções de erros e melhorias • Garante que as IDs do cliente MQTT não sejam duplicadas. • Adiciona leitura e gravação de arquivos mais robustas para evitar e se recuperar da corrupção. • Tenta novamente o docker image pull em caso de erros específicos relacionados à rede. • Adiciona a `noProxyAddresses` opção de conexão MQTT.
2.9.2	Correções de erros e melhorias • Corrige um problema em que a configuração `interpolateComponentConfiguration` não se aplica a uma implantação contínua. • Usa o OSHI para listar todos os processos secundários.
2.9.1	Correções de erros e melhorias • Adiciona uma correção em que o Greengrass é reiniciado se uma implantação remover um componente de plug-in.
2.9.0	Novos atributos • Adiciona a capacidade de criar subimplantações que repetem implantações com um subconjunto menor de dispositivos. Esse recurso cria uma maneira mais eficiente de testar e resolver implantações malsucedidas. Correções de erros e melhorias • Melhora o suporte para sistemas que não têm `useraddgroupadd`, `usermod` e. • Pequenas correções e melhorias adicionais.

Version (Versão)	Alterações
2.8.1	Correções de erros e melhorias • Corrige um problema em que os códigos de erro de implantação não eram gerados corretamente a partir dos erros da API do Greengrass. • Corrige um problema em que as atualizações de status da frota enviam informações imprecisas quando um componente atinge um `ERRORED` estado durante uma implantação. • Corrige um problema em que as implantações não podiam ser concluídas quando o Greengrass tinha mais de 50 assinaturas existente s.

Version (Versão)	Alterações
2.8.0	Novos atributos • Atualiza o núcleo do Greengrass para relatar uma resposta do status de integridade da implantação que inclui códigos de erro detalhado s quando há um problema na implantação de componentes em um dispositivo principal. Para ter mais informações, consulte Códigos de erro de implantação detalhados. • Atualiza o núcleo do Greengrass para relatar uma resposta ao status de integridade do componente que inclui códigos de erro detalhados quando um componente entra no `BROKEN` estado ou. `ERRORED` Para ter mais informações, consulte Códigos detalhados de status do component e. • Expande os campos de mensagens de status para melhorar as informações de disponibilidade na nuvem para dispositivos. • Melhora a robustez do serviço de status da frota. Correções de erros e melhorias • Permite que um componente quebrado seja reinstalado quando sua configuração for alterada. • Corrige um problema em que a reinicialização do núcleo durante a implantação do bootstrap causa uma falha na implantação. • Corrige um problema no Windows em que a instalação falha quando um caminho raiz contém espaços. • Corrige um problema em que um componente desligado durante uma implantação usa o script de desligamento da nova versão. • Várias melhorias no desligamento. • Pequenas correções e melhorias adicionais.

Version (Versão)	Alterações
2.7.0	Novos atributos • Atualiza o núcleo do Greengrass para enviar atualizações de status para a AWS IoT Greengrass nuvem quando o dispositivo principal aplica uma implantação local. • Adiciona suporte para certificados de cliente assinados por uma autoridade de certificação (CA) personalizada, na qual a CA não está registrada AWS IoT. Para usar esse recurso, você pode definir a nova opção `greengrassDataPlaneEndpoint` de configuração como`iotdata`. Para ter mais informações, consulte Use um certificado de dispositivo assinado por uma CA privada. Correções de erros e melhorias • Corrige um problema em que o núcleo do Greengrass reverte uma implantação em determinados cenários quando o núcleo é interromp ido ou reiniciado. O núcleo agora retoma a implantação após a reinicial ização do núcleo. • Atualiza o instalador do Greengrass para respeitar o `--start` argumento quando você especifica a configuração do software como um serviço do sistema. • Atualiza o comportamento de SubscribeToComponentUpdates definir o ID de implantação em eventos em que o núcleo atualizou um componente. • Pequenas correções e melhorias adicionais.

Version (Versão)	Alterações
2.6.0	Novos atributos • Adiciona suporte para curingas MQTT quando você assina tópicos locais de publicação/assinatura. Para obter mais informações, consulte Publique/assine mensagens locais e SubscribeToTopic. • Adiciona suporte para variáveis de receita em configurações de componentes, além da variável de *component_dependency_name* :configuration: *json_pointer* receita. Você pode usar essas variáveis de receitas ao definir um componente DefaultConfiguration em uma receita ou ao configurar um componente e em uma implantação. Para ativar esse recurso, defina a opção de ComponentConfiguration configuração de interpolação como. true Para obter mais informações, consulte Variáveis da receita e Use variáveis de receita em atualizações de mesclagem. • Adiciona suporte total ao caractere * curinga nas políticas de autorização de comunicação entre processos (IPC). Agora você pode especificar o * caractere em uma string de recursos para corresponder a qualquer combinação de caracteres. Para ter mais informações, consulte Caracteres curingas nas políticas de autorização. • Adiciona suporte para componentes personalizados para chamar operações de IPC que a CLI do Greengrass usa. Você pode usar essas operações de IPC para gerenciar implantações locais, visualizar detalhes do componente e gerar uma senha que pode ser usada para entrar no console de depuração local. Para obter mais informações, consulte IPC: gerenciar implantações e componentes locais. Correções de erros e melhorias • Corrige um problema em que os componentes dependentes não reagiam quando suas dependências rígidas reiniciavam ou mudavam de estado em determinados cenários. • Melhora as mensagens de erro que o dispositivo principal reporta ao serviço de AWS IoT Greengrass nuvem quando uma implantação falha. • Corrige um problema em que o núcleo do Greengrass aplicava a implantação de uma coisa duas vezes em determinados cenários quando o núcleo era reiniciado.

Version (Versão)	Alterações
	• Pequenas correções e melhorias adicionais. Para obter mais informaçõ es, consulte os lançamentos em GitHub.
2.5.6	Novos atributos • Adiciona suporte para módulos de segurança de hardware que usam chaves ECC. Você pode usar um módulo de segurança de hardware (HSM) para armazenar com segurança a chave privada e o certifica do do dispositivo. Para ter mais informações, consulte Integração de segurança de hardware. Correções de erros e melhorias • Corrige um problema em que a implantação nunca é concluída quando você implanta um componente com um script de instalação incorreto em determinados cenários. • Melhora o desempenho durante a inicialização. • Pequenas correções e melhorias adicionais.
2.5.5	Novos atributos • Adiciona a variável de `GG_ROOT_CA_PATH` ambiente para component es, para que você possa acessar o certificado de autoridade de certifica ção (CA) raiz em componentes personalizados. Correções de erros e melhorias • Adiciona suporte para dispositivos Windows que usam um idioma de exibição diferente do inglês. • Atualiza como o núcleo do Greengrass analisa os argumentos do instalador booleano, para que você possa especificar um argumento booleano sem um valor booleano para especificar um valor. `true` Por exemplo, agora você pode especificar, `--provision` em vez de `--provision true` instalar, com o provisionamento automático de recursos. • Corrige um problema em que o dispositivo principal não reportava seu status ao serviço de AWS IoT Greengrass nuvem após o provision amento em determinados cenários. • Pequenas correções e melhorias adicionais.

Version (Versão)	Alterações
2.5.4	Correções de erros e melhorias • Melhorias e correções de erros gerais.
2.5.3	Novos atributos • Adiciona suporte para integração de segurança de hardware. Você pode usar um módulo de segurança de hardware (HSM) para armazenar com segurança a chave privada e o certificado do dispositivo. Para ter mais informações, consulte Integração de segurança de hardware. Correções de erros e melhorias • Corrige um problema com exceções de tempo de execução enquanto o núcleo estabelece conexões MQTT com o. AWS IoT Core
2.5.2	Correções de erros e melhorias • Corrige um problema em que, após as atualizações do núcleo do Greengrass, o serviço Windows não é iniciado novamente depois que você o interrompe ou reinicializa o dispositivo.

Version (Versão)	Alterações
2.5.1	**Warning** Essa versão não está mais disponível. As melhorias nesta versão estão disponíveis em versões posteriores desse componente. Correções de erros e melhorias • Adiciona suporte para versões de 32 bits do Java Runtime Environment (JRE) no Windows. • Altera o comportamento de remoção de grupos de coisas para dispositivos principais cuja AWS IoT política não concede a `greengrass:ListThingGroupsForCoreDevice` permissão. Com essa versão, a implantação continua, registra um aviso e não remove componentes quando você remove o dispositivo principal de um grupo de coisas. Para ter mais informações, consulte Implemente AWS IoT Greengrass componentes em dispositivos. • Corrige um problema com as variáveis de ambiente do sistema que o núcleo do Greengrass disponibiliza para os processos de componentes do Greengrass. Agora você pode reiniciar um componente para que ele use as variáveis de ambiente do sistema mais recentes.

Version (Versão)	Alterações
2.5.0	Novos atributos • Adiciona suporte para dispositivos principais que executam o Windows. • Mude o comportamento da remoção de grupos de coisas. Com essa versão, você pode remover um dispositivo principal de um grupo de coisas para desinstalar os componentes desse grupo de coisas na próxima implantação. Como resultado dessa alteração, a AWS IoT política de um dispositivo principal deve ter a `greengrass:ListThingGroupsForCoreDevice` permissão. Se você usou o instalador do software AWS IoT Greengrass Core para provisionar recursos, a AWS IoT política padrão permite `greengrass:*` , o que inclui essa permissão. Para ter mais informações, consulte Autorização e autenticação do dispositivo para o AWS IoT Greengrass. • Adiciona suporte para configurações de proxy HTTPS. Para ter mais informações, consulte Conectar-se à porta 443 ou por meio de um proxy de rede. • Adiciona o novo parâmetro `windowsUser` de configuração. Você pode usar esse parâmetro para especificar o usuário padrão a ser usado para executar componentes em um dispositivo principal do Windows. Para ter mais informações, consulte Configurar o usuário que executa os componentes. • Adiciona as novas opções de `httpClient` configuração que você pode usar para personalizar os tempos limite de solicitação HTTP para melhorar o desempenho em redes lentas. Para obter mais informações, consulte o parâmetro de configuração HttpClient. Correções de erros e melhorias • Corrige a opção de ciclo de vida do bootstrap para reiniciar o dispositivo principal a partir de um componente. • Adiciona suporte para hífens nas variáveis da receita. • Corrige a autorização de IPC para componentes da função Lambda sob demanda.

Version (Versão)	Alterações
	• Melhora as mensagens de registro e altera os registros não críticos de `INFO` um para `DEBUG` outro, então os registros são mais úteis. • Remove a `iot:DescribeCertificate` permissão da função padrão de troca de tokens que o núcleo do Greengrass cria quando você instala o software AWS IoT Greengrass Core com provisionamento automático. Essa permissão não é usada pelo núcleo Greengrass. • Corrige um problema para que o script de provisionamento automático não exija a `iam:GetPolicy` permissão se `iam:CreatePolicy` estiver disponível para a mesma política. • Pequenas correções e melhorias adicionais.

Version (Versão)	Alterações
2.4.0	Novos atributos • Adiciona suporte aos limites de recursos do sistema. Você pode configurar a quantidade máxima de uso de CPU e RAM que os processos de cada componente podem usar no dispositivo principal. Para ter mais informações, consulte Configurar limites de recursos do sistema para componentes. • Adiciona operações de IPC para pausar e retomar componentes. Para obter mais informações, consulte PauseComponent e ResumeComponent. • Adiciona suporte para plug-ins de provisionamento. Você pode especificar um arquivo JAR a ser executado durante a instalação para provisionar AWS os recursos necessários para um dispositivo principal do Greengrass. O núcleo do Greengrass inclui uma interface que você pode implementar para desenvolver plug-ins de provisionamento personalizados. Para ter mais informações, consulte Instale o software AWS IoT Greengrass principal com provisionamento personalizado de recursos. • Adiciona o `thing-name-policy` argumento opcional ao instalador do software AWS IoT Greengrass Core. Você pode usar essa opção para especificar uma AWS IoT política existente ou personalizada ao instalar o software AWS IoT Greengrass Core com provisionamento automático de recursos. Correções de erros e melhorias • Atualiza a configuração de registro na inicialização. Isso corrige um problema em que a configuração de registro não foi aplicada na inicialização. • Atualiza o link simbólico do carregador de núcleo para apontar para o armazenamento de componentes na pasta raiz do Greengrass durante a instalação. Essa atualização permite que você exclua o arquivo JAR e outros artefatos do núcleo que você baixa ao instalar o software AWS IoT Greengrass Core.

Version (Versão)	Alterações
	• Pequenas correções e melhorias adicionais. Para obter mais informaçõ es, consulte os lançamentos em GitHub.
2.3.0	Novos atributos • Adiciona suporte para documentos de configuração de implantação de até 10 MB, acima de 7 KB (para implantações direcionadas a itens) ou 31 KB (para implantações direcionadas a grupos de itens). Para usar esse recurso, a AWS IoT política de um dispositivo principal deve permitir a `greengrass:GetDeploymentConfiguration` permissão. Se você usou o instalador do software AWS IoT Greengras s Core para provisionar recursos, a AWS IoT política do seu dispositivo principal permite`greengrass:*` , o que inclui essa permissão. Para ter mais informações, consulte Autorização e autenticação do dispositivo para o AWS IoT Greengrass. • Adiciona a variável da `iot:thingName` receita. Você pode usar essa variável de receita para obter o nome do dispositivo AWS IoT principal em uma receita. Para ter mais informações, consulte Variáveis da receita. Correções de erros e melhorias • Pequenas correções e melhorias adicionais. Para obter mais informaçõ es, consulte os lançamentos em GitHub.
2.2.0	Novos atributos • Adiciona operações de IPC para gerenciamento local de sombras. Correções de erros e melhorias • Reduz o tamanho do arquivo JAR. • Reduz o uso da memória. • Corrige problemas em que a configuração do log não foi atualizada em certos casos. • Pequenas correções e melhorias adicionais. Para obter mais informaçõ es, consulte os lançamentos em GitHub.

Version (Versão)	Alterações

2.1.0	Novos atributos
	• Suporta o download de imagens do Docker de repositórios privados no Amazon ECR.
	• Adiciona os seguintes parâmetros para personalizar a configuração do MQTT nos dispositivos principais:
	  • `maxInFlightPublishes` — O número máximo de mensagens de QoS 1 não confirmadas do MQTT que podem estar em andamento ao mesmo tempo.
	  • `maxPublishRetry` — O número máximo de vezes para repetir uma mensagem que não foi publicada.
	• Adiciona o parâmetro de `fleetstatusservice` configuração para configurar o intervalo no qual o dispositivo principal publica o status do dispositivo no Nuvem AWS.
	• Pequenas correções e melhorias adicionais. Para obter mais informaçõ es, consulte os lançamentos em GitHub.
	Correções de erros e melhorias
	• Corrige um problema que fazia com que as implantações de sombra fossem duplicadas quando o núcleo era reiniciado.
	• Corrige um problema que fazia com que o núcleo falhasse ao encontrar uma exceção de carga de serviço.
	• Melhora a resolução de dependências de componentes para falhar em uma implantação que inclui uma dependência circular.
	• Corrige um problema que impedia que um componente de plug-in fosse reimplantado se esse componente tivesse sido removido anteriormente do dispositivo principal.
	• Corrija um problema que fazia com que a variável de `HOME` ambiente fosse definida no */greengrass/v2* `/work` diretório dos componentes do Lambda ou dos componentes executados como raiz. Agora, a `HOME` variável está definida corretamente no diretório inicial do usuário que executa o componente.

Version (Versão)	Alterações
	• Pequenas correções e melhorias adicionais. Para obter mais informaçõ es, consulte os lançamentos em GitHub.
2.0.5	Correções de erros e melhorias • Roteia corretamente o tráfego por meio de um proxy de rede configura do ao baixar os componentes AWS fornecidos. • Use o endpoint correto do plano de dados Greengrass nas AWS regiões da China.
2.0.4	Novos atributos • Habilita o tráfego HTTPS pela porta 443. Você pode usar o novo parâmetro de `greengrassDataPlanePort` configuração da versão 2.0.4 do componente nucleus para configurar a comunicação HTTPS para viajar pela porta 443 em vez da porta padrão 8443. Para ter mais informações, consulte Configurar HTTPS pela porta 443. • Adiciona a variável de receita do caminho de trabalho. Você pode usar essa variável de receita para obter o caminho para as pastas de trabalho dos componentes, que você pode usar para compartilhar arquivos entre componentes e suas dependências. Para obter mais informações, consulte a variável de receita do caminho de trabalho. Correções de erros e melhorias • Impede a criação da política de função de troca de tokens AWS Identity and Access Management (IAM) se uma política de função já existir. Como resultado dessa alteração, o instalador agora exige o `iam:GetPolicy` e `sts:GetCallerIdentity` quando executado com `--provision true` . Para ter mais informações, consulte Política mínima de IAM para o instalador provisionar recursos. • Lida corretamente com o cancelamento de uma implantação que ainda não foi registrada com sucesso. • Atualiza a configuração para remover entradas mais antigas com carimbos de data/hora mais recentes ao reverter uma implantação. • Pequenas correções e melhorias adicionais. Para obter mais informaçõ es, consulte os lançamentos em GitHub.

Version (Versão)	Alterações
2.0.3	Versão inicial.

Autenticação do dispositivo cliente

O componente de autenticação do dispositivo cliente (`aws.greengrass.clientdevices.Auth`) autentica os dispositivos do cliente e autoriza as ações do dispositivo cliente.

> **Note**
>
> Os dispositivos cliente são dispositivos IoT locais que se conectam a um dispositivo principal do Greengrass para enviar mensagens e dados MQTT para processamento. Para ter mais informações, consulte Interaja com dispositivos IoT locais.

Tópicos

Versões

> **Note**
>
> A versão 2.3.0 da autenticação do dispositivo cliente foi descontinuada. É altamente recomendável que você atualize para a autenticação do dispositivo cliente versão 2.3.1 ou posterior.

Esse componente tem as seguintes versões:

- 2.4.x
- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

Tipo

Este componente é um componente de plug-in (`aws.greengrass.plugin`). O núcleo do Greengrass executa esse componente na mesma Java Virtual Machine (JVM) do núcleo. O núcleo reinicia quando você altera a versão desse componente no dispositivo principal.

Esse componente usa o mesmo arquivo de log do núcleo do Greengrass. Para ter mais informações, consulte Monitore AWS IoT Greengrass os registros.

Para ter mais informações, consulte Tipos de componentes.

Sistema operacional

Esse componente pode ser instalado em dispositivos principais que executam os seguintes sistemas operacionais:

- Linux
- Windows

Requisitos

Esse componente tem os seguintes requisitos:

- A função de serviço do Greengrass deve estar associada à sua Conta da AWS e permitir a `iot:DescribeCertificate` permissão.
- A AWS IoT política do dispositivo principal deve permitir as seguintes permissões:
  - `greengrass:GetConnectivityInfo`, onde os recursos incluem o ARN do dispositivo principal que executa esse componente

- `greengrass:VerifyClientDeviceIoTCertificateAssociation`, onde os recursos incluem o Amazon Resource Name (ARN) de cada dispositivo cliente que se conecta ao dispositivo principal
- `greengrass:VerifyClientDeviceIdentity`
- `greengrass:PutCertificateAuthorities`
- `iot:Publish`, onde os recursos incluem o ARN do seguinte tópico do MQTT:
  - `$aws/things/`*`coreDeviceThingName`*`*-gci/shadow/get`
- `iot:Subscribe`, onde os recursos incluem os ARNs dos seguintes filtros de tópicos do MQTT:
  - `$aws/things/`*`coreDeviceThingName`*`*-gci/shadow/update/delta`
  - `$aws/things/`*`coreDeviceThingName`*`*-gci/shadow/get/accepted`
- `iot:Receive`, onde os recursos incluem os ARNs dos seguintes tópicos do MQTT:
  - `$aws/things/`*`coreDeviceThingName`*`*-gci/shadow/update/delta`
  - `$aws/things/`*`coreDeviceThingName`*`*-gci/shadow/get/accepted`

  Para obter mais informações, consulte Políticas do AWS IoT para operações de plano de dados e AWS IoTPolítica mínima para oferecer suporte aos dispositivos do cliente.
- (Opcional) Para usar a autenticação off-line, a função AWS Identity and Access Management (IAM) usada pelo AWS IoT Greengrass serviço deve conter a seguinte permissão:
  - `greengrass:ListClientDevicesAssociatedWithCoreDevice`para permitir que o dispositivo principal liste clientes para autenticação offline.
- O componente de autenticação do dispositivo cliente tem suporte para execução em uma VPC. Para implantar esse componente em uma VPC, é necessário o seguinte.
  - O componente de autenticação do dispositivo cliente deve ter conectividade com AWS IoT data, AWS IoT Credentials e Amazon S3.

Endpoints e portas

Esse componente deve ser capaz de realizar solicitações de saída para os seguintes endpoints e portas, além dos endpoints e portas necessários para a operação básica. Para ter mais informações, consulte Permitir o tráfego de dispositivos por meio de um proxy ou firewall.

Endpoint	Porta	Obrigatório	Descrição
`iot.region.amazonaws.com`	443	Sim	Usado para obter informações sobre certificados de AWS IoT coisas.

Dependências

Quando você implanta um componente, AWS IoT Greengrass também implanta versões compatíveis de suas dependências. Isso significa que você deve atender aos requisitos do componente e de todas as suas dependências para implantá-lo com êxito. Esta seção lista as dependências das versões lançadas desse componente e as restrições de versão semântica que definem as versões do componente para cada dependência. Você também pode visualizar as dependências de cada versão do componente no AWS IoT Greengrass console. Na página de detalhes do componente, procure a lista de dependências.

2.4.4

A tabela a seguir lista as dependências da versão 2.4.4 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.6.0 <2.13.0	Flexível

2.4.3

A tabela a seguir lista as dependências da versão 2.4.3 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2,6,0 <2,12,0	Flexível

2.4.1 and 2.4.2

A tabela a seguir lista as dependências das versões 2.4.1 e 2.4.2 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.6.0 <2.11.0	Flexível

2.3.0 – 2.4.0

A tabela a seguir lista as dependências das versões 2.3.0 a 2.4.0 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.6.0 <2.10.0	Flexível

2.3.0

A tabela a seguir lista as dependências da versão 2.3.0 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.6.0 <2.10.0	Flexível

2.2.3

A tabela a seguir lista as dependências da versão 2.2.3 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2,6,0 <=2,9,0	Flexível

2.2.2

A tabela a seguir lista as dependências da versão 2.2.2 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2,6,0 <=2,8,0	Flexível

2.2.1

A tabela a seguir lista as dependências da versão 2.2.1 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.6.0 <2.8.0	Flexível

2.2.0

A tabela a seguir lista as dependências da versão 2.2.0 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.6.0 <2.7.0	Flexível

2.1.0

A tabela a seguir lista as dependências da versão 2.1.0 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.2.0 <2.7.0	Flexível

2.0.4

A tabela a seguir lista as dependências da versão 2.0.4 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.2.0 <2.6.0	Flexível

2.0.2 and 2.0.3

A tabela a seguir lista as dependências das versões 2.0.2 e 2.0.3 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.2.0 <2.5.0	Flexível

2.0.1

A tabela a seguir lista as dependências da versão 2.0.1 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.2.0 <2.4.0	Flexível

2.0.0

A tabela a seguir lista as dependências da versão 2.0.0 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.2.0 <2.3.0	Flexível

Para obter mais informações sobre dependências de componentes, consulte a referência da receita do componente.

Configuração

Esse componente fornece os seguintes parâmetros de configuração que você pode personalizar ao implantar o componente.

> **Note**
>
> A permissão de assinatura é avaliada durante uma solicitação de assinatura do cliente para a corretora local da MQTT. Se a permissão de inscrição existente do cliente for revogada, o cliente não poderá mais se inscrever em um tópico. No entanto, ele continuará recebendo

mensagens de todos os tópicos inscritos anteriormente. Para evitar esse comportamento, o agente MQTT local deve ser reiniciado após revogar a permissão de assinatura para forçar a reautorização dos clientes.
Para o componente MQTT 5 broker (EMQX), atualize a `restartIdentifier` configuração para reiniciar o broker MQTT 5. Para obter mais informações, consulte a configuração do componente do broker MQTT 5.
Para o componente do broker MQTT 3.1.1 (Moquette), ele reinicia semanalmente por padrão quando o certificado do servidor é alterado, forçando os clientes a reautorizarem. Você pode forçar uma reinicialização alterando as informações de conectividade (endereços IP) do dispositivo principal ou fazendo uma implantação para remover o componente intermediário e implantá-lo novamente mais tarde.

v2.5.0

`deviceGroups`

Grupos de dispositivos são grupos de dispositivos clientes que têm permissões para se conectar e se comunicar com um dispositivo principal. Use regras de seleção para identificar grupos de dispositivos clientes e definir políticas de autorização de dispositivos clientes que especifiquem as permissões para cada grupo de dispositivos.

Esse objeto contém as seguintes informações:

`formatVersion`

A versão do formato desse objeto de configuração.

Escolha uma das seguintes opções:

- `2021-03-05`

`definitions`

Os grupos de dispositivos desse dispositivo principal. Cada definição especifica uma regra de seleção para avaliar se um dispositivo cliente é membro do grupo. Cada definição também especifica a política de permissões a ser aplicada aos dispositivos clientes que correspondem à regra de seleção. Se um dispositivo cliente for membro de vários grupos de dispositivos, as permissões do dispositivo serão compostas pela política de permissões de cada grupo.

Esse objeto contém as seguintes informações:

groupNameKey

O nome desse grupo de dispositivos. *groupNameKey*Substitua por um nome que ajude a identificar esse grupo de dispositivos.

Esse objeto contém as seguintes informações:

`selectionRule`

A consulta que especifica quais dispositivos cliente são membros desse grupo de dispositivos. Quando um dispositivo cliente se conecta, o dispositivo principal avalia essa regra de seleção para determinar se o dispositivo cliente é membro desse grupo de dispositivos. Se o dispositivo cliente for membro, o dispositivo principal usa a política desse grupo de dispositivos para autorizar as ações do dispositivo cliente.

Cada regra de seleção compreende pelo menos uma cláusula de regra de seleção, que é uma consulta de expressão única que pode corresponder aos dispositivos do cliente. As regras de seleção usam a mesma sintaxe de consulta da indexação de AWS IoT frotas. Para obter mais informações sobre a sintaxe da regra de seleção, consulte a sintaxe da consulta de indexação de AWS IoT frotas no Guia do AWS IoT Core desenvolvedor.

Use o `*` curinga para combinar vários dispositivos clientes com uma cláusula de regra de seleção. Você pode usar esse caractere curinga no início e no final do nome da coisa para corresponder aos dispositivos clientes cujos nomes começam ou terminam com a string especificada. Você também pode usar esse caractere curinga para corresponder a todos os dispositivos clientes.

> **Note**
>
> Para selecionar um valor que contenha um caractere de dois pontos (`:`), escape dos dois pontos com um caractere de barra invertida (). `\` Em formatos como JSON, você deve escapar dos caracteres de barra invertida, então você insere dois caracteres de barra invertida antes do caractere de dois pontos. Por exemplo, especifique `thingName:MyTeam\\:ClientDevice1` para selecionar uma coisa cujo nome seja`MyTeam:ClientDevice1`.

Você pode especificar o seguinte seletor:

- `thingName`— O nome da AWS IoT coisa de um dispositivo cliente.

  Example Exemplo de regra de seleção

  A regra de seleção a seguir corresponde aos dispositivos clientes cujos nomes são `MyClientDevice1` ou`MyClientDevice2`.

  ```
  thingName: MyClientDevice1 OR thingName: MyClientDevice2
  ```

  Example Exemplo de regra de seleção (use curingas)

  A regra de seleção a seguir corresponde aos dispositivos clientes cujos nomes começam com`MyClientDevice`.

  ```
  thingName: MyClientDevice*
  ```

  Example Exemplo de regra de seleção (use curingas)

  A regra de seleção a seguir corresponde aos dispositivos clientes cujos nomes terminam com`MyClientDevice`.

  ```
  thingName: *MyClientDevice
  ```

  Example Exemplo de regra de seleção (corresponda a todos os dispositivos)

  A regra de seleção a seguir corresponde a todos os dispositivos clientes.

  ```
  thingName: *
  ```

`policyName`

A política de permissões que se aplica aos dispositivos cliente nesse grupo de dispositivos. Especifique o nome de uma política que você define no `policies` objeto.

`policies`

As políticas de autorização do dispositivo cliente para dispositivos cliente que se conectam ao dispositivo principal. Cada política de autorização especifica um conjunto de ações e os recursos nos quais um dispositivo cliente pode realizar essas ações.

Esse objeto contém as seguintes informações:

policyNameKey

O nome dessa política de autorização. *policyNameKey*Substitua por um nome que ajude a identificar essa política de autorização. Você usa esse nome de política para definir qual política se aplica a um grupo de dispositivos.

Esse objeto contém as seguintes informações:

statementNameKey

O nome dessa declaração de política. *statementNameKey*Substitua por um nome que ajude a identificar essa declaração de política.

Esse objeto contém as seguintes informações:

`operations`

A lista de operações para permitir os recursos desta política.

Você pode incluir qualquer uma das seguintes operações:

- `mqtt:connect`— Concede permissão para se conectar ao dispositivo principal. Os dispositivos cliente devem ter essa permissão para se conectar a um dispositivo principal.

  Essa operação oferece suporte aos seguintes recursos:

  - `mqtt:clientId:`*deviceClientId*— Restrinja o acesso com base na ID do cliente que um dispositivo cliente usa para se conectar ao agente MQTT do dispositivo principal. *deviceClientId*Substitua pela ID do cliente a ser usada.
- `mqtt:publish`— Concede permissão para publicar mensagens MQTT em tópicos.

  Essa operação oferece suporte aos seguintes recursos:

  - `mqtt:topic:`*mqttTopic*— Restrinja o acesso com base no tópico do MQTT em que um dispositivo cliente publica uma mensagem. Substitua *MqttTopic* pelo tópico a ser usado.

    Esse recurso não é compatível com curingas de tópicos do MQTT.
- `mqtt:subscribe`— Concede permissão para assinar os filtros de tópicos do MQTT para receber mensagens.

Essa operação oferece suporte aos seguintes recursos:

- `mqtt:topicfilter:`*`mqttTopicFilter`*— Restrinja o acesso com base nos tópicos do MQTT em que um dispositivo cliente pode assinar mensagens. *`mqttTopicFilter`*Substitua pelo filtro de tópicos a ser usado.

  Esse recurso oferece suporte aos curingas do tópico + e do # MQTT. Para obter mais informações, consulte os tópicos do MQTT no Guia do AWS IoT Core desenvolvedor.

  O dispositivo cliente pode assinar os filtros de tópicos exatos que você permite. Por exemplo, se você permitir que o dispositivo cliente assine o `mqtt:topicfilter:client/+/status` recurso, o dispositivo cliente poderá se inscrever, `client/+/status` mas não`client/client1/status`.

Você pode especificar o * caractere curinga para permitir o acesso a todas as ações.

`resources`

A lista de recursos para permitir as operações desta política. Especifique os recursos que correspondem às operações dessa política. Por exemplo, você pode especificar uma lista de recursos de tópicos do MQTT (`mqtt:topic:`*`mqttTopic`*) em uma política que especifica a `mqtt:publish` operação.

Você pode especificar o * curinga em qualquer lugar dentro da variável de recurso para permitir o acesso a todos os recursos. Por exemplo, você pode especificar **`mqtt:topic:my*`** para permitir o acesso aos recursos que correspondam a essa entrada.

A seguinte variável de recurso é suportada:

- `mqtt:topic:${iot:Connection.Thing.ThingName}`

  Isso se resume ao nome da coisa no AWS IoT Core registro para a qual a política está sendo avaliada. AWS IoT Core usa o certificado que o dispositivo apresenta ao se autenticar para determinar qual coisa usar para verificar a conexão. Essa variável de política só está disponível quando um dispositivo se conecta pelo MQTT ou pelo MQTT pelo WebSocket protocolo.

`statementDescription`

(Opcional) Uma descrição desta declaração de política.

`certificates`

(Opcional) As opções de configuração do certificado para esse dispositivo principal. Esse objeto contém as seguintes informações:

`serverCertificateValiditySeconds`

(Opcional) A quantidade de tempo (em segundos) após a expiração do certificado do servidor MQTT local. Você pode configurar essa opção para personalizar a frequência com que os dispositivos cliente se desconectam e se reconectam ao dispositivo principal.

Esse componente gira o certificado do servidor MQTT local 24 horas antes de expirar. O agente MQTT, como o componente do agente Moquette MQTT, gera um novo certificado e reinicia. Quando isso acontece, todos os dispositivos clientes conectados a esse dispositivo principal são desconectados. Os dispositivos cliente podem se reconectar ao dispositivo principal após um curto período de tempo.

Padrão: `604800` (7 dias)

Valor mínimo: `172800` (2 dias)

Valor máximo: `864000` (10 dias)

`performance`

(Opcional) As opções de configuração de desempenho desse dispositivo principal. Esse objeto contém as seguintes informações:

`maxActiveAuthTokens`

(Opcional) O número máximo de tokens ativos de autorização do dispositivo cliente. Você pode aumentar esse número para permitir que um número maior de dispositivos clientes se conecte a um único dispositivo principal, sem precisar reautenticá-los.

Padrão: `2500`

`cloudRequestQueueSize`

(Opcional) O número máximo de Nuvem AWS solicitações a serem colocadas na fila antes que esse componente rejeite as solicitações.

Padrão: `100`

`maxConcurrentCloudRequests`

(Opcional) O número máximo de solicitações simultâneas a serem enviadas para o. Nuvem AWS Você pode aumentar esse número para melhorar o desempenho da autenticação em dispositivos principais nos quais você conecta um grande número de dispositivos clientes.

Padrão: 1

`certificateAuthority`

(Opcional) Opções de configuração da autoridade de certificação para substituir a autoridade intermediária do dispositivo principal pela sua própria autoridade de certificação intermediária.

> **Note**
>
> Se você configurar seu dispositivo principal do Greengrass com uma autoridade de certificação (CA) personalizada e usar a mesma CA para emitir certificados de dispositivo cliente, o Greengrass ignorará as verificações da política de autorização para operações de MQTT do dispositivo cliente. O componente de autenticação do dispositivo cliente confia totalmente nos clientes usando certificados assinados pela CA que ele está configurado para usar.
> Para restringir esse comportamento ao usar uma CA personalizada, crie e assine dispositivos cliente usando outra CA ou CA intermediária e, em seguida, ajuste os `certificateChainUri` campos `certificateUri` e para apontar para a CA intermediária correta.

Esse objeto contém as seguintes informações.

URI do certificado

A localização do certificado. Pode ser um URI do sistema de arquivos ou um URI que aponta para um certificado armazenado em um módulo de segurança de hardware.

`certificateChainUri`

A localização da cadeia de certificados da CA do dispositivo principal. Essa deve ser a cadeia completa de certificados de volta à sua CA raiz. Pode ser um URI do sistema de

arquivos ou um URI que aponta para uma cadeia de certificados armazenada em um módulo de segurança de hardware.

`privateKeyUri`

A localização da chave privada do dispositivo principal. Isso pode ser um URI do sistema de arquivos ou um URI que aponta para uma chave privada de certificado armazenada em um módulo de segurança de hardware.

`security`

(Opcional) Opções de configuração de segurança para esse dispositivo principal. Esse objeto contém as seguintes informações.

`clientDeviceTrustDurationMinutes`

A duração em minutos em que as informações de autenticação de um dispositivo cliente podem ser confiáveis antes que seja necessário reautenticar com o dispositivo principal. O valor padrão é 1.

`metrics`

(Opcional) As opções de métricas para esse dispositivo principal. As métricas de erro só serão exibidas se houver um erro com a autenticação do dispositivo cliente. Esse objeto contém as seguintes informações:

`disableMetrics`

Se o `disableMetrics` campo for definido como`true`, a autenticação do dispositivo cliente não coletará métricas.

Padrão: `false`

`aggregatePeriodSeconds`

O período de agregação em segundos que determina com que frequência a autenticação do dispositivo cliente agrega métricas e as envia ao agente de telemetria. Isso não altera a frequência com que as métricas são publicadas porque o agente de telemetria ainda as publica uma vez por dia.

Padrão: `3600`

startupTimeoutSeconds

(Opcional) O tempo máximo em segundos para o componente iniciar. O estado do componente muda para `BROKEN` se ele exceder esse tempo limite.

Padrão: `120`

Example Exemplo: atualização de mesclagem de configurações (usando uma política restritiva)

O exemplo de configuração a seguir especifica para permitir que dispositivos clientes cujos nomes comecem com `MyClientDevice` se conectem e publiquem ou se inscrevam em todos os tópicos.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
        "policyName": "MyRestrictivePolicy"
      }
    },
    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:test/topic"
          ]
        },
```

```
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "mqtt:topicfilter:test/topic/response"
          ]
        }
      }
    }
  }
}
```

Example Exemplo: atualização de mesclagem de configurações (usando uma política permissiva)

O exemplo de configuração a seguir especifica para permitir que todos os dispositivos cliente se conectem e publiquem ou se inscrevam em todos os tópicos.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
        "selectionRule": "thingName: *",
        "policyName": "MyPermissivePolicy"
      }
    },
    "policies": {
      "MyPermissivePolicy": {
        "AllowAll": {
          "statementDescription": "Allow client devices to perform all actions.",
          "operations": [
            "*"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
```

```
}
```

Example Exemplo: atualização de mesclagem de configuração (usando uma política de nome de coisa)

O exemplo de configuração a seguir permite que dispositivos cliente publiquem tópicos que começam com o nome do item do dispositivo cliente e terminam com a string`topic`.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "myThing": {
        "selectionRule": "thingName: *",
        "policyName": "MyThingNamePolicy"
      }
    },
    "policies": {
      "MyThingNamePolicy": {
        "policyStatement": {
          "statementDescription": "mqtt publish",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:${iot:Connection.Thing.ThingName}/*/topic"
          ]
        }
      }
    }
  }
}
```

v2.4.5

`deviceGroups`

Grupos de dispositivos são grupos de dispositivos clientes que têm permissões para se conectar e se comunicar com um dispositivo principal. Use regras de seleção para identificar grupos de dispositivos clientes e definir políticas de autorização de dispositivos clientes que especifiquem as permissões para cada grupo de dispositivos.

Esse objeto contém as seguintes informações:

`formatVersion`

A versão do formato desse objeto de configuração.

Escolha uma das seguintes opções:

- `2021-03-05`

`definitions`

Os grupos de dispositivos desse dispositivo principal. Cada definição especifica uma regra de seleção para avaliar se um dispositivo cliente é membro do grupo. Cada definição também especifica a política de permissões a ser aplicada aos dispositivos clientes que correspondem à regra de seleção. Se um dispositivo cliente for membro de vários grupos de dispositivos, as permissões do dispositivo serão compostas pela política de permissões de cada grupo.

Esse objeto contém as seguintes informações:

`groupNameKey`

O nome desse grupo de dispositivos. *`groupNameKey`*Substitua por um nome que ajude a identificar esse grupo de dispositivos.

Esse objeto contém as seguintes informações:

`selectionRule`

A consulta que especifica quais dispositivos cliente são membros desse grupo de dispositivos. Quando um dispositivo cliente se conecta, o dispositivo principal avalia essa regra de seleção para determinar se o dispositivo cliente é membro desse grupo de dispositivos. Se o dispositivo cliente for membro, o dispositivo principal usa a política desse grupo de dispositivos para autorizar as ações do dispositivo cliente.

Cada regra de seleção compreende pelo menos uma cláusula de regra de seleção, que é uma consulta de expressão única que pode corresponder aos dispositivos do cliente. As regras de seleção usam a mesma sintaxe de consulta da indexação de AWS IoT frotas. Para obter mais informações sobre a sintaxe da regra de seleção, consulte a sintaxe da consulta de indexação de AWS IoT frotas no Guia do AWS IoT Core desenvolvedor.

Use o * curinga para combinar vários dispositivos clientes com uma cláusula de regra de seleção. Você pode usar esse caractere curinga no início e no final do nome da coisa para corresponder aos dispositivos clientes cujos nomes começam ou terminam com a string especificada. Você também pode usar esse caractere curinga para corresponder a todos os dispositivos clientes.

> **Note**
>
> Para selecionar um valor que contenha um caractere de dois pontos (`:`), escape dos dois pontos com um caractere de barra invertida (). `\` Em formatos como JSON, você deve escapar dos caracteres de barra invertida, então você insere dois caracteres de barra invertida antes do caractere de dois pontos. Por exemplo, especifique `thingName:MyTeam\\:ClientDevice1` para selecionar uma coisa cujo nome seja`MyTeam:ClientDevice1`.

Você pode especificar o seguinte seletor:

- `thingName`— O nome da AWS IoT coisa de um dispositivo cliente.

Example Exemplo de regra de seleção

A regra de seleção a seguir corresponde aos dispositivos clientes cujos nomes são `MyClientDevice1` ou`MyClientDevice2`.

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example Exemplo de regra de seleção (use curingas)

A regra de seleção a seguir corresponde aos dispositivos clientes cujos nomes começam com`MyClientDevice`.

```
thingName: MyClientDevice*
```

Example Exemplo de regra de seleção (use curingas)

A regra de seleção a seguir corresponde aos dispositivos clientes cujos nomes terminam com`MyClientDevice`.

```
thingName: *MyClientDevice
```

Example Exemplo de regra de seleção (corresponda a todos os dispositivos)

A regra de seleção a seguir corresponde a todos os dispositivos clientes.

```
thingName: *
```

`policyName`

A política de permissões que se aplica aos dispositivos cliente nesse grupo de dispositivos. Especifique o nome de uma política que você define no `policies` objeto.

`policies`

As políticas de autorização do dispositivo cliente para dispositivos cliente que se conectam ao dispositivo principal. Cada política de autorização especifica um conjunto de ações e os recursos nos quais um dispositivo cliente pode realizar essas ações.

Esse objeto contém as seguintes informações:

`policyNameKey`

O nome dessa política de autorização. *`policyNameKey`*Substitua por um nome que ajude a identificar essa política de autorização. Você usa esse nome de política para definir qual política se aplica a um grupo de dispositivos.

Esse objeto contém as seguintes informações:

`statementNameKey`

O nome dessa declaração de política. *`statementNameKey`*Substitua por um nome que ajude a identificar essa declaração de política.

Esse objeto contém as seguintes informações:

`operations`

A lista de operações para permitir os recursos desta política.

Você pode incluir qualquer uma das seguintes operações:

- `mqtt:connect`— Concede permissão para se conectar ao dispositivo principal. Os dispositivos cliente devem ter essa permissão para se conectar a um dispositivo principal.

  Essa operação oferece suporte aos seguintes recursos:

  - `mqtt:clientId:`*`deviceClientId`*— Restrinja o acesso com base na ID do cliente que um dispositivo cliente usa para se conectar ao agente MQTT do dispositivo principal. *`deviceClientId`*Substitua pela ID do cliente a ser usada.

- `mqtt:publish`— Concede permissão para publicar mensagens MQTT em tópicos.

  Essa operação oferece suporte aos seguintes recursos:

  - `mqtt:topic:`*`mqttTopic`*— Restrinja o acesso com base no tópico do MQTT em que um dispositivo cliente publica uma mensagem. Substitua *`MqttTopic`* pelo tópico a ser usado.

    Esse recurso não é compatível com curingas de tópicos do MQTT.

- `mqtt:subscribe`— Concede permissão para assinar os filtros de tópicos do MQTT para receber mensagens.

  Essa operação oferece suporte aos seguintes recursos:

  - `mqtt:topicfilter:`*`mqttTopicFilter`*— Restrinja o acesso com base nos tópicos do MQTT em que um dispositivo cliente pode assinar mensagens. *`mqttTopicFilter`*Substitua pelo filtro de tópicos a ser usado.

    Esse recurso oferece suporte aos curingas do tópico + e do # MQTT. Para obter mais informações, consulte os tópicos do MQTT no Guia do AWS IoT Core desenvolvedor.

    O dispositivo cliente pode assinar os filtros de tópicos exatos que você permite. Por exemplo, se você permitir que o dispositivo cliente assine o `mqtt:topicfilter:client/+/status` recurso, o dispositivo cliente poderá se inscrever, `client/+/status` mas não`client/client1/status`.

Você pode especificar o * caractere curinga para permitir o acesso a todas as ações.

`resources`

A lista de recursos para permitir as operações desta política. Especifique os recursos que correspondem às operações dessa política. Por exemplo, você pode especificar uma lista de recursos de tópicos do MQTT (`mqtt:topic:`*`mqttTopic`*) em uma política que especifica a `mqtt:publish` operação.

Você pode especificar o * caractere curinga para permitir o acesso a todos os recursos. Você não pode usar o * caractere curinga para corresponder a identificadores parciais de recursos. Por exemplo, você pode especificar**`"resources": "*"`**, mas não pode especificar**`"resources": "mqtt:clientId:*"`**.

`statementDescription`

(Opcional) Uma descrição desta declaração de política.

`certificates`

(Opcional) As opções de configuração do certificado para esse dispositivo principal. Esse objeto contém as seguintes informações:

`serverCertificateValiditySeconds`

(Opcional) A quantidade de tempo (em segundos) após a expiração do certificado do servidor MQTT local. Você pode configurar essa opção para personalizar a frequência com que os dispositivos cliente se desconectam e se reconectam ao dispositivo principal.

Esse componente gira o certificado do servidor MQTT local 24 horas antes de expirar. O agente MQTT, como o componente do agente Moquette MQTT, gera um novo certificado e reinicia. Quando isso acontece, todos os dispositivos clientes conectados a esse dispositivo principal são desconectados. Os dispositivos cliente podem se reconectar ao dispositivo principal após um curto período de tempo.

Padrão: `604800` (7 dias)

Valor mínimo: `172800` (2 dias)

Valor máximo: `864000` (10 dias)

`performance`

(Opcional) As opções de configuração de desempenho desse dispositivo principal. Esse objeto contém as seguintes informações:

`maxActiveAuthTokens`

(Opcional) O número máximo de tokens ativos de autorização do dispositivo cliente. Você pode aumentar esse número para permitir que um número maior de dispositivos clientes se conecte a um único dispositivo principal, sem precisar reautenticá-los.

Padrão: `2500`

`cloudRequestQueueSize`

(Opcional) O número máximo de Nuvem AWS solicitações a serem colocadas na fila antes que esse componente rejeite as solicitações.

Padrão: `100`

`maxConcurrentCloudRequests`

(Opcional) O número máximo de solicitações simultâneas a serem enviadas para o. Nuvem AWS Você pode aumentar esse número para melhorar o desempenho da autenticação em dispositivos principais nos quais você conecta um grande número de dispositivos clientes.

Padrão: 1

`certificateAuthority`

(Opcional) Opções de configuração da autoridade de certificação para substituir a autoridade intermediária do dispositivo principal pela sua própria autoridade de certificação intermediária.

> **Note**
>
> Se você configurar seu dispositivo principal do Greengrass com uma autoridade de certificação (CA) personalizada e usar a mesma CA para emitir certificados de dispositivo cliente, o Greengrass ignorará as verificações da política de autorização para operações de MQTT do dispositivo cliente. O componente de autenticação do dispositivo cliente confia totalmente nos clientes usando certificados assinados pela CA que ele está configurado para usar.

Para restringir esse comportamento ao usar uma CA personalizada, crie e assine dispositivos cliente usando outra CA ou CA intermediária e, em seguida, ajuste os `certificateChainUri` campos `certificateUri` e para apontar para a CA intermediária correta.

Esse objeto contém as seguintes informações.

URI do certificado

A localização do certificado. Pode ser um URI do sistema de arquivos ou um URI que aponta para um certificado armazenado em um módulo de segurança de hardware.

`certificateChainUri`

A localização da cadeia de certificados da CA do dispositivo principal. Essa deve ser a cadeia completa de certificados de volta à sua CA raiz. Pode ser um URI do sistema de arquivos ou um URI que aponta para uma cadeia de certificados armazenada em um módulo de segurança de hardware.

`privateKeyUri`

A localização da chave privada do dispositivo principal. Isso pode ser um URI do sistema de arquivos ou um URI que aponta para uma chave privada de certificado armazenada em um módulo de segurança de hardware.

`security`

(Opcional) Opções de configuração de segurança para esse dispositivo principal. Esse objeto contém as seguintes informações.

`clientDeviceTrustDurationMinutes`

A duração em minutos em que as informações de autenticação de um dispositivo cliente podem ser confiáveis antes que seja necessário reautenticar com o dispositivo principal. O valor padrão é 1.

`metrics`

(Opcional) As opções de métricas para esse dispositivo principal. As métricas de erro só serão exibidas se houver um erro com a autenticação do dispositivo cliente. Esse objeto contém as seguintes informações:

`disableMetrics`

Se o `disableMetrics` campo for definido como`true`, a autenticação do dispositivo cliente não coletará métricas.

Padrão: `false`

`aggregatePeriodSeconds`

O período de agregação em segundos que determina com que frequência a autenticação do dispositivo cliente agrega métricas e as envia ao agente de telemetria. Isso não altera a frequência com que as métricas são publicadas porque o agente de telemetria ainda as publica uma vez por dia.

Padrão: `3600`

startupTimeoutSeconds

(Opcional) O tempo máximo em segundos para o componente iniciar. O estado do componente muda para `BROKEN` se ele exceder esse tempo limite.

Padrão: `120`

Example Exemplo: atualização de mesclagem de configurações (usando uma política restritiva)

O exemplo de configuração a seguir especifica para permitir que dispositivos clientes cujos nomes comecem com `MyClientDevice` se conectem e publiquem ou se inscrevam em todos os tópicos.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
        "policyName": "MyRestrictivePolicy"
      }
    },
    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
```

```
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:test/topic"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "mqtt:topicfilter:test/topic/response"
          ]
        }
      }
    }
  }
}
```

Example Exemplo: atualização de mesclagem de configurações (usando uma política permissiva)

O exemplo de configuração a seguir especifica para permitir que todos os dispositivos cliente se conectem e publiquem ou se inscrevam em todos os tópicos.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
        "selectionRule": "thingName: *",
        "policyName": "MyPermissivePolicy"
```

```
      }
    },
    "policies": {
      "MyPermissivePolicy": {
        "AllowAll": {
          "statementDescription": "Allow client devices to perform all actions.",
          "operations": [
            "*"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
}
```

v2.4.2 - v2.4.4

`deviceGroups`

Grupos de dispositivos são grupos de dispositivos clientes que têm permissões para se conectar e se comunicar com um dispositivo principal. Use regras de seleção para identificar grupos de dispositivos clientes e definir políticas de autorização de dispositivos clientes que especifiquem as permissões para cada grupo de dispositivos.

Esse objeto contém as seguintes informações:

`formatVersion`

A versão do formato desse objeto de configuração.

Escolha uma das seguintes opções:

- `2021-03-05`

`definitions`

Os grupos de dispositivos desse dispositivo principal. Cada definição especifica uma regra de seleção para avaliar se um dispositivo cliente é membro do grupo. Cada definição também especifica a política de permissões a ser aplicada aos dispositivos clientes que correspondem à regra de seleção. Se um dispositivo cliente for membro de vários grupos

de dispositivos, as permissões do dispositivo serão compostas pela política de permissões de cada grupo.

Esse objeto contém as seguintes informações:

groupNameKey

O nome desse grupo de dispositivos. *groupNameKey*Substitua por um nome que ajude a identificar esse grupo de dispositivos.

Esse objeto contém as seguintes informações:

`selectionRule`

A consulta que especifica quais dispositivos cliente são membros desse grupo de dispositivos. Quando um dispositivo cliente se conecta, o dispositivo principal avalia essa regra de seleção para determinar se o dispositivo cliente é membro desse grupo de dispositivos. Se o dispositivo cliente for membro, o dispositivo principal usa a política desse grupo de dispositivos para autorizar as ações do dispositivo cliente.

Cada regra de seleção compreende pelo menos uma cláusula de regra de seleção, que é uma consulta de expressão única que pode corresponder aos dispositivos do cliente. As regras de seleção usam a mesma sintaxe de consulta da indexação de AWS IoT frotas. Para obter mais informações sobre a sintaxe da regra de seleção, consulte a sintaxe da consulta de indexação de AWS IoT frotas no Guia do AWS IoT Core desenvolvedor.

Use o `*` curinga para combinar vários dispositivos clientes com uma cláusula de regra de seleção. Você pode usar esse caractere curinga no final do nome da coisa para corresponder aos dispositivos clientes cujos nomes começam com uma string especificada por você. Você também pode usar esse caractere curinga para corresponder a todos os dispositivos clientes.

> **Note**
>
> Para selecionar um valor que contenha um caractere de dois pontos (`:`), escape dos dois pontos com um caractere de barra invertida (). `\\` Em formatos como JSON, você deve escapar dos caracteres de barra invertida, então você insere dois caracteres de barra invertida antes do caractere de dois pontos. Por exemplo, especifique `thingName:`

`MyTeam\\\\:ClientDevice1` para selecionar uma coisa cujo nome seja`MyTeam:ClientDevice1`.

Você pode especificar o seguinte seletor:

- `thingName`— O nome da AWS IoT coisa de um dispositivo cliente.

Example Exemplo de regra de seleção

A regra de seleção a seguir corresponde aos dispositivos clientes cujos nomes são `MyClientDevice1` ou`MyClientDevice2`.

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example Exemplo de regra de seleção (use curingas)

A regra de seleção a seguir corresponde aos dispositivos clientes cujos nomes começam com`MyClientDevice`.

```
thingName: MyClientDevice*
```

Example Exemplo de regra de seleção (corresponda a todos os dispositivos)

A regra de seleção a seguir corresponde a todos os dispositivos clientes.

```
thingName: *
```

`policyName`

A política de permissões que se aplica aos dispositivos cliente nesse grupo de dispositivos. Especifique o nome de uma política que você define no `policies` objeto.

`policies`

As políticas de autorização do dispositivo cliente para dispositivos cliente que se conectam ao dispositivo principal. Cada política de autorização especifica um conjunto de ações e os recursos nos quais um dispositivo cliente pode realizar essas ações.

Esse objeto contém as seguintes informações:

policyNameKey

O nome dessa política de autorização. *policyNameKey*Substitua por um nome que ajude a identificar essa política de autorização. Você usa esse nome de política para definir qual política se aplica a um grupo de dispositivos.

Esse objeto contém as seguintes informações:

statementNameKey

O nome dessa declaração de política. *statementNameKey*Substitua por um nome que ajude a identificar essa declaração de política.

Esse objeto contém as seguintes informações:

`operations`

A lista de operações para permitir os recursos desta política.

Você pode incluir qualquer uma das seguintes operações:

- `mqtt:connect`— Concede permissão para se conectar ao dispositivo principal. Os dispositivos cliente devem ter essa permissão para se conectar a um dispositivo principal.

  Essa operação oferece suporte aos seguintes recursos:

  - `mqtt:clientId:`*deviceClientId*— Restrinja o acesso com base na ID do cliente que um dispositivo cliente usa para se conectar ao agente MQTT do dispositivo principal. *deviceClientId*Substitua pela ID do cliente a ser usada.
- `mqtt:publish`— Concede permissão para publicar mensagens MQTT em tópicos.

  Essa operação oferece suporte aos seguintes recursos:

  - `mqtt:topic:`*mqttTopic*— Restrinja o acesso com base no tópico do MQTT em que um dispositivo cliente publica uma mensagem. Substitua *MqttTopic* pelo tópico a ser usado.

    Esse recurso não é compatível com curingas de tópicos do MQTT.
- `mqtt:subscribe`— Concede permissão para assinar os filtros de tópicos do MQTT para receber mensagens.

Essa operação oferece suporte aos seguintes recursos:

- `mqtt:topicfilter:`*`mqttTopicFilter`*— Restrinja o acesso com base nos tópicos do MQTT em que um dispositivo cliente pode assinar mensagens. *`mqttTopicFilter`*Substitua pelo filtro de tópicos a ser usado.

  Esse recurso oferece suporte aos curingas do tópico + e do # MQTT. Para obter mais informações, consulte os tópicos do MQTT no Guia do AWS IoT Core desenvolvedor.

  O dispositivo cliente pode assinar os filtros de tópicos exatos que você permite. Por exemplo, se você permitir que o dispositivo cliente assine o `mqtt:topicfilter:client/+/status` recurso, o dispositivo cliente poderá se inscrever, `client/+/status` mas não`client/client1/status`.

Você pode especificar o * caractere curinga para permitir o acesso a todas as ações.

`resources`

A lista de recursos para permitir as operações desta política. Especifique os recursos que correspondem às operações dessa política. Por exemplo, você pode especificar uma lista de recursos de tópicos do MQTT (`mqtt:topic:`*`mqttTopic`*) em uma política que especifica a `mqtt:publish` operação.

Você pode especificar o * caractere curinga para permitir o acesso a todos os recursos. Você não pode usar o * caractere curinga para corresponder a identificadores parciais de recursos. Por exemplo, você pode especificar**`"resources": "*"`**, mas não pode especificar**`"resources": "mqtt:clientId:*"`**.

`statementDescription`

(Opcional) Uma descrição desta declaração de política.

`certificates`

(Opcional) As opções de configuração do certificado para esse dispositivo principal. Esse objeto contém as seguintes informações:

`serverCertificateValiditySeconds`

(Opcional) A quantidade de tempo (em segundos) após a expiração do certificado do servidor MQTT local. Você pode configurar essa opção para personalizar a frequência com que os dispositivos cliente se desconectam e se reconectam ao dispositivo principal.

Esse componente gira o certificado do servidor MQTT local 24 horas antes de expirar. O agente MQTT, como o componente do agente Moquette MQTT, gera um novo certificado e reinicia. Quando isso acontece, todos os dispositivos clientes conectados a esse dispositivo principal são desconectados. Os dispositivos cliente podem se reconectar ao dispositivo principal após um curto período de tempo.

Padrão: `604800` (7 dias)

Valor mínimo: `172800` (2 dias)

Valor máximo: `864000` (10 dias)

`performance`

(Opcional) As opções de configuração de desempenho desse dispositivo principal. Esse objeto contém as seguintes informações:

`maxActiveAuthTokens`

(Opcional) O número máximo de tokens ativos de autorização do dispositivo cliente. Você pode aumentar esse número para permitir que um número maior de dispositivos clientes se conecte a um único dispositivo principal, sem precisar reautenticá-los.

Padrão: `2500`

`cloudRequestQueueSize`

(Opcional) O número máximo de Nuvem AWS solicitações a serem colocadas na fila antes que esse componente rejeite as solicitações.

Padrão: `100`

`maxConcurrentCloudRequests`

(Opcional) O número máximo de solicitações simultâneas a serem enviadas para o. Nuvem AWS Você pode aumentar esse número para melhorar o desempenho da autenticação em dispositivos principais nos quais você conecta um grande número de dispositivos clientes.

Padrão: 1

`certificateAuthority`

(Opcional) Opções de configuração da autoridade de certificação para substituir a autoridade intermediária do dispositivo principal pela sua própria autoridade de certificação intermediária.

> **Note**
>
> Se você configurar seu dispositivo principal do Greengrass com uma autoridade de certificação (CA) personalizada e usar a mesma CA para emitir certificados de dispositivo cliente, o Greengrass ignorará as verificações da política de autorização para operações de MQTT do dispositivo cliente. O componente de autenticação do dispositivo cliente confia totalmente nos clientes usando certificados assinados pela CA que ele está configurado para usar.
> Para restringir esse comportamento ao usar uma CA personalizada, crie e assine dispositivos cliente usando outra CA ou CA intermediária e, em seguida, ajuste os `certificateChainUri` campos `certificateUri` e para apontar para a CA intermediária correta.

Esse objeto contém as seguintes informações.

URI do certificado

A localização do certificado. Pode ser um URI do sistema de arquivos ou um URI que aponta para um certificado armazenado em um módulo de segurança de hardware.

`certificateChainUri`

A localização da cadeia de certificados da CA do dispositivo principal. Essa deve ser a cadeia completa de certificados de volta à sua CA raiz. Pode ser um URI do sistema de arquivos ou um URI que aponta para uma cadeia de certificados armazenada em um módulo de segurança de hardware.

`privateKeyUri`

A localização da chave privada do dispositivo principal. Isso pode ser um URI do sistema de arquivos ou um URI que aponta para uma chave privada de certificado armazenada em um módulo de segurança de hardware.

`security`

(Opcional) Opções de configuração de segurança para esse dispositivo principal. Esse objeto contém as seguintes informações.

`clientDeviceTrustDurationMinutes`

A duração em minutos em que as informações de autenticação de um dispositivo cliente podem ser confiáveis antes que seja necessário reautenticar com o dispositivo principal. O valor padrão é 1.

`metrics`

(Opcional) As opções de métricas para esse dispositivo principal. As métricas de erro só serão exibidas se houver um erro com a autenticação do dispositivo cliente. Esse objeto contém as seguintes informações:

`disableMetrics`

Se o `disableMetrics` campo for definido como`true`, a autenticação do dispositivo cliente não coletará métricas.

Padrão: `false`

`aggregatePeriodSeconds`

O período de agregação em segundos que determina com que frequência a autenticação do dispositivo cliente agrega métricas e as envia ao agente de telemetria. Isso não altera a frequência com que as métricas são publicadas porque o agente de telemetria ainda as publica uma vez por dia.

Padrão: `3600`

startupTimeoutSeconds

(Opcional) O tempo máximo em segundos para o componente iniciar. O estado do componente muda para `BROKEN` se ele exceder esse tempo limite.

Padrão: `120`

Example Exemplo: atualização de mesclagem de configurações (usando uma política restritiva)

O exemplo de configuração a seguir especifica para permitir que dispositivos clientes cujos nomes comecem com `MyClientDevice` se conectem e publiquem ou se inscrevam em todos os tópicos.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
        "policyName": "MyRestrictivePolicy"
      }
    },
    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:test/topic"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "mqtt:topicfilter:test/topic/response"
```

```
          ]
        }
      }
    }
  }
}
```

Example Exemplo: atualização de mesclagem de configurações (usando uma política permissiva)

O exemplo de configuração a seguir especifica para permitir que todos os dispositivos cliente se conectem e publiquem ou se inscrevam em todos os tópicos.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
        "selectionRule": "thingName: *",
        "policyName": "MyPermissivePolicy"
      }
    },
    "policies": {
      "MyPermissivePolicy": {
        "AllowAll": {
          "statementDescription": "Allow client devices to perform all actions.",
          "operations": [
            "*"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
}
```

v2.4.0 - v2.4.1

`deviceGroups`

Grupos de dispositivos são grupos de dispositivos clientes que têm permissões para se conectar e se comunicar com um dispositivo principal. Use regras de seleção para identificar

grupos de dispositivos clientes e definir políticas de autorização de dispositivos clientes que especifiquem as permissões para cada grupo de dispositivos.

Esse objeto contém as seguintes informações:

`formatVersion`

A versão do formato desse objeto de configuração.

Escolha uma das seguintes opções:

- `2021-03-05`

`definitions`

Os grupos de dispositivos desse dispositivo principal. Cada definição especifica uma regra de seleção para avaliar se um dispositivo cliente é membro do grupo. Cada definição também especifica a política de permissões a ser aplicada aos dispositivos clientes que correspondem à regra de seleção. Se um dispositivo cliente for membro de vários grupos de dispositivos, as permissões do dispositivo serão compostas pela política de permissões de cada grupo.

Esse objeto contém as seguintes informações:

`groupNameKey`

O nome desse grupo de dispositivos. *`groupNameKey`*Substitua por um nome que ajude a identificar esse grupo de dispositivos.

Esse objeto contém as seguintes informações:

`selectionRule`

A consulta que especifica quais dispositivos cliente são membros desse grupo de dispositivos. Quando um dispositivo cliente se conecta, o dispositivo principal avalia essa regra de seleção para determinar se o dispositivo cliente é membro desse grupo de dispositivos. Se o dispositivo cliente for membro, o dispositivo principal usa a política desse grupo de dispositivos para autorizar as ações do dispositivo cliente.

Cada regra de seleção compreende pelo menos uma cláusula de regra de seleção, que é uma consulta de expressão única que pode corresponder aos dispositivos do cliente. As regras de seleção usam a mesma sintaxe de consulta da indexação de AWS IoT frotas. Para obter mais informações sobre a sintaxe da regra de seleção,

consulte a sintaxe da consulta de indexação de AWS IoT frotas no Guia do AWS IoT Core desenvolvedor.

Use o * caractere curinga para combinar vários dispositivos clientes com uma cláusula de regra de seleção. Você pode usar esse caractere curinga no final do nome da coisa para corresponder aos dispositivos clientes cujos nomes começam com uma string especificada por você. Você também pode usar esse caractere curinga para corresponder a todos os dispositivos clientes.

> **Note**
>
> Para selecionar um valor que contenha um caractere de dois pontos (`:`), escape dos dois pontos com um caractere de barra invertida (). `\\` Em formatos como JSON, você deve escapar dos caracteres de barra invertida, então você insere dois caracteres de barra invertida antes do caractere de dois pontos. Por exemplo, especifique `thingName:MyTeam\\\\:ClientDevice1` para selecionar uma coisa cujo nome seja`MyTeam:ClientDevice1`.

Você pode especificar o seguinte seletor:

- `thingName`— O nome da AWS IoT coisa de um dispositivo cliente.

Example Exemplo de regra de seleção

A regra de seleção a seguir corresponde aos dispositivos clientes cujos nomes são `MyClientDevice1` ou`MyClientDevice2`.

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example Exemplo de regra de seleção (use curingas)

A regra de seleção a seguir corresponde aos dispositivos clientes cujos nomes começam com`MyClientDevice`.

```
thingName: MyClientDevice*
```

Example Exemplo de regra de seleção (corresponda a todos os dispositivos)

A regra de seleção a seguir corresponde a todos os dispositivos clientes.

```
thingName: *
```

`policyName`

A política de permissões que se aplica aos dispositivos cliente nesse grupo de dispositivos. Especifique o nome de uma política que você define no `policies` objeto.

`policies`

As políticas de autorização do dispositivo cliente para dispositivos cliente que se conectam ao dispositivo principal. Cada política de autorização especifica um conjunto de ações e os recursos nos quais um dispositivo cliente pode realizar essas ações.

Esse objeto contém as seguintes informações:

`policyNameKey`

O nome dessa política de autorização. *`policyNameKey`*Substitua por um nome que ajude a identificar essa política de autorização. Você usa esse nome de política para definir qual política se aplica a um grupo de dispositivos.

Esse objeto contém as seguintes informações:

`statementNameKey`

O nome dessa declaração de política. *`statementNameKey`*Substitua por um nome que ajude a identificar essa declaração de política.

Esse objeto contém as seguintes informações:

`operations`

A lista de operações para permitir os recursos desta política.

Você pode incluir qualquer uma das seguintes operações:

- `mqtt:connect`— Concede permissão para se conectar ao dispositivo principal. Os dispositivos cliente devem ter essa permissão para se conectar a um dispositivo principal.

  Essa operação oferece suporte aos seguintes recursos:

  - `mqtt:clientId:`*`deviceClientId`*— Restrinja o acesso com base na ID do cliente que um dispositivo cliente usa para se conectar ao agente MQTT

do dispositivo principal. *deviceClientId*Substitua pela ID do cliente a ser usada.

- `mqtt:publish`— Concede permissão para publicar mensagens MQTT em tópicos.

  Essa operação oferece suporte aos seguintes recursos:

  - `mqtt:topic:`*mqttTopic*— Restrinja o acesso com base no tópico do MQTT em que um dispositivo cliente publica uma mensagem. Substitua *MqttTopic* pelo tópico a ser usado.

    Esse recurso não é compatível com curingas de tópicos do MQTT.

- `mqtt:subscribe`— Concede permissão para assinar os filtros de tópicos do MQTT para receber mensagens.

  Essa operação oferece suporte aos seguintes recursos:

  - `mqtt:topicfilter:`*mqttTopicFilter*— Restrinja o acesso com base nos tópicos do MQTT em que um dispositivo cliente pode assinar mensagens. *mqttTopicFilter*Substitua pelo filtro de tópicos a ser usado.

    Esse recurso oferece suporte aos curingas do tópico + e do # MQTT. Para obter mais informações, consulte os tópicos do MQTT no Guia do AWS IoT Core desenvolvedor.

    O dispositivo cliente pode assinar os filtros de tópicos exatos que você permite. Por exemplo, se você permitir que o dispositivo cliente assine o `mqtt:topicfilter:client/+/status` recurso, o dispositivo cliente poderá se inscrever, `client/+/status` mas não`client/client1/status`.

  Você pode especificar o * caractere curinga para permitir o acesso a todas as ações.

`resources`

A lista de recursos para permitir as operações desta política. Especifique os recursos que correspondem às operações dessa política. Por exemplo, você pode especificar uma lista de recursos de tópicos do MQTT (`mqtt:topic:`*mqttTopic*) em uma política que especifica a `mqtt:publish` operação.

Você pode especificar o * caractere curinga para permitir o acesso a todos os recursos. Você não pode usar o * caractere curinga para corresponder a identificadores parciais de recursos. Por exemplo, você pode especificar**`"resources": "*"`**, mas não pode especificar**`"resources": "mqtt:clientId:*"`**.

`statementDescription`

(Opcional) Uma descrição desta declaração de política.

`certificates`

(Opcional) As opções de configuração do certificado para esse dispositivo principal. Esse objeto contém as seguintes informações:

`serverCertificateValiditySeconds`

(Opcional) A quantidade de tempo (em segundos) após a expiração do certificado do servidor MQTT local. Você pode configurar essa opção para personalizar a frequência com que os dispositivos cliente se desconectam e se reconectam ao dispositivo principal.

Esse componente gira o certificado do servidor MQTT local 24 horas antes de expirar. O agente MQTT, como o componente do agente Moquette MQTT, gera um novo certificado e reinicia. Quando isso acontece, todos os dispositivos clientes conectados a esse dispositivo principal são desconectados. Os dispositivos cliente podem se reconectar ao dispositivo principal após um curto período de tempo.

Padrão: `604800` (7 dias)

Valor mínimo: `172800` (2 dias)

Valor máximo: `864000` (10 dias)

`performance`

(Opcional) As opções de configuração de desempenho desse dispositivo principal. Esse objeto contém as seguintes informações:

`maxActiveAuthTokens`

(Opcional) O número máximo de tokens ativos de autorização do dispositivo cliente. Você pode aumentar esse número para permitir que um número maior de dispositivos clientes se conecte a um único dispositivo principal, sem precisar reautenticá-los.

Padrão: `2500`

`cloudRequestQueueSize`

(Opcional) O número máximo de Nuvem AWS solicitações a serem colocadas na fila antes que esse componente rejeite as solicitações.

Padrão: `100`

`maxConcurrentCloudRequests`

(Opcional) O número máximo de solicitações simultâneas a serem enviadas para o. Nuvem AWS Você pode aumentar esse número para melhorar o desempenho da autenticação em dispositivos principais nos quais você conecta um grande número de dispositivos clientes.

Padrão: 1

`certificateAuthority`

(Opcional) Opções de configuração da autoridade de certificação para substituir a autoridade intermediária do dispositivo principal pela sua própria autoridade de certificação intermediária. Esse objeto contém as seguintes informações.

Esse objeto contém as seguintes informações:

URI do certificado

A localização do certificado. Pode ser um URI do sistema de arquivos ou um URI que aponta para um certificado armazenado em um módulo de segurança de hardware.

`certificateChainUri`

A localização da cadeia de certificados da CA do dispositivo principal. Essa deve ser a cadeia completa de certificados de volta à sua CA raiz. Pode ser um URI do sistema de arquivos ou um URI que aponta para uma cadeia de certificados armazenada em um módulo de segurança de hardware.

`privateKeyUri`

A localização da chave privada do dispositivo principal. Isso pode ser um URI do sistema de arquivos ou um URI que aponta para uma chave privada de certificado armazenada em um módulo de segurança de hardware.

`security`

(Opcional) Opções de configuração de segurança para esse dispositivo principal. Esse objeto contém as seguintes informações.

`clientDeviceTrustDurationMinutes`

A duração em minutos em que as informações de autenticação de um dispositivo cliente podem ser confiáveis antes que seja necessário reautenticar com o dispositivo principal. O valor padrão é 1.

`metrics`

(Opcional) As opções de métricas para esse dispositivo principal. As métricas de erro só serão exibidas se houver um erro com a autenticação do dispositivo cliente. Esse objeto contém as seguintes informações:

`disableMetrics`

Se o `disableMetrics` campo for definido como`true`, a autenticação do dispositivo cliente não coletará métricas.

Padrão: `false`

`aggregatePeriodSeconds`

O período de agregação em segundos que determina com que frequência a autenticação do dispositivo cliente agrega métricas e as envia ao agente de telemetria. Isso não altera a frequência com que as métricas são publicadas porque o agente de telemetria ainda as publica uma vez por dia.

Padrão: `3600`

Example Exemplo: atualização de mesclagem de configurações (usando uma política restritiva)

O exemplo de configuração a seguir especifica para permitir que dispositivos clientes cujos nomes comecem com `MyClientDevice` se conectem e publiquem ou se inscrevam em todos os tópicos.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
```

```
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
        "policyName": "MyRestrictivePolicy"
      }
    },
    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:test/topic"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "mqtt:topicfilter:test/topic/response"
          ]
        }
      }
    }
  }
}
```

Example Exemplo: atualização de mesclagem de configurações (usando uma política permissiva)

O exemplo de configuração a seguir especifica para permitir que todos os dispositivos cliente se conectem e publiquem ou se inscrevam em todos os tópicos.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
        "selectionRule": "thingName: *",
        "policyName": "MyPermissivePolicy"
      }
    },
    "policies": {
      "MyPermissivePolicy": {
        "AllowAll": {
          "statementDescription": "Allow client devices to perform all actions.",
          "operations": [
            "*"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
}
```

v2.3.x

`deviceGroups`

Grupos de dispositivos são grupos de dispositivos clientes que têm permissões para se conectar e se comunicar com um dispositivo principal. Use regras de seleção para identificar grupos de dispositivos clientes e definir políticas de autorização de dispositivos clientes que especifiquem as permissões para cada grupo de dispositivos.

Esse objeto contém as seguintes informações:

`formatVersion`

A versão do formato desse objeto de configuração.

Escolha uma das seguintes opções:

- `2021-03-05`

`definitions`

Os grupos de dispositivos desse dispositivo principal. Cada definição especifica uma regra de seleção para avaliar se um dispositivo cliente é membro do grupo. Cada definição também especifica a política de permissões a ser aplicada aos dispositivos clientes que correspondem à regra de seleção. Se um dispositivo cliente for membro de vários grupos de dispositivos, as permissões do dispositivo serão compostas pela política de permissões de cada grupo.

Esse objeto contém as seguintes informações:

`groupNameKey`

O nome desse grupo de dispositivos. *`groupNameKey`*Substitua por um nome que ajude a identificar esse grupo de dispositivos.

Esse objeto contém as seguintes informações:

`selectionRule`

A consulta que especifica quais dispositivos cliente são membros desse grupo de dispositivos. Quando um dispositivo cliente se conecta, o dispositivo principal avalia essa regra de seleção para determinar se o dispositivo cliente é membro desse grupo de dispositivos. Se o dispositivo cliente for membro, o dispositivo principal usa a política desse grupo de dispositivos para autorizar as ações do dispositivo cliente.

Cada regra de seleção compreende pelo menos uma cláusula de regra de seleção, que é uma consulta de expressão única que pode corresponder aos dispositivos do cliente. As regras de seleção usam a mesma sintaxe de consulta da indexação de AWS IoT frotas. Para obter mais informações sobre a sintaxe da regra de seleção, consulte a sintaxe da consulta de indexação de AWS IoT frotas no Guia do AWS IoT Core desenvolvedor.

Use o * caractere curinga para combinar vários dispositivos clientes com uma cláusula de regra de seleção. Você pode usar esse caractere curinga no final do nome da coisa para corresponder aos dispositivos clientes cujos nomes começam com uma string especificada por você. Você também pode usar esse caractere curinga para corresponder a todos os dispositivos clientes.

> **Note**
>
> Para selecionar um valor que contenha um caractere de dois pontos (`:`), escape dos dois pontos com um caractere de barra invertida (). `\\` Em formatos como JSON, você deve escapar dos caracteres de barra invertida, então você insere dois caracteres de barra invertida antes do caractere de dois pontos. Por exemplo, especifique `thingName:MyTeam\\\\:ClientDevice1` para selecionar uma coisa cujo nome seja`MyTeam:ClientDevice1`.

Você pode especificar o seguinte seletor:

- `thingName`— O nome da AWS IoT coisa de um dispositivo cliente.

Example Exemplo de regra de seleção

A regra de seleção a seguir corresponde aos dispositivos clientes cujos nomes são `MyClientDevice1` ou`MyClientDevice2`.

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example Exemplo de regra de seleção (use curingas)

A regra de seleção a seguir corresponde aos dispositivos clientes cujos nomes começam com`MyClientDevice`.

```
thingName: MyClientDevice*
```

Example Exemplo de regra de seleção (corresponda a todos os dispositivos)

A regra de seleção a seguir corresponde a todos os dispositivos clientes.

```
thingName: *
```

`policyName`

A política de permissões que se aplica aos dispositivos cliente nesse grupo de dispositivos. Especifique o nome de uma política que você define no `policies` objeto.

`policies`

As políticas de autorização do dispositivo cliente para dispositivos cliente que se conectam ao dispositivo principal. Cada política de autorização especifica um conjunto de ações e os recursos nos quais um dispositivo cliente pode realizar essas ações.

Esse objeto contém as seguintes informações:

`policyNameKey`

O nome dessa política de autorização. *`policyNameKey`*Substitua por um nome que ajude a identificar essa política de autorização. Você usa esse nome de política para definir qual política se aplica a um grupo de dispositivos.

Esse objeto contém as seguintes informações:

`statementNameKey`

O nome dessa declaração de política. *`statementNameKey`*Substitua por um nome que ajude a identificar essa declaração de política.

Esse objeto contém as seguintes informações:

`operations`

A lista de operações para permitir os recursos desta política.

Você pode incluir qualquer uma das seguintes operações:

- `mqtt:connect`— Concede permissão para se conectar ao dispositivo principal. Os dispositivos cliente devem ter essa permissão para se conectar a um dispositivo principal.

  Essa operação oferece suporte aos seguintes recursos:

  - `mqtt:clientId:`*`deviceClientId`*— Restrinja o acesso com base na ID do cliente que um dispositivo cliente usa para se conectar ao agente MQTT do dispositivo principal. *`deviceClientId`*Substitua pela ID do cliente a ser usada.
- `mqtt:publish`— Concede permissão para publicar mensagens MQTT em tópicos.

  Essa operação oferece suporte aos seguintes recursos:

- `mqtt:topic:`*`mqttTopic`*— Restrinja o acesso com base no tópico do MQTT em que um dispositivo cliente publica uma mensagem. Substitua *`MqttTopic`* pelo tópico a ser usado.

  Esse recurso não é compatível com curingas de tópicos do MQTT.

- `mqtt:subscribe`— Concede permissão para assinar os filtros de tópicos do MQTT para receber mensagens.

  Essa operação oferece suporte aos seguintes recursos:

  - `mqtt:topicfilter:`*`mqttTopicFilter`*— Restrinja o acesso com base nos tópicos do MQTT em que um dispositivo cliente pode assinar mensagens. *`mqttTopicFilter`*Substitua pelo filtro de tópicos a ser usado.

    Esse recurso oferece suporte aos curingas do tópico + e do # MQTT. Para obter mais informações, consulte os tópicos do MQTT no Guia do AWS IoT Core desenvolvedor.

    O dispositivo cliente pode assinar os filtros de tópicos exatos que você permite. Por exemplo, se você permitir que o dispositivo cliente assine o `mqtt:topicfilter:client/+/status` recurso, o dispositivo cliente poderá se inscrever, `client/+/status` mas não`client/client1/status`.

  Você pode especificar o * caractere curinga para permitir o acesso a todas as ações.

`resources`

A lista de recursos para permitir as operações desta política. Especifique os recursos que correspondem às operações dessa política. Por exemplo, você pode especificar uma lista de recursos de tópicos do MQTT (`mqtt:topic:`*`mqttTopic`*) em uma política que especifica a `mqtt:publish` operação.

Você pode especificar o * caractere curinga para permitir o acesso a todos os recursos. Você não pode usar o * caractere curinga para corresponder a identificadores parciais de recursos. Por exemplo, você pode especificar**`"resources": "*"`**, mas não pode especificar**`"resources": "mqtt:clientId:*"`**.

`statementDescription`

(Opcional) Uma descrição desta declaração de política.

`certificates`

(Opcional) As opções de configuração do certificado para esse dispositivo principal. Esse objeto contém as seguintes informações:

`serverCertificateValiditySeconds`

(Opcional) A quantidade de tempo (em segundos) após a expiração do certificado do servidor MQTT local. Você pode configurar essa opção para personalizar a frequência com que os dispositivos cliente se desconectam e se reconectam ao dispositivo principal.

Esse componente gira o certificado do servidor MQTT local 24 horas antes de expirar. O agente MQTT, como o componente do agente Moquette MQTT, gera um novo certificado e reinicia. Quando isso acontece, todos os dispositivos clientes conectados a esse dispositivo principal são desconectados. Os dispositivos cliente podem se reconectar ao dispositivo principal após um curto período de tempo.

Padrão: `604800` (7 dias)

Valor mínimo: `172800` (2 dias)

Valor máximo: `864000` (10 dias)

`performance`

(Opcional) As opções de configuração de desempenho desse dispositivo principal. Esse objeto contém as seguintes informações:

`maxActiveAuthTokens`

(Opcional) O número máximo de tokens ativos de autorização do dispositivo cliente. Você pode aumentar esse número para permitir que um número maior de dispositivos clientes se conecte a um único dispositivo principal sem precisar reautenticá-los.

Padrão: `2500`

`cloudRequestQueueSize`

(Opcional) O número máximo de Nuvem AWS solicitações a serem colocadas na fila antes que esse componente rejeite as solicitações.

Padrão: `100`

`maxConcurrentCloudRequests`

(Opcional) O número máximo de solicitações simultâneas a serem enviadas para o. Nuvem AWS Você pode aumentar esse número para melhorar o desempenho da autenticação em dispositivos principais nos quais você conecta um grande número de dispositivos clientes.

Padrão: 1

`certificateAuthority`

(Opcional) Opções de configuração da autoridade de certificação para substituir a autoridade intermediária do dispositivo principal pela sua própria autoridade de certificação intermediária. Esse objeto contém as seguintes informações.

URI do certificado

A localização do certificado. Pode ser um URI do sistema de arquivos ou um URI que aponta para um certificado armazenado em um módulo de segurança de hardware.

`certificateChainUri`

A localização da cadeia de certificados da CA do dispositivo principal. Essa deve ser a cadeia completa de certificados de volta à sua CA raiz. Pode ser um URI do sistema de arquivos ou um URI que aponta para uma cadeia de certificados armazenada em um módulo de segurança de hardware.

`privateKeyUri`

A localização da chave privada do dispositivo principal. Isso pode ser um URI do sistema de arquivos ou um URI que aponta para uma chave privada de certificado armazenada em um módulo de segurança de hardware.

`security`

(Opcional) Opções de configuração de segurança para esse dispositivo principal. Esse objeto contém as seguintes informações.

`clientDeviceTrustDurationMinutes`

A duração em minutos em que as informações de autenticação de um dispositivo cliente podem ser confiáveis antes que seja necessário se autenticar novamente com o dispositivo principal. O valor padrão é 1.

Example Exemplo: atualização de mesclagem de configurações (usando uma política restritiva)

O exemplo de configuração a seguir especifica para permitir que dispositivos clientes cujos nomes comecem com `MyClientDevice` se conectem e publiquem ou se inscrevam em todos os tópicos.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
        "policyName": "MyRestrictivePolicy"
      }
    },
    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:test/topic"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "mqtt:topicfilter:test/topic/response"
```

```
          ]
        }
      }
    }
  }
}
```

Example Exemplo: atualização de mesclagem de configurações (usando uma política permissiva)

O exemplo de configuração a seguir especifica para permitir que todos os dispositivos cliente se conectem e publiquem ou se inscrevam em todos os tópicos.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
        "selectionRule": "thingName: *",
        "policyName": "MyPermissivePolicy"
      }
    },
    "policies": {
      "MyPermissivePolicy": {
        "AllowAll": {
          "statementDescription": "Allow client devices to perform all actions.",
          "operations": [
            "*"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
}
```

v2.2.x

`deviceGroups`

Grupos de dispositivos são grupos de dispositivos clientes que têm permissões para se conectar e se comunicar com um dispositivo principal. Use regras de seleção para identificar

grupos de dispositivos clientes e definir políticas de autorização de dispositivos clientes que especifiquem as permissões para cada grupo de dispositivos.

Esse objeto contém as seguintes informações:

`formatVersion`

A versão do formato desse objeto de configuração.

Escolha uma das seguintes opções:

- `2021-03-05`

`definitions`

Os grupos de dispositivos desse dispositivo principal. Cada definição especifica uma regra de seleção para avaliar se um dispositivo cliente é membro do grupo. Cada definição também especifica a política de permissões a ser aplicada aos dispositivos clientes que correspondem à regra de seleção. Se um dispositivo cliente for membro de vários grupos de dispositivos, as permissões do dispositivo serão compostas pela política de permissões de cada grupo.

Esse objeto contém as seguintes informações:

`groupNameKey`

O nome desse grupo de dispositivos. *`groupNameKey`*Substitua por um nome que ajude a identificar esse grupo de dispositivos.

Esse objeto contém as seguintes informações:

`selectionRule`

A consulta que especifica quais dispositivos cliente são membros desse grupo de dispositivos. Quando um dispositivo cliente se conecta, o dispositivo principal avalia essa regra de seleção para determinar se o dispositivo cliente é membro desse grupo de dispositivos. Se o dispositivo cliente for membro, o dispositivo principal usa a política desse grupo de dispositivos para autorizar as ações do dispositivo cliente.

Cada regra de seleção compreende pelo menos uma cláusula de regra de seleção, que é uma consulta de expressão única que pode corresponder aos dispositivos do cliente. As regras de seleção usam a mesma sintaxe de consulta da indexação de AWS IoT frotas. Para obter mais informações sobre a sintaxe da regra de seleção,

consulte a sintaxe da consulta de indexação de AWS IoT frotas no Guia do AWS IoT Core desenvolvedor.

Use o * caractere curinga para combinar vários dispositivos clientes com uma cláusula de regra de seleção. Você pode usar esse caractere curinga no final do nome da coisa para corresponder aos dispositivos clientes cujos nomes começam com uma string especificada por você. Você também pode usar esse caractere curinga para corresponder a todos os dispositivos clientes.

> **Note**
>
> Para selecionar um valor que contenha um caractere de dois pontos (`:`), escape dos dois pontos com um caractere de barra invertida (). `\\` Em formatos como JSON, você deve escapar dos caracteres de barra invertida, então você insere dois caracteres de barra invertida antes do caractere de dois pontos. Por exemplo, especifique `thingName:MyTeam\\\\:ClientDevice1` para selecionar uma coisa cujo nome seja`MyTeam:ClientDevice1`.

Você pode especificar o seguinte seletor:

- `thingName`— O nome da AWS IoT coisa de um dispositivo cliente.

Example Exemplo de regra de seleção

A regra de seleção a seguir corresponde aos dispositivos clientes cujos nomes são `MyClientDevice1` ou`MyClientDevice2`.

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example Exemplo de regra de seleção (use curingas)

A regra de seleção a seguir corresponde aos dispositivos clientes cujos nomes começam com`MyClientDevice`.

```
thingName: MyClientDevice*
```

Example Exemplo de regra de seleção (corresponda a todos os dispositivos)

A regra de seleção a seguir corresponde a todos os dispositivos clientes.

```
thingName: *
```

`policyName`

A política de permissões que se aplica aos dispositivos cliente nesse grupo de dispositivos. Especifique o nome de uma política que você define no `policies` objeto.

`policies`

As políticas de autorização do dispositivo cliente para dispositivos cliente que se conectam ao dispositivo principal. Cada política de autorização especifica um conjunto de ações e os recursos nos quais um dispositivo cliente pode realizar essas ações.

Esse objeto contém as seguintes informações:

`policyNameKey`

O nome dessa política de autorização. *`policyNameKey`*Substitua por um nome que ajude a identificar essa política de autorização. Você usa esse nome de política para definir qual política se aplica a um grupo de dispositivos.

Esse objeto contém as seguintes informações:

`statementNameKey`

O nome dessa declaração de política. *`statementNameKey`*Substitua por um nome que ajude a identificar essa declaração de política.

Esse objeto contém as seguintes informações:

`operations`

A lista de operações para permitir os recursos desta política.

Você pode incluir qualquer uma das seguintes operações:

- `mqtt:connect`— Concede permissão para se conectar ao dispositivo principal. Os dispositivos cliente devem ter essa permissão para se conectar a um dispositivo principal.

  Essa operação oferece suporte aos seguintes recursos:

  - `mqtt:clientId:`*`deviceClientId`*— Restrinja o acesso com base na ID do cliente que um dispositivo cliente usa para se conectar ao agente MQTT

do dispositivo principal. *deviceClientId*Substitua pela ID do cliente a ser usada.

- `mqtt:publish`— Concede permissão para publicar mensagens MQTT em tópicos.

  Essa operação oferece suporte aos seguintes recursos:

  - `mqtt:topic:`*mqttTopic*— Restrinja o acesso com base no tópico do MQTT em que um dispositivo cliente publica uma mensagem. Substitua *MqttTopic* pelo tópico a ser usado.

    Esse recurso não é compatível com curingas de tópicos do MQTT.

- `mqtt:subscribe`— Concede permissão para assinar os filtros de tópicos do MQTT para receber mensagens.

  Essa operação oferece suporte aos seguintes recursos:

  - `mqtt:topicfilter:`*mqttTopicFilter*— Restrinja o acesso com base nos tópicos do MQTT em que um dispositivo cliente pode assinar mensagens. *mqttTopicFilter*Substitua pelo filtro de tópicos a ser usado.

    Esse recurso oferece suporte aos curingas do tópico + e do # MQTT. Para obter mais informações, consulte os tópicos do MQTT no Guia do AWS IoT Core desenvolvedor.

    O dispositivo cliente pode assinar os filtros de tópicos exatos que você permite. Por exemplo, se você permitir que o dispositivo cliente assine o `mqtt:topicfilter:client/+/status` recurso, o dispositivo cliente poderá se inscrever, `client/+/status` mas não`client/client1/status`.

  Você pode especificar o * caractere curinga para permitir o acesso a todas as ações.

`resources`

A lista de recursos para permitir as operações desta política. Especifique os recursos que correspondem às operações dessa política. Por exemplo, você pode especificar uma lista de recursos de tópicos do MQTT (`mqtt:topic:`*mqttTopic*) em uma política que especifica a `mqtt:publish` operação.

Você pode especificar o * caractere curinga para permitir o acesso a todos os recursos. Você não pode usar o * caractere curinga para corresponder a identificadores parciais de recursos. Por exemplo, você pode especificar**`"resources": "*"`**, mas não pode especificar**`"resources": "mqtt:clientId:*"`**.

`statementDescription`

(Opcional) Uma descrição desta declaração de política.

`certificates`

(Opcional) As opções de configuração do certificado para esse dispositivo principal. Esse objeto contém as seguintes informações:

`serverCertificateValiditySeconds`

(Opcional) A quantidade de tempo (em segundos) após a expiração do certificado do servidor MQTT local. Você pode configurar essa opção para personalizar a frequência com que os dispositivos cliente se desconectam e se reconectam ao dispositivo principal.

Esse componente gira o certificado do servidor MQTT local 24 horas antes de expirar. O agente MQTT, como o componente do agente Moquette MQTT, gera um novo certificado e reinicia. Quando isso acontece, todos os dispositivos clientes conectados a esse dispositivo principal são desconectados. Os dispositivos cliente podem se reconectar ao dispositivo principal após um curto período de tempo.

Padrão: `604800` (7 dias)

Valor mínimo: `172800` (2 dias)

Valor máximo: `864000` (10 dias)

`performance`

(Opcional) As opções de configuração de desempenho desse dispositivo principal. Esse objeto contém as seguintes informações:

`maxActiveAuthTokens`

(Opcional) O número máximo de tokens ativos de autorização do dispositivo cliente. Você pode aumentar esse número para permitir que um número maior de dispositivos clientes se conecte a um único dispositivo principal sem precisar reautenticá-los.

Padrão: `2500`

`cloudRequestQueueSize`

(Opcional) O número máximo de Nuvem AWS solicitações a serem colocadas na fila antes que esse componente rejeite as solicitações.

Padrão: `100`

`maxConcurrentCloudRequests`

(Opcional) O número máximo de solicitações simultâneas a serem enviadas para o. Nuvem AWS Você pode aumentar esse número para melhorar o desempenho da autenticação em dispositivos principais nos quais você conecta um grande número de dispositivos clientes.

Padrão: 1

Example Exemplo: atualização de mesclagem de configurações (usando uma política restritiva)

O exemplo de configuração a seguir especifica para permitir que dispositivos clientes cujos nomes comecem com `MyClientDevice` se conectem e publiquem ou se inscrevam em todos os tópicos.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
        "policyName": "MyRestrictivePolicy"
      }
    },
    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
```

```
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:test/topic"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "mqtt:topicfilter:test/topic/response"
          ]
        }
      }
    }
  }
}
```

Example Exemplo: atualização de mesclagem de configurações (usando uma política permissiva)

O exemplo de configuração a seguir especifica para permitir que todos os dispositivos cliente se conectem e publiquem ou se inscrevam em todos os tópicos.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
        "selectionRule": "thingName: *",
        "policyName": "MyPermissivePolicy"
      }
    },
    "policies": {
      "MyPermissivePolicy": {
        "AllowAll": {
```

```
          "statementDescription": "Allow client devices to perform all actions.",
          "operations": [
            "*"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
}
```

v2.1.x

`deviceGroups`

Grupos de dispositivos são grupos de dispositivos clientes que têm permissões para se conectar e se comunicar com um dispositivo principal. Use regras de seleção para identificar grupos de dispositivos clientes e definir políticas de autorização de dispositivos clientes que especifiquem as permissões para cada grupo de dispositivos.

Esse objeto contém as seguintes informações:

`formatVersion`

A versão do formato desse objeto de configuração.

Escolha uma das seguintes opções:

- `2021-03-05`

`definitions`

Os grupos de dispositivos desse dispositivo principal. Cada definição especifica uma regra de seleção para avaliar se um dispositivo cliente é membro do grupo. Cada definição também especifica a política de permissões a ser aplicada aos dispositivos clientes que correspondem à regra de seleção. Se um dispositivo cliente for membro de vários grupos de dispositivos, as permissões do dispositivo serão compostas pela política de permissões de cada grupo.

Esse objeto contém as seguintes informações:

groupNameKey

O nome desse grupo de dispositivos. *groupNameKey*Substitua por um nome que ajude a identificar esse grupo de dispositivos.

Esse objeto contém as seguintes informações:

`selectionRule`

A consulta que especifica quais dispositivos cliente são membros desse grupo de dispositivos. Quando um dispositivo cliente se conecta, o dispositivo principal avalia essa regra de seleção para determinar se o dispositivo cliente é membro desse grupo de dispositivos. Se o dispositivo cliente for membro, o dispositivo principal usa a política desse grupo de dispositivos para autorizar as ações do dispositivo cliente.

Cada regra de seleção compreende pelo menos uma cláusula de regra de seleção, que é uma consulta de expressão única que pode corresponder aos dispositivos do cliente. As regras de seleção usam a mesma sintaxe de consulta da indexação de AWS IoT frotas. Para obter mais informações sobre a sintaxe da regra de seleção, consulte a sintaxe da consulta de indexação de AWS IoT frotas no Guia do AWS IoT Core desenvolvedor.

Use o * caractere curinga para combinar vários dispositivos clientes com uma cláusula de regra de seleção. Você pode usar esse caractere curinga no final do nome da coisa para corresponder aos dispositivos clientes cujos nomes começam com uma string especificada por você. Você também pode usar esse caractere curinga para corresponder a todos os dispositivos clientes.

> **Note**
>
> Para selecionar um valor que contenha um caractere de dois pontos (`:`), escape dos dois pontos com um caractere de barra invertida (). `\\` Em formatos como JSON, você deve escapar dos caracteres de barra invertida, então você insere dois caracteres de barra invertida antes do caractere de dois pontos. Por exemplo, especifique `thingName:MyTeam\\\\:ClientDevice1` para selecionar uma coisa cujo nome seja`MyTeam:ClientDevice1`.

Você pode especificar o seguinte seletor:

- `thingName`— O nome da AWS IoT coisa de um dispositivo cliente.

  Example Exemplo de regra de seleção

  A regra de seleção a seguir corresponde aos dispositivos clientes cujos nomes são `MyClientDevice1` ou`MyClientDevice2`.

  ```
  thingName: MyClientDevice1 OR thingName: MyClientDevice2
  ```

  Example Exemplo de regra de seleção (use curingas)

  A regra de seleção a seguir corresponde aos dispositivos clientes cujos nomes começam com`MyClientDevice`.

  ```
  thingName: MyClientDevice*
  ```

  Example Exemplo de regra de seleção (corresponda a todos os dispositivos)

  A regra de seleção a seguir corresponde a todos os dispositivos clientes.

  ```
  thingName: *
  ```

`policyName`

A política de permissões que se aplica aos dispositivos cliente nesse grupo de dispositivos. Especifique o nome de uma política que você define no `policies` objeto.

`policies`

As políticas de autorização do dispositivo cliente para dispositivos cliente que se conectam ao dispositivo principal. Cada política de autorização especifica um conjunto de ações e os recursos nos quais um dispositivo cliente pode realizar essas ações.

Esse objeto contém as seguintes informações:

`policyNameKey`

O nome dessa política de autorização. *`policyNameKey`*Substitua por um nome que ajude a identificar essa política de autorização. Você usa esse nome de política para definir qual política se aplica a um grupo de dispositivos.

Esse objeto contém as seguintes informações:

statementNameKey

O nome dessa declaração de política. *statementNameKey*Substitua por um nome que ajude a identificar essa declaração de política.

Esse objeto contém as seguintes informações:

`operations`

A lista de operações para permitir os recursos desta política.

Você pode incluir qualquer uma das seguintes operações:

- `mqtt:connect`— Concede permissão para se conectar ao dispositivo principal. Os dispositivos cliente devem ter essa permissão para se conectar a um dispositivo principal.

  Essa operação oferece suporte aos seguintes recursos:

  - `mqtt:clientId:`*deviceClientId*— Restrinja o acesso com base na ID do cliente que um dispositivo cliente usa para se conectar ao agente MQTT do dispositivo principal. *deviceClientId*Substitua pela ID do cliente a ser usada.
- `mqtt:publish`— Concede permissão para publicar mensagens MQTT em tópicos.

  Essa operação oferece suporte aos seguintes recursos:

  - `mqtt:topic:`*mqttTopic*— Restrinja o acesso com base no tópico do MQTT em que um dispositivo cliente publica uma mensagem. Substitua *MqttTopic* pelo tópico a ser usado.

    Esse recurso não é compatível com curingas de tópicos do MQTT.
- `mqtt:subscribe`— Concede permissão para assinar os filtros de tópicos do MQTT para receber mensagens.

  Essa operação oferece suporte aos seguintes recursos:

  - `mqtt:topicfilter:`*mqttTopicFilter*— Restrinja o acesso com base nos tópicos do MQTT em que um dispositivo cliente pode assinar mensagens. *mqttTopicFilter*Substitua pelo filtro de tópicos a ser usado.

Esse recurso oferece suporte aos curingas do tópico + e do # MQTT. Para obter mais informações, consulte os tópicos do MQTT no Guia do AWS IoT Core desenvolvedor.

O dispositivo cliente pode assinar os filtros de tópicos exatos que você permite. Por exemplo, se você permitir que o dispositivo cliente assine o `mqtt:topicfilter:client/+/status` recurso, o dispositivo cliente poderá se inscrever, `client/+/status` mas não`client/client1/status`.

Você pode especificar o * caractere curinga para permitir o acesso a todas as ações.

`resources`

A lista de recursos para permitir as operações desta política. Especifique os recursos que correspondem às operações dessa política. Por exemplo, você pode especificar uma lista de recursos de tópicos do MQTT (`mqtt:topic:`*`mqttTopic`*) em uma política que especifica a `mqtt:publish` operação.

Você pode especificar o * caractere curinga para permitir o acesso a todos os recursos. Você não pode usar o * caractere curinga para corresponder a identificadores parciais de recursos. Por exemplo, você pode especificar**`"resources": "*"`**, mas não pode especificar**`"resources": "mqtt:clientId:*"`**.

`statementDescription`

(Opcional) Uma descrição desta declaração de política.

`certificates`

(Opcional) As opções de configuração do certificado para esse dispositivo principal. Esse objeto contém as seguintes informações:

`serverCertificateValiditySeconds`

(Opcional) A quantidade de tempo (em segundos) após a expiração do certificado do servidor MQTT local. Você pode configurar essa opção para personalizar a frequência com que os dispositivos cliente se desconectam e se reconectam ao dispositivo principal.

Esse componente gira o certificado do servidor MQTT local 24 horas antes de expirar. O agente MQTT, como o componente do agente Moquette MQTT, gera um novo certificado e reinicia. Quando isso acontece, todos os dispositivos clientes conectados a esse dispositivo principal são desconectados. Os dispositivos cliente podem se reconectar ao dispositivo principal após um curto período de tempo.

Padrão: `604800` (7 dias)

Valor mínimo: `172800` (2 dias)

Valor máximo: `864000` (10 dias)

Example Exemplo: atualização de mesclagem de configurações (usando uma política restritiva)

O exemplo de configuração a seguir especifica para permitir que dispositivos clientes cujos nomes comecem com `MyClientDevice` se conectem e publiquem ou se inscrevam em todos os tópicos.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
        "policyName": "MyRestrictivePolicy"
      }
    },
    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
```

```
          ],
          "resources": [
            "mqtt:topic:test/topic"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "mqtt:topicfilter:test/topic/response"
          ]
        }
      }
    }
  }
}
```

Example Exemplo: atualização de mesclagem de configurações (usando uma política permissiva)

O exemplo de configuração a seguir especifica para permitir que todos os dispositivos cliente se conectem e publiquem ou se inscrevam em todos os tópicos.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
        "selectionRule": "thingName: *",
        "policyName": "MyPermissivePolicy"
      }
    },
    "policies": {
      "MyPermissivePolicy": {
        "AllowAll": {
          "statementDescription": "Allow client devices to perform all actions.",
          "operations": [
            "*"
          ],
          "resources": [
            "*"
```

```
          ]
        }
      }
    }
  }
}
```

v2.0.x

`deviceGroups`

Grupos de dispositivos são grupos de dispositivos clientes que têm permissões para se conectar e se comunicar com um dispositivo principal. Use regras de seleção para identificar grupos de dispositivos clientes e definir políticas de autorização de dispositivos clientes que especifiquem as permissões para cada grupo de dispositivos.

Esse objeto contém as seguintes informações:

`formatVersion`

A versão do formato desse objeto de configuração.

Escolha uma das seguintes opções:

- `2021-03-05`

`definitions`

Os grupos de dispositivos desse dispositivo principal. Cada definição especifica uma regra de seleção para avaliar se um dispositivo cliente é membro do grupo. Cada definição também especifica a política de permissões a ser aplicada aos dispositivos clientes que correspondem à regra de seleção. Se um dispositivo cliente for membro de vários grupos de dispositivos, as permissões do dispositivo serão compostas pela política de permissões de cada grupo.

Esse objeto contém as seguintes informações:

`groupNameKey`

O nome desse grupo de dispositivos. *`groupNameKey`*Substitua por um nome que ajude a identificar esse grupo de dispositivos.

Esse objeto contém as seguintes informações:

`selectionRule`

A consulta que especifica quais dispositivos cliente são membros desse grupo de dispositivos. Quando um dispositivo cliente se conecta, o dispositivo principal avalia essa regra de seleção para determinar se o dispositivo cliente é membro desse grupo de dispositivos. Se o dispositivo cliente for membro, o dispositivo principal usa a política desse grupo de dispositivos para autorizar as ações do dispositivo cliente.

Cada regra de seleção compreende pelo menos uma cláusula de regra de seleção, que é uma consulta de expressão única que pode corresponder aos dispositivos do cliente. As regras de seleção usam a mesma sintaxe de consulta da indexação de AWS IoT frotas. Para obter mais informações sobre a sintaxe da regra de seleção, consulte a sintaxe da consulta de indexação de AWS IoT frotas no Guia do AWS IoT Core desenvolvedor.

Use o * caractere curinga para combinar vários dispositivos clientes com uma cláusula de regra de seleção. Você pode usar esse caractere curinga no final do nome da coisa para corresponder aos dispositivos clientes cujos nomes começam com uma string especificada por você. Você também pode usar esse caractere curinga para corresponder a todos os dispositivos clientes.

> **Note**
>
> Para selecionar um valor que contenha um caractere de dois pontos (`:`), escape dos dois pontos com um caractere de barra invertida (). `\\` Em formatos como JSON, você deve escapar dos caracteres de barra invertida, então você insere dois caracteres de barra invertida antes do caractere de dois pontos. Por exemplo, especifique `thingName:MyTeam\\\\:ClientDevice1` para selecionar uma coisa cujo nome seja`MyTeam:ClientDevice1`.

Você pode especificar o seguinte seletor:

- `thingName`— O nome da AWS IoT coisa de um dispositivo cliente.

Example Exemplo de regra de seleção

A regra de seleção a seguir corresponde aos dispositivos clientes cujos nomes são `MyClientDevice1` ou`MyClientDevice2`.

```
thingName: MyClientDevice1 OR thingName: MyClientDevice2
```

Example Exemplo de regra de seleção (use curingas)

A regra de seleção a seguir corresponde aos dispositivos clientes cujos nomes começam com`MyClientDevice`.

```
thingName: MyClientDevice*
```

Example Exemplo de regra de seleção (corresponda a todos os dispositivos)

A regra de seleção a seguir corresponde a todos os dispositivos clientes.

```
thingName: *
```

`policyName`

A política de permissões que se aplica aos dispositivos cliente nesse grupo de dispositivos. Especifique o nome de uma política que você define no `policies` objeto.

`policies`

As políticas de autorização do dispositivo cliente para dispositivos cliente que se conectam ao dispositivo principal. Cada política de autorização especifica um conjunto de ações e os recursos nos quais um dispositivo cliente pode realizar essas ações.

Esse objeto contém as seguintes informações:

`policyNameKey`

O nome dessa política de autorização. *`policyNameKey`*Substitua por um nome que ajude a identificar essa política de autorização. Você usa esse nome de política para definir qual política se aplica a um grupo de dispositivos.

Esse objeto contém as seguintes informações:

`statementNameKey`

O nome dessa declaração de política. *`statementNameKey`*Substitua por um nome que ajude a identificar essa declaração de política.

Esse objeto contém as seguintes informações:

`operations`

A lista de operações para permitir os recursos desta política.

Você pode incluir qualquer uma das seguintes operações:

- `mqtt:connect`— Concede permissão para se conectar ao dispositivo principal. Os dispositivos cliente devem ter essa permissão para se conectar a um dispositivo principal.

  Essa operação oferece suporte aos seguintes recursos:

  - `mqtt:clientId:`*`deviceClientId`*— Restrinja o acesso com base na ID do cliente que um dispositivo cliente usa para se conectar ao agente MQTT do dispositivo principal. *`deviceClientId`*Substitua pela ID do cliente a ser usada.
- `mqtt:publish`— Concede permissão para publicar mensagens MQTT em tópicos.

  Essa operação oferece suporte aos seguintes recursos:

  - `mqtt:topic:`*`mqttTopic`*— Restrinja o acesso com base no tópico do MQTT em que um dispositivo cliente publica uma mensagem. Substitua *`MqttTopic`* pelo tópico a ser usado.

    Esse recurso não é compatível com curingas de tópicos do MQTT.
- `mqtt:subscribe`— Concede permissão para assinar os filtros de tópicos do MQTT para receber mensagens.

  Essa operação oferece suporte aos seguintes recursos:

  - `mqtt:topicfilter:`*`mqttTopicFilter`*— Restrinja o acesso com base nos tópicos do MQTT em que um dispositivo cliente pode assinar mensagens. *`mqttTopicFilter`*Substitua pelo filtro de tópicos a ser usado.

    Esse recurso oferece suporte aos curingas do tópico + e do # MQTT. Para obter mais informações, consulte os tópicos do MQTT no Guia do AWS IoT Core desenvolvedor.

    O dispositivo cliente pode assinar os filtros de tópicos exatos que você permite. Por exemplo, se você permitir que o dispositivo cliente assine o `mqtt:topicfilter:client/+/status` recurso, o dispositivo cliente

poderá se inscrever, `client/+/status` mas não`client/client1/status`.

Você pode especificar o `*` caractere curinga para permitir o acesso a todas as ações.

`resources`

A lista de recursos para permitir as operações desta política. Especifique os recursos que correspondem às operações dessa política. Por exemplo, você pode especificar uma lista de recursos de tópicos do MQTT (`mqtt:topic:`*`mqttTopic`*) em uma política que especifica a `mqtt:publish` operação.

Você pode especificar o `*` caractere curinga para permitir o acesso a todos os recursos. Você não pode usar o `*` caractere curinga para corresponder a identificadores parciais de recursos. Por exemplo, você pode especificar**`"resources": "*"`**, mas não pode especificar**`"resources": "mqtt:clientId:*"`**.

`statementDescription`

(Opcional) Uma descrição desta declaração de política.

Example Exemplo: atualização de mesclagem de configurações (usando uma política restritiva)

O exemplo de configuração a seguir especifica para permitir que dispositivos clientes cujos nomes comecem com `MyClientDevice` se conectem e publiquem ou se inscrevam em todos os tópicos.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyDeviceGroup": {
        "selectionRule": "thingName: MyClientDevice*",
        "policyName": "MyRestrictivePolicy"
      }
    },
    "policies": {
      "MyRestrictivePolicy": {
        "AllowConnect": {
```

```
          "statementDescription": "Allow client devices to connect.",
          "operations": [
            "mqtt:connect"
          ],
          "resources": [
            "*"
          ]
        },
        "AllowPublish": {
          "statementDescription": "Allow client devices to publish on test/topic.",
          "operations": [
            "mqtt:publish"
          ],
          "resources": [
            "mqtt:topic:test/topic"
          ]
        },
        "AllowSubscribe": {
          "statementDescription": "Allow client devices to subscribe to test/topic/
response.",
          "operations": [
            "mqtt:subscribe"
          ],
          "resources": [
            "mqtt:topicfilter:test/topic/response"
          ]
        }
      }
    }
  }
}
```

Example Exemplo: atualização de mesclagem de configurações (usando uma política permissiva)

O exemplo de configuração a seguir especifica para permitir que todos os dispositivos cliente se conectem e publiquem ou se inscrevam em todos os tópicos.

```
{
  "deviceGroups": {
    "formatVersion": "2021-03-05",
    "definitions": {
      "MyPermissiveDeviceGroup": {
        "selectionRule": "thingName: *",
```

```
        "policyName": "MyPermissivePolicy"
      }
    },
    "policies": {
      "MyPermissivePolicy": {
        "AllowAll": {
          "statementDescription": "Allow client devices to perform all actions.",
          "operations": [
            "*"
          ],
          "resources": [
            "*"
          ]
        }
      }
    }
  }
}
```

Arquivo de log local

Esse componente usa o mesmo arquivo de log do componente do núcleo do Greengrass.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

Para ver os registros desse componente

- Execute o comando a seguir no dispositivo principal para visualizar o arquivo de log desse componente em tempo real. Substitua */greengrass/v2* ou *C:\greengrass\v2* pelo caminho para a pasta AWS IoT Greengrass raiz.

Linux

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
```

Changelog

A tabela a seguir descreve as alterações em cada versão do componente.

Version (Versão)	Alterações
2.5.0	Novos atributos • Permite a substituição de `${iot:Connection.Thing.ThingName}` variáveis por recursos de política. • Permite recursos de política com curingas, como `mqtt:topic:my*` .
2.4.5	Novos atributos Adiciona suporte para prefixos curinga para selecionar nomes de itens com o `selectionRule` parâmetro. Correções de erros e melhorias Corrige um problema em que os certificados não são atualizados com novas informações de conectividade em certos casos.
2.4.4	Versão atualizada para a versão 2.12.0 do Greengrass nucleus.
2.4.3	Versão atualizada para a versão 2.11.0 do Greengrass nucleus.
2.4.2	Novos atributos Adiciona uma nova opção `startupTimeoutSeconds` de configuração.

Version (Versão)	Alterações
2.4.1	Versão atualizada para a versão 2.10.0 do Greengrass nucleus.
2.4.0	Novos atributos • Adiciona suporte à autenticação do dispositivo cliente para emitir métricas operacionais que serão publicadas pelo agente de telemetria. Correções de erros e melhorias • Corrige um problema em que a autenticação do dispositivo cliente leva mais de 10 segundos para verificar a identidade do dispositivo cliente. • Pequenas correções e melhorias adicionais.
2.3.2	Correções de erros e melhorias • Adiciona suporte para armazenar em cache as informações do nome do host para que o componente gere corretamente os assuntos do certifica do quando reiniciado quando estiver off-line.
2.3.1	Correções de erros e melhorias • Corrige um vazamento de memória.
2.3.0	**Warning** Essa versão não está mais disponível. As melhorias nesta versão estão disponíveis em versões posteriores desse componente. Novos atributos • Adiciona suporte para autenticação offline de dispositivos cliente para que eles possam continuar se conectando ao dispositivo principal quando o dispositivo principal não estiver conectado à Internet. • Adiciona suporte à autoridade de certificação fornecida pelo cliente que o dispositivo principal usa como certificado raiz para gerar certificados de agente MQTT.
2.2.3	Versão atualizada para a versão 2.8.0 do Greengrass nucleus.

Version (Versão)	Alterações
2.2.2	Correções de erros e melhorias • Corrige um problema em que o certificado do servidor MQTT local gira com mais frequência do que o pretendido em determinados cenários.
2.2.1	Versão atualizada para a versão 2.7.0 do Greengrass nucleus.
2.2.0	Novos atributos • Adiciona suporte a componentes personalizados para chamar operações de comunicação entre processos (IPC) para autenticar e autorizar dispositivos clientes. Você pode usar essas operações em um componente personalizado do broker MQTT, por exemplo. Para obter mais informações, consulte IPC: autenticar e autorizar dispositivos cliente. • Adiciona as `threadPoolSize` opções `maxActiveAuthTokens` `cloudQueueSize` ,, e que você pode configurar para ajustar o desempenho desse componente.
2.1.0	Novos atributos • Adiciona a `serverCertificateValiditySeconds` opção que você pode configurar para personalizar quando o certificado do servidor do agente MQTT expirar. Você pode configurar o certificado do servidor para expirar após 2 a 10 dias. Correções de erros e melhorias • Corrige problemas com a forma como esse componente lida com as atualizações de redefinição de configuração. • Corrige um problema em que o certificado do servidor MQTT local gira com mais frequência do que o pretendido em determinados cenários. Para aplicar essa correção, você também deve usar a versão 2.1.0 ou posterior do componente de corretor Moquette MQTT. • Melhora as mensagens que esse componente registra ao alternar certificados. • Versão atualizada para a versão 2.6.0 do Greengrass nucleus.

Version (Versão)	Alterações
2.0.4	Versão atualizada para a versão 2.5.0 do Greengrass nucleus.
2.0.3	Correções de erros e melhorias • As credenciais agora são atualizadas se você alternar a chave privada do dispositivo principal. • Atualizações para tornar as mensagens de registro mais claras.
2.0.2	Versão atualizada para a versão 2.4.0 do Greengrass nucleus.
2.0.1	Versão atualizada para a versão 2.3.0 do Greengrass nucleus.
2.0.0	Versão inicial.

CloudWatch métricas

O componente de CloudWatch métricas da Amazon (`aws.greengrass.Cloudwatch`) publica métricas personalizadas dos principais dispositivos do Greengrass na Amazon. CloudWatch O componente permite que os componentes publiquem CloudWatch métricas, que você pode usar para monitorar e analisar o ambiente do dispositivo principal do Greengrass. Para obter mais informações, consulte Usando CloudWatch métricas da Amazon no Guia CloudWatch do usuário da Amazon.

Para publicar uma CloudWatch métrica com esse componente, publique uma mensagem em um tópico em que esse componente se inscreva. Por padrão, esse componente se inscreve no tópico `cloudwatch/metric/put` local de publicação/assinatura. Você pode especificar outros tópicos, incluindo tópicos do AWS IoT Core MQTT, ao implantar esse componente.

Esse componente agrupa métricas que estão no mesmo namespace e as publica em intervalos regulares. CloudWatch

> **Note**
>
> Esse componente fornece funcionalidade semelhante ao conector de CloudWatch métricas na AWS IoT Greengrass V1. Para obter mais informações, consulte o conector de CloudWatch métricas no Guia do desenvolvedor AWS IoT Greengrass V1.

Tópicos

Versões

Esse componente tem as seguintes versões:

- 3.1.x
- 3.0.x
- 2.1.x
- 2.0.x

Para obter informações sobre mudanças em cada versão do componente, consulte o changelog.

Tipo

v3.x

> Este componente é um componente genérico (`aws.greengrass.generic`). O núcleo do Greengrass executa os scripts do ciclo de vida do componente.

v2.x

Esse componente é um componente Lambda () `aws.greengrass.lambda`. O núcleo do Greengrass executa a função Lambda desse componente usando o componente lançador Lambda.

Para ter mais informações, consulte Tipos de componentes.

Sistema operacional

v3.x

Esse componente pode ser instalado em dispositivos principais que executam os seguintes sistemas operacionais:

- Linux
- Windows

v2.x

Esse componente pode ser instalado somente nos dispositivos principais do Linux.

Requisitos

Esse componente tem os seguintes requisitos:

3.x

- Python versão 3.7 instalado no dispositivo de núcleo e adicionado à variável de ambiente PATH.
- A função de dispositivo do Greengrass deve permitir a `cloudwatch:PutMetricData` ação, conforme mostrado no exemplo de política do IAM a seguir.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "cloudwatch:PutMetricData"
```

```
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

Para obter mais informações, consulte a referência de CloudWatch permissões da Amazon no Guia CloudWatch do usuário da Amazon.

2.x

- Seu dispositivo principal deve atender aos requisitos para executar as funções do Lambda. Se você quiser que o dispositivo principal execute funções Lambda em contêineres, o dispositivo deve atender aos requisitos para fazer isso. Para ter mais informações, consulte Requisitos da função do Lambda.
- Python versão 3.7 instalado no dispositivo de núcleo e adicionado à variável de ambiente PATH.
- A função de dispositivo do Greengrass deve permitir a `cloudwatch:PutMetricData` ação, conforme mostrado no exemplo de política do IAM a seguir.

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

Para obter mais informações, consulte a referência de CloudWatch permissões da Amazon no Guia CloudWatch do usuário da Amazon.

- Para receber dados de saída desse componente, você deve mesclar a seguinte atualização de configuração para o componente antigo do roteador de assinatura

(`aws.greengrass.LegacySubscriptionRouter`) ao implantar esse componente. Essa configuração especifica o tópico em que esse componente publica as respostas.

Legacy subscription router v2.1.x

```
{
  "subscriptions": {
    "aws-greengrass-cloudwatch": {
      "id": "aws-greengrass-cloudwatch",
      "source": "component:aws.greengrass.Cloudwatch",
      "subject": "cloudwatch/metric/put/status",
      "target": "cloud"
    }
  }
}
```

Legacy subscription router v2.0.x

```
{
  "subscriptions": {
    "aws-greengrass-cloudwatch": {
      "id": "aws-greengrass-cloudwatch",
      "source": "arn:aws:lambda:region:aws:function:aws-greengrass-
cloudwatch:version",
      "subject": "cloudwatch/metric/put/status",
      "target": "cloud"
    }
  }
}
```

- Substitua a *região* pela Região da AWS que você usa.
- *Substitua a versão pela versão da função Lambda que esse componente executa.* Para encontrar a versão da função Lambda, você deve visualizar a receita da versão desse componente que você deseja implantar. Abra a página de detalhes desse componente no AWS IoT Greengrass console e procure o par de valores-chave da função Lambda. Esse par de valores-chave contém o nome e a versão da função Lambda.

> **Important**
>
> Você deve atualizar a versão da função Lambda no roteador de assinatura legado sempre que implantar esse componente. Isso garante que você use a versão correta da função Lambda para a versão do componente que você implanta.

Para ter mais informações, consulte Criar implantações.

Endpoints e portas

Esse componente deve ser capaz de realizar solicitações de saída para os seguintes endpoints e portas, além dos endpoints e portas necessários para a operação básica. Para ter mais informações, consulte Permitir o tráfego de dispositivos por meio de um proxy ou firewall.

Endpoint	Porta	Obrigatório	Descrição
`monitoring. region.amazonaws.com`	443	Sim	CloudWatch Métricas de upload.

Dependências

Quando você implanta um componente, AWS IoT Greengrass também implanta versões compatíveis de suas dependências. Isso significa que você deve atender aos requisitos do componente e de todas as suas dependências para implantá-lo com êxito. Esta seção lista as dependências das versões lançadas desse componente e as restrições de versão semântica que definem as versões dos componentes para cada dependência. Você também pode visualizar as dependências de cada versão do componente no AWS IoT Greengrass console. Na página de detalhes do componente, procure a lista de dependências.

3.0.0 - 3.1.0

A tabela a seguir lista as dependências das versões 3.0.0 a 3.1.0 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <3.0.0	Flexível
Serviço de troca de tokens	>=0,0,0	Rígido

2.1.2 and 2.1.3

A tabela a seguir lista as dependências das versões 2.1.2 e 2.1.3 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.8.0	Rígido
Lançador Lambda	^2.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	^2.0.0	Flexível
Serviço de troca de tokens	^2.0.0	Rígido

2.1.1

A tabela a seguir lista as dependências da versão 2.1.1 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.7.0	Rígido
Lançador Lambda	^2.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	^2.0.0	Flexível
Serviço de troca de tokens	^2.0.0	Rígido

2.0.8 - 2.1.0

A tabela a seguir lista as dependências das versões 2.0.8 a 2.1.0 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.6.0	Rígido
Lançador Lambda	^2.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	^2.0.0	Flexível
Serviço de troca de tokens	^2.0.0	Rígido

2.0.7

A tabela a seguir lista as dependências da versão 2.0.7 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.5.0	Rígido
Lançador Lambda	^2.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	^2.0.0	Flexível
Serviço de troca de tokens	^2.0.0	Rígido

2.0.6

A tabela a seguir lista as dependências da versão 2.0.6 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.4.0	Rígido

Dependência	Versões compatíveis	Tipo de dependência
Lançador Lambda	^2.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	^2.0.0	Flexível
Serviço de troca de tokens	^2.0.0	Rígido

2.0.5

A tabela a seguir lista as dependências da versão 2.0.5 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.3.0	Rígido
Lançador Lambda	^2.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	^2.0.0	Flexível
Serviço de troca de tokens	^2.0.0	Rígido

2.0.4

A tabela a seguir lista as dependências da versão 2.0.4 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.2.0	Rígido
Lançador Lambda	^2.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	^2.0.0	Flexível
Serviço de troca de tokens	^2.0.0	Rígido

2.0.3

A tabela a seguir lista as dependências da versão 2.0.3 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.3 <2.1.0	Rígido
Lançador Lambda	>=1.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	>=1.0.0	Flexível
Serviço de troca de tokens	>=1.0.0	Rígido

Para obter mais informações sobre dependências de componentes, consulte a referência da receita do componente.

Configuração

Esse componente fornece os seguintes parâmetros de configuração que você pode personalizar ao implantar o componente.

v3.x

`PublishInterval`

(Opcional) O número máximo de segundos de espera antes que o componente publique métricas em lote para um determinado namespace. Para configurar o componente para publicar métricas à medida que as recebe, ou seja, sem agrupamento em lotes, especifique`0`.

O componente é publicado CloudWatch após receber 20 métricas no mesmo namespace ou após o intervalo especificado por você.

> **Note**
>
> O componente não especifica a ordem na qual os eventos são publicados.

Esse valor pode ser de no máximo 900 segundos.

Padrão: 10 segundos

`MaxMetricsToRetain`

(Opcional) O número máximo de métricas em todos os namespaces a serem salvas na memória antes que o componente as substitua por métricas mais recentes.

Esse limite se aplica quando o dispositivo principal não tem uma conexão com a Internet, então o componente armazena em buffer as métricas para publicar posteriormente. Quando o buffer está cheio, o componente substitui as métricas mais antigas pelas mais recentes. As métricas em um determinado namespace substituem somente as métricas no mesmo namespace.

> **Note**
>
> Se o processo de hospedagem do componente for interrompido, o componente não salvará as métricas. Isso pode acontecer durante uma implantação ou quando o dispositivo principal é reiniciado, por exemplo.

Esse valor deve ser de pelo menos 2.000 métricas.

Padrão: 5.000 métricas

`InputTopic`

(Opcional) O tópico no qual o componente se inscreve para receber mensagens. Se você especificar `true` para`PubSubToIoTCore`, poderá usar curingas MQTT (+ e #) neste tópico.

Padrão: `cloudwatch/metric/put`

`OutputTopic`

(Opcional) O tópico no qual o componente publica respostas de status.

Padrão: `cloudwatch/metric/put/status`

`PubSubToIoTCore`

(Opcional) Valor da string que define se você deve publicar e assinar tópicos do AWS IoT Core MQTT. Os valores compatíveis são `true` e `false`.

Padrão: `false`

`UseInstaller`

(Opcional) Valor booleano que define se o script do instalador deve ser usado nesse componente para instalar as dependências do SDK desse componente.

Defina esse valor como `false` se você quiser usar um script personalizado para instalar dependências ou se quiser incluir dependências de tempo de execução em uma imagem Linux pré-criada. Para usar esse componente, você deve instalar as seguintes bibliotecas, incluindo quaisquer dependências, e disponibilizá-las para o usuário padrão do sistema Greengrass.

- AWS IoT Device SDK v2 para Python
- AWS SDK for Python (Boto3)

Padrão: `true`

`PublishRegion`

(Opcional) A Região da AWS CloudWatch métrica para a qual publicar. Esse valor substitui a região padrão do dispositivo principal. Esse parâmetro é necessário somente para métricas entre regiões.

`accessControl`

(Opcional) O objeto que contém a política de autorização que permite que o componente publique e assine os tópicos especificados. Se você especificar valores personalizados para `InputTopic` e`OutputTopic`, deverá atualizar os valores dos recursos nesse objeto.

Padrão:

```
{
  "aws.greengrass.ipc.pubsub": {
    "aws.greengrass.Cloudwatch:pubsub:1": {
      "policyDescription": "Allows access to subscribe to input topics.",
      "operations": [
        "aws.greengrass#SubscribeToTopic"
      ],
      "resources": [
        "cloudwatch/metric/put"
      ]
    },
    "aws.greengrass.Cloudwatch:pubsub:2": {
```

```
      "policyDescription": "Allows access to publish to output topics.",
      "operations": [
        "aws.greengrass#PublishToTopic"
      ],
      "resources": [
        "cloudwatch/metric/put/status"
      ]
    }
  },
  "aws.greengrass.ipc.mqttproxy": {
    "aws.greengrass.Cloudwatch:mqttproxy:1": {
      "policyDescription": "Allows access to subscribe to input topics.",
      "operations": [
        "aws.greengrass#SubscribeToIoTCore"
      ],
      "resources": [
        "cloudwatch/metric/put"
      ]
    },
    "aws.greengrass.Cloudwatch:mqttproxy:2": {
      "policyDescription": "Allows access to publish to output topics.",
      "operations": [
        "aws.greengrass#PublishToIoTCore"
      ],
      "resources": [
        "cloudwatch/metric/put/status"
      ]
    }
  }
}
```

Example Exemplo: atualização da mesclagem de configurações

```
{
  "PublishInterval": 0,
  "PubSubToIoTCore": true
}
```

v2.x

> **Note**
>
> A configuração padrão desse componente inclui parâmetros da função Lambda. Recomendamos que você edite somente os parâmetros a seguir para configurar esse componente em seus dispositivos.

`lambdaParams`

Um objeto que contém os parâmetros da função Lambda desse componente. Esse objeto contém as seguintes informações:

`EnvironmentVariables`

Um objeto que contém os parâmetros da função Lambda. Esse objeto contém as seguintes informações:

`PUBLISH_INTERVAL`

(Opcional) O número máximo de segundos de espera antes que o componente publique métricas em lote para um determinado namespace. Para configurar o componente para publicar métricas à medida que as recebe, ou seja, sem agrupamento em lotes, especifique`0`.

O componente é publicado CloudWatch após receber 20 métricas no mesmo namespace ou após o intervalo especificado por você.

> **Note**
>
> O componente não garante a ordem na qual os eventos são publicados.

Esse valor pode ser de no máximo 900 segundos.

Padrão: 10 segundos

`MAX_METRICS_TO_RETAIN`

(Opcional) O número máximo de métricas em todos os namespaces a serem salvas na memória antes que o componente as substitua por métricas mais recentes.

Esse limite se aplica quando o dispositivo principal não tem uma conexão com a Internet, então o componente armazena em buffer as métricas para publicar posteriormente. Quando o buffer está cheio, o componente substitui as métricas mais antigas pelas mais recentes. As métricas em um determinado namespace substituem somente as métricas no mesmo namespace.

> **Note**
>
> Se o processo de hospedagem do componente for interrompido, o componente não salvará as métricas. Isso pode acontecer durante uma implantação ou quando o dispositivo principal é reiniciado, por exemplo.

Esse valor deve ser de pelo menos 2.000 métricas.

Padrão: 5.000 métricas

`PUBLISH_REGION`

(Opcional) A Região da AWS CloudWatch métrica para a qual publicar. Esse valor substitui a região padrão do dispositivo principal. Esse parâmetro é necessário somente para métricas entre regiões.

`containerMode`

(Opcional) O modo de conteinerização desse componente. Escolha uma das seguintes opções:

- `NoContainer`— O componente não é executado em um ambiente de execução isolado.
- `GreengrassContainer`— O componente é executado em um ambiente de execução isolado dentro do AWS IoT Greengrass contêiner.

Padrão: `GreengrassContainer`

`containerParams`

(Opcional) Um objeto que contém os parâmetros do contêiner desse componente. O componente usa esses parâmetros se você especificar `GreengrassContainer` para`containerMode`.

Esse objeto contém as seguintes informações:

`memorySize`

(Opcional) A quantidade de memória (em kilobytes) a ser alocada para o componente.

O padrão é 64 MB (65.535 KB).

`pubsubTopics`

(Opcional) Um objeto que contém os tópicos em que o componente se inscreve para receber mensagens. Você pode especificar cada tópico e se o componente se inscreve nos tópicos do MQTT AWS IoT Core ou nos tópicos locais de publicação/assinatura.

Esse objeto contém as seguintes informações:

`0`— Este é um índice de matriz como uma string.

Um objeto que contém as seguintes informações:

`type`

(Opcional) O tipo de mensagem de publicação/assinatura que esse componente usa para assinar mensagens. Escolha uma das seguintes opções:

- `PUB_SUB` – Assine mensagens locais de publicar/assinar. Se você escolher essa opção, o tópico não poderá conter curingas do MQTT. Para obter mais informações sobre como enviar mensagens do componente personalizado ao especificar essa opção, consultePublique/assine mensagens locais.
- `IOT_CORE`— Assine as mensagens do AWS IoT Core MQTT. Se você escolher essa opção, o tópico poderá conter curingas do MQTT. Para obter mais informações sobre como enviar mensagens de componentes personalizados ao especificar essa opção, consultePublique/assine mensagens MQTT AWS IoT Core.

Padrão: `PUB_SUB`

`topic`

(Opcional) O tópico no qual o componente se inscreve para receber mensagens. Se você especificar `IotCore` para`type`, poderá usar curingas MQTT (+e#) neste tópico.

Example Exemplo: atualização da mesclagem de configuração (modo contêiner)

```
{
  "containerMode": "GreengrassContainer"
}
```

Example Exemplo: atualização de mesclagem de configuração (sem modo de contêiner)

```
{
  "containerMode": "NoContainer"
}
```

Dados de entrada

Esse componente aceita métricas no tópico a seguir e as publica em. CloudWatch Por padrão, esse componente assina mensagens locais de publicação/assinatura. Para obter mais informações sobre como publicar mensagens nesse componente a partir de seus componentes personalizados, consulte Publique/assine mensagens locais.

A partir da versão v3.0.0 do componente, você pode opcionalmente configurar esse componente para assinar um tópico do MQTT definindo o `PubSubToIoTCore` parâmetro de configuração como. `true` Para obter mais informações sobre a publicação de mensagens em um tópico do MQTT em seus componentes personalizados, consulte Publique/assine mensagens MQTT AWS IoT Core.

Tópico padrão: `cloudwatch/metric/put`

A mensagem aceita as seguintes propriedades. As mensagens de entrada devem estar no formato JSON.

`request`

A métrica nesta mensagem.

O objeto de solicitação contém os dados de métrica para publicação no CloudWatch. Os valores métricos devem atender às especificações da `PutMetricData` operação.

Tipo: `object` que contém as seguintes informações:

`namespace`

O namespace definido pelo usuário para os dados métricos nessa solicitação. CloudWatch usa namespaces como contêineres para pontos de dados métricos.

> **Note**
>
> Você não pode especificar um namespace que comece com a string reservada `AWS/`.

Digite: `string`

Padrão válido: `[^:].*`

`metricData`

Os dados para a métrica.

Tipo: `object` que contém as seguintes informações:

`metricName`

O nome da métrica.

Tipo: `string`

`value`

O valor para a métrica.

> **Note**
>
> CloudWatch rejeita valores muito pequenos ou muito grandes. O valor deve estar entre `8.515920e-109` e `1.174271e+108` (Base 10) ou `2e-360` e `2e360` (Base 2). CloudWatch não suporta valores especiais como `NaN`+`Infinity`, `-Infinity` e.

Tipo: `double`

`dimensions`

(Opcional) As dimensões da métrica. As dimensões fornecem informações adicionais sobre a métrica e seus dados. Uma métrica pode definir até 10 dimensões.

Esse componente inclui automaticamente uma dimensão chamada`coreName`, em que o valor é o nome do dispositivo principal.

Tipo: `array` de objetos em que cada um contém as seguintes informações:

`name`

(Opcional) O nome da dimensão.

Tipo: `string`

`value`

(Opcional) O valor da dimensão.

Tipo: `string`

`timestamp`

(Opcional) A hora em que os dados métricos foram recebidos, expressa em segundos no horário da época do Unix.

O padrão é a hora em que o componente recebe a mensagem.

Tipo: `double`

> **Note**
>
> Se você usar entre as versões 2.0.3 e 2.0.7 desse componente, recomendamos que você recupere o timestamp separadamente para cada métrica ao enviar várias métricas de uma única fonte. Não use uma variável para armazenar o timestamp.

`unit`

(Opcional) A unidade da métrica.

Tipo: `string`

Valores válidos:
`Seconds`,`Microseconds`,`Milliseconds`,`Bytes`,,`Kilobytes`,`Megabytes`,`Gigabytes`,`Teraby`
`Second`,`Kilobytes/Second`,`Megabytes/Second`,`Gigabytes/Second`,,`Terabytes/`
`Second`,`Bits/Second`,`Kilobits/Second`,`Megabits/Second`,`Gigabits/`
`Second`,`Terabits/Second`,`Count/Second`, `None`

Padronizado como `None`.

> **Note**
>
> Todas as cotas que se aplicam à CloudWatch `PutMetricData` API se aplicam às métricas que você publica com esse componente. As seguintes cotas são especialmente importantes:

- Limite de 40 KB na carga útil da API
- 20 métricas por solicitação de API
- 150 transações por segundo (TPS) para a API `PutMetricData`

Para obter mais informações, consulte as cotas de CloudWatch serviço no Guia do CloudWatch usuário.

Example Exemplo de entrada

```
{
  "request": {
    "namespace": "Greengrass",
    "metricData": {
      "metricName": "latency",
      "dimensions": [
        {
          "name": "hostname",
          "value": "test_hostname"
        }
      ],
      "timestamp": 1539027324,
      "value": 123.0,
      "unit": "Seconds"
    }
  }
}
```

Dados de saída

Por padrão, esse componente publica respostas como dados de saída no seguinte tópico local de publicação/assinatura. Para obter mais informações sobre como assinar mensagens sobre esse tópico em seus componentes personalizados, consultePublique/assine mensagens locais.

Opcionalmente, você pode configurar esse componente para publicar em um tópico do MQTT definindo o parâmetro de `PubSubToIoTCore` configuração como. `true` Para obter mais informações sobre como assinar mensagens sobre um tópico do MQTT em seus componentes personalizados, consulte. Publique/assine mensagens MQTT AWS IoT Core

> **Note**
>
> As versões 2.0.x do componente publicam respostas como dados de saída em um tópico do MQTT por padrão. Você deve especificar o tópico conforme a `subject` configuração do componente antigo do roteador de assinatura.

Tópico padrão: `cloudwatch/metric/put/status`

Example Exemplo de resultado: sucesso

A resposta inclui o namespace dos dados métricos e o `RequestId` campo da CloudWatch resposta.

```
{
  "response": {
    "cloudwatch_rid": "70573243-d723-11e8-b095-75ff2EXAMPLE",
    "namespace": "Greengrass",
    "status": "success"
  }
}
```

Example Exemplo de resultado: falha

```
{
  "response" : {
    "namespace": "Greengrass",
    "error": "InvalidInputException",
    "error_message": "cw metric is invalid",
    "status": "fail"
  }
}
```

> **Note**
>
> Se o componente detectar um erro que possa ser repetido, como um erro de conexão, ele tentará publicar novamente no próximo lote.

Licenças

Esse componente inclui o seguinte software/licenciamento de terceiros:

- AWS SDK for Python (Boto3)/Licença Apache 2.0
- botocore/Licença Apache 2.0
- dateutil/Licença PSF
- docutils/Licença BSD, GNU Licença pública geral (GPL), Licença Python Software Foundation, Domínio público
- jmespath/Licença MIT
- s3transfer/Licença Apache 2.0
- urllib3/Licença MIT

Esse componente é lançado de acordo com o Contrato de Licença de Software Principal do Greengrass.

Arquivo de log local

Esse componente usa o seguinte arquivo de log.

Linux

```
/greengrass/v2/logs/aws.greengrass.Cloudwatch.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.Cloudwatch.log
```

Para ver os registros desse componente

- Execute o comando a seguir no dispositivo principal para visualizar o arquivo de log desse componente em tempo real. Substitua */greengrass/v2* ou *C:\greengrass\v2* pelo caminho para a pasta AWS IoT Greengrass raiz.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.greengrass.Cloudwatch.log
  ```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\aws.greengrass.Cloudwatch.log -Tail 10 -Wait
```

Changelog

A tabela a seguir descreve as alterações em cada versão do componente.

v3.x

Version (Versão)	Alterações
3.1.0	Correções de erros e melhorias • Adiciona suporte para configurações de proxy de rede HTTPS. Para obter mais informações, consulte Permita que o dispositivo principal confie em um proxy HTTPS e Conectar-se à porta 443 ou por meio de um proxy de rede.
3.0.0	Essa versão do componente de CloudWatch métricas espera parâmetros de configuração diferentes da versão 2.x. Se você usar uma configuração não padrão para a versão 2.x e quiser atualizar da v2.x para a v3.x, deverá atualizar a configuração do componente. Para obter mais informações, consulte a configuração do componente de CloudWatch métricas. Novos atributos • Adiciona suporte para dispositivos principais que executam o Windows. • Altera o tipo de componente de componente Lambda para componente genérico. Agora, esse componente não depende mais do componente antigo do roteador de assinatura para criar assinaturas. • Adiciona um novo parâmetro de `InputTopic` configuração para especificar o tópico no qual o componente se inscreve para receber mensagens.

Version (Versão)	Alterações
	• Adiciona um novo parâmetro de `OutputTopic` configuração para especificar o tópico no qual o componente publica respostas de status. • Adiciona um novo parâmetro de `PubSubToIoTCore` configuração para especificar se deseja publicar e assinar tópicos do AWS IoT Core MQTT. • Adiciona o novo parâmetro `UseInstaller` de configuração que permite desativar opcionalmente o script de instalação que instala as dependências dos componentes. Correções de erros e melhorias Adiciona suporte para carimbos de data/hora duplicados nos dados de entrada.

v2.x

Version (Versão)	Alterações
2.1.3	Versão atualizada para a versão 2.11.0 do Greengrass nucleus.
2.1.2	Versão atualizada para a versão 2.7.0 do Greengrass nucleus.
2.1.1	Versão atualizada para a versão 2.6.0 do Greengrass nucleus.
2.1.0	Novos atributos • Adiciona suporte para configurações de proxy de rede HTTPS. Para obter mais informações, consulte Permita que o dispositivo principal confie em um proxy HTTPS e Conectar-se à porta 443 ou por meio de um proxy de rede.
2.0.8	Correções de erros e melhorias • Adiciona suporte para carimbos de data/hora duplicados nos dados de entrada.

Version (Versão)	Alterações
	• Versão atualizada para a versão 2.5.0 do Greengrass nucleus.
2.0.7	Versão atualizada para a versão 2.4.0 do Greengrass nucleus.
2.0.6	Versão atualizada para a versão 2.3.0 do Greengrass nucleus.
2.0.5	Versão atualizada para a versão 2.2.0 do Greengrass nucleus.
2.0.4	Versão atualizada para a versão 2.1.0 do Greengrass nucleus.
2.0.3	Versão inicial.

Consulte também

- Usando CloudWatch métricas da Amazon no Guia do CloudWatch usuário da Amazon
- PutMetricDatana Amazon CloudWatch API Reference

AWS IoT Device Defender

O AWS IoT Device Defender componente (`aws.greengrass.DeviceDefender`) notifica os administradores sobre mudanças no estado dos dispositivos principais do Greengrass. Isso pode ajudar a identificar comportamento incomum e, assim, indicar um dispositivo comprometido. Para obter mais informações, consulte AWS IoT Device Defender no AWS IoT Core Guia do desenvolvedor.

Esse componente lê as métricas do sistema no dispositivo principal. Em seguida, ele publica as métricas para AWS IoT Device Defender. Para obter mais informações sobre como ler e interpretar as métricas que esse componente relata, consulte Especificação do documento de métricas do dispositivo no Guia do AWS IoT Core desenvolvedor.

> **Note**
>
> Esse componente fornece funcionalidade semelhante ao conector do Device Defender em AWS IoT Greengrass V1. Para obter mais informações, consulte o conector do Device Defender no Guia do AWS IoT Greengrass V1 desenvolvedor.

Tópicos

Versões

Esse componente tem as seguintes versões:

- 3.1.x
- 3.0.x
- 2.0.x

Para obter informações sobre mudanças em cada versão do componente, consulte o changelog.

Tipo

v3.x

Este componente é um componente genérico (`aws.greengrass.generic`). O núcleo do Greengrass executa os scripts do ciclo de vida do componente.

v2.x

Esse componente é um componente Lambda () `aws.greengrass.lambda`. O núcleo do Greengrass executa a função Lambda desse componente usando o componente lançador Lambda.

Para ter mais informações, consulte Tipos de componentes.

Sistema operacional

v3.x

Esse componente pode ser instalado em dispositivos principais que executam os seguintes sistemas operacionais:

- Linux
- Windows

v2.x

Esse componente pode ser instalado somente nos dispositivos principais do Linux.

Requisitos

Esse componente tem os seguintes requisitos:

v3.x

- Python versão 3.7 instalado no dispositivo de núcleo e adicionado à variável de ambiente PATH.
- AWS IoT Device Defender configurado para usar o recurso Detectar para monitorar violações. Para obter mais informações, consulte Detectar no Guia do desenvolvedor do AWS IoT Core .

v2.x

- Seu dispositivo principal deve atender aos requisitos para executar as funções do Lambda. Se você quiser que o dispositivo principal execute funções Lambda em contêineres, o dispositivo deve atender aos requisitos para fazer isso. Para ter mais informações, consulte Requisitos da função do Lambda.
- Python versão 3.7 instalado no dispositivo de núcleo e adicionado à variável de ambiente PATH.
- AWS IoT Device Defender configurado para usar o recurso Detectar para monitorar violações. Para obter mais informações, consulte Detectar no Guia do desenvolvedor do AWS IoT Core .

- A biblioteca psutil instalada no dispositivo principal. A versão 5.7.0 é a versão mais recente verificada para funcionar com o componente.
- A biblioteca cbor instalada no dispositivo principal. A versão 1.0.0 é a versão mais recente verificada para funcionar com o componente.
- Para receber dados de saída desse componente, você deve mesclar a seguinte atualização de configuração para o componente antigo do roteador de assinatura (`aws.greengrass.LegacySubscriptionRouter`) ao implantar esse componente. Essa configuração especifica o tópico em que esse componente publica as respostas.

  Legacy subscription router v2.1.x

  ```
  {
    "subscriptions": {
      "aws-greengrass-device-defender": {
        "id": "aws-greengrass-device-defender",
        "source": "component:aws.greengrass.DeviceDefender",
        "subject": "$aws/things/+/defender/metrics/json",
        "target": "cloud"
      }
    }
  }
  ```

  Legacy subscription router v2.0.x

  ```
  {
    "subscriptions": {
      "aws-greengrass-device-defender": {
        "id": "aws-greengrass-device-defender",
        "source": "arn:aws:lambda:region:aws:function:aws-greengrass-device-
  defender:version",
        "subject": "$aws/things/+/defender/metrics/json",
        "target": "cloud"
      }
    }
  }
  ```

  - Substitua a *região* pela Região da AWS que você usa.
  - *Substitua a versão pela versão da função Lambda que esse componente executa.* Para encontrar a versão da função Lambda, você deve visualizar a receita da versão desse componente que você deseja implantar. Abra a

página de detalhes desse componente no AWS IoT Greengrass console e procure o par de valores-chave da função Lambda. Esse par de valores-chave contém o nome e a versão da função Lambda.

> **Important**
>
> Você deve atualizar a versão da função Lambda no roteador de assinatura legado sempre que implantar esse componente. Isso garante que você use a versão correta da função Lambda para a versão do componente que você implanta.

Para ter mais informações, consulte Criar implantações.

Dependências

Quando você implanta um componente, AWS IoT Greengrass também implanta versões compatíveis de suas dependências. Isso significa que você deve atender aos requisitos do componente e de todas as suas dependências para implantá-lo com êxito. Esta seção lista as dependências das versões lançadas desse componente e as restrições de versão semântica que definem as versões do componente para cada dependência. Você também pode visualizar as dependências de cada versão do componente no AWS IoT Greengrass console. Na página de detalhes do componente, procure a lista de dependências.

3.1.1

A tabela a seguir lista as dependências da versão 3.1.1 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <3.0.0	Flexível
Serviço de troca de tokens	>=0,0,0	Rígido

3.0.0 - 3.0.2

A tabela a seguir lista as dependências das versões 3.0.0 a 3.0.2 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <3.0.0	Flexível
Serviço de troca de tokens	>=0,0,0	Rígido

2.0.10 and 2.0.11

A tabela a seguir lista as dependências das versões 2.0.10 e 2.0.11 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.8.0	Rígido
Lançador Lambda	^2.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	^2.0.0	Flexível
Serviço de troca de tokens	^2.0.0	Rígido

2.0.9

A tabela a seguir lista as dependências da versão 2.0.9 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.7.0	Rígido
Lançador Lambda	^2.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	^2.0.0	Flexível
Serviço de troca de tokens	^2.0.0	Rígido

2.0.8

A tabela a seguir lista as dependências da versão 2.0.8 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.6.0	Rígido
Lançador Lambda	^2.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	^2.0.0	Flexível
Serviço de troca de tokens	^2.0.0	Rígido

2.0.7

A tabela a seguir lista as dependências da versão 2.0.7 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.5.0	Rígido
Lançador Lambda	^2.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	^2.0.0	Flexível
Serviço de troca de tokens	^2.0.0	Rígido

2.0.6

A tabela a seguir lista as dependências da versão 2.0.6 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.4.0	Rígido

Dependência	Versões compatíveis	Tipo de dependência
Lançador Lambda	^2.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	^2.0.0	Flexível
Serviço de troca de tokens	^2.0.0	Rígido

2.0.5

A tabela a seguir lista as dependências da versão 2.0.5 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.3.0	Rígido
Lançador Lambda	^2.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	^2.0.0	Flexível
Serviço de troca de tokens	^2.0.0	Rígido

2.0.4

A tabela a seguir lista as dependências da versão 2.0.4 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.2.0	Rígido
Lançador Lambda	^2.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	^2.0.0	Flexível
Serviço de troca de tokens	^2.0.0	Rígido

2.0.3

A tabela a seguir lista as dependências da versão 2.0.3 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.3 <2.1.0	Rígido
Lançador Lambda	>=1.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	>=1.0.0	Flexível
Serviço de troca de tokens	>=1.0.0	Rígido

Para obter mais informações sobre dependências de componentes, consulte a referência da receita do componente.

Configuração

Esse componente fornece os seguintes parâmetros de configuração que você pode personalizar ao implantar o componente.

v3.x

`PublishRetryCount`

A quantidade de vezes que a publicação será repetida. Esse recurso está disponível na versão 3.1.1.

O mínimo é 0.

O máximo é 72.

Padrão: 5

`SampleIntervalSeconds`

(Opcional) A quantidade de tempo em segundos entre cada ciclo em que o componente coleta e relata métricas.

O valor mínimo é 300 segundos (5 minutos).

Padrão: 300 segundos

`UseInstaller`

(Opcional) Valor booleano que define se o script do instalador deve ser usado nesse componente para instalar as dependências desse componente.

Defina esse valor como `false` se você quiser usar um script personalizado para instalar dependências ou se quiser incluir dependências de tempo de execução em uma imagem Linux pré-criada. Para usar esse componente, você deve instalar as seguintes bibliotecas, incluindo quaisquer dependências, e disponibilizá-las para o usuário padrão do sistema Greengrass.

- AWS IoT Device SDK v2 para Python
- biblioteca Cbor. A versão 1.0.0 é a versão mais recente verificada para funcionar com o componente.
- biblioteca psutil. A versão 5.7.0 é a versão mais recente verificada para funcionar com o componente.

> **Note**
>
> Se você usa a versão 3.0.0 ou 3.0.1 desse componente em dispositivos principais que você configura para usar um proxy HTTPS, defina esse valor como. `false` O script do instalador não oferece suporte à operação por trás de um proxy HTTPS nessas versões desse componente.

Padrão: `true`

v2.x

> **Note**
>
> A configuração padrão desse componente inclui parâmetros da função Lambda. Recomendamos que você edite somente os parâmetros a seguir para configurar esse componente em seus dispositivos.

`lambdaParams`

Um objeto que contém os parâmetros da função Lambda desse componente. Esse objeto contém as seguintes informações:

`EnvironmentVariables`

Um objeto que contém os parâmetros da função Lambda. Esse objeto contém as seguintes informações:

`PROCFS_PATH`

(Opcional) O caminho para a `/proc` pasta.

- Para executar esse componente em um contêiner, use o valor padrão,`/host-proc`. O componente é executado em um contêiner por padrão.
- Para executar esse componente no modo sem contêiner, especifique `/proc` esse parâmetro.

Padrão: `/host-proc`. Esse é o caminho padrão em que esse componente monta a `/proc` pasta no contêiner.

> **Note**
>
> Esse componente tem acesso somente para leitura a essa pasta.

`SAMPLE_INTERVAL_SECONDS`

(Opcional) A quantidade de tempo em segundos entre cada ciclo em que o componente coleta e relata métricas.

O valor mínimo é 300 segundos (5 minutos).

Padrão: 300 segundos

`containerMode`

(Opcional) O modo de conteinerização desse componente. Escolha uma das seguintes opções:

- `GreengrassContainer`— O componente é executado em um ambiente de execução isolado dentro do AWS IoT Greengrass contêiner.
- `NoContainer`— O componente não é executado em um ambiente de execução isolado.

Se você especificar essa opção, deverá especificar `/proc` para o parâmetro da variável de `PROCFS_PATH` ambiente.

Padrão: `GreengrassContainer`

`containerParams`

(Opcional) Um objeto que contém os parâmetros do contêiner desse componente. O componente usa esses parâmetros se você especificar `GreengrassContainer` para`containerMode`.

Esse objeto contém as seguintes informações:

`memorySize`

(Opcional) A quantidade de memória (em kilobytes) a ser alocada para o componente.

O padrão é 50.000 KB.

`pubsubTopics`

(Opcional) Um objeto que contém os tópicos em que o componente se inscreve para receber mensagens. Você pode especificar cada tópico e se o componente se inscreve nos tópicos do MQTT AWS IoT Core ou nos tópicos locais de publicação/assinatura.

Esse objeto contém as seguintes informações:

`0`— Este é um índice de matriz como uma string.

Um objeto que contém as seguintes informações:

`type`

(Opcional) O tipo de mensagem de publicação/assinatura que esse componente usa para assinar mensagens. Escolha uma das seguintes opções:

- `PUB_SUB` – Assine mensagens locais de publicar/assinar. Se você escolher essa opção, o tópico não poderá conter curingas do MQTT. Para obter mais informações sobre como enviar mensagens do componente personalizado ao especificar essa opção, consulte[Publique/assine mensagens locais](#).
- `IOT_CORE`— Assine as mensagens do AWS IoT Core MQTT. Se você escolher essa opção, o tópico poderá conter curingas do MQTT. Para obter mais informações sobre como enviar mensagens de componentes personalizados ao especificar essa opção, consulte[Publique/assine mensagens MQTT AWS IoT Core](#).

Padrão: PUB_SUB

`topic`

(Opcional) O tópico no qual o componente se inscreve para receber mensagens. Se você especificar `IotCore` para`type`, poderá usar curingas MQTT (+e#) neste tópico.

Example Exemplo: atualização da mesclagem de configuração (modo contêiner)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "PROCFS_PATH": "/host_proc"
    }
  },
  "containerMode": "GreengrassContainer"
}
```

Example Exemplo: atualização de mesclagem de configuração (sem modo de contêiner)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "PROCFS_PATH": "/proc"
    }
  },
  "containerMode": "NoContainer"
}
```

Dados de entrada

Esse componente não aceita mensagens como dados de entrada.

Dados de saída

Esse componente publica métricas de segurança no seguinte tópico reservado para AWS IoT Device Defender. Esse componente é *`coreDeviceName`*substituído pelo nome do dispositivo principal quando ele publica as métricas.

Tópico (AWS IoT Core MQTT): `$aws/things/`*`coreDeviceName`*`/defender/metrics/json`

Example Exemplo de saída


```
{
  "header": {
    "report_id": 1529963534,
    "version": "1.0"
  },
  "metrics": {
    "listening_tcp_ports": {
      "ports": [
        {
          "interface": "eth0",
          "port": 24800
        },
        {
          "interface": "eth0",
          "port": 22
        },
        {
          "interface": "eth0",
          "port": 53
        }
      ],
      "total": 3
    },
    "listening_udp_ports": {
      "ports": [
        {
          "interface": "eth0",
          "port": 5353
        },
        {
          "interface": "eth0",
          "port": 67
        }
      ],
      "total": 2
    },
    "network_stats": {
      "bytes_in": 1157864729406,
      "bytes_out": 1170821865,
      "packets_in": 693092175031,
      "packets_out": 738917180
    },
```

```
    "tcp_connections": {
      "established_connections":{
        "connections": [
          {
            "local_interface": "eth0",
            "local_port": 80,
            "remote_addr": "192.168.0.1:8000"
          },
          {
            "local_interface": "eth0",
            "local_port": 80,
            "remote_addr": "192.168.0.1:8000"
          }
        ],
        "total": 2
      }
    }
  }
}
```

Para obter mais informações sobre as métricas que esse componente relata, consulte Especificação do documento de métricas do dispositivo no Guia do AWS IoT Core desenvolvedor.

Arquivo de log local

Esse componente usa o seguinte arquivo de log.

Linux

```
/greengrass/v2/logs/aws.greengrass.DeviceDefender.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.DeviceDefender.log
```

Para ver os registros desse componente

- Execute o comando a seguir no dispositivo principal para visualizar o arquivo de log desse componente em tempo real. Substitua */greengrass/v2* ou *C:\greengrass\v2* pelo caminho para a pasta AWS IoT Greengrass raiz.

Linux

```
sudo tail -f /greengrass/v2/logs/aws.greengrass.DeviceDefender.log
```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\aws.greengrass.DeviceDefender.log -Tail 10 -
Wait
```

Licenças

Esse componente é lançado de acordo com o Contrato de Licença de Software Principal do Greengrass.

Changelog

A tabela a seguir descreve as alterações em cada versão do componente.

v3.x

Version (Versão)	Alterações
3.1.1	Correções de erros e melhorias • Adiciona novas tentativas para a conexão do cliente quando a conexão não se recupera após uma interrupção na rede. • Adiciona uma nova tentativa configurável para publicar métricas.
3.1.0	Correções de erros e melhorias • Adiciona suporte para configurações de proxy de rede HTTPS. Para obter mais informações, consulte Permita que o dispositivo principal confie em um proxy HTTPS e Conectar-se à porta 443 ou por meio de um proxy de rede.
3.0.1	Corrige um problema na forma como o componente calcula valores delta para métricas.

Version (Versão)	Alterações
3.0.0	**Warning** Essa versão não está mais disponível. As melhorias nesta versão estão disponíveis em versões posteriores desse componente. Versão inicial.

v2.x

Version (Versão)	Alterações
2.0.11	Versão atualizada para a versão 2.11.0 do Greengrass nucleus.
2.0.10	Versão atualizada para a versão 2.7.0 do Greengrass nucleus.
2.0.9	Versão atualizada para a versão 2.6.0 do Greengrass nucleus.
2.0.8	Versão atualizada para a versão 2.5.0 do Greengrass nucleus.
2.0.7	Versão atualizada para a versão 2.4.0 do Greengrass nucleus.
2.0.6	Versão atualizada para a versão 2.3.0 do Greengrass nucleus.
2.0.5	Versão atualizada para a versão 2.2.0 do Greengrass nucleus.
2.0.4	Versão atualizada para a versão 2.1.0 do Greengrass nucleus.
2.0.3	Versão inicial.

Spooler de disco

O componente spooler de disco (`aws.greengrass.DiskSpooler`) oferece uma opção de armazenamento persistente para mensagens enviadas dos dispositivos principais do Greengrass para o. AWS IoT Core Esse componente armazenará essas mensagens de saída no disco.

Tópicos

Versões

Esse componente tem as seguintes versões:

- 1,0.x

Tipo

Este componente é um componente de plug-in (`aws.greengrass.plugin`). O núcleo do Greengrass executa esse componente na mesma Java Virtual Machine (JVM) do núcleo. O núcleo reinicia quando você altera a versão desse componente no dispositivo principal.

Esse componente usa o mesmo arquivo de log do núcleo do Greengrass. Para ter mais informações, consulte Monitore AWS IoT Greengrass os registros.

Para ter mais informações, consulte Tipos de componentes.

Sistema operacional

Esse componente pode ser instalado em dispositivos principais que executam os seguintes sistemas operacionais:

- Linux
- Windows

Requisitos

Esse componente tem os seguintes requisitos:

- `storageType`deve ser configurado `Disk` para usar esse componente. Você pode definir isso na configuração do núcleo do Greengrass.
- `maxSizeInBytes`não deve ser configurado para ser maior do que o espaço disponível no dispositivo. Você pode definir isso na configuração do núcleo do Greengrass.
- O componente spooler de disco tem suporte para execução em uma VPC.

Dependências

Quando você implanta um componente, AWS IoT Greengrass também implanta versões compatíveis de suas dependências. Isso significa que você deve atender aos requisitos do componente e de todas as suas dependências para implantá-lo com êxito. Esta seção lista as dependências das versões lançadas desse componente e as restrições de versão semântica que definem as versões do componente para cada dependência. Você também pode visualizar as dependências de cada versão do componente no AWS IoT Greengrass console. Na página de detalhes do componente, procure a lista de dependências.

1.0.1 – 1.0.3

A tabela a seguir lista as dependências das versões 1.0.1 a 1.0.3 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.11.0 <2.13.0	Rígido

1.0.0

A tabela a seguir lista as dependências da versão 1.0.0 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2,11,0 <2,12,0	Rígido

Para obter mais informações sobre dependências de componentes, consulte a referência da receita do componente.

Uso

Para usar o componente spooler de disco, ele `aws.greengrass.DiskSpooler` deve ser implantado.

Para configurar e usar esse componente, você deve `pluginName` definir `aws.greengrass.DiskSpooler` o.

Arquivo de log local

Esse componente usa o mesmo arquivo de log do componente do núcleo do Greengrass.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

Para ver os registros desse componente

- Execute o comando a seguir no dispositivo principal para visualizar o arquivo de log desse componente em tempo real. Substitua */greengrass/v2* ou *C:\greengrass\v2* pelo caminho para a pasta AWS IoT Greengrass raiz.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
```

Changelog

A tabela a seguir descreve as alterações em cada versão do componente.

Version (Versão)	Alterações
1.0.3	Correções de erros e melhorias Melhora o desempenho ao reutilizar conexões de banco de dados.
1.0.2	Correções de erros e melhorias Corrige um problema em que o campo de formato de mensagem MQTT não persiste em certos casos.
1.0.1	Versão atualizada para a versão 2.12.0 do Greengrass nucleus.
1.0.0	Versão inicial.

Gerenciador de aplicativos Docker

O componente Docker Application Manager (`aws.greengrass.DockerApplicationManager`) permite AWS IoT Greengrass baixar imagens do Docker de registros públicos de imagens e registros privados hospedados no Amazon Elastic Container Registry (Amazon ECR). Ele também permite AWS IoT Greengrass gerenciar credenciais automaticamente para baixar imagens com segurança de repositórios privados no Amazon ECR.

Ao desenvolver um componente personalizado que executa um contêiner Docker, inclua o gerenciador de aplicativos Docker como uma dependência para baixar as imagens do Docker que são especificadas como artefatos em seu componente. Para ter mais informações, consulte Execute um contêiner Docker.

Tópicos

Versões

Esse componente tem as seguintes versões:

- 2.0.x

Tipo

Este componente é um componente genérico (`aws.greengrass.generic`). O núcleo do Greengrass executa os scripts do ciclo de vida do componente.

Para ter mais informações, consulte Tipos de componentes.

Sistema operacional

Esse componente pode ser instalado em dispositivos principais que executam os seguintes sistemas operacionais:

- Linux
- Windows

Requisitos

Esse componente tem os seguintes requisitos:

- Docker Engine 1.9.1 ou posterior instalado no dispositivo principal do Greengrass. A versão 20.10 é a versão mais recente verificada para funcionar com o software AWS IoT Greengrass Core. Você

deve instalar o Docker diretamente no dispositivo principal antes de implantar componentes que executam contêineres do Docker.

- O daemon do Docker foi iniciado e executado no dispositivo principal antes de você implantar esse componente.
- Imagens do Docker armazenadas em uma das seguintes fontes de imagem compatíveis:
  - Repositórios de imagens públicos e privados no Amazon Elastic Container Registry (Amazon ECR)
  - Repositório público do Docker Hub
  - Registro confiável do Public Docker
- Imagens do Docker incluídas como artefatos em seus componentes personalizados do contêiner Docker. Use os seguintes formatos de URI para especificar suas imagens do Docker:
  - Imagem privada do Amazon ECR: `docker:`*`account-id`*`.dkr.ecr.`*`region`*`.amazonaws.com/`*`repository`*`/`*`image`*`[`*`:tag|@digest`*`]`
  - Imagem pública do Amazon ECR: `docker:public.ecr.aws/`*`repository`*`/`*`image`*`[`*`:tag|@digest`*`]`
  - Imagem pública do Docker Hub: `docker:`*`name`*`[`*`:tag|@digest`*`]`

  Para ter mais informações, consulte Execute um contêiner Docker.

  > **Note**
  >
  > Se você não especificar a tag da imagem ou o resumo da imagem no URI do artefato de uma imagem, o gerenciador de aplicativos Docker extrairá a versão mais recente disponível dessa imagem quando você implanta seu componente de contêiner personalizado do Docker. Para garantir que todos os seus dispositivos principais executem a mesma versão de uma imagem, recomendamos que você inclua a tag da imagem ou o resumo da imagem no URI do artefato.

- O usuário do sistema que executa um componente de contêiner do Docker deve ter permissões de raiz ou administrador, ou você deve configurar o Docker para executá-lo como usuário não raiz ou não administrador.
  - Em dispositivos Linux, você pode adicionar um usuário ao `docker` grupo para chamar `docker` comandos sem`sudo`.
  - Em dispositivos Windows, você pode adicionar um usuário ao `docker-users` grupo para chamar `docker` comandos sem privilégios de administrador.

Linux or Unix

Para adicionar `ggc_user` ou o usuário não root que você usa para executar componentes de contêiner do Docker ao `docker` grupo, execute o comando a seguir.

```
sudo usermod -aG docker ggc_user
```

Para obter mais informações, consulte Gerenciar o Docker como usuário não root.

Windows Command Prompt (CMD)

Para adicionar `ggc_user` ou o usuário que você usa para executar componentes de contêiner do Docker ao `docker-users` grupo, execute o comando a seguir como administrador.

```
net localgroup docker-users ggc_user /add
```

Windows PowerShell

Para adicionar `ggc_user` ou o usuário que você usa para executar componentes de contêiner do Docker ao `docker-users` grupo, execute o comando a seguir como administrador.

```
Add-LocalGroupMember -Group docker-users -Member ggc_user
```

- Se você configurar o software AWS IoT Greengrass Core para usar um proxy de rede, deverá configurar o Docker para usar o mesmo servidor proxy.
- Se suas imagens do Docker estiverem armazenadas em um registro privado do Amazon ECR, você deverá incluir o componente do serviço de troca de tokens como uma dependência no componente de contêiner do Docker. Além disso, a função de dispositivo do Greengrass deve permitir as `ecr:GetDownloadUrlForLayer` ações`ecr:GetAuthorizationToken`,`ecr:BatchGetImage`, e, conforme mostrado no exemplo de política do IAM a seguir.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "ecr:GetAuthorizationToken",
        "ecr:BatchGetImage",
        "ecr:GetDownloadUrlForLayer"
```

```
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

- O componente docker application manager tem suporte para execução em uma VPC. Para implantar esse componente em uma VPC, é necessário o seguinte.
  - O componente docker application manager deve ter conectividade para baixar imagens. Por exemplo, se você usa o ECR, deve ter conectividade com os seguintes endpoints.
    - `*.dkr.ecr.region.amazonaws.com`(VPC endpoint) `com.amazonaws.region.ecr.dkr`
    - `api.ecr.region.amazonaws.com`(VPC endpoint) `com.amazonaws.region.ecr.api`

Endpoints e portas

Esse componente deve ser capaz de realizar solicitações de saída para os seguintes endpoints e portas, além dos endpoints e portas necessários para a operação básica. Para ter mais informações, consulte Permitir o tráfego de dispositivos por meio de um proxy ou firewall.

Endpoint	Porta	Obrigatório	Descrição
`ecr.region.amazonaws.com`	443	Não	Obrigatório se você baixar imagens do Docker do Amazon ECR.

`hub.docker.com`
 	
`registry.hub.docker.com/v1`
	443	Não	Obrigatório se você baixar imagens do Docker

Endpoint	Porta	Obrigatório	Descrição
			do Docker Hub.

Dependências

Quando você implanta um componente, AWS IoT Greengrass também implanta versões compatíveis de suas dependências. Isso significa que você deve atender aos requisitos do componente e de todas as suas dependências para implantá-lo com êxito. Esta seção lista as dependências das versões lançadas desse componente e as restrições de versão semântica que definem as versões dos componentes para cada dependência. Você também pode visualizar as dependências de cada versão do componente no AWS IoT Greengrassconsole. Na página de detalhes do componente, procure a lista de dependências.

2.0.11

A tabela a seguir lista as dependências da versão 2.0.11 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.1.0 <2.13.0	Flexível

2.0.10

A tabela a seguir lista as dependências da versão 2.0.10 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.1.0 <2.12.0	Flexível

2.0.9

A tabela a seguir lista as dependências da versão 2.0.9 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.1.0 <2.11.0	Flexível

2.0.8

A tabela a seguir lista as dependências da versão 2.0.8 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.1.0 <2.10.0	Flexível

2.0.7

A tabela a seguir lista as dependências da versão 2.0.7 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.1.0 <2.9.0	Flexível

2.0.6

A tabela a seguir lista as dependências da versão 2.0.6 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.1.0 <2.8.0	Flexível

2.0.5

A tabela a seguir lista as dependências da versão 2.0.5 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.1.0 <2.7.0	Flexível

2.0.4

A tabela a seguir lista as dependências da versão 2.0.4 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.1.0 <2.6.0	Flexível

2.0.3

A tabela a seguir lista as dependências da versão 2.0.3 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.1.0 <2.5.0	Flexível

2.0.2

A tabela a seguir lista as dependências da versão 2.0.2 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.1.0 <2.4.0	Flexível

2.0.1

A tabela a seguir lista as dependências da versão 2.0.1 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.1.0 <2.3.0	Flexível

2.0.0

A tabela a seguir lista as dependências da versão 2.0.0 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.1.0 <2.2.0	Flexível

Para obter mais informações sobre dependências de componentes, consulte a referência da receita do componente.

Configuração

Esse componente não tem nenhum parâmetro de configuração.

Arquivo de log local

Esse componente usa o mesmo arquivo de log do componente do núcleo do Greengrass.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

Para ver os registros desse componente

- Execute o comando a seguir no dispositivo principal para visualizar o arquivo de log desse componente em tempo real. Substitua */greengrass/v2* ou *C:\greengrass\v2* pelo caminho para a pasta AWS IoT Greengrass raiz.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

Changelog

A tabela a seguir descreve as alterações em cada versão do componente.

Version (Versão)	Alterações
2.0.11	Versão atualizada para a versão 2.12.0 do Greengrass nucleus.
2.0.10	Versão atualizada para a versão 2.11.0 do Greengrass nucleus.
2.0.9	Versão atualizada para a versão 2.10.0 do Greengrass nucleus.
2.0.8	Versão atualizada para a versão 2.9.0 do Greengrass nucleus.
2.0.7	Versão atualizada para a versão 2.8.0 do Greengrass nucleus.
2.0.6	Versão atualizada para a versão 2.7.0 do Greengrass nucleus.
2.0.5	Versão atualizada para a versão 2.6.0 do Greengrass nucleus.
2.0.4	Versão atualizada para a versão 2.5.0 do Greengrass nucleus.
2.0.3	Versão atualizada para a versão 2.4.0 do Greengrass nucleus.
2.0.2	Versão atualizada para a versão 2.3.0 do Greengrass nucleus.
2.0.1	Versão atualizada para a versão 2.2.0 do Greengrass nucleus.
2.0.0	Versão inicial.

Consulte também

- Execute um contêiner Docker

Conector Edge para Kinesis Video Streams

O conector de borda para o `aws.iot.EdgeConnectorForKVS` componente Kinesis Video Streams () lê os feeds de vídeo das câmeras locais e publica os streams no Kinesis Video Streams. Você pode configurar esse componente para ler feeds de vídeo de câmeras IP (Internet Protocol) usando o Real Time Streaming Protocol (RTSP). Em seguida, você pode configurar painéis no Amazon

Managed Grafana ou nos servidores locais do Grafana para monitorar e interagir com os streams de vídeo.

Você pode integrar esse componente AWS IoT TwinMaker para exibir e controlar fluxos de vídeo nos painéis da Grafana. AWS IoT TwinMaker é um AWS serviço que permite criar gêmeos digitais operacionais de sistemas físicos. Você pode usar AWS IoT TwinMaker para visualizar dados de sensores, câmeras e aplicativos corporativos para rastrear suas fábricas físicas, edifícios ou plantas industriais. Você também pode usar esses dados para monitorar operações, diagnosticar erros e reparar erros. Para obter mais informações, consulte O que é AWS IoT TwinMaker? no Guia do AWS IoT TwinMaker usuário.

Esse componente armazena sua configuração em AWS IoT SiteWise, que é um AWS serviço que modela e armazena dados industriais. Em AWS IoT SiteWise, os ativos representam objetos como dispositivos, equipamentos ou grupos de outros objetos. Para configurar e usar esse componente, você cria um AWS IoT SiteWise ativo para cada dispositivo principal do Greengrass e para cada câmera IP conectada a cada dispositivo principal. Cada ativo tem propriedades que você configura para controlar recursos, como transmissão ao vivo, upload sob demanda e armazenamento em cache local. Para especificar o URL de cada câmera, você cria um segredo AWS Secrets Manager que contém o URL da câmera. Se a câmera exigir autenticação, você também especifica um nome de usuário e uma senha na URL. Em seguida, você especifica esse segredo em uma propriedade de ativo para a câmera IP.

Esse componente carrega o stream de vídeo de cada câmera em um stream de vídeo do Kinesis. Você especifica o nome do stream de vídeo do Kinesis de destino na configuração do AWS IoT SiteWise ativo para cada câmera. Se o stream de vídeo do Kinesis não existir, esse componente o cria para você.

AWS IoT TwinMaker fornece um script que você pode executar para criar esses AWS IoT SiteWise ativos e segredos do Secrets Manager. Para obter mais informações sobre como criar esses recursos e como instalar, configurar e usar esse componente, consulte a integração de AWS IoT TwinMaker vídeo no Guia do AWS IoT TwinMaker usuário.

> **Note**
>
> O conector edge para o componente Kinesis Video Streams está disponível somente no seguinte: Regiões da AWS
>
> - Leste dos EUA (Norte da Virgínia)
> - Oeste dos EUA (Oregon)

- Europa (Frankfurt)
- Europa (Irlanda)
- Ásia-Pacífico (Singapura)

Tópicos

Versões

Esse componente tem as seguintes versões:

- 1,0.x

Tipo

Este componente é um componente genérico (`aws.greengrass.generic`). O núcleo do Greengrass executa os scripts do ciclo de vida do componente.

Para ter mais informações, consulte Tipos de componentes.

Sistema operacional

Esse componente pode ser instalado somente nos dispositivos principais do Linux.

Requisitos

Esse componente tem os seguintes requisitos:

- Você pode implantar esse componente somente em dispositivos de núcleo único, porque a configuração do componente deve ser exclusiva para cada dispositivo principal. Você não pode implantar esse componente em grupos de dispositivos principais.
- GStreamer 1.18.4 ou posterior instalado no dispositivo principal. Para obter mais informações, consulte Instalando o GStreamer.

  Em um dispositivo com `apt`, você pode executar os seguintes comandos para instalar o GStreamer.

  ```
  sudo apt install -y libgstreamer1.0-dev libgstreamer-plugins-base1.0-dev
   gstreamer1.0-plugins-base-apps
  sudo apt install -y gstreamer1.0-libav
  sudo apt install -y gstreamer1.0-plugins-bad gstreamer1.0-plugins-good gstreamer1.0-
  plugins-ugly gstreamer1.0-tools
  ```
- Um AWS IoT SiteWise ativo para cada dispositivo principal. Esse AWS IoT SiteWise ativo representa o dispositivo principal. Para obter mais informações sobre como criar esse ativo, consulte a integração de AWS IoT TwinMaker vídeo no Guia AWS IoT TwinMaker do usuário.
- Um AWS IoT SiteWise ativo para cada câmera IP que você conecta a cada dispositivo principal. Esses AWS IoT SiteWise ativos representam as câmeras que transmitem vídeo para cada dispositivo principal. Cada ativo da câmera deve estar associado ao ativo do dispositivo principal que se conecta à câmera. Os ativos da câmera têm propriedades que você pode configurar para especificar um stream de vídeo do Kinesis, um segredo de autenticação e parâmetros de streaming de vídeo. Para obter mais informações sobre como criar e configurar ativos de câmera, consulte a integração de AWS IoT TwinMaker vídeo no Guia AWS IoT TwinMaker do usuário.
- Um AWS Secrets Manager segredo para cada câmera IP. Esse segredo deve definir um par de valores-chave, onde está `RTSPStreamUrl` a chave e o valor é a URL da câmera. Se a câmera exigir autenticação, inclua o nome de usuário e a senha nesse URL. Você pode usar um script para criar um segredo ao criar os recursos necessários para esse componente. Para obter mais informações, consulte a integração de AWS IoT TwinMaker vídeo no Guia AWS IoT TwinMaker do usuário.

Você também pode usar o console e a API do Secrets Manager para criar segredos adicionais. Para obter mais informações, consulte Criar um segredo no Guia AWS Secrets Manager do usuário.

- A função de troca de tokens do Greengrass deve permitir as seguintes ações AWS Secrets Manager, AWS IoT SiteWise, e do Kinesis Video Streams, conforme mostrado no exemplo de política do IAM a seguir.

> **Note**
>
> Este exemplo de política permite que o dispositivo obtenha o valor dos segredos chamados **`IPCamera1Url`** **`IPCamera2Url`** e. Ao configurar cada câmera IP, você especifica um segredo que contém a URL dessa câmera. Se a câmera exigir autenticação, você também especifica um nome de usuário e uma senha na URL. A função de troca de tokens do dispositivo principal deve permitir o acesso ao segredo para que cada câmera IP se conecte.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "secretsmanager:GetSecretValue"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:secretsmanager:region:account-id:secret:IPCamera1Url",
        "arn:aws:secretsmanager:region:account-id:secret:IPCamera2Url"
      ]
    },
    {
      "Action": [
        "iotsitewise:BatchPutAssetPropertyValue",
        "iotsitewise:DescribeAsset",
        "iotsitewise:DescribeAssetModel",
        "iotsitewise:DescribeAssetProperty",
        "iotsitewise:GetAssetPropertyValue",
        "iotsitewise:ListAssetRelationships",
        "iotsitewise:ListAssets",
```

```
        "iotsitewise:ListAssociatedAssets",
        "kinesisvideo:CreateStream",
        "kinesisvideo:DescribeStream",
        "kinesisvideo:GetDataEndpoint",
        "kinesisvideo:PutMedia",
        "kinesisvideo:TagStream"
      ],
      "Effect": "Allow",
      "Resource": [
        "*"
      ]
    }
  ]
}
```

> **Note**
>
> Se você usar uma AWS Key Management Service chave gerenciada pelo cliente para criptografar segredos, a função do dispositivo também deverá permitir a `kms:Decrypt` ação.

Endpoints e portas

Esse componente deve ser capaz de realizar solicitações de saída para os seguintes endpoints e portas, além dos endpoints e portas necessários para a operação básica. Para ter mais informações, consulte Permitir o tráfego de dispositivos por meio de um proxy ou firewall.

Endpoint	Porta	Obrigatório	Descrição
`kinesisvideo.` *`region`*`.amazonaws.com`	443	Sim	Faça upload de dados para o Kinesis Video Streams.
`data.iotsitewise.` *`region`*`.amazonaws.com`	443	Sim	Publique metadados do stream

Endpoint	Porta	Obrigatório	Descrição
			de vídeo em AWS IoT SiteWise.
`secretsmanager. region.amazonaws.com`	443	Sim	Baixe os segredos do URL da câmera para o dispositivo principal.

Dependências

Quando você implanta um componente, AWS IoT Greengrass também implanta versões compatíveis de suas dependências. Isso significa que você deve atender aos requisitos do componente e de todas as suas dependências para implantá-lo com êxito. Esta seção lista as dependências das versões lançadas desse componente e as restrições de versão semântica que definem as versões do componente para cada dependência. Você também pode visualizar as dependências de cada versão do componente no AWS IoT Greengrass console. Na página de detalhes do componente, procure a lista de dependências.

A tabela a seguir lista as dependências das versões 1.0.0 a 1.0.5 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Serviço de troca de tokens	>=2.0.3	Rígido
Gerenciador de streams	>=2.0.9	Rígido

Para obter mais informações sobre dependências de componentes, consulte a referência da receita do componente.

Configuração

Esse componente fornece os seguintes parâmetros de configuração que você pode personalizar ao implantar o componente.

`SiteWiseAssetIdForHub`

O ID do AWS IoT SiteWise ativo que representa esse dispositivo principal. Para obter mais informações sobre como criar esse ativo e usá-lo para interagir com esse componente, consulte a integração de AWS IoT TwinMaker vídeo no Guia do AWS IoT TwinMaker usuário.

Example Exemplo: atualização da mesclagem de configurações

```
{
  "SiteWiseAssetIdForHub": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
}
```

Licenças

Esse componente inclui o seguinte software/licenciamento de terceiros:

- Quartz Job Scheduler/Licença Apache 2.0
- Ligações Java para o GStreamer 1.x/GNU Lesser General Public License v3.0

Uso

Para configurar e interagir com esse componente, você pode definir propriedades nos AWS IoT SiteWise ativos que representam o dispositivo principal e as câmeras IP às quais ele se conecta. Você também pode visualizar e interagir com streams de vídeo nos painéis da Grafana por meio de. AWS IoT TwinMaker Para obter mais informações, consulte a integração de AWS IoT TwinMaker vídeo no Guia AWS IoT TwinMaker do usuário.

Arquivo de log local

Esse componente usa o seguinte arquivo de log.

```
/greengrass/v2/logs/aws.iot.EdgeConnectorForKVS.log
```

Para ver os registros desse componente

- Execute o comando a seguir no dispositivo principal para visualizar o arquivo de log desse componente em tempo real. */greengrass/v2*Substitua pelo caminho para a pasta AWS IoT Greengrass raiz.

```
sudo tail -f /greengrass/v2/logs/aws.iot.EdgeConnectorForKVS.log
```

Changelog

A tabela a seguir descreve as alterações em cada versão do componente.

Version (Versão)	Alterações
1.0.5	Melhorias e correções de erros gerais.
1.0.4	Correções de erros e melhorias • Corrige um problema que fazia com que o upload ao vivo parasse.
1.0.3	Melhorias e correções de erros gerais.
1.0.1	Melhorias e correções de erros gerais.
1.0.0	Versão inicial.

Consulte também

- O que AWS IoT TwinMaker é no Guia do AWS IoT TwinMaker usuário
- AWS IoT TwinMaker integração de vídeo no Guia AWS IoT TwinMaker do usuário
- O que AWS IoT SiteWiseé no Guia do AWS IoT SiteWise usuário
- Atualização de valores de atributos no Guia AWS IoT SiteWise do usuário
- O que é o AWS Secrets Manager? no AWS Secrets Manager Guia do usuário
- Crie e gerencie segredos no Guia AWS Secrets Manager do usuário

CLI do Greengrass

O componente CLI do Greengrass (`aws.greengrass.Cli`) fornece uma interface de linha de comando local que você pode usar nos dispositivos principais para desenvolver e depurar componentes localmente. A CLI do Greengrass permite criar implantações locais e reiniciar componentes no dispositivo principal, por exemplo.

Você pode instalar esse componente ao instalar o software AWS IoT Greengrass Core. Para ter mais informações, consulte Tutorial: Conceitos básicos do AWS IoT Greengrass V2.

> **Important**
>
> Recomendamos que você use esse componente somente em ambientes de desenvolvimento, não em ambientes de produção. Esse componente fornece acesso a informações e operações que você normalmente não precisará em um ambiente de produção. Siga o princípio do menor privilégio implantando esse componente somente nos dispositivos principais onde você precisar dele.

Depois de instalar esse componente, execute o comando a seguir para ver sua documentação de ajuda. Quando esse componente é instalado, ele adiciona um link simbólico `greengrass-cli` à *`/greengrass/v2`*`/bin` pasta. Você pode executar a CLI do Greengrass a partir desse caminho ou adicioná-la à sua variável de `PATH` ambiente para ser executada `greengrass-cli` sem seu caminho absoluto.

Linux or Unix

```
/greengrass/v2/bin/greengrass-cli help
```

Windows

```
C:\greengrass\v2\bin\greengrass-cli help
```

O comando a seguir reinicia um componente chamado`com.example.HelloWorld`, por exemplo.

Linux or Unix

```
sudo /greengrass/v2/bin/greengrass-cli component restart --names
 "com.example.HelloWorld"
```

Windows

```
C:\greengrass\v2\bin\greengrass-cli component restart --names
 "com.example.HelloWorld"
```

Para ter mais informações, consulte Interface de linha de comando do Greengrass.

Tópicos

- Versões
- Tipo
- Sistema operacional
- Requisitos
- Dependências
- Configuração
- Arquivo de log local
- Changelog

Versões

Esse componente tem as seguintes versões:

- 2.12.x
- 2.11.x
- 2.10.x
- 2.9.x
- 2,8.x
- 2.7.x
- 2.6.x
- 2,5.x

- 2.4.x
- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

Tipo

Este componente é um componente de plug-in (`aws.greengrass.plugin`). O núcleo do Greengrass executa esse componente na mesma Java Virtual Machine (JVM) do núcleo. O núcleo reinicia quando você altera a versão desse componente no dispositivo principal.

Esse componente usa o mesmo arquivo de log do núcleo do Greengrass. Para ter mais informações, consulte Monitore AWS IoT Greengrass os registros.

Para ter mais informações, consulte Tipos de componentes.

Sistema operacional

Esse componente pode ser instalado em dispositivos principais que executam os seguintes sistemas operacionais:

- Linux
- Windows

Requisitos

Esse componente tem os seguintes requisitos:

- Você deve estar autorizado a usar a CLI do Greengrass para interagir com o software principal. AWS IoT Greengrass Faça o seguinte para usar a CLI do Greengrass:
  - Use o usuário do sistema que executa o software AWS IoT Greengrass Core.
  - Use um usuário com permissões de root ou administrativas. Nos dispositivos principais do Linux, você pode usar `sudo` para obter permissões de root.
  - Use um usuário do sistema que esteja em um grupo que você especifica nos parâmetros de `AuthorizedWindowsGroups` configuração `AuthorizedPosixGroups` ou ao implantar

o componente. Para obter mais informações, consulte Configuração do componente CLI do Greengrass.

- O componente Greengrass CLI tem suporte para ser executado em uma VPC.

Dependências

Quando você implanta um componente, AWS IoT Greengrass também implanta versões compatíveis de suas dependências. Isso significa que você deve atender aos requisitos do componente e de todas as suas dependências para implantá-lo com êxito. Esta seção lista as dependências das versões lançadas desse componente e as restrições de versão semântica que definem as versões do componente para cada dependência. Você também pode visualizar as dependências de cada versão do componente no AWS IoT Greengrass console. Na página de detalhes do componente, procure a lista de dependências.

2.12.0 – 2.12.6

A tabela a seguir lista as dependências da versão 2.12.0 a 2.12.6 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2,12,0 <2,13,0	Flexível

2.11.0 – 2.11.3

A tabela a seguir lista as dependências das versões 2.11.0 a 2.11.3 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2,11,0 <2,12,0	Flexível

2.10.0 – 2.10.3

A tabela a seguir lista as dependências das versões 2.10.0 a 2.10.3 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2,5.0 <2,11.0	Flexível

2.9.0 – 2.9.6

A tabela a seguir lista as dependências das versões 2.9.0 a 2.9.6 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.5.0 <2.10.0	Flexível

2.8.0 – 2.8.1

A tabela a seguir lista as dependências das versões 2.8.0 e 2.8.1 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2,5.0 <2,9,0	Flexível

2.7.0

A tabela a seguir lista as dependências da versão 2.7.0 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2,5.0 <2,8,0	Flexível

2.6.0

A tabela a seguir lista as dependências da versão 2.6.0 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2,5.0 <2,7,0	Flexível

2.5.0 – 2.5.6

A tabela a seguir lista as dependências das versões 2.5.0 a 2.5.6 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2,5.0 <2,6,0	Flexível

2.4.0

A tabela a seguir lista as dependências da versão 2.4.0 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.1.0 <2.5.0	Flexível

2.3.0

A tabela a seguir lista as dependências da versão 2.3.0 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.1.0 <2.4.0	Flexível

2.2.0

A tabela a seguir lista as dependências da versão 2.2.0 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.1.0 <2.3.0	Flexível

2.1.0

A tabela a seguir lista as dependências da versão 2.1.0 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.1.0 <2.2.0	Flexível

2.0.x

A tabela a seguir lista as dependências da versão 2.0.x desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.1.0	Flexível

> **Note**
>
> A versão mínima compatível do núcleo do Greengrass corresponde à versão de patch do componente CLI do Greengrass.

Para obter mais informações sobre dependências de componentes, consulte a referência da receita do componente.

Configuração

Esse componente fornece os seguintes parâmetros de configuração que você pode personalizar ao implantar o componente.

2.5.x - 2.12.x

`AuthorizedPosixGroups`

(Opcional) Uma string que contém uma lista separada por vírgulas dos grupos do sistema. Você autoriza esses grupos de sistema a usar a CLI do Greengrass para interagir com AWS IoT Greengrass o software principal. Você pode especificar nomes de grupos ou IDs de grupos. Por exemplo, `group1,1002,group3` autoriza três grupos do sistema (`group1`1002, `egroup3`) a usar a CLI do Greengrass.

Se você não especificar nenhum grupo para autorizar, poderá usar a CLI do Greengrass como `sudo` usuário raiz () ou como usuário do sistema que AWS IoT Greengrass executa o software Core.

`AuthorizedWindowsGroups`

(Opcional) Uma string que contém uma lista separada por vírgulas dos grupos do sistema. Você autoriza esses grupos de sistema a usar a CLI do Greengrass para interagir com AWS

IoT Greengrass o software principal. Você pode especificar nomes de grupos ou IDs de grupos. Por exemplo, `group1,1002,group3` autoriza três grupos do sistema (`group11002`, `egroup3`) a usar a CLI do Greengrass.

Se você não especificar nenhum grupo para autorizar, poderá usar a CLI do Greengrass como administrador ou como usuário do sistema que AWS IoT Greengrass executa o software Core.

Example Exemplo: atualização da mesclagem de configurações

O exemplo de configuração a seguir especifica a autorização de três grupos de sistema POSIX (`group11002`, `egroup3`) e dois grupos de usuários do Windows (`Device OperatorseQA Engineers`) para usar a CLI do Greengrass.

```
{
  "AuthorizedPosixGroups": "group1,1002,group3",
  "AuthorizedWindowsGroups": "Device Operators,QA Engineers"
}
```

2.4.x - 2.0.x

`AuthorizedPosixGroups`

(Opcional) Uma string que contém uma lista separada por vírgulas dos grupos do sistema. Você autoriza esses grupos de sistema a usar a CLI do Greengrass para interagir com AWS IoT Greengrass o software principal. Você pode especificar nomes de grupos ou IDs de grupos. Por exemplo, `group1,1002,group3` autoriza três grupos do sistema (`group11002`, `egroup3`) a usar a CLI do Greengrass.

Se você não especificar nenhum grupo para autorizar, poderá usar a CLI do Greengrass como `sudo` usuário raiz () ou como usuário do sistema que AWS IoT Greengrass executa o software Core.

Example Exemplo: atualização da mesclagem de configurações

O exemplo de configuração a seguir especifica a autorização de três grupos do sistema (`group11002`, `egroup3`) a usar a CLI do Greengrass.

```
{
  "AuthorizedPosixGroups": "group1,1002,group3"
}
```

Arquivo de log local

Esse componente usa o mesmo arquivo de log do componente do núcleo do Greengrass.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

Para ver os registros desse componente

- Execute o comando a seguir no dispositivo principal para visualizar o arquivo de log desse componente em tempo real. Substitua */greengrass/v2* ou *C:\greengrass\v2* pelo caminho para a pasta AWS IoT Greengrass raiz.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

Changelog

A tabela a seguir descreve as alterações em cada versão do componente.

Version (Versão)	Alterações
2.12.6	Versão atualizada para a versão 2.12.6 do Greengrass nucleus.
2.12.5	Versão atualizada para a versão 2.12.5 do Greengrass nucleus.
2.12.4	Versão atualizada para a versão 2.12.4 do Greengrass nucleus.

Version (Versão)	Alterações
2.12.3	**Warning** Essa versão não está mais disponível. As melhorias nesta versão estão disponíveis em versões posteriores desse componente. Versão atualizada para a versão 2.12.3 do Greengrass nucleus.
2.12.2	Versão atualizada para a versão 2.12.2 do Greengrass nucleus.
2.12.1	Versão atualizada para a versão 2.12.1 do Greengrass nucleus.
2.12.0	Versão atualizada para a versão 2.12.0 do Greengrass nucleus.
2.11.3	Versão atualizada para a versão 2.11.3 do Greengrass nucleus.
2.11.2	Versão atualizada para a versão 2.11.2 do Greengrass nucleus.
2.11.1	Versão atualizada para a versão 2.11.1 do Greengrass nucleus.
2.11.0	Novos atributos • Permite que você cancele uma implantação local. • Permite que você configure uma política de tratamento de falhas para uma implantação local. • Melhora os relatórios detalhados do status de implantação.
2.10.3	Versão atualizada para a versão 2.10.3 do Greengrass nucleus.
2.10.2	Versão atualizada para a versão 2.10.2 do Greengrass nucleus.
2.10.1	Versão atualizada para a versão 2.10.1 do Greengrass nucleus.
2.10.0	Versão atualizada para a versão 2.10.0 do Greengrass nucleus.
2.9.6	Versão atualizada para a versão 2.9.6 do Greengrass nucleus.
2.9.5	Versão atualizada para a versão 2.9.5 do Greengrass nucleus.

Version (Versão)	Alterações
2.9.4	Versão atualizada para a versão 2.9.4 do Greengrass nucleus.
2.9.3	Versão atualizada para a versão 2.9.3 do Greengrass nucleus.
2.9.2	Versão atualizada para a versão 2.9.2 do Greengrass nucleus.
2.9.1	Versão atualizada para a versão 2.9.1 do Greengrass nucleus.
2.9.0	Versão atualizada para a versão 2.9.0 do Greengrass nucleus.
2.8.1	Versão atualizada para a versão 2.8.1 do Greengrass nucleus.
2.8.0	Versão atualizada para a versão 2.8.0 do Greengrass nucleus.
2.7.0	Versão atualizada para a versão 2.7.0 do Greengrass nucleus.
2.6.0	Novos atributos • Adiciona suporte para componentes personalizados para chamar operações de comunicação entre processos (IPC) que a CLI do Greengrass usa. Você pode usar essas operações de IPC para gerenciar implantações locais, visualizar detalhes do componente e gerar uma senha que pode ser usada para entrar no console de depuração local. Para obter mais informações, consulte IPC: gerenciar implantações e componentes locais. Correções de erros e melhorias • Pequenas correções e melhorias adicionais.
2.5.6	Versão atualizada para a versão 2.5.6 do Greengrass nucleus.
2.5.5	Versão atualizada para a versão 2.5.5 do Greengrass nucleus.
2.5.4	Versão atualizada para a versão 2.5.4 do Greengrass nucleus.
2.5.3	Versão atualizada para a versão 2.5.3 do Greengrass nucleus.
2.5.2	Versão atualizada para a versão 2.5.2 do Greengrass nucleus.
2.5.1	Versão atualizada para a versão 2.5.1 do Greengrass nucleus.

Version (Versão)	Alterações
2.5.0	Novos atributos • Adiciona suporte para dispositivos principais que executam o Windows. • Adiciona o novo parâmetro de `AuthorizedWindowsGroups` configuração que você pode especificar para autorizar grupos do sistema a usar a CLI do Greengrass em dispositivos Windows. • Adiciona o `windowsUser` parâmetro para implantações locais. Você pode usar esse parâmetro para especificar o usuário a ser usado para executar componentes em um dispositivo principal do Windows.
2.4.0	Novos atributos • Adiciona suporte aos limites de recursos do sistema. Ao criar uma implantação local, você pode configurar a quantidade máxima de uso de CPU e RAM que os processos de cada componente podem usar no dispositivo principal. Para obter mais informações, consulte Configura r limites de recursos do sistema para componentes e o comando deployment create.
2.3.0	Versão atualizada para a versão 2.3.0 do Greengrass nucleus.
2.2.0	Versão atualizada para a versão 2.2.0 do Greengrass nucleus.
2.1.0	Versão atualizada para a versão 2.1.0 do Greengrass nucleus.
2.0.5	Versão atualizada para a versão 2.0.5 do Greengrass nucleus.
2.0.4	Versão atualizada para a versão 2.0.4 do Greengrass nucleus.
2.0.3	Versão inicial.

Detector IP

O componente detector de IP (`aws.greengrass.clientdevices.IPDetector`) faz o seguinte:

- Monitora as informações de conectividade de rede do dispositivo principal do Greengrass. Essas informações incluem os endpoints de rede do dispositivo principal e a porta em que um agente MQTT opera.

- Atualiza as informações de conectividade do dispositivo principal no serviço de AWS IoT Greengrass nuvem.

Os dispositivos clientes podem usar o Greengrass Cloud Discovery para recuperar as informações de conectividade dos dispositivos principais associados. Em seguida, os dispositivos cliente podem tentar se conectar a cada dispositivo principal até que se conectem com sucesso.

> **Note**
>
> Os dispositivos cliente são dispositivos IoT locais que se conectam a um dispositivo central do Greengrass para enviar mensagens e dados MQTT para processamento. Para ter mais informações, consulte Interaja com dispositivos IoT locais.

O componente detector de IP substitui as informações de conectividade existentes de um dispositivo principal pelas informações que ele detecta. Como esse componente remove as informações existentes, você pode usar o componente detector de IP ou gerenciar manualmente as informações de conectividade.

> **Note**
>
> O componente detector de IP detecta somente endereços IPv4.

Tópicos

Versões

Esse componente tem as seguintes versões:

- 2.1.x
- 2.0.x

Tipo

Este componente é um componente de plug-in (`aws.greengrass.plugin`). O núcleo do Greengrass executa esse componente na mesma Java Virtual Machine (JVM) do núcleo. O núcleo reinicia quando você altera a versão desse componente no dispositivo principal.

Esse componente usa o mesmo arquivo de log do núcleo do Greengrass. Para ter mais informações, consulte Monitore AWS IoT Greengrass os registros.

Para ter mais informações, consulte Tipos de componentes.

Sistema operacional

Esse componente pode ser instalado em dispositivos principais que executam os seguintes sistemas operacionais:

- Linux
- Windows

Requisitos

Esse componente tem os seguintes requisitos:

- A função de serviço do Greengrass deve estar associada às suas Conta da AWS e permitir as permissões `iot:GetThingShadow` e. `iot:UpdateThingShadow`
- A AWS IoT política do dispositivo principal deve permitir a `greengrass:UpdateConnectivityInfo` permissão. Para obter mais informações, consulte Políticas do AWS IoT para operações de plano de dados e AWS IoTPolítica mínima para oferecer suporte aos dispositivos do cliente.
- Se você configurar o componente intermediário MQTT do dispositivo principal para usar uma porta diferente da porta padrão 8883, deverá usar o detector de IP v2.1.0 ou posterior. Configure-o para relatar a porta em que o agente opera.

- Se você tiver uma configuração de rede complexa, o componente detector de IP talvez não consiga identificar os endpoints nos quais os dispositivos cliente podem se conectar ao dispositivo principal. Se o componente detector de IP não puder gerenciar os endpoints, você deverá gerenciar manualmente os endpoints do dispositivo principal. Por exemplo, se o dispositivo principal estiver atrás de um roteador que encaminha a porta do agente MQTT para ele, você deverá especificar o endereço IP do roteador como um endpoint para o dispositivo principal. Para ter mais informações, consulte Gerencie os endpoints principais do dispositivo.
- O componente detector de IP tem suporte para ser executado em uma VPC.

Dependências

Quando você implanta um componente, AWS IoT Greengrass também implanta versões compatíveis de suas dependências. Isso significa que você deve atender aos requisitos do componente e de todas as suas dependências para implantá-lo com êxito. Esta seção lista as dependências das versões lançadas desse componente e as restrições de versão semântica que definem as versões dos componentes para cada dependência. Você também pode visualizar as dependências de cada versão do componente no AWS IoT Greengrass console. Na página de detalhes do componente, procure a lista de dependências.

2.1.8 – 2.1.9

A tabela a seguir lista as dependências das versões 2.1.8 e 2.1.9 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.2.0 <2.13.0	Flexível

2.1.7

A tabela a seguir lista as dependências da versão 2.1.7 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.2.0 <2.12.0	Flexível

2.1.6

A tabela a seguir lista as dependências da versão 2.1.6 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.2.0 <2.11.0	Flexível

2.1.5

A tabela a seguir lista as dependências da versão 2.1.5 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.2.0 <2.10.0	Flexível

2.1.4

A tabela a seguir lista as dependências da versão 2.1.4 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.2.0 <2.9.0	Flexível

2.1.3

A tabela a seguir lista as dependências da versão 2.1.3 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.2.0 <2.8.0	Flexível

2.1.2

A tabela a seguir lista as dependências da versão 2.1.2 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.2.0 <2.7.0	Flexível

2.1.1

A tabela a seguir lista as dependências da versão 2.1.1 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.2.0 <2.6.0	Flexível

2.1.0 and 2.0.2

A tabela a seguir lista as dependências das versões 2.1.0 e 2.0.2 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.2.0 <2.5.0	Flexível

2.0.1

A tabela a seguir lista as dependências da versão 2.0.1 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.2.0 <2.4.0	Flexível

2.0.0

A tabela a seguir lista as dependências da versão 2.0.0 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.2.0 <2.3.0	Flexível

Para obter mais informações sobre dependências de componentes, consulte a referência da receita do componente.

Configuração

Esse componente fornece os seguintes parâmetros de configuração que você pode personalizar ao implantar o componente.

2.1.x

`defaultPort`

(Opcional) A porta do agente MQTT para relatar quando esse componente detecta endereços IP. Você deve especificar esse parâmetro se configurar o agente MQTT para usar uma porta diferente da porta padrão 8883.

Padrão: `8883`

`includeIPv4LoopbackAddrs`

(Opcional) Você pode ativar essa opção para detectar e relatar endereços de loopback IPv4. Esses são endereços IP, como, por exemplo`localhost`, onde um dispositivo pode se comunicar consigo mesmo. Use essa opção em ambientes de teste em que o dispositivo principal e o dispositivo cliente são executados no mesmo sistema.

Padrão: `false`

`includeIPv4LinkLocalAddrs`

(Opcional) Você pode ativar essa opção para detectar e relatar endereços locais de links IPv4. Use essa opção se a rede do dispositivo principal não tiver o Dynamic Host Configuration Protocol (DHCP) ou endereços IP atribuídos estaticamente.

Padrão: `false`

2.0.x

`includeIPv4LoopbackAddrs`

(Opcional) Você pode ativar essa opção para detectar e relatar endereços de loopback IPv4. Esses são endereços IP, como, por exemplo`localhost`, onde um dispositivo pode

se comunicar consigo mesmo. Use essa opção em ambientes de teste em que o dispositivo principal e o dispositivo cliente são executados no mesmo sistema.

Padrão: `false`

`includeIPv4LinkLocalAddrs`

(Opcional) Você pode ativar essa opção para detectar e relatar endereços locais de links IPv4. Use essa opção se a rede do dispositivo principal não tiver o Dynamic Host Configuration Protocol (DHCP) ou endereços IP atribuídos estaticamente.

Padrão: `false`

Arquivo de log local

Esse componente usa o mesmo arquivo de log do componente do núcleo do Greengrass.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

Para ver os registros desse componente

- Execute o comando a seguir no dispositivo principal para visualizar o arquivo de log desse componente em tempo real. Substitua */greengrass/v2* ou *C:\greengrass\v2* pelo caminho para a pasta AWS IoT Greengrass raiz.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

Changelog

A tabela a seguir descreve as alterações em cada versão do componente.

Version (Versão)	Alterações
2.1.9	Correções de erros e melhorias • Ajusta a etapa de aquisição do IP para enviar somente registros no nível do registro de depuração.
2.1.8	Versão atualizada para a versão 2.12.0 do Greengrass nucleus.
2.1.7	Versão atualizada para a versão 2.11.0 do Greengrass nucleus.
2.1.6	Versão atualizada para a versão 2.10.0 do Greengrass nucleus.
2.1.5	Versão atualizada para a versão 2.9.0 do Greengrass nucleus.
2.1.4	Versão atualizada para a versão 2.8.0 do Greengrass nucleus.
2.1.3	Versão atualizada para a versão 2.7.0 do Greengrass nucleus.
2.1.2	Correções de erros e melhorias • Melhora as mensagens de erro que esse componente registra em determinados cenários. • Versão atualizada para a versão 2.6.0 do Greengrass nucleus.
2.1.1	Versão atualizada para a versão 2.5.0 do Greengrass nucleus.
2.1.0	Melhorias • Adiciona o `defaultPort` parâmetro, que permite usar uma porta de agente MQTT não padrão. • Atualizações para tornar as mensagens de registro mais claras.
2.0.2	Versão atualizada para a versão 2.4.0 do Greengrass nucleus.
2.0.1	Versão atualizada para a versão 2.3.0 do Greengrass nucleus.
2.0.0	Versão inicial.

Firehose

O componente Firehose (`aws.greengrass.KinesisFirehose`) publica dados por meio de fluxos de entrega do Amazon Data Firehose para destinos, como Amazon S3, Amazon Redshift e Amazon Service. OpenSearch Para obter mais informações, consulte O que é o Amazon Data Firehose? no Guia do desenvolvedor do Amazon Data Firehose.

Para publicar em um stream de entrega do Kinesis com esse componente, publique uma mensagem em um tópico em que esse componente se inscreva. Por padrão, esse componente se inscreve nos tópicos de `kinesisfirehose/message` publicação/assinatura `kinesisfirehose/message/binary/#` locais. Você pode especificar outros tópicos, incluindo tópicos do AWS IoT Core MQTT, ao implantar esse componente.

> **Note**
>
> Esse componente fornece funcionalidade semelhante ao conector Firehose na AWS IoT Greengrass V1. Para obter mais informações, consulte o conector Firehose no Guia do desenvolvedor AWS IoT Greengrass V1.

Tópicos

Versões

Esse componente tem as seguintes versões:

- 2.1.x
- 2.0.x

Tipo

Esse componente é um componente Lambda () `aws.greengrass.lambda`. O núcleo do Greengrass executa a função Lambda desse componente usando o componente lançador Lambda.

Para ter mais informações, consulte Tipos de componentes.

Sistema operacional

Esse componente pode ser instalado somente nos dispositivos principais do Linux.

Requisitos

Esse componente tem os seguintes requisitos:

- Seu dispositivo principal deve atender aos requisitos para executar as funções do Lambda. Se você quiser que o dispositivo principal execute funções Lambda em contêineres, o dispositivo deve atender aos requisitos para fazer isso. Para ter mais informações, consulte Requisitos da função do Lambda.
- Python versão 3.7 instalado no dispositivo de núcleo e adicionado à variável de ambiente PATH.
- A função de dispositivo do Greengrass deve permitir as `firehose:PutRecordBatch` ações `firehose:PutRecord` e, conforme mostrado no exemplo de política do IAM a seguir.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "firehose:PutRecord",
        "firehose:PutRecordBatch"
      ],
      "Effect": "Allow",
      "Resource": [
```

```
        "arn:aws:firehose:region:account-id:deliverystream/stream-name"
      ]
    }
  ]
}
```

Você pode substituir dinamicamente o fluxo de entrega padrão na carga da mensagem de entrada desse componente. Se seu aplicativo usa esse recurso, a política do IAM deve incluir todos os fluxos de destino como recursos. Você pode conceder acesso granular ou condicional aos recursos (por exemplo, usando um esquema de nomeação * curinga).

- Para receber dados de saída desse componente, você deve mesclar a seguinte atualização de configuração para o componente antigo do roteador de assinatura (`aws.greengrass.LegacySubscriptionRouter`) ao implantar esse componente. Essa configuração especifica o tópico em que esse componente publica respostas.

  Legacy subscription router v2.1.x

  ```
  {
    "subscriptions": {
      "aws-greengrass-kinesisfirehose": {
        "id": "aws-greengrass-kinesisfirehose",
        "source": "component:aws.greengrass.KinesisFirehose",
        "subject": "kinesisfirehose/message/status",
        "target": "cloud"
      }
    }
  }
  ```

  Legacy subscription router v2.0.x

  ```
  {
    "subscriptions": {
      "aws-greengrass-kinesisfirehose": {
        "id": "aws-greengrass-kinesisfirehose",
        "source": "arn:aws:lambda:region:aws:function:aws-greengrass-
  kinesisfirehose:version",
        "subject": "kinesisfirehose/message/status",
        "target": "cloud"
      }
    }
  }
  ```

- Substitua a *região* pela Região da AWS que você usa.
- *Substitua a versão pela versão da função Lambda que esse componente executa.* Para encontrar a versão da função Lambda, você deve visualizar a receita da versão desse componente que você deseja implantar. Abra a página de detalhes desse componente no AWS IoT Greengrass console e procure o par de valores-chave da função Lambda. Esse par de valores-chave contém o nome e a versão da função Lambda.

> **Important**
>
> Você deve atualizar a versão da função Lambda no roteador de assinatura legado sempre que implantar esse componente. Isso garante que você use a versão correta da função Lambda para a versão do componente que você implanta.

Para ter mais informações, consulte Criar implantações.

- O componente Firehose pode ser executado em uma VPC. Para implantar esse componente em uma VPC, é necessário o seguinte.
  - O componente Firehose deve ter conectividade com a `firehose.region.amazonaws.com` qual tenha o VPC endpoint de. `com.amazonaws.region.kinesis-firehose`

Endpoints e portas

Esse componente deve ser capaz de realizar solicitações de saída para os seguintes endpoints e portas, além dos endpoints e portas necessários para a operação básica. Para ter mais informações, consulte Permitir o tráfego de dispositivos por meio de um proxy ou firewall.

Endpoint	Porta	Obrigatório	Descrição
`firehose.` *region*`.amazonaws.com`	443	Sim	Faça upload de dados para o Firehose.

Dependências

Quando você implanta um componente, AWS IoT Greengrass também implanta versões compatíveis de suas dependências. Isso significa que você deve atender aos requisitos do componente e de todas as suas dependências para implantá-lo com êxito. Esta seção lista as dependências das versões lançadas desse componente e as restrições de versão semântica que definem as versões dos componentes para cada dependência. Você também pode visualizar as dependências de cada versão do componente no AWS IoT Greengrass console. Na página de detalhes do componente, procure a lista de dependências.

2.1.7

A tabela a seguir lista as dependências da versão 2.1.7 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.13.0	Rígido
Lançador Lambda	^2.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	^2.0.0	Flexível
Serviço de troca de tokens	^2.0.0	Rígido

2.1.6

A tabela a seguir lista as dependências da versão 2.1.6 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.12.0	Rígido
Lançador Lambda	^2.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	^2.0.0	Flexível
Serviço de troca de tokens	^2.0.0	Rígido

2.1.5

A tabela a seguir lista as dependências da versão 2.1.5 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.11.0	Rígido
Lançador Lambda	^2.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	^2.0.0	Flexível
Serviço de troca de tokens	^2.0.0	Rígido

2.1.4

A tabela a seguir lista as dependências da versão 2.1.4 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.10.0	Rígido
Lançador Lambda	^2.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	^2.0.0	Flexível
Serviço de troca de tokens	^2.0.0	Rígido

2.1.3

A tabela a seguir lista as dependências da versão 2.1.3 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.9.0	Rígido

Dependência	Versões compatíveis	Tipo de dependência
Lançador Lambda	^2.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	^2.0.0	Flexível
Serviço de troca de tokens	^2.0.0	Rígido

2.1.2

A tabela a seguir lista as dependências da versão 2.1.2 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.8.0	Rígido
Lançador Lambda	^2.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	^2.0.0	Flexível
Serviço de troca de tokens	^2.0.0	Rígido

2.1.1

A tabela a seguir lista as dependências da versão 2.1.1 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.7.0	Rígido
Lançador Lambda	^2.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	^2.0.0	Flexível
Serviço de troca de tokens	^2.0.0	Rígido

2.0.8 - 2.1.0

A tabela a seguir lista as dependências das versões 2.0.8 e 2.1.0 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.6.0	Rígido
Lançador Lambda	^2.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	^2.0.0	Flexível
Serviço de troca de tokens	^2.0.0	Rígido

2.0.7

A tabela a seguir lista as dependências da versão 2.0.7 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.5.0	Rígido
Lançador Lambda	^2.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	^2.0.0	Flexível
Serviço de troca de tokens	^2.0.0	Rígido

2.0.6

A tabela a seguir lista as dependências da versão 2.0.6 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.4.0	Rígido

Dependência	Versões compatíveis	Tipo de dependência
Lançador Lambda	^2.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	^2.0.0	Flexível
Serviço de troca de tokens	^2.0.0	Rígido

2.0.5

A tabela a seguir lista as dependências da versão 2.0.5 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.3.0	Rígido
Lançador Lambda	^2.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	^2.0.0	Flexível
Serviço de troca de tokens	^2.0.0	Rígido

2.0.4

A tabela a seguir lista as dependências da versão 2.0.4 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.2.0	Rígido
Lançador Lambda	^2.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	^2.0.0	Flexível
Serviço de troca de tokens	^2.0.0	Rígido

2.0.3

A tabela a seguir lista as dependências da versão 2.0.3 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.3 <2.1.0	Rígido
Lançador Lambda	>=1.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	>=1.0.0	Flexível
Serviço de troca de tokens	>=1.0.0	Rígido

Para obter mais informações sobre dependências de componentes, consulte a referência da receita do componente.

Configuração

Esse componente fornece os seguintes parâmetros de configuração que você pode personalizar ao implantar o componente.

> **Note**
>
> A configuração padrão desse componente inclui parâmetros da função Lambda. Recomendamos que você edite somente os parâmetros a seguir para configurar esse componente em seus dispositivos.

`lambdaParams`

Um objeto que contém os parâmetros da função Lambda desse componente. Esse objeto contém as seguintes informações:

`EnvironmentVariables`

Um objeto que contém os parâmetros da função Lambda. Esse objeto contém as seguintes informações:

`DEFAULT_DELIVERY_STREAM_ARN`

O ARN do stream de entrega padrão do Firehose para o qual o componente envia dados. Você pode substituir o fluxo de destino pela `delivery_stream_arn` propriedade na carga da mensagem de entrada.

> **Note**
>
> A função principal do dispositivo deve permitir as ações necessárias em todos os fluxos de entrega de destino. Para ter mais informações, consulte Requisitos.

`PUBLISH_INTERVAL`

(Opcional) O número máximo de segundos de espera antes que o componente publique os dados em lote no Firehose. Para configurar o componente para publicar métricas à medida que as recebe, ou seja, sem agrupamento em lotes, especifique`0`.

Esse valor pode ser de no máximo 900 segundos.

Padrão: 10 segundos

`DELIVERY_STREAM_QUEUE_SIZE`

(Opcional) O número máximo de registros a serem retidos na memória antes que o componente rejeite novos registros para o mesmo fluxo de entrega.

Esse valor deve ter pelo menos 2.000 registros.

Padrão: 5.000 registros

`containerMode`

(Opcional) O modo de conteinerização desse componente. Escolha uma das seguintes opções:

- `NoContainer`— O componente não é executado em um ambiente de execução isolado.
- `GreengrassContainer`— O componente é executado em um ambiente de execução isolado dentro do AWS IoT Greengrass contêiner.

Padrão: `GreengrassContainer`

`containerParams`

(Opcional) Um objeto que contém os parâmetros do contêiner desse componente. O componente usa esses parâmetros se você especificar `GreengrassContainer` para`containerMode`.

Esse objeto contém as seguintes informações:

`memorySize`

(Opcional) A quantidade de memória (em kilobytes) a ser alocada para o componente.

O padrão é 64 MB (65.535 KB).

`pubsubTopics`

(Opcional) Um objeto que contém os tópicos em que o componente se inscreve para receber mensagens. Você pode especificar cada tópico e se o componente se inscreve nos tópicos do MQTT AWS IoT Core ou nos tópicos locais de publicação/assinatura.

Esse objeto contém as seguintes informações:

`0`— Este é um índice de matriz como uma string.

Um objeto que contém as seguintes informações:

`type`

(Opcional) O tipo de mensagem de publicação/assinatura que esse componente usa para assinar mensagens. Escolha uma das seguintes opções:

- `PUB_SUB` – Assine mensagens locais de publicar/assinar. Se você escolher essa opção, o tópico não poderá conter curingas MQTT. Para obter mais informações sobre como enviar mensagens do componente personalizado ao especificar essa opção, consulte Publique/assine mensagens locais.
- `IOT_CORE`— Assine as mensagens do AWS IoT Core MQTT. Se você escolher essa opção, o tópico poderá conter curingas do MQTT. Para obter mais informações sobre como enviar mensagens de componentes personalizados ao especificar essa opção, consulte Publique/assine mensagens MQTT AWS IoT Core.

Padrão: `PUB_SUB`

`topic`

(Opcional) O tópico no qual o componente se inscreve para receber mensagens. Se você especificar `IotCore` para `type`, poderá usar curingas MQTT (+e#) neste tópico.

Example Exemplo: atualização da mesclagem de configuração (modo contêiner)

```
{
  "lambdaExecutionParameters": {
```

```
    "EnvironmentVariables": {
      "DEFAULT_DELIVERY_STREAM_ARN": "arn:aws:firehose:us-
west-2:123456789012:deliverystream/mystream"
    }
  },
  "containerMode": "GreengrassContainer"
}
```

Example Exemplo: atualização de mesclagem de configuração (sem modo de contêiner)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "DEFAULT_DELIVERY_STREAM_ARN": "arn:aws:firehose:us-
west-2:123456789012:deliverystream/mystream"
    }
  },
  "containerMode": "NoContainer"
}
```

Dados de entrada

Esse componente aceita conteúdo de stream nos tópicos a seguir e envia o conteúdo para o stream de entrega de destino. O componente aceita dois tipos de dados de entrada:

- Dados JSON no tópico `kinesisfirehose/message`.
- Dados binários no tópico `kinesisfirehose/message/binary/#`.

Tópico padrão para dados JSON (publicação/assinatura local): `kinesisfirehose/message`

A mensagem aceita as seguintes propriedades. As mensagens de entrada devem estar no formato JSON.

`request`

Os dados a serem enviados para o fluxo de entrega e o fluxo de entrega de destino, se diferentes do fluxo padrão.

Tipo: `object` que contém as seguintes informações:

`data`

Os dados a serem enviados para o fluxo de entrega.

Tipo: `string`

`delivery_stream_arn`

(Opcional) O ARN do stream de entrega do Firehose de destino. Especifique essa propriedade para substituir o fluxo de entrega padrão.

Tipo: `string`

`id`

Um ID arbitrário para a solicitação. Use essa propriedade para mapear uma solicitação de entrada para uma resposta de saída. Quando você especifica essa propriedade, o componente define a `id` propriedade no objeto de resposta com esse valor.

Tipo: `string`

Example Exemplo de entrada

```
{
  "request": {
    "delivery_stream_arn": "arn:aws:firehose:region:account-id:deliverystream/
stream2-name",
    "data": "Data to send to the delivery stream."
  },
  "id": "request123"
}
```

Tópico padrão para dados binários (publicação/assinatura local): `kinesisfirehose/message/binary/#`

Use esse tópico para enviar uma mensagem que contenha dados binários. O componente não analisa dados binários. O componente transmite os dados como estão.

Para mapear a solicitação de entrada para uma resposta de saída, substitua o curinga # na mensagem do tópico por um ID de solicitação arbitrário. Por exemplo, se você publicar uma mensagem no `kinesisfirehose/message/binary/request123`, a propriedade `id` no objeto de resposta será definida como `request123`.

Se você não deseja mapear uma solicitação para uma resposta, é possível publicar suas mensagens em `kinesisfirehose/message/binary/`. Certifique-se de incluir a barra final ()/.

Dados de saída

Por padrão, esse componente publica respostas como dados de saída no seguinte tópico do MQTT. Você deve especificar esse tópico conforme a `subject` configuração do componente antigo do roteador de assinatura. Para obter mais informações sobre como assinar mensagens sobre esse tópico em seus componentes personalizados, consultePublique/assine mensagens MQTT AWS IoT Core.

Tópico padrão (AWS IoT Core MQTT): `kinesisfirehose/message/status`

Example Exemplo de saída

A resposta contém o status de cada registro de dados enviado no lote.

```
{
  "response": [
    {
      "ErrorCode": "error",
      "ErrorMessage": "test error",
      "id": "request123",
      "status": "fail"
    },
    {
      "firehose_record_id": "xyz2",
      "id": "request456",
      "status": "success"
    },
    {
      "firehose_record_id": "xyz3",
      "id": "request890",
      "status": "success"
    }
  ]
}
```

> **Note**
>
> Se o componente detectar um erro que possa ser repetido, como um erro de conexão, ele tentará publicar novamente no próximo lote.

Arquivo de log local

Esse componente usa o seguinte arquivo de log.

```
/greengrass/v2/logs/aws.greengrass.KinesisFirehose.log
```

Para ver os registros desse componente

- Execute o comando a seguir no dispositivo principal para visualizar o arquivo de log desse componente em tempo real. */greengrass/v2*Substitua pelo caminho para a pasta AWS IoT Greengrass raiz.

  ```
  sudo tail -f /greengrass/v2/logs/aws.greengrass.KinesisFirehose.log
  ```

Licenças

Esse componente inclui o seguinte software/licenciamento de terceiros:

- AWS SDK for Python (Boto3)/Licença Apache 2.0
- botocore/Licença Apache 2.0
- dateutil/Licença PSF
- docutils/Licença BSD, GNU Licença pública geral (GPL), Licença Python Software Foundation, Domínio público
- jmespath/Licença MIT
- s3transfer/Licença Apache 2.0
- urllib3/Licença MIT

Esse componente é lançado de acordo com o Contrato de Licença de Software Principal do Greengrass.

Changelog

A tabela a seguir descreve as alterações em cada versão do componente.

Version (Versão)	Alterações
2.1.7	Versão atualizada para a versão 2.12.0 do Greengrass nucleus.
2.1.6	Versão atualizada para a versão 2.11.0 do Greengrass nucleus.
2.1.5	Versão atualizada para a versão 2.10.0 do Greengrass nucleus.
2.1.4	Versão atualizada para a versão 2.9.0 do Greengrass nucleus.
2.1.3	Versão atualizada para a versão 2.8.0 do Greengrass nucleus.
2.1.2	Versão atualizada para a versão 2.7.0 do Greengrass nucleus.
2.1.1	Versão atualizada para a versão 2.6.0 do Greengrass nucleus.
2.1.0	Novos atributos • Adiciona suporte para configurações de proxy de rede HTTPS. Para obter mais informações, consulte Permita que o dispositivo principal confie em um proxy HTTPS e Conectar-se à porta 443 ou por meio de um proxy de rede.
2.0.8	Versão atualizada para a versão 2.5.0 do Greengrass nucleus.
2.0.7	Versão atualizada para a versão 2.4.0 do Greengrass nucleus.
2.0.6	Versão atualizada para a versão 2.3.0 do Greengrass nucleus.
2.0.5	Versão atualizada para a versão 2.2.0 do Greengrass nucleus.
2.0.4	Versão atualizada para a versão 2.1.0 do Greengrass nucleus.
2.0.3	Versão inicial.

Consulte também

- O que é o Amazon Data Firehose? no Guia do desenvolvedor do Amazon Data Firehose

Lançador Lambda

O componente Lambda launcher (`aws.greengrass.LambdaLauncher`) inicia e interrompe AWS Lambda as funções nos AWS IoT Greengrass dispositivos principais. Esse componente também configura qualquer conteinerização e executa processos como os usuários que você especificar.

> **Note**
>
> Quando você implanta um componente da função Lambda em um dispositivo principal, a implantação também inclui esse componente. Para ter mais informações, consulte Executar AWS Lambda funções.

Tópicos

Versões

Esse componente tem as seguintes versões:

- 2.0.x

Tipo

Este componente é um componente genérico (`aws.greengrass.generic`). O núcleo do Greengrass executa os scripts do ciclo de vida do componente.

Para ter mais informações, consulte Tipos de componentes.

Sistema operacional

Esse componente pode ser instalado somente nos dispositivos principais do Linux.

Requisitos

Esse componente tem os seguintes requisitos:

- Seu dispositivo principal deve atender aos requisitos para executar as funções do Lambda. Se você quiser que o dispositivo principal execute funções Lambda em contêineres, o dispositivo deve atender aos requisitos para fazer isso. Para ter mais informações, consulte Requisitos da função do Lambda.
- O componente Lambda launcher é compatível com a execução em uma VPC.

Dependências

Quando você implanta um componente, AWS IoT Greengrass também implanta versões compatíveis de suas dependências. Isso significa que você deve atender aos requisitos do componente e de todas as suas dependências para implantá-lo com êxito. Esta seção lista as dependências das versões lançadas desse componente e as restrições de versão semântica que definem as versões dos componentes para cada dependência. Você também pode visualizar as dependências de cada versão do componente no AWS IoT Greengrass console. Na página de detalhes do componente, procure a lista de dependências.

2.0.11 – 2.0.13

A tabela a seguir lista as dependências das versões 2.0.11 a 2.0.13 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Gerente do Lambda	>=2.0.0 <2.4.0	Rígido

2.0.9 – 2.0.10

A tabela a seguir lista as dependências das versões 2.0.9 a 2.0.10 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Gerente do Lambda	>=2.0.0 <2.3.0	Rígido

2.0.4 - 2.0.8

A tabela a seguir lista as dependências das versões 2.0.4 a 2.0.8 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Gerente do Lambda	>=2.0.0 <2.2.0	Rígido

2.0.3

A tabela a seguir lista as dependências da versão 2.0.3 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Gerente do Lambda	>=2.0.3 <2.1.0	Rígido

Para obter mais informações sobre dependências de componentes, consulte a referência da receita do componente.

Configuração

Esse componente não tem nenhum parâmetro de configuração.

Arquivo de log local

Esse componente usa o seguinte arquivo de log.

```
/greengrass/v2/logs/lambdaFunctionComponentName.log
```

Para ver os registros desse componente

- Execute o comando a seguir no dispositivo principal para visualizar o arquivo de log desse componente em tempo real. */greengrass/v2*Substitua pelo caminho para a pasta AWS IoT

Greengrass raiz e substitua *lambdaFunctionComponentName pelo nome* do componente da função Lambda que esse componente inicia.

```
sudo tail -f /greengrass/v2/logs/lambdaFunctionComponentName.log
```

Changelog

A tabela a seguir descreve as alterações em cada versão do componente.

Version (Versão)	Alterações
2.0.13	Correções de erros e melhorias Melhorias e correções de erros gerais.
2.0.12	Correções de erros e melhorias Corrige um problema em que o lançador Lambda poderia gerar um erro se o processo anterior não fosse interrompido corretamente.
2.0.11	Support para o Lambda manager 2.3.0.
2.0.10	Correções de erros e melhorias • Melhorias e correções de erros gerais.
2.0.9	Versão atualizada para a versão 2.5.0 do Greengrass nucleus.
2.0.8	Versão atualizada para a versão 2.4.0 do Greengrass nucleus.
2.0.7	Versão atualizada para a versão 2.3.0 do Greengrass nucleus.
2.0.6	Melhorias no desempenho geral e correções de erros.
2.0.4	Correções de erros e melhorias • Corrige um problema em que o componente não passa corretamente `AddGroupOwner` para o contêiner da função Lambda.
2.0.3	Versão inicial.

Gerente do Lambda

O componente Lambda Manager (`aws.greengrass.LambdaManager`) gerencia itens de trabalho e comunicação entre processos para AWS Lambda funções que são executadas no dispositivo principal do Greengrass.

> **Note**
>
> Quando você implanta um componente da função Lambda em um dispositivo principal, a implantação também inclui esse componente. Para ter mais informações, consulte Executar AWS Lambda funções.

Tópicos

Versões

Esse componente tem as seguintes versões:

- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

Sistema operacional

Esse componente pode ser instalado somente nos dispositivos principais do Linux.

Tipo

Este componente é um componente de plug-in (`aws.greengrass.plugin`). O núcleo do Greengrass executa esse componente na mesma Java Virtual Machine (JVM) do núcleo. O núcleo reinicia quando você altera a versão desse componente no dispositivo principal.

Esse componente usa o mesmo arquivo de log do núcleo do Greengrass. Para ter mais informações, consulte Monitore AWS IoT Greengrass os registros.

Para ter mais informações, consulte Tipos de componentes.

Requisitos

Esse componente tem os seguintes requisitos:

- Seu dispositivo principal deve atender aos requisitos para executar as funções do Lambda. Se você quiser que o dispositivo principal execute funções Lambda em contêineres, o dispositivo deve atender aos requisitos para fazer isso. Para ter mais informações, consulte Requisitos da função do Lambda.
- O componente do gerenciador Lambda tem suporte para execução em uma VPC.

Dependências

Quando você implanta um componente, AWS IoT Greengrass também implanta versões compatíveis de suas dependências. Isso significa que você deve atender aos requisitos do componente e de todas as suas dependências para implantá-lo com êxito. Esta seção lista as dependências das versões lançadas desse componente e as restrições de versão semântica que definem as versões do componente para cada dependência. Você também pode visualizar as dependências de cada versão do componente no AWS IoT Greengrass console. Na página de detalhes do componente, procure a lista de dependências.

2.3.2 and 2.3.3

A tabela a seguir lista as dependências das versões 2.3.2 e 2.3.3 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.13.0	Flexível

2.2.10 and 2.3.1

A tabela a seguir lista as dependências das versões 2.2.10 e 2.3.1 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.12.0	Flexível

2.2.8 and 2.2.9

A tabela a seguir lista as dependências das versões 2.2.8 e 2.2.9 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.11.0	Flexível

2.2.7

A tabela a seguir lista as dependências da versão 2.2.7 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.10.0	Flexível

2.2.6

A tabela a seguir lista as dependências da versão 2.2.6 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.9.0	Flexível

2.2.5

A tabela a seguir lista as dependências da versão 2.2.5 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.8.0	Flexível

2.2.4

A tabela a seguir lista as dependências da versão 2.2.4 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.7.0	Flexível

2.2.1 - 2.2.3

A tabela a seguir lista as dependências das versões 2.2.1 a 2.2.3 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.6.0	Flexível

2.2.0

A tabela a seguir lista as dependências da versão 2.2.0 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2,5.0 <2,6,0	Flexível

2.1.3 and 2.1.4

A tabela a seguir lista as dependências das versões 2.1.3 e 2.1.4 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.5.0	Flexível

2.1.2

A tabela a seguir lista as dependências da versão 2.1.2 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.4.0	Flexível

2.1.1

A tabela a seguir lista as dependências da versão 2.1.1 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.3.0	Flexível

2.1.0

A tabela a seguir lista as dependências da versão 2.1.0 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.2.0	Flexível

2.0.x

A tabela a seguir lista as dependências da versão 2.0.x desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.3 <2.1.0	Flexível

Para obter mais informações sobre dependências de componentes, consulte a referência da receita do componente.

Configuração

Esse componente fornece os seguintes parâmetros de configuração que você pode personalizar ao implantar o componente.

`logHandlerMode`

> **Note**
>
> Somente para as versões 2.3.0+ do gerenciador lambda

Usado para escolher a implementação do gerenciador de registros Lambda a ser usada. Defina o valor `optimized` para usar menos threads para ler registros lambda.

`getResultTimeoutInSecond`

(Opcional) O tempo máximo em segundos que as funções Lambda podem ser executadas antes de atingirem o tempo limite.

Padrão: `60`

Arquivo de log local

Esse componente usa o mesmo arquivo de log do componente do núcleo do Greengrass.

```
/greengrass/v2/logs/greengrass.log
```

Para ver os registros desse componente

- Execute o comando a seguir no dispositivo principal para visualizar o arquivo de log desse componente em tempo real. */greengrass/v2*Substitua pelo caminho para a pasta AWS IoT Greengrass raiz.

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

Changelog

A tabela a seguir descreve as alterações em cada versão do componente.

Version (Versão)	Alterações
2.3.3	Correções de erros e melhorias • Melhorias e correções de erros gerais.
2.3.2	Versão atualizada para a versão 2.12.0 do Greengrass nucleus.
2.3.1	Correções de erros e melhorias • Ajusta os níveis de log para determinados erros.
2.3.0	Novos atributos • O manipulador de registros foi otimizado para reduzir a carga da CPU. Use esse recurso definindo a opção de configuração `logHandler Mode comooptimized .` Correções de erros e melhorias • Não registra mais o stacktrace completo`WorkQueueFullException` , melhorando os registros e o desempenho. • Define o tempo limite de desligamento lambda de 15 segundos a 300 segundos para evitar tempos limite de desligamento. • Corrige um problema em que lambdas sob demanda podem falhar ao reiniciar após alterar a configuração.
2.2.11	Correções de erros e melhorias • Corrige um problema em que a LegacySubscriptionRouter configuração não é atualizada quando a configuração do Lambda é alterada.
2.2.10	Versão atualizada para a versão 2.11.0 do Greengrass nucleus.
2.2.9	Correções de erros e melhorias Corrige um problema em que o número da porta está corrompido devido a um relógio distorcido.
2.2.8	Versão atualizada para a versão 2.10.0 do Greengrass nucleus.
2.2.7	Versão atualizada para a versão 2.9.0 do Greengrass nucleus.

Version (Versão)	Alterações
2.2.6	Versão atualizada para a versão 2.8.0 do Greengrass nucleus.
2.2.5	Novos atributos • Adiciona suporte para curingas de tópicos do MQTT em fontes de eventos nas quais você assina mensagens locais de publicação/assinatura. Esse recurso requer a versão v2.6.0 ou posterior do componente do núcleo do Greengrass. • Versão atualizada para a versão 2.7.0 do Greengrass nucleus.
2.2.4	Versão atualizada para a versão 2.6.0 do Greengrass nucleus.
2.2.3	Correções de erros e melhorias • Corrige um problema em que várias instâncias de uma função Lambda compartilham um único cgroup. Esse componente usa cgroups para gerenciar o uso de recursos para funções do Lambda.
2.2.2	Correções de erros e melhorias • Corrige um problema em que componentes fixos da função Lambda são reiniciados inesperadamente em determinados cenários.
2.2.1	Correções de erros e melhorias • Altera as restrições de versão da dependência do núcleo Greengrass desse componente para corrigir um problema de resolução de dependências.
2.2.0	Correções de erros e melhorias • Corrige um problema em que as funções Lambda não conseguiam gravar registros após uma reinicialização. • Corrige um problema em que o roteador de assinatura antigo envia mensagens duplicadas quando há curingas no tópico. • Corrige um problema em que funções Lambda não fixadas não podiam usar a biblioteca de comunicação entre processos (IPC) do Greengrass no. AWS IoT Device SDK

Version (Versão)	Alterações
2.1.4	Correções de erros e melhorias • Corrige um problema que fazia com que as funções Lambda que usam tempos de execução do NodeJS processassem somente uma mensagem. • Versão atualizada para a versão 2.5.0 do Greengrass nucleus.
2.1.3	Versão atualizada para a versão 2.4.0 do Greengrass nucleus.
2.1.2	Versão atualizada para a versão 2.3.0 do Greengrass nucleus.
2.1.1	Versão atualizada para a versão 2.2.0 do Greengrass nucleus.
2.1.0	Versão atualizada para a versão 2.1.0 do Greengrass nucleus.
2.0.3	Versão inicial.

Runtimes do Lambda

O componente Lambda runtimes (`aws.greengrass.LambdaRuntimes`) fornece os tempos de execução que os dispositivos principais do Greengrass usam para executar funções. AWS Lambda

> **Note**
>
> Quando você implanta um componente da função Lambda em um dispositivo principal, a implantação também inclui esse componente. Para ter mais informações, consulte Executar AWS Lambda funções.

Tópicos

- Versões
- Tipo
- Sistema operacional
- Requisitos
- Dependências

- Configuração
- Arquivo de log local
- Changelog

Versões

Esse componente tem as seguintes versões:

- 2.0.x

Tipo

Este componente é um componente genérico (`aws.greengrass.generic`). O núcleo do Greengrass executa os scripts do ciclo de vida do componente.

Para ter mais informações, consulte Tipos de componentes.

Sistema operacional

Esse componente pode ser instalado somente nos dispositivos principais do Linux.

Requisitos

Esse componente tem os seguintes requisitos:

- Seu dispositivo principal deve atender aos requisitos para executar as funções do Lambda. Se você quiser que o dispositivo principal execute funções Lambda em contêineres, o dispositivo deve atender aos requisitos para fazer isso. Para ter mais informações, consulte Requisitos da função do Lambda.
- O componente Lambda Runtimes tem suporte para execução em uma VPC.

Dependências

Esse componente não tem nenhuma dependência.

Configuração

Esse componente não tem nenhum parâmetro de configuração.

Arquivo de log local

Esse componente não gera registros.

Changelog

A tabela a seguir descreve as alterações em cada versão do componente.

Version (Versão)	Alterações
2.0.8	Versão atualizada para a versão 2.5.0 do Greengrass nucleus.
2.0.7	Versão atualizada para a versão 2.4.0 do Greengrass nucleus.
2.0.6	Versão atualizada para a versão 2.3.0 do Greengrass nucleus.
2.0.5	Versão atualizada para a versão 2.2.0 do Greengrass nucleus.
2.0.4	Versão atualizada para a versão 2.1.0 do Greengrass nucleus.
2.0.3	Versão inicial.

Roteador de assinatura antigo

O roteador de assinatura antigo (`aws.greengrass.LegacySubscriptionRouter`) gerencia as assinaturas no dispositivo principal do Greengrass. As assinaturas são um recurso da AWS IoT Greengrass V1 que define os tópicos que as funções Lambda podem usar para mensagens MQTT em um dispositivo principal. Para obter mais informações, consulte Assinaturas gerenciadas no fluxo de trabalho de mensagens MQTT no Guia do desenvolvedor AWS IoT GreengrassV1.

Você pode usar esse componente para habilitar assinaturas para componentes de conectores e componentes da função Lambda que usam o SDK principal. AWS IoT Greengrass

> **Note**
>
> O componente antigo do roteador de assinatura é necessário somente se sua função Lambda usar a `publish()` função no SDK AWS IoT Greengrass principal. Se você atualizar o código da função Lambda para usar a interface de comunicação entre processos (IPC) na AWS IoT Device SDK V2, não precisará implantar o componente legado do roteador de

assinatura. Para obter mais informações, consulte os seguintes serviços de comunicação entre processos:

- Publique/assine mensagens locais
- Publique/assine mensagens MQTT AWS IoT Core

Tópicos

Versões

Esse componente tem as seguintes versões:

- 2.1.x
- 2.0.x

Tipo

Este componente é um componente genérico (`aws.greengrass.generic`). O núcleo do Greengrass executa os scripts do ciclo de vida do componente.

Para ter mais informações, consulte Tipos de componentes.

Sistema operacional

Esse componente pode ser instalado somente nos dispositivos principais do Linux.

Requisitos

Esse componente tem os seguintes requisitos:

- O roteador de assinatura legado tem suporte para ser executado em uma VPC.

Dependências

Quando você implanta um componente, AWS IoT Greengrass também implanta versões compatíveis de suas dependências. Isso significa que você deve atender aos requisitos do componente e de todas as suas dependências para implantá-lo com êxito. Esta seção lista as dependências das versões lançadas desse componente e as restrições de versão semântica que definem as versões do componente para cada dependência. Você também pode visualizar as dependências de cada versão do componente no AWS IoT Greengrassconsole. Na página de detalhes do componente, procure a lista de dependências.

2.1.11

A tabela a seguir lista as dependências da versão 2.1.11 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.13.0	Flexível

2.1.10

A tabela a seguir lista as dependências da versão 2.1.10 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.12.0	Flexível

2.1.9

A tabela a seguir lista as dependências da versão 2.1.9 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.11.0	Flexível

2.1.8

A tabela a seguir lista as dependências da versão 2.1.8 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.10.0	Flexível

2.1.7

A tabela a seguir lista as dependências da versão 2.1.7 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.9.0	Flexível

2.1.6

A tabela a seguir lista as dependências da versão 2.1.6 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.8.0	Flexível

2.1.5

A tabela a seguir lista as dependências da versão 2.1.5 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.7.0	Flexível

2.1.4

A tabela a seguir lista as dependências da versão 2.1.4 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.6.0	Flexível

2.1.3

A tabela a seguir lista as dependências da versão 2.1.3 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.5.0	Flexível

2.1.2

A tabela a seguir lista as dependências da versão 2.1.2 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.4.0	Flexível

2.1.1

A tabela a seguir lista as dependências da versão 2.1.1 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.3.0	Flexível

2.1.0

A tabela a seguir lista as dependências da versão 2.1.0 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.2.0	Flexível

2.0.3

A tabela a seguir lista as dependências da versão 2.0.3 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.3 <2.1.0	Flexível

Para obter mais informações sobre dependências de componentes, consulte a referência da receita do componente.

Configuração

Esse componente fornece os seguintes parâmetros de configuração que você pode personalizar ao implantar o componente.

v2.1.x

`subscriptions`

(Opcional) As assinaturas a serem ativadas no dispositivo principal. Esse é um objeto, em que cada chave é uma ID exclusiva e cada valor é um objeto que define a assinatura desse conector. Você deve configurar uma assinatura ao implantar um componente de conector V1 ou uma função Lambda que usa AWS IoT Greengrass o SDK principal.

Cada objeto de assinatura contém as seguintes informações:

`id`

O ID exclusivo dessa assinatura. Esse ID deve corresponder à chave desse objeto de assinatura.

`source`

A função Lambda que usa o SDK AWS IoT Greengrass principal para publicar mensagens MQTT sobre os tópicos que você especifica em. `subject` Especifique um dos seguintes:

- O nome de um componente da função Lambda no dispositivo principal. Especifique o nome do componente com o `component:` prefixo, como**`component:com.example.HelloWorldLambda`**.
- O Amazon Resource Name (ARN) de uma função Lambda no dispositivo principal.

> **Important**
>
> Se a versão da função Lambda mudar, você deverá configurar a assinatura com a nova versão da função. Caso contrário, esse componente não roteará as mensagens até que a versão corresponda à assinatura.
> Você deve especificar um Amazon Resource Name (ARN) que inclua a versão da função a ser importada. Você não pode usar aliases de versão, como `$LATEST`.

Para implantar uma assinatura para um componente do conector V1, especifique o nome do componente ou o ARN da função Lambda do componente do conector.

`subject`

O tópico ou filtro de tópicos do MQTT no qual a fonte e o destino podem publicar e receber mensagens. Esse valor suporta os curingas + e o # tópico.

`target`

O alvo que recebe as mensagens do MQTT sobre os tópicos que você especifica em`subject`. A assinatura especifica que a `source` função publica mensagens MQTT em AWS IoT Core ou para uma função Lambda no dispositivo principal. Especifique um dos seguintes:

- `cloud`. A `source` função publica mensagens MQTT para. AWS IoT Core
- O nome de um componente da função Lambda no dispositivo principal. Especifique o nome do componente com o `component:` prefixo, como**`component:com.example.HelloWorldLambda`**.
- O Amazon Resource Name (ARN) de uma função Lambda no dispositivo principal.

> **Important**
>
> Se a versão da função Lambda mudar, você deverá configurar a assinatura com a nova versão da função. Caso contrário, esse componente não roteará as mensagens até que a versão corresponda à assinatura.

Você deve especificar um Amazon Resource Name (ARN) que inclua a versão da função a ser importada. Você não pode usar aliases de versão, como `$LATEST`.

Padrão: Sem assinaturas

Example Exemplo de atualização de configuração (definindo uma assinatura paraAWS IoT Core)

O exemplo a seguir especifica que o componente `com.example.HelloWorldLambda` da função Lambda publica a mensagem AWS IoT Core MQTT no tópico. `hello/world`

```
{
  "subscriptions": {
    "Greengrass_HelloWorld_to_cloud": {
      "id": "Greengrass_HelloWorld_to_cloud",
      "source": "component:com.example.HelloWorldLambda",
      "subject": "hello/world",
      "target": "cloud"
    }
  }
}
```

Example Exemplo de atualização de configuração (definindo uma assinatura para outra função Lambda)

O exemplo a seguir especifica que o componente `com.example.HelloWorldLambda` da função Lambda publica mensagens MQTT no componente da função `com.example.MessageRelay` Lambda sobre o tópico. `hello/world`

```
{
  "subscriptions": {
    "Greengrass_HelloWorld_to_MessageRelay": {
      "id": "Greengrass_HelloWorld_to_MessageRelay",
      "source": "component:com.example.HelloWorldLambda",
      "subject": "hello/world",
      "target": "component:com.example.MessageRelay"
    }
  }
}
```

v2.0.x

`subscriptions`

(Opcional) As assinaturas a serem ativadas no dispositivo principal. Esse é um objeto, em que cada chave é uma ID exclusiva e cada valor é um objeto que define a assinatura desse conector. Você deve configurar uma assinatura ao implantar um componente de conector V1 ou uma função Lambda que usa AWS IoT Greengrass o SDK principal.

Cada objeto de assinatura contém as seguintes informações:

`id`

O ID exclusivo dessa assinatura. Esse ID deve corresponder à chave desse objeto de assinatura.

`source`

A função Lambda que usa o SDK AWS IoT Greengrass principal para publicar mensagens MQTT sobre os tópicos que você especifica em. `subject` Especifique o seguinte:

- O Amazon Resource Name (ARN) de uma função Lambda no dispositivo principal.

  > **Important**
  >
  > Se a versão da função Lambda mudar, você deverá configurar a assinatura com a nova versão da função. Caso contrário, esse componente não roteará as mensagens até que a versão corresponda à assinatura.
  > Você deve especificar um Amazon Resource Name (ARN) que inclua a versão da função a ser importada. Você não pode usar aliases de versão, como `$LATEST`.

Para implantar uma assinatura para um componente do conector V1, especifique o ARN da função Lambda do componente do conector.

`subject`

O tópico ou filtro de tópicos do MQTT no qual a fonte e o destino podem publicar e receber mensagens. Esse valor suporta os curingas + e o # tópico.

`target`

O alvo que recebe as mensagens do MQTT sobre os tópicos que você especifica em`subject`. A assinatura especifica que a `source` função publica mensagens MQTT em

AWS IoT Core ou para uma função Lambda no dispositivo principal. Especifique um dos seguintes:

- `cloud`. A `source` função publica mensagens MQTT para. AWS IoT Core
- O Amazon Resource Name (ARN) de uma função Lambda no dispositivo principal.

> **Important**
>
> Se a versão da função Lambda mudar, você deverá configurar a assinatura com a nova versão da função. Caso contrário, esse componente não roteará as mensagens até que a versão corresponda à assinatura.
> Você deve especificar um Amazon Resource Name (ARN) que inclua a versão da função a ser importada. Você não pode usar aliases de versão, como `$LATEST`.

Padrão: Sem assinaturas

Example Exemplo de atualização de configuração (definindo uma assinatura paraAWS IoT Core)

O exemplo a seguir especifica que a `Greengrass_HelloWorld` função publica a mensagem MQTT AWS IoT Core no tópico. `hello/world`

```
"subscriptions": {
  "Greengrass_HelloWorld_to_cloud": {
    "id": "Greengrass_HelloWorld_to_cloud",
    "source": "arn:aws:lambda:us-
west-2:123456789012:function:Greengrass_HelloWorld:5",
    "subject": "hello/world",
    "target": "cloud"
  }
}
```

Example Exemplo de atualização de configuração (definindo uma assinatura para outra função Lambda)

O exemplo a seguir especifica que a `Greengrass_HelloWorld` função publica mensagens MQTT `Greengrass_MessageRelay` no tópico. `hello/world`

```
"subscriptions": {
```

```
  "Greengrass_HelloWorld_to_MessageRelay": {
    "id": "Greengrass_HelloWorld_to_MessageRelay",
    "source": "arn:aws:lambda:us-
west-2:123456789012:function:Greengrass_HelloWorld:5",
    "subject": "hello/world",
    "target": "arn:aws:lambda:us-
west-2:123456789012:function:Greengrass_MessageRelay:5"
  }
}
```

Arquivo de log local

Esse componente não gera registros.

Changelog

A tabela a seguir descreve as alterações em cada versão do componente.

Version (Versão)	Alterações
2.1.11	Versão atualizada para a versão 2.12.0 do Greengrass nucleus.
2.1.10	Versão atualizada para a versão 2.11.0 do Greengrass nucleus.
2.1.9	Versão atualizada para a versão 2.10.0 do Greengrass nucleus.
2.1.8	Versão atualizada para a versão 2.9.0 do Greengrass nucleus.
2.1.7	Versão atualizada para a versão 2.8.0 do Greengrass nucleus.
2.1.6	Versão atualizada para a versão 2.7.0 do Greengrass nucleus.
2.1.5	Versão atualizada para a versão 2.6.0 do Greengrass nucleus.
2.1.4	Versão atualizada para a versão 2.5.0 do Greengrass nucleus.
2.1.3	Versão atualizada para a versão 2.4.0 do Greengrass nucleus.
2.1.2	Versão atualizada para a versão 2.3.0 do Greengrass nucleus.
2.1.1	Versão atualizada para a versão 2.2.0 do Greengrass nucleus.

Version (Versão)	Alterações
2.1.0	Correções de erros e melhorias • Adiciona suporte para especificar nomes de componentes em vez de ARNs para `source` e. `target` Se você especificar um nome de componente para uma assinatura, não precisará reconfigurar a assinatura sempre que a versão da função Lambda for alterada.
2.0.3	Versão inicial.

Console de depuração local

O componente do console de depuração local (`aws.greengrass.LocalDebugConsole`) fornece um painel local que exibe informações sobre seus dispositivos AWS IoT Greengrass principais e seus componentes. Você pode usar esse painel para depurar seu dispositivo principal e gerenciar componentes locais.

> **Important**
>
> Recomendamos que você use esse componente somente em ambientes de desenvolvimento, não em ambientes de produção. Esse componente fornece acesso a informações e operações que você normalmente não precisará em um ambiente de produção. Siga o princípio do menor privilégio implantando esse componente somente nos dispositivos principais onde você precisar dele.

Tópicos

- Changelog

Versões

Esse componente tem as seguintes versões:

- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

Tipo

Este componente é um componente de plug-in (`aws.greengrass.plugin`). O núcleo do Greengrass executa esse componente na mesma Java Virtual Machine (JVM) do núcleo. O núcleo reinicia quando você altera a versão desse componente no dispositivo principal.

Esse componente usa o mesmo arquivo de log do núcleo do Greengrass. Para ter mais informações, consulte Monitore AWS IoT Greengrass os registros.

Para ter mais informações, consulte Tipos de componentes.

Sistema operacional

Esse componente pode ser instalado em dispositivos principais que executam os seguintes sistemas operacionais:

- Linux
- Windows

Requisitos

Esse componente tem os seguintes requisitos:

- Você usa um nome de usuário e senha para entrar no painel. O nome de usuário, que é `debug`, é fornecido para você. Você deve usar a AWS IoT Greengrass CLI para criar uma senha temporária que autentique você com o painel em um dispositivo principal. Você deve ser capaz de usar a AWS IoT Greengrass CLI para usar o console de depuração local. Para obter mais informações,

consulte os requisitos da CLI do Greengrass. Para obter mais informações sobre como gerar a senha e fazer login, consulte Uso do componente do console de depuração local.

- O componente do console de depuração local tem suporte para execução em uma VPC.

Dependências

Quando você implanta um componente, AWS IoT Greengrass também implanta versões compatíveis de suas dependências. Isso significa que você deve atender aos requisitos do componente e de todas as suas dependências para implantá-lo com êxito. Esta seção lista as dependências das versões lançadas desse componente e as restrições de versão semântica que definem as versões dos componentes para cada dependência. Você também pode visualizar as dependências de cada versão do componente no AWS IoT Greengrass console. Na página de detalhes do componente, procure a lista de dependências.

2.4.1 – 2.4.2

A tabela a seguir lista as dependências das versões 2.4.1 a 2.4.2 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.10.0 <2.13.0	Rígido
CLI do Greengrass	>=2.10.0 <2.13.0	Rígido

2.4.0

A tabela a seguir lista as dependências da versão 2.4.0 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2,10,0 <2,12,0	Rígido
CLI do Greengrass	>=2,10,0 <2,12,0	Rígido

2.3.0 and 2.3.1

A tabela a seguir lista as dependências das versões 2.3.0 e 2.3.1 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2,10,0 <2,12,0	Rígido
CLI do Greengrass	>=2,10,0 <2,12,0	Rígido

2.2.9

A tabela a seguir lista as dependências da versão 2.2.9 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.1.0 <2.12.0	Rígido
CLI do Greengrass	>=2.1.0 <2.12.0	Rígido

2.2.8

A tabela a seguir lista as dependências da versão 2.2.8 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.1.0 <2.11.0	Rígido
CLI do Greengrass	>=2.1.0 <2.11.0	Rígido

2.2.7

A tabela a seguir lista as dependências da versão 2.2.7 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.1.0 <2.10.0	Rígido
CLI do Greengrass	>=2.1.0 <2.10.0	Rígido

2.2.6

A tabela a seguir lista as dependências da versão 2.2.6 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.1.0 <2.9.0	Rígido
CLI do Greengrass	>=2.1.0 <2.9.0	Rígido

2.2.5

A tabela a seguir lista as dependências da versão 2.2.5 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.1.0 <2.8.0	Rígido
CLI do Greengrass	>=2.1.0 <2.8.0	Rígido

2.2.4

A tabela a seguir lista as dependências da versão 2.2.4 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.1.0 <2.7.0	Rígido
CLI do Greengrass	>=2.1.0 <2.7.0	Rígido

2.2.3

A tabela a seguir lista as dependências da versão 2.2.3 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.1.0 <2.6.0	Rígido

Dependência	Versões compatíveis	Tipo de dependência
CLI do Greengrass	>=2.1.0 <2.6.0	Rígido

2.2.2

A tabela a seguir lista as dependências da versão 2.2.2 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.1.0 <2.5.0	Rígido
CLI do Greengrass	>=2.1.0 <2.5.0	Rígido

2.2.1

A tabela a seguir lista as dependências da versão 2.2.1 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.1.0 <2.4.0	Rígido
CLI do Greengrass	>=2.1.0 <2.4.0	Rígido

2.2.0

A tabela a seguir lista as dependências da versão 2.2.0 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.1.0 <2.3.0	Rígido
CLI do Greengrass	>=2.1.0 <2.3.0	Rígido

2.1.0

A tabela a seguir lista as dependências da versão 2.1.0 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.1.0 <2.2.0	Rígido
CLI do Greengrass	>=2.1.0 <2.2.0	Rígido

2.0.x

A tabela a seguir lista as dependências da versão 2.0.x desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.3 <2.1.0	Flexível
CLI do Greengrass	>=2.0.3 <2.1.0	Flexível

Para obter mais informações sobre dependências de componentes, consulte a referência da receita do componente.

Configuração

Esse componente fornece os seguintes parâmetros de configuração que você pode personalizar ao implantar o componente.

v2.1.x - v2.4.x

`httpsEnabled`

(Opcional) Você pode ativar a comunicação HTTPS para o console de depuração local. Se você habilitar a comunicação HTTPS, o console de depuração local criará um certificado autoassinado. Os navegadores da Web mostram avisos de segurança para sites que usam certificados autoassinados, portanto, você deve verificar o certificado manualmente. Em seguida, você pode ignorar o aviso. Para ter mais informações, consulte Uso.

Padrão: `true`

`port`

(Opcional) A porta na qual fornecer o console de depuração local.

Padrão: `1441`

`websocketPort`

(Opcional) A porta do websocket a ser usada para o console de depuração local.

Padrão: `1442`

`bindHostname`

(Opcional) O nome do host a ser usado no console de depuração local.

Se você executar o software AWS IoT Greengrass Core em um contêiner do Docker, defina esse parâmetro como `0.0.0.0`, para poder abrir o console de depuração local fora do contêiner do Docker.

Padrão: `localhost`

Example Exemplo: atualização da mesclagem de configurações

O exemplo de configuração a seguir especifica a abertura do console de depuração local em portas não padrão e a desativação do HTTPS.

```
{
  "httpsEnabled": false,
  "port": "10441",
  "websocketPort": "10442"
}
```

v2.0.x

`port`

(Opcional) A porta na qual fornecer o console de depuração local.

Padrão: `1441`

`websocketPort`

(Opcional) A porta do websocket a ser usada para o console de depuração local.

Padrão: `1442`

`bindHostname`

(Opcional) O nome do host a ser usado no console de depuração local.

Se você executar o software AWS IoT Greengrass Core em um contêiner do Docker, defina esse parâmetro como `0.0.0.0`, para poder abrir o console de depuração local fora do contêiner do Docker.

Padrão: `localhost`

Example Exemplo: atualização da mesclagem de configurações

O exemplo de configuração a seguir especifica a abertura do console de depuração local em portas não padrão.

```
{
  "port": "10441",
  "websocketPort": "10442"
}
```

Uso

Para usar o console de depuração local, crie uma sessão a partir da CLI do Greengrass. Quando você cria uma sessão, a CLI do Greengrass fornece um nome de usuário e uma senha temporária que você pode usar para entrar no console de depuração local.

Siga estas instruções para abrir o console de depuração local em seu dispositivo principal ou em seu computador de desenvolvimento.

v2.1.x - v2.4.x

Nas versões 2.1.0 e posteriores, o console de depuração local usa HTTPS por padrão. Quando o HTTPS está habilitado, o console de depuração local cria um certificado autoassinado para proteger a conexão. Seu navegador mostra um aviso de segurança quando você abre o console de depuração local devido a esse certificado autoassinado. Quando você cria uma sessão com a CLI do Greengrass, a saída inclui as impressões digitais do certificado, para que você possa verificar se o certificado é legítimo e se a conexão é segura.

Você pode desativar o HTTPS. Para obter mais informações, consulte Configuração do console de depuração local.

Para abrir o console de depuração local

1. (Opcional) Para visualizar o console de depuração local em seu computador de desenvolvimento, você pode encaminhar a porta do console via SSH. No entanto, você deve primeiro ativar a `AllowTcpForwarding` opção no arquivo de configuração SSH do seu dispositivo principal. Essa opção é habilitada por padrão. Execute o comando a seguir em seu computador de desenvolvimento para visualizar o painel `localhost:1441` em seu computador de desenvolvimento.

   ```
   ssh -L 1441:localhost:1441 -L 1442:localhost:1442 username@core-device-ip-address
   ```

   > **Note**
   >
   > Você pode alterar as portas padrão de `1441` `1442` e. Para obter mais informações, consulte Configuração do console de depuração local.

2. Crie uma sessão para usar o console de depuração local. Ao criar uma sessão, você gera uma senha que usa para autenticar. O console de depuração local exige uma senha para aumentar a segurança, pois você pode usar esse componente para visualizar informações importantes e realizar operações no dispositivo principal. O console de depuração local também cria um certificado para proteger a conexão se você habilitar HTTPS na configuração do componente. O HTTPS está habilitado por padrão.

   Use a AWS IoT Greengrass CLI para criar a sessão. Esse comando gera uma senha aleatória de 43 caracteres que expira após 8 horas. Substitua */greengrass/v2* ou *C:\greengrass\v2* pelo caminho para a pasta AWS IoT Greengrass V2 raiz.

   Linux or Unix

   ```
   sudo /greengrass/v2/bin/greengrass-cli get-debug-password
   ```

   Windows

   ```
   C:\greengrass\v2\bin\greengrass-cli get-debug-password
   ```

A saída do comando se parece com o exemplo a seguir se você tiver configurado o console de depuração local para usar HTTPS. Você usa as impressões digitais do certificado para verificar se a conexão é segura ao abrir o console de depuração local.

```
Username: debug
Password: bEDp3MOHdj8ou2w5de_sCBI2XAaguy3a8XxREXAMPLE
Password expires at: 2021-04-01T17:01:43.921999931-07:00
The local debug console is configured to use TLS security. The certificate is
 self-signed so you will need to bypass your web browser's security warnings to
 open the console.
Before you bypass the security warning, verify that the certificate fingerprint
 matches the following fingerprints.
SHA-256: 15 0B 2C E2 54 8B 22 DE 08 46 54 8A B1 2B 25 DE FB 02 7D 01 4E 4A 56 67
 96 DA A6 CC B1 D2 C4 1B
SHA-1: BC 3E 16 04 D3 80 70 DA E0 47 25 F9 90 FA D6 02 80 3E B5 C1
```

O componente debug view cria uma sessão que dura 8 horas. Depois disso, você deve gerar uma nova senha para visualizar novamente o console de depuração local.

3. Abra e faça login no painel. Visualize o painel em seu dispositivo principal do Greengrass ou em seu computador de desenvolvimento se você encaminhar a porta por SSH. Execute um destes procedimentos:

   - Se você habilitou o HTTPS no console de depuração local, que é a configuração padrão, faça o seguinte:

     a. Abra `https://localhost:1441` em seu dispositivo principal ou em seu computador de desenvolvimento se você encaminhou a porta por SSH.

        Seu navegador pode mostrar um aviso de segurança sobre um certificado de segurança inválido.

     b. Se seu navegador mostrar um aviso de segurança, verifique se o certificado é legítimo e ignore o aviso de segurança. Faça o seguinte:

        i. Encontre a impressão digital SHA-256 ou SHA-1 do certificado e verifique se ela corresponde à impressão digital SHA-256 ou SHA-1 que o comando imprimiu anteriormente. `get-debug-password` Seu navegador pode fornecer uma ou ambas as impressões digitais. Consulte a documentação do seu navegador para ver o certificado e suas impressões digitais. Em alguns

navegadores, a impressão digital do certificado é chamada de impressão digital.

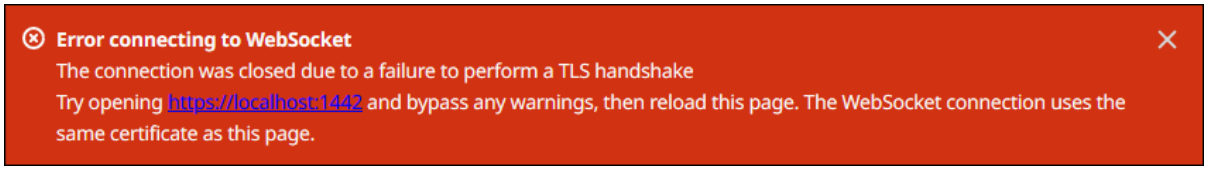
> **Note**
>
> Se a impressão digital do certificado não corresponder, acesse Step 2 para criar uma nova sessão. Se a impressão digital do certificado ainda não corresponder, sua conexão pode estar insegura.

ii. Se a impressão digital do certificado corresponder, ignore o aviso de segurança do seu navegador para abrir o console de depuração local. Consulte a documentação do seu navegador para ignorar o aviso de segurança do navegador.

c. Faça login no site usando o nome de usuário e a senha que o `get-debug-password` comando imprimiu anteriormente.

O console de depuração local é aberto.

d. Se o console de depuração local mostrar um erro informando que não é possível se conectar ao WebSocket devido a uma falha no handshake de TLS, você deverá ignorar o aviso de segurança autoassinado do URL. WebSocket



Faça o seguinte:

i. Abra `https://localhost:1442` no mesmo navegador em que você abriu o console de depuração local.

ii. Verifique o certificado e ignore o aviso de segurança.

Seu navegador pode mostrar uma página HTTP 404 depois que você ignorar o aviso.

iii. Abra `https://localhost:1441` novamente.

O console de depuração local mostra informações sobre o dispositivo principal.

- Se você desativou o HTTPS no console de depuração local, faça o seguinte:

a. Abra `http://localhost:1441` em seu dispositivo principal ou abra-o em seu computador de desenvolvimento se você encaminhou a porta por SSH.

b. Faça login no site usando o nome de usuário e a senha que o `get-debug-password` comando imprimiu anteriormente.

   O console de depuração local é aberto.

v2.0.x

Para abrir o console de depuração local

1. (Opcional) Para visualizar o console de depuração local em seu computador de desenvolvimento, você pode encaminhar a porta do console via SSH. No entanto, você deve primeiro ativar a `AllowTcpForwarding` opção no arquivo de configuração SSH do seu dispositivo principal. Essa opção é habilitada por padrão. Execute o comando a seguir em seu computador de desenvolvimento para visualizar o painel `localhost:1441` em seu computador de desenvolvimento.

   ```
   ssh -L 1441:localhost:1441 -L 1442:localhost:1442 username@core-device-ip-address
   ```

   > **Note**
   >
   > Você pode alterar as portas padrão de `1441` `1442` e. Para obter mais informações, consulte Configuração do console de depuração local.

2. Crie uma sessão para usar o console de depuração local. Ao criar uma sessão, você gera uma senha que usa para autenticar. O console de depuração local exige uma senha para aumentar a segurança, pois você pode usar esse componente para visualizar informações importantes e realizar operações no dispositivo principal.

   Use a AWS IoT Greengrass CLI para criar a sessão. Esse comando gera uma senha aleatória de 43 caracteres que expira após 8 horas. Substitua */greengrass/v2* ou *C:\greengrass\v2* pelo caminho para a pasta AWS IoT Greengrass V2 raiz.

Linux or Unix

```
sudo /greengrass/v2/bin/greengrass-cli get-debug-password
```

Windows

```
C:\greengrass\v2\bin\greengrass-cli get-debug-password
```

A saída do comando se parece com o exemplo a seguir.

```
Username: debug
Password: bEDp3MOHdj8ou2w5de_sCBI2XAaguy3a8XxREXAMPLE
Password will expire at: 2021-04-01T17:01:43.921999931-07:00
```

O componente de visualização de depuração cria uma sessão que dura 4 horas e, em seguida, você deve gerar uma nova senha para visualizar novamente o console de depuração local.

3. Abra `http://localhost:1441` em seu dispositivo principal ou abra-o em seu computador de desenvolvimento se você encaminhou a porta por SSH.
4. Faça login no site usando o nome de usuário e a senha que o `get-debug-password` comando imprimiu anteriormente.

   O console de depuração local é aberto.

Arquivo de log local

Esse componente usa o mesmo arquivo de log do componente do núcleo do Greengrass.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

Para ver os registros desse componente

- Execute o comando a seguir no dispositivo principal para visualizar o arquivo de log desse componente em tempo real. Substitua */greengrass/v2* ou *C:\greengrass\v2* pelo caminho para a pasta AWS IoT Greengrass raiz.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

Changelog

A tabela a seguir descreve as alterações em cada versão do componente.

Version (Versão)	Alterações
2.4.2	Correções de erros e melhorias • Melhorias e correções de erros gerais.
2.4.1	Versão atualizada para a versão 2.12.0 do Greengrass nucleus.
2.4.0	Novos atributos • Adiciona o console de depuração do gerenciador de streams.
2.3.1	Versão atualizada para a versão 2.11.0 do Greengrass nucleus.
2.3.0	Versão atualizada para a versão 2.10.0 do Greengrass nucleus. Novos atributos • Inclui um PubSub cliente de depuração AWS IoT Core MQTT.
2.2.7	Versão atualizada para a versão 2.9.0 do Greengrass nucleus.
2.2.6	Versão atualizada para a versão 2.8.0 do Greengrass nucleus.

Version (Versão)	Alterações
2.2.5	Versão atualizada para a versão 2.7.0 do Greengrass nucleus.
2.2.4	Versão atualizada para a versão 2.6.0 do Greengrass nucleus.
2.2.3	Correções de erros e melhorias • Corrige um problema que impedia a inicialização quando o component e não conseguia descriptografar o repositório de chaves que contém a chave privada SSL. • Versão atualizada para a versão 2.5.0 do Greengrass nucleus.
2.2.2	Versão atualizada para a versão 2.4.0 do Greengrass nucleus.
2.2.1	Versão atualizada para a versão 2.3.0 do Greengrass nucleus.
2.2.0	Versão atualizada para a versão 2.2.0 do Greengrass nucleus.
2.1.0	Novos atributos • Usa HTTPS para proteger sua conexão com o console de depuração local. O HTTPS está habilitado por padrão. Correções de erros e melhorias • Você pode ignorar as mensagens da barra de flash no editor de configuração.
2.0.3	Versão inicial.

Gerenciador de registros

O componente do gerenciador de registros (`aws.greengrass.LogManager`) carrega registros dos dispositivos AWS IoT Greengrass principais para o Amazon CloudWatch Logs. Você pode carregar registros do núcleo do Greengrass, de outros componentes do Greengrass e de outros aplicativos e serviços que não sejam componentes do Greengrass. Para obter mais informações sobre como monitorar registros no CloudWatch Logs e no sistema de arquivos local, consulte[Monitore AWS IoT Greengrass os registros](#).

As considerações a seguir se aplicam quando você usa o componente do gerenciador de CloudWatch registros para gravar em registros:

- Atrasos no registro

> **Note**
>
> Recomendamos que você atualize para a versão 2.3.0 do gerenciador de registros, que reduz os atrasos nos arquivos de log rotacionados e ativos. Ao atualizar para o log manager 2.3.0, recomendamos que você também atualize para o Greengrass nucleus 2.9.1.

A versão 2.2.8 (e anterior) do componente gerenciador de registros processa e carrega registros somente de arquivos de log rotacionados. Por padrão, o software AWS IoT Greengrass Core gira os arquivos de log a cada hora ou depois de atingirem 1.024 KB. Como resultado, o componente do gerenciador de registros carrega registros somente depois que o software AWS IoT Greengrass Core ou um componente do Greengrass grava mais de 1.024 KB de registros. Você pode configurar um limite menor de tamanho de arquivo de log para fazer com que os arquivos de log girem com mais frequência. Isso faz com que o componente do gerenciador de registros faça upload de registros para o CloudWatch Logs com mais frequência.

A versão 2.3.0 (e posterior) do componente gerenciador de registros processa e carrega todos os registros. Quando você grava um novo registro, a versão 2.3.0 (e posterior) do gerenciador de registros processa e carrega diretamente esse arquivo de log ativo em vez de esperar que ele seja rotacionado. Isso significa que você pode visualizar o novo registro em 5 minutos ou menos.

O componente do gerenciador de registros carrega novos registros periodicamente. Por padrão, o componente do gerenciador de registros carrega novos registros a cada 5 minutos. Você pode configurar um intervalo de upload menor, para que o componente do gerenciador de registros faça o upload dos CloudWatch registros para o Logs com mais frequência configurando o. `periodicUploadIntervalSec` Para obter mais informações sobre como configurar esse intervalo periódico, consulte Configuração.

Os registros podem ser carregados quase em tempo real a partir do mesmo sistema de arquivos do Greengrass. Se você precisar observar os registros em tempo real, considere usar os registros do sistema de arquivos.

> **Note**
>
> Se você estiver usando sistemas de arquivos diferentes para gravar registros, o gerenciador de registros retornará ao comportamento nas versões 2.2.8 e anteriores do componente do gerenciador de registros. Para obter informações sobre como acessar registros do sistema de arquivos, consulte Acessar registros do sistema de arquivos.

- Inclinação do relógio

  O componente do gerenciador de registros usa o processo de assinatura padrão do Signature versão 4 para criar solicitações de API para o CloudWatch Logs. Se a hora do sistema em um dispositivo principal estiver fora de sincronia por mais de 15 minutos, o CloudWatch Logs rejeitará as solicitações. Para obter mais informações, consulte Processo de assinatura do Signature versão 4 na Referência geral da AWS.

Para obter informações sobre os grupos e fluxos de registros para os quais esse componente carrega registros, consulte. Uso

Tópicos

Versões

Esse componente tem as seguintes versões:

- 2.3.x

- 2.2.x
- 2.1.x
- 2.0.x

Tipo

Este componente é um componente de plug-in (`aws.greengrass.plugin`). O núcleo do Greengrass executa esse componente na mesma Java Virtual Machine (JVM) do núcleo. O núcleo reinicia quando você altera a versão desse componente no dispositivo principal.

Esse componente usa o mesmo arquivo de log do núcleo do Greengrass. Para ter mais informações, consulte Monitore AWS IoT Greengrass os registros.

Para ter mais informações, consulte Tipos de componentes.

Sistema operacional

Esse componente pode ser instalado em dispositivos principais que executam os seguintes sistemas operacionais:

- Linux
- Windows

Requisitos

Esse componente tem os seguintes requisitos:

- A função de dispositivo do Greengrass deve permitir as `logs:DescribeLogStreams` ações`logs:CreateLogGroup`,`logs:CreateLogStream`, `elogs:PutLogEvents`, conforme mostrado no exemplo de política do IAM a seguir.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
```

```
        "logs:DescribeLogStreams"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:logs:*:*:*"
    }
  ]
}
```

> **Note**
>
> A função de dispositivo do Greengrass que você cria ao instalar o software AWS IoT Greengrass Core inclui as permissões neste exemplo de política por padrão.

Para obter mais informações, consulte Uso de políticas baseadas em identidade (políticas do IAM) para CloudWatch registros no Guia do usuário do Amazon CloudWatch Logs.

- O componente do gerenciador de registros é compatível com a execução em uma VPC. Para implantar esse componente em uma VPC, é necessário o seguinte.
  - O componente do gerenciador de registros deve ter conectividade com a `logs.region.amazonaws.com` qual tenha o VPC endpoint de. `com.amazonaws.us-east-1.logs`

Endpoints e portas

Esse componente deve ser capaz de realizar solicitações de saída para os seguintes endpoints e portas, além dos endpoints e portas necessários para a operação básica. Para ter mais informações, consulte Permitir o tráfego de dispositivos por meio de um proxy ou firewall.

Endpoint	Porta	Obrigatório	Descrição
`logs.region.amazonaws.com`	443	Não	Obrigatório se você gravar registros em CloudWatch Logs.

Dependências

Quando você implanta um componente, AWS IoT Greengrass também implanta versões compatíveis de suas dependências. Isso significa que você deve atender aos requisitos do componente e de todas as suas dependências para implantá-lo com êxito. Esta seção lista as dependências das versões lançadas desse componente e as restrições de versão semântica que definem as versões dos componentes para cada dependência. Você também pode visualizar as dependências de cada versão do componente no AWS IoT Greengrassconsole. Na página de detalhes do componente, procure a lista de dependências.

2.3.7

A tabela a seguir lista as dependências da versão 2.3.7 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.1.0 <2.13.0	Flexível

2.3.5 and 2.3.6

A tabela a seguir lista as dependências das versões 2.3.5 e 2.3.6 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.1.0 <2.12.0	Flexível

2.3.3 – 2.3.4

A tabela a seguir lista as dependências das versões 2.3.3 a 2.3.4 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.1.0 <2.11.0	Flexível

2.2.8 – 2.3.2

A tabela a seguir lista as dependências das versões 2.2.8 a 2.3.2 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.1.0 <2.10.0	Flexível

2.2.7

A tabela a seguir lista as dependências da versão 2.2.7 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.1.0 <2.9.0	Flexível

2.2.6

A tabela a seguir lista as dependências da versão 2.2.6 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.1.0 <2.8.0	Flexível

2.2.5

A tabela a seguir lista as dependências da versão 2.2.5 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.1.0 <2.7.0	Flexível

2.2.1 - 2.2.4

A tabela a seguir lista as dependências das versões 2.2.1 a 2.2.4 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.1.0 <2.6.0	Flexível

2.1.3 and 2.2.0

A tabela a seguir lista as dependências das versões 2.1.3 e 2.2.0 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.1.0 <2.5.0	Flexível

2.1.2

A tabela a seguir lista as dependências da versão 2.1.2 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.1.0 <2.4.0	Flexível

2.1.1

A tabela a seguir lista as dependências da versão 2.1.1 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.1.0 <2.3.0	Flexível

2.1.0

A tabela a seguir lista as dependências da versão 2.1.0 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.1.0 <2.2.0	Flexível

2.0.x

A tabela a seguir lista as dependências da versão 2.0.x desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.3 <2.1.0	Flexível

Para obter mais informações sobre dependências de componentes, consulte a referência da receita do componente.

Configuração

Esse componente fornece os seguintes parâmetros de configuração que você pode personalizar ao implantar o componente.

v2.3.6 – v2.3.7

`logsUploaderConfiguration`

(Opcional) A configuração dos registros que o componente do gerenciador de registros carrega. Esse objeto contém as seguintes informações:

`systemLogsConfiguration`

(Opcional) A configuração dos registros do sistema de software AWS IoT Greengrass Core, que incluem registros do núcleo do Greengrass e dos componentes do plug-in. Especifique essa configuração para permitir que o componente do gerenciador de registros gerencie os registros do sistema. Esse objeto contém as seguintes informações:

`uploadToCloudWatch`

(Opcional) Você pode fazer upload dos registros do sistema para o CloudWatch Logs.

Padrão: `false`

`minimumLogLevel`

(Opcional) O nível mínimo de mensagens de registro a serem carregadas. Esse nível mínimo se aplica somente se você configurar o componente nucleus do Greengrass para gerar registros no formato JSON. Para ativar os registros no formato JSON, especifique `JSON` o parâmetro de formato de registro (`logging.format`).

Escolha entre os seguintes níveis de registro, listados aqui em ordem de nível:

- `DEBUG`

- `INFO`
- `WARN`
- `ERROR`

Padrão: `INFO`

`diskSpaceLimit`

(Opcional) O tamanho total máximo dos arquivos de log do sistema Greengrass, na unidade em que você especifica. `diskSpaceLimitUnit` Depois que o tamanho total dos arquivos de log do sistema Greengrass exceder esse tamanho total máximo, o software AWS IoT Greengrass Core exclui os arquivos de log mais antigos do sistema Greengrass.

Esse parâmetro é equivalente ao parâmetro de limite de tamanho de log (`totalLogsSizeKB`) do componente do núcleo do Greengrass. O software AWS IoT Greengrass Core usa o mínimo dos dois valores como o tamanho máximo total do log do sistema Greengrass.

`diskSpaceLimitUnit`

(Opcional) A unidade para `diskSpaceLimit` o. Escolha uma das seguintes opções:

- `KB`— kilobytes
- `MB`— megabytes
- `GB`— gigabytes

Padrão: `KB`

`deleteLogFileAfterCloudUpload`

(Opcional) Você pode excluir um arquivo de registro depois que o componente gerenciador de registros fizer o upload dos registros para o CloudWatch Logs.

Padrão: `false`

`componentLogsConfigurationMap`

(Opcional) Um mapa das configurações de log para componentes no dispositivo principal. Cada `componentName` objeto nesse mapa define a configuração de log para o componente ou aplicativo. O componente gerenciador de registros carrega esses registros de componentes no CloudWatch Logs.

> **Important**
>
> É altamente recomendável usar uma única chave de configuração por componente. Você só deve segmentar um grupo de arquivos que tenha somente um arquivo de log que esteja sendo gravado ativamente ao usar `logFileRegex` o. Não seguir essa recomendação pode fazer com que registros duplicados sejam enviados para CloudWatch. Se você tem como alvo vários arquivos de log ativos com uma única regex, recomendamos que você atualize para o gerenciador de registros v2.3.1 ou posterior e considere alterar sua configuração usando a configuração de exemplo.

> **Note**
>
> Se você estiver atualizando de uma versão do gerenciador de registros anterior à v2.2.0, poderá continuar usando a `componentLogsConfiguration` lista em vez de. `componentLogsConfigurationMap` No entanto, é altamente recomendável que você use o formato de mapa para poder mesclar e redefinir atualizações para modificar as configurações de componentes específicos. Para obter informações sobre o `componentLogsConfiguration` parâmetro, consulte os parâmetros de configuração para a versão 2.1.x desse componente.

`componentName`

A configuração de log para o *`componentName`*componente ou aplicativo para essa configuração de log. Você pode especificar o nome de um componente do Greengrass ou outro valor para identificar esse grupo de registros.

Cada objeto contém as seguintes informações:

`minimumLogLevel`

(Opcional) O nível mínimo de mensagens de registro a serem carregadas. Esse nível mínimo se aplica somente se os registros desse componente usarem um formato JSON específico, que você pode encontrar no repositório do módulo de AWS IoT Greengrass registro em. GitHub

Escolha entre os seguintes níveis de registro, listados aqui em ordem de nível:

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

Padrão: `INFO`

`diskSpaceLimit`

(Opcional) O tamanho total máximo de todos os arquivos de log desse componente, na unidade especificada`diskSpaceLimitUnit`. Depois que o tamanho total dos arquivos de log desse componente exceder esse tamanho total máximo, o software AWS IoT Greengrass Core exclui os arquivos de log mais antigos desse componente.

Esse parâmetro está relacionado ao parâmetro de limite de tamanho de log (`totalLogsSizeKB`) do componente do núcleo do Greengrass. O software AWS IoT Greengrass Core usa o mínimo dos dois valores como o tamanho máximo total do log para esse componente.

`diskSpaceLimitUnit`

(Opcional) A unidade para `diskSpaceLimit` o. Escolha uma das seguintes opções:

- `KB`— kilobytes
- `MB`— megabytes
- `GB`— gigabytes

Padrão: `KB`

`logFileDirectoryPath`

(Opcional) O caminho para a pasta que contém os arquivos de log desse componente.

Você não precisa especificar esse parâmetro para componentes do Greengrass que imprimem na saída padrão (stdout) e no erro padrão (stderr).

Padrão: */greengrass/v2*`/logs`.

`logFileRegex`

(Opcional) Uma expressão regular que especifica o formato do nome do arquivo de log usado pelo componente ou aplicativo. O componente gerenciador de registros usa essa expressão regular para identificar arquivos de log na pasta em`logFileDirectoryPath`.

Você não precisa especificar esse parâmetro para componentes do Greengrass que imprimem na saída padrão (stdout) e no erro padrão (stderr).

Se seu componente ou aplicativo rotacionar arquivos de log, especifique um regex que corresponda aos nomes dos arquivos de log rotacionados. Por exemplo, você pode especificar **`hello_world\\\\w*.log`** o upload de registros para um aplicativo Hello World. O `\\\\w*` padrão corresponde a zero ou mais caracteres de palavras, o que inclui caracteres alfanuméricos e sublinhados. Esse regex corresponde aos arquivos de log com e sem carimbos de data/hora em seus nomes. Neste exemplo, o gerenciador de registros carrega os seguintes arquivos de log:

- `hello_world.log`— O arquivo de log mais recente do aplicativo Hello World.
- `hello_world_2020_12_15_17_0.log`— Um arquivo de log mais antigo para o aplicativo Hello World.

Padrão:*`componentName`*`\\\\w*.log`, em que *`componentName`* é o nome do componente dessa configuração de log.

`deleteLogFileAfterCloudUpload`

(Opcional) Você pode excluir um arquivo de registro depois que o componente gerenciador de registros fizer o upload dos registros para o CloudWatch Logs.

Padrão: `false`

`multiLineStartPattern`

(Opcional) Uma expressão regular que identifica quando uma mensagem de log em uma nova linha é uma nova mensagem de log. Se a expressão regular não corresponder à nova linha, o componente do gerenciador de registros anexará a nova linha à mensagem de registro da linha anterior.

Por padrão, o componente do gerenciador de registros verifica se a linha começa com um caractere de espaço em branco, como uma tabulação ou espaço. Caso

contrário, o gerenciador de registros trata essa linha como uma nova mensagem de registro. Caso contrário, ele anexa essa linha à mensagem de log atual. Esse comportamento garante que o componente do gerenciador de registros não divida mensagens que se estendem por várias linhas, como rastreamentos de pilha.

`periodicUploadIntervalSec`

(Opcional) O período em segundos no qual o componente do gerenciador de registros verifica se há novos arquivos de log a serem carregados.

Padrão: `300` (5 minutos)

Mínimo: `0.000001` (1 microssegundo)

`deprecatedVersionSupport`

Indica se o gerenciador de registros deve usar as melhorias de velocidade de registro introduzidas no gerenciador de registros v2.3.5. Defina o valor `false` para usar as melhorias.

Se você definir esse valor para `false` quando fizer o upgrade do gerenciador de registros v2.3.1 ou anterior, entradas de registro duplicadas poderão ser carregadas.

O padrão é `true`.

Example Exemplo: atualização da mesclagem de configurações

O exemplo de configuração a seguir especifica o upload de registros do sistema e de `com.example.HelloWorld` componentes para o CloudWatch Logs.

```
{
  "logsUploaderConfiguration": {
    "systemLogsConfiguration": {
      "uploadToCloudWatch": "true",
      "minimumLogLevel": "INFO",
      "diskSpaceLimit": "10",
      "diskSpaceLimitUnit": "MB",
      "deleteLogFileAfterCloudUpload": "false"
    },
    "componentLogsConfigurationMap": {
      "com.example.HelloWorld": {
        "minimumLogLevel": "INFO",
        "diskSpaceLimit": "20",
```

```
        "diskSpaceLimitUnit": "MB",
        "deleteLogFileAfterCloudUpload": "false"
      }
    }
  },
  "periodicUploadIntervalSec": "300",
  "deprecatedVersionSupport": "false"
}
```

Example Exemplo: configuração para carregar vários arquivos de log ativos usando o gerenciador de registros v2.3.1

O exemplo de configuração a seguir é o exemplo recomendado se você quiser segmentar vários arquivos de log ativos. Este exemplo de configuração especifica para quais arquivos de log ativos você deseja CloudWatch carregar. Usando este exemplo de configuração, a configuração também carregará todos os arquivos rotacionados que correspondam a`logFileRegex`. Esse exemplo de configuração é compatível com o gerenciador de registros v2.3.1.

```
{
  "logsUploaderConfiguration": {
    "componentLogsConfigurationMap": {
      "com.example.A": {
        "logFileRegex": "com.example.A\\w*.log",
        "deleteLogFileAfterCloudUpload": "false"
      }
      "com.example.B": {
        "logFileRegex": "com.example.B\\w*.log",
        "deleteLogFileAfterCloudUpload": "false"
      }
    }
  },
  "periodicUploadIntervalSec": "10"
}
```

v2.3.x

`logsUploaderConfiguration`

(Opcional) A configuração dos registros que o componente do gerenciador de registros carrega. Esse objeto contém as seguintes informações:

`systemLogsConfiguration`

(Opcional) A configuração dos registros do sistema de software AWS IoT Greengrass Core, que incluem registros do núcleo do Greengrass e dos componentes do plug-in. Especifique essa configuração para permitir que o componente do gerenciador de registros gerencie os registros do sistema. Esse objeto contém as seguintes informações:

`uploadToCloudWatch`

(Opcional) Você pode fazer upload dos registros do sistema para o CloudWatch Logs.

Padrão: `false`

`minimumLogLevel`

(Opcional) O nível mínimo de mensagens de registro a serem carregadas. Esse nível mínimo se aplica somente se você configurar o componente nucleus do Greengrass para gerar registros no formato JSON. Para ativar os registros no formato JSON, especifique `JSON` o parâmetro de formato de registro (`logging.format`).

Escolha entre os seguintes níveis de registro, listados aqui em ordem de nível:

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

Padrão: `INFO`

`diskSpaceLimit`

(Opcional) O tamanho total máximo dos arquivos de log do sistema Greengrass, na unidade em que você especifica. `diskSpaceLimitUnit` Depois que o tamanho total dos arquivos de log do sistema Greengrass exceder esse tamanho total máximo, o software AWS IoT Greengrass Core exclui os arquivos de log mais antigos do sistema Greengrass.

Esse parâmetro é equivalente ao parâmetro de limite de tamanho de log (`totalLogsSizeKB`) do componente do núcleo do Greengrass. O software AWS IoT Greengrass Core usa o mínimo dos dois valores como o tamanho máximo total do log do sistema Greengrass.

`diskSpaceLimitUnit`

(Opcional) A unidade para `diskSpaceLimit` o. Escolha uma das seguintes opções:

- `KB`— kilobytes
- `MB`— megabytes
- `GB`— gigabytes

Padrão: `KB`

`deleteLogFileAfterCloudUpload`

(Opcional) Você pode excluir um arquivo de registro depois que o componente gerenciador de registros fizer o upload dos registros para o CloudWatch Logs.

Padrão: `false`

`componentLogsConfigurationMap`

(Opcional) Um mapa das configurações de log para componentes no dispositivo principal. Cada `componentName` objeto nesse mapa define a configuração de log para o componente ou aplicativo. O componente gerenciador de registros carrega esses registros de componentes no CloudWatch Logs.

> **Important**
>
> É altamente recomendável usar uma única chave de configuração por componente. Você só deve segmentar um grupo de arquivos que tenha somente um arquivo de log que esteja sendo gravado ativamente ao usar `logFileRegex` o. Não seguir essa recomendação pode fazer com que registros duplicados sejam enviados para CloudWatch. Se você tem como alvo vários arquivos de log ativos com uma única regex, recomendamos que você atualize para o log manager v2.3.1 e considere alterar sua configuração usando a configuração de exemplo.

> **Note**
>
> Se você estiver atualizando de uma versão do gerenciador de registros anterior à v2.2.0, poderá continuar usando a `componentLogsConfiguration` lista em vez de. `componentLogsConfigurationMap` No entanto, é altamente recomendável

que você use o formato de mapa para poder mesclar e redefinir atualizações para modificar as configurações de componentes específicos. Para obter informações sobre o `componentLogsConfiguration` parâmetro, consulte os parâmetros de configuração para a versão 2.1.x desse componente.

`componentName`

A configuração de log para o *`componentName`*componente ou aplicativo para essa configuração de log. Você pode especificar o nome de um componente do Greengrass ou outro valor para identificar esse grupo de registros.

Cada objeto contém as seguintes informações:

`minimumLogLevel`

(Opcional) O nível mínimo de mensagens de registro a serem carregadas. Esse nível mínimo se aplica somente se os registros desse componente usarem um formato JSON específico, que você pode encontrar no repositório do módulo de AWS IoT Greengrass registro em. GitHub

Escolha entre os seguintes níveis de registro, listados aqui em ordem de nível:

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

Padrão: `INFO`

`diskSpaceLimit`

(Opcional) O tamanho total máximo de todos os arquivos de log desse componente, na unidade especificada`diskSpaceLimitUnit`. Depois que o tamanho total dos arquivos de log desse componente exceder esse tamanho total máximo, o software AWS IoT Greengrass Core exclui os arquivos de log mais antigos desse componente.

Esse parâmetro está relacionado ao parâmetro de limite de tamanho de log (`totalLogsSizeKB`) do componente do núcleo do Greengrass. O software AWS

IoT Greengrass Core usa o mínimo dos dois valores como o tamanho máximo total do log para esse componente.

`diskSpaceLimitUnit`

(Opcional) A unidade para `diskSpaceLimit` o. Escolha uma das seguintes opções:

- `KB`— kilobytes
- `MB`— megabytes
- `GB`— gigabytes

Padrão: `KB`

`logFileDirectoryPath`

(Opcional) O caminho para a pasta que contém os arquivos de log desse componente.

Você não precisa especificar esse parâmetro para componentes do Greengrass que imprimem na saída padrão (stdout) e no erro padrão (stderr).

Padrão: */greengrass/v2*`/logs`.

`logFileRegex`

(Opcional) Uma expressão regular que especifica o formato do nome do arquivo de log usado pelo componente ou aplicativo. O componente gerenciador de registros usa essa expressão regular para identificar arquivos de log na pasta em`logFileDirectoryPath`.

Você não precisa especificar esse parâmetro para componentes do Greengrass que imprimem na saída padrão (stdout) e no erro padrão (stderr).

Se seu componente ou aplicativo rotacionar arquivos de log, especifique um regex que corresponda aos nomes dos arquivos de log rotacionados. Por exemplo, você pode especificar **`hello_world\\\\w*.log`** o upload de registros para um aplicativo Hello World. O `\\\\w*` padrão corresponde a zero ou mais caracteres de palavras, o que inclui caracteres alfanuméricos e sublinhados. Esse regex corresponde aos arquivos de log com e sem carimbos de data/hora em seus nomes. Neste exemplo, o gerenciador de registros carrega os seguintes arquivos de log:

- `hello_world.log`— O arquivo de log mais recente do aplicativo Hello World.
- `hello_world_2020_12_15_17_0.log`— Um arquivo de log mais antigo para o aplicativo Hello World.

Padrão:*componentName*`\\\\w*.log`, em que *componentName* é o nome do componente dessa configuração de log.

`deleteLogFileAfterCloudUpload`

(Opcional) Você pode excluir um arquivo de registro depois que o componente gerenciador de registros fizer o upload dos registros para o CloudWatch Logs.

Padrão: `false`

`multiLineStartPattern`

(Opcional) Uma expressão regular que identifica quando uma mensagem de log em uma nova linha é uma nova mensagem de log. Se a expressão regular não corresponder à nova linha, o componente do gerenciador de registros anexará a nova linha à mensagem de registro da linha anterior.

Por padrão, o componente do gerenciador de registros verifica se a linha começa com um caractere de espaço em branco, como uma tabulação ou espaço. Caso contrário, o gerenciador de registros trata essa linha como uma nova mensagem de registro. Caso contrário, ele anexa essa linha à mensagem de log atual. Esse comportamento garante que o componente do gerenciador de registros não divida mensagens que se estendem por várias linhas, como rastreamentos de pilha.

`periodicUploadIntervalSec`

(Opcional) O período em segundos no qual o componente do gerenciador de registros verifica se há novos arquivos de log a serem carregados.

Padrão: `300` (5 minutos)

Mínimo: `0.000001` (1 microssegundo)

Example Exemplo: atualização da mesclagem de configurações

O exemplo de configuração a seguir especifica o upload de registros do sistema e de `com.example.HelloWorld` componentes para o CloudWatch Logs.

```
{
  "logsUploaderConfiguration": {
    "systemLogsConfiguration": {
      "uploadToCloudWatch": "true",
      "minimumLogLevel": "INFO",
      "diskSpaceLimit": "10",
      "diskSpaceLimitUnit": "MB",
      "deleteLogFileAfterCloudUpload": "false"
    },
    "componentLogsConfigurationMap": {
      "com.example.HelloWorld": {
        "minimumLogLevel": "INFO",
        "diskSpaceLimit": "20",
        "diskSpaceLimitUnit": "MB",
        "deleteLogFileAfterCloudUpload": "false"
      }
    }
  },
  "periodicUploadIntervalSec": "300"
}
```

Example Exemplo: configuração para carregar vários arquivos de log ativos usando o gerenciador de registros v2.3.1

O exemplo de configuração a seguir é o exemplo recomendado se você quiser segmentar vários arquivos de log ativos. Este exemplo de configuração especifica para quais arquivos de log ativos você deseja CloudWatch carregar. Usando este exemplo de configuração, a configuração também carregará todos os arquivos rotacionados que correspondam a`logFileRegex`. Esse exemplo de configuração é compatível com o gerenciador de registros v2.3.1.

```
{
  "logsUploaderConfiguration": {
    "componentLogsConfigurationMap": {
      "com.example.A": {
        "logFileRegex": "com.example.A\\w*.log",
        "deleteLogFileAfterCloudUpload": "false"
      }
      "com.example.B": {
        "logFileRegex": "com.example.B\\w*.log",
        "deleteLogFileAfterCloudUpload": "false"
      }
    }
```

```
  },
  "periodicUploadIntervalSec": "10"
}
```

v2.2.x

`logsUploaderConfiguration`

(Opcional) A configuração dos registros que o componente do gerenciador de registros carrega. Esse objeto contém as seguintes informações:

`systemLogsConfiguration`

(Opcional) A configuração dos registros do sistema de software AWS IoT Greengrass Core, que incluem registros do núcleo do Greengrass e dos componentes do plug-in. Especifique essa configuração para permitir que o componente do gerenciador de registros gerencie os registros do sistema. Esse objeto contém as seguintes informações:

`uploadToCloudWatch`

(Opcional) Você pode fazer upload dos registros do sistema para o CloudWatch Logs.

Padrão: `false`

`minimumLogLevel`

(Opcional) O nível mínimo de mensagens de registro a serem carregadas. Esse nível mínimo se aplica somente se você configurar o componente nucleus do Greengrass para gerar registros no formato JSON. Para ativar os registros no formato JSON, especifique `JSON` o parâmetro de formato de registro (`logging.format`).

Escolha entre os seguintes níveis de registro, listados aqui em ordem de nível:

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

Padrão: `INFO`

`diskSpaceLimit`

(Opcional) O tamanho total máximo dos arquivos de log do sistema Greengrass, na unidade em que você especifica. `diskSpaceLimitUnit` Depois que o tamanho total

dos arquivos de log do sistema Greengrass exceder esse tamanho total máximo, o software AWS IoT Greengrass Core exclui os arquivos de log mais antigos do sistema Greengrass.

Esse parâmetro é equivalente ao parâmetro de limite de tamanho de log (`totalLogsSizeKB`) do componente do núcleo do Greengrass. O software AWS IoT Greengrass Core usa o mínimo dos dois valores como o tamanho máximo total do log do sistema Greengrass.

`diskSpaceLimitUnit`

(Opcional) A unidade para `diskSpaceLimit` o. Escolha uma das seguintes opções:

- `KB`— kilobytes
- `MB`— megabytes
- `GB`— gigabytes

Padrão: `KB`

`deleteLogFileAfterCloudUpload`

(Opcional) Você pode excluir um arquivo de registro depois que o componente gerenciador de registros fizer o upload dos registros para o CloudWatch Logs.

Padrão: `false`

`componentLogsConfigurationMap`

(Opcional) Um mapa das configurações de log para componentes no dispositivo principal. Cada `componentName` objeto nesse mapa define a configuração de log para o componente ou aplicativo. O componente gerenciador de registros carrega esses registros de componentes no CloudWatch Logs.

> **Note**
>
> Se você estiver atualizando de uma versão do gerenciador de registros anterior à v2.2.0, poderá continuar usando a `componentLogsConfiguration` lista em vez de. `componentLogsConfigurationMap` No entanto, é altamente recomendável que você use o formato de mapa para poder mesclar e redefinir atualizações para modificar as configurações de componentes específicos. Para obter informações sobre o `componentLogsConfiguration` parâmetro, consulte os parâmetros de configuração para a versão 2.1.x desse componente.

componentName

A configuração de log para o *componentName*componente ou aplicativo para essa configuração de log. Você pode especificar o nome de um componente do Greengrass ou outro valor para identificar esse grupo de registros.

Cada objeto contém as seguintes informações:

`minimumLogLevel`

(Opcional) O nível mínimo de mensagens de registro a serem carregadas. Esse nível mínimo se aplica somente se os registros desse componente usarem um formato JSON específico, que você pode encontrar no repositório do módulo de AWS IoT Greengrass registro em. GitHub

Escolha entre os seguintes níveis de registro, listados aqui em ordem de nível:

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

Padrão: `INFO`

`diskSpaceLimit`

(Opcional) O tamanho total máximo de todos os arquivos de log desse componente, na unidade especificada`diskSpaceLimitUnit`. Depois que o tamanho total dos arquivos de log desse componente exceder esse tamanho total máximo, o software AWS IoT Greengrass Core exclui os arquivos de log mais antigos desse componente.

Esse parâmetro está relacionado ao parâmetro de limite de tamanho de log (`totalLogsSizeKB`) do componente do núcleo do Greengrass. O software AWS IoT Greengrass Core usa o mínimo dos dois valores como o tamanho máximo total do log para esse componente.

`diskSpaceLimitUnit`

(Opcional) A unidade para `diskSpaceLimit` o. Escolha uma das seguintes opções:

- `KB`— kilobytes
- `MB`— megabytes
- `GB`— gigabytes

Padrão: `KB`

`logFileDirectoryPath`

(Opcional) O caminho para a pasta que contém os arquivos de log desse componente.

Você não precisa especificar esse parâmetro para componentes do Greengrass que imprimem na saída padrão (stdout) e no erro padrão (stderr).

Padrão: */greengrass/v2*`/logs`.

`logFileRegex`

(Opcional) Uma expressão regular que especifica o formato do nome do arquivo de log usado pelo componente ou aplicativo. O componente gerenciador de registros usa essa expressão regular para identificar arquivos de log na pasta em`logFileDirectoryPath`.

Você não precisa especificar esse parâmetro para componentes do Greengrass que imprimem na saída padrão (stdout) e no erro padrão (stderr).

Se seu componente ou aplicativo rotacionar arquivos de log, especifique um regex que corresponda aos nomes dos arquivos de log rotacionados. Por exemplo, você pode especificar **`hello_world\\\\w*.log`** o upload de registros para um aplicativo Hello World. O `\\\\w*` padrão corresponde a zero ou mais caracteres de palavras, o que inclui caracteres alfanuméricos e sublinhados. Esse regex corresponde aos arquivos de log com e sem carimbos de data/hora em seus nomes. Neste exemplo, o gerenciador de registros carrega os seguintes arquivos de log:

- `hello_world.log`— O arquivo de log mais recente do aplicativo Hello World.
- `hello_world_2020_12_15_17_0.log`— Um arquivo de log mais antigo para o aplicativo Hello World.

Padrão:*componentName*`\\\\w*.log`, em que *componentName* é o nome do componente dessa configuração de log.

`deleteLogFileAfterCloudUpload`

(Opcional) Você pode excluir um arquivo de registro depois que o componente gerenciador de registros fizer o upload dos registros para o CloudWatch Logs.

Padrão: `false`

`multiLineStartPattern`

(Opcional) Uma expressão regular que identifica quando uma mensagem de log em uma nova linha é uma nova mensagem de log. Se a expressão regular não corresponder à nova linha, o componente do gerenciador de registros anexará a nova linha à mensagem de registro da linha anterior.

Por padrão, o componente do gerenciador de registros verifica se a linha começa com um caractere de espaço em branco, como uma tabulação ou espaço. Caso contrário, o gerenciador de registros trata essa linha como uma nova mensagem de registro. Caso contrário, ele anexa essa linha à mensagem de log atual. Esse comportamento garante que o componente do gerenciador de registros não divida mensagens que se estendem por várias linhas, como rastreamentos de pilha.

`periodicUploadIntervalSec`

(Opcional) O período em segundos no qual o componente do gerenciador de registros verifica se há novos arquivos de log a serem carregados.

Padrão: `300` (5 minutos)

Mínimo: `0.000001` (1 microssegundo)

Example Exemplo: atualização da mesclagem de configurações

O exemplo de configuração a seguir especifica o upload de registros do sistema e de `com.example.HelloWorld` componentes para o CloudWatch Logs.

```
{
  "logsUploaderConfiguration": {
    "systemLogsConfiguration": {
      "uploadToCloudWatch": "true",
      "minimumLogLevel": "INFO",
      "diskSpaceLimit": "10",
      "diskSpaceLimitUnit": "MB",
```

```
      "deleteLogFileAfterCloudUpload": "false"
    },
    "componentLogsConfigurationMap": {
      "com.example.HelloWorld": {
        "minimumLogLevel": "INFO",
        "diskSpaceLimit": "20",
        "diskSpaceLimitUnit": "MB",
        "deleteLogFileAfterCloudUpload": "false"
      }
    }
  },
  "periodicUploadIntervalSec": "300"
}
```

v2.1.x

`logsUploaderConfiguration`

(Opcional) A configuração dos registros que o componente do gerenciador de registros carrega. Esse objeto contém as seguintes informações:

`systemLogsConfiguration`

(Opcional) A configuração dos registros do sistema de software AWS IoT Greengrass Core, que incluem registros do núcleo do Greengrass e dos componentes do plug-in. Especifique essa configuração para permitir que o componente do gerenciador de registros gerencie os registros do sistema. Esse objeto contém as seguintes informações:

`uploadToCloudWatch`

(Opcional) Você pode fazer upload dos registros do sistema para o CloudWatch Logs.

Padrão: `false`

`minimumLogLevel`

(Opcional) O nível mínimo de mensagens de registro a serem carregadas. Esse nível mínimo se aplica somente se você configurar o componente nucleus do Greengrass para gerar registros no formato JSON. Para ativar os registros no formato JSON, especifique `JSON` o parâmetro de formato de registro (`logging.format`).

Escolha entre os seguintes níveis de registro, listados aqui em ordem de nível:

- `DEBUG`

- `INFO`
- `WARN`
- `ERROR`

Padrão: `INFO`

`diskSpaceLimit`

(Opcional) O tamanho total máximo dos arquivos de log do sistema Greengrass, na unidade em que você especifica. `diskSpaceLimitUnit` Depois que o tamanho total dos arquivos de log do sistema Greengrass exceder esse tamanho total máximo, o software AWS IoT Greengrass Core exclui os arquivos de log mais antigos do sistema Greengrass.

Esse parâmetro é equivalente ao parâmetro de limite de tamanho de log (`totalLogsSizeKB`) do componente do núcleo do Greengrass. O software AWS IoT Greengrass Core usa o mínimo dos dois valores como o tamanho máximo total do log do sistema Greengrass.

`diskSpaceLimitUnit`

(Opcional) A unidade para `diskSpaceLimit` o. Escolha uma das seguintes opções:

- `KB`— kilobytes
- `MB`— megabytes
- `GB`— gigabytes

Padrão: `KB`

`deleteLogFileAfterCloudUpload`

(Opcional) Você pode excluir um arquivo de registro depois que o componente gerenciador de registros fizer o upload dos registros para o CloudWatch Logs.

Padrão: `false`

`componentLogsConfiguration`

(Opcional) Uma lista de configurações de log para componentes no dispositivo principal. Cada configuração nessa lista define a configuração de log de um componente ou aplicativo. O componente gerenciador de registros carrega esses registros de componentes para CloudWatch o Logs.

Cada objeto contém as seguintes informações:

`componentName`

O nome do componente ou aplicativo para essa configuração de log. Você pode especificar o nome de um componente do Greengrass ou outro valor para identificar esse grupo de registros.

`minimumLogLevel`

(Opcional) O nível mínimo de mensagens de registro a serem carregadas. Esse nível mínimo se aplica somente se os registros desse componente usarem um formato JSON específico, que você pode encontrar no repositório do módulo de AWS IoT Greengrass registro em. GitHub

Escolha entre os seguintes níveis de registro, listados aqui em ordem de nível:

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

Padrão: `INFO`

`diskSpaceLimit`

(Opcional) O tamanho total máximo de todos os arquivos de log desse componente, na unidade especificada`diskSpaceLimitUnit`. Depois que o tamanho total dos arquivos de log desse componente exceder esse tamanho total máximo, o software AWS IoT Greengrass Core exclui os arquivos de log mais antigos desse componente.

Esse parâmetro está relacionado ao parâmetro de limite de tamanho de log (`totalLogsSizeKB`) do componente do núcleo do Greengrass. O software AWS IoT Greengrass Core usa o mínimo dos dois valores como o tamanho máximo total do log para esse componente.

`diskSpaceLimitUnit`

(Opcional) A unidade para `diskSpaceLimit` o. Escolha uma das seguintes opções:

- `KB`— kilobytes
- `MB`— megabytes

- `GB`— gigabytes

Padrão: `KB`

`logFileDirectoryPath`

(Opcional) O caminho para a pasta que contém os arquivos de log desse componente.

Você não precisa especificar esse parâmetro para componentes do Greengrass que imprimem na saída padrão (stdout) e no erro padrão (stderr).

Padrão: *`/greengrass/v2`*`/logs`.

`logFileRegex`

(Opcional) Uma expressão regular que especifica o formato do nome do arquivo de log usado pelo componente ou aplicativo. O componente gerenciador de registros usa essa expressão regular para identificar arquivos de log na pasta em`logFileDirectoryPath`.

Você não precisa especificar esse parâmetro para componentes do Greengrass que imprimem na saída padrão (stdout) e no erro padrão (stderr).

Se seu componente ou aplicativo rotacionar arquivos de log, especifique um regex que corresponda aos nomes dos arquivos de log rotacionados. Por exemplo, você pode especificar **`hello_world\\\\w*.log`** o upload de registros para um aplicativo Hello World. O `\\\\w*` padrão corresponde a zero ou mais caracteres de palavras, o que inclui caracteres alfanuméricos e sublinhados. Esse regex corresponde aos arquivos de log com e sem carimbos de data/hora em seus nomes. Neste exemplo, o gerenciador de registros carrega os seguintes arquivos de log:

- `hello_world.log`— O arquivo de log mais recente do aplicativo Hello World.
- `hello_world_2020_12_15_17_0.log`— Um arquivo de log mais antigo para o aplicativo Hello World.

Padrão:*`componentName`*`\\\\w*.log`, em que *`componentName`* é o nome do componente dessa configuração de log.

`deleteLogFileAfterCloudUpload`

(Opcional) Você pode excluir um arquivo de registro depois que o componente gerenciador de registros fizer o upload dos registros para o CloudWatch Logs.

Padrão: `false`

`multiLineStartPattern`

(Opcional) Uma expressão regular que identifica quando uma mensagem de log em uma nova linha é uma nova mensagem de log. Se a expressão regular não corresponder à nova linha, o componente do gerenciador de registros anexará a nova linha à mensagem de registro da linha anterior.

Por padrão, o componente do gerenciador de registros verifica se a linha começa com um caractere de espaço em branco, como uma tabulação ou espaço. Caso contrário, o gerenciador de registros trata essa linha como uma nova mensagem de registro. Caso contrário, ele anexa essa linha à mensagem de log atual. Esse comportamento garante que o componente do gerenciador de registros não divida mensagens que se estendem por várias linhas, como rastreamentos de pilha.

`periodicUploadIntervalSec`

(Opcional) O período em segundos no qual o componente do gerenciador de registros verifica se há novos arquivos de log a serem carregados.

Padrão: `300` (5 minutos)

Mínimo: `0.000001` (1 microssegundo)

Example Exemplo: atualização da mesclagem de configurações

O exemplo de configuração a seguir especifica o upload de registros do sistema e de `com.example.HelloWorld` componentes para o CloudWatch Logs.

```
{
  "logsUploaderConfiguration": {
    "systemLogsConfiguration": {
      "uploadToCloudWatch": "true",
      "minimumLogLevel": "INFO",
      "diskSpaceLimit": "10",
      "diskSpaceLimitUnit": "MB",
      "deleteLogFileAfterCloudUpload": "false"
    },
    "componentLogsConfiguration": [
      {
```

```
        "componentName": "com.example.HelloWorld",
        "minimumLogLevel": "INFO",
        "diskSpaceLimit": "20",
        "diskSpaceLimitUnit": "MB",
        "deleteLogFileAfterCloudUpload": "false"
      }
    ]
  },
  "periodicUploadIntervalSec": "300"
}
```

v2.0.x

`logsUploaderConfiguration`

(Opcional) A configuração dos registros que o componente do gerenciador de registros carrega. Esse objeto contém as seguintes informações:

`systemLogsConfiguration`

(Opcional) A configuração dos registros do sistema de software AWS IoT Greengrass Core. Especifique essa configuração para permitir que o componente do gerenciador de registros gerencie os registros do sistema. Esse objeto contém as seguintes informações:

`uploadToCloudWatch`

(Opcional) Você pode fazer upload dos registros do sistema para o CloudWatch Logs.

Padrão: `false`

`minimumLogLevel`

(Opcional) O nível mínimo de mensagens de registro a serem carregadas. Esse nível mínimo se aplica somente se você configurar o componente nucleus do Greengrass para gerar registros no formato JSON. Para ativar os registros no formato JSON, especifique `JSON` o parâmetro de formato de registro (`logging.format`).

Escolha entre os seguintes níveis de registro, listados aqui em ordem de nível:

- `DEBUG`
- `INFO`
- `WARN`

- `ERROR`

Padrão: `INFO`

`diskSpaceLimit`

(Opcional) O tamanho total máximo dos arquivos de log do sistema Greengrass, na unidade em que você especifica. `diskSpaceLimitUnit` Depois que o tamanho total dos arquivos de log do sistema Greengrass exceder esse tamanho total máximo, o software AWS IoT Greengrass Core exclui os arquivos de log mais antigos do sistema Greengrass.

Esse parâmetro é equivalente ao parâmetro de limite de tamanho de log (`totalLogsSizeKB`) do componente do núcleo do Greengrass. O software AWS IoT Greengrass Core usa o mínimo dos dois valores como o tamanho máximo total do log do sistema Greengrass.

`diskSpaceLimitUnit`

(Opcional) A unidade para `diskSpaceLimit` o. Escolha uma das seguintes opções:

- `KB`— kilobytes
- `MB`— megabytes
- `GB`— gigabytes

Padrão: `KB`

`deleteLogFileAfterCloudUpload`

(Opcional) Você pode excluir um arquivo de registro depois que o componente gerenciador de registros fizer o upload dos registros para o CloudWatch Logs.

Padrão: `false`

`componentLogsConfiguration`

(Opcional) Uma lista de configurações de log para componentes no dispositivo principal. Cada configuração nessa lista define a configuração de log de um componente ou aplicativo. O componente gerenciador de registros carrega esses registros de componentes para CloudWatch o Logs.

Cada objeto contém as seguintes informações:

`componentName`

O nome do componente ou aplicativo para essa configuração de log. Você pode especificar o nome de um componente do Greengrass ou outro valor para identificar esse grupo de registros.

`minimumLogLevel`

(Opcional) O nível mínimo de mensagens de registro a serem carregadas. Esse nível mínimo se aplica somente se os registros desse componente usarem um formato JSON específico, que você pode encontrar no repositório do módulo de AWS IoT Greengrass registro em. GitHub

Escolha entre os seguintes níveis de registro, listados aqui em ordem de nível:

- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

Padrão: `INFO`

`diskSpaceLimit`

(Opcional) O tamanho total máximo de todos os arquivos de log desse componente, na unidade especificada`diskSpaceLimitUnit`. Depois que o tamanho total dos arquivos de log desse componente exceder esse tamanho total máximo, o software AWS IoT Greengrass Core exclui os arquivos de log mais antigos desse componente.

Esse parâmetro está relacionado ao parâmetro de limite de tamanho de log (`totalLogsSizeKB`) do componente do núcleo do Greengrass. O software AWS IoT Greengrass Core usa o mínimo dos dois valores como o tamanho máximo total do log para esse componente.

`diskSpaceLimitUnit`

(Opcional) A unidade para `diskSpaceLimit` o. Escolha uma das seguintes opções:

- `KB`— kilobytes
- `MB`— megabytes
- `GB`— gigabytes

Padrão: KB

`logFileDirectoryPath`

O caminho para a pasta que contém os arquivos de log desse componente.

Para carregar os registros de um componente do Greengrass***/greengrass/v2*****/logs**, especifique e */greengrass/v2* substitua pela pasta raiz do Greengrass.

`logFileRegex`

Uma expressão regular que especifica o formato do nome do arquivo de log usado pelo componente ou aplicativo. O componente gerenciador de registros usa essa expressão regular para identificar arquivos de log na pasta em`logFileDirectoryPath`.

Para carregar os registros de um componente do Greengrass, especifique um regex que corresponda aos nomes dos arquivos de log rotacionados. Por exemplo, você pode especificar **`com.example.HelloWorld\\w*.log`** o upload de registros para um componente Hello World. O `\\w*` padrão corresponde a zero ou mais caracteres de palavras, o que inclui caracteres alfanuméricos e sublinhados. Esse regex corresponde aos arquivos de log com e sem carimbos de data/hora em seus nomes. Neste exemplo, o gerenciador de registros carrega os seguintes arquivos de log:

- `com.example.HelloWorld.log`— O arquivo de log mais recente do componente Hello World.
- `com.example.HelloWorld_2020_12_15_17_0.log`— Um arquivo de log antigo para o componente Hello World. O núcleo do Greengrass adiciona um timestamp rotativo aos arquivos de log.

`deleteLogFileAfterCloudUpload`

(Opcional) Você pode excluir um arquivo de registro depois que o componente gerenciador de registros fizer o upload dos registros para o CloudWatch Logs.

Padrão: `false`

`multiLineStartPattern`

(Opcional) Uma expressão regular que identifica quando uma mensagem de log em uma nova linha é uma nova mensagem de log. Se a expressão regular não corresponder à nova linha, o componente do gerenciador de registros anexará a nova linha à mensagem de registro da linha anterior.

Por padrão, o componente do gerenciador de registros verifica se a linha começa com um caractere de espaço em branco, como uma tabulação ou espaço. Caso contrário, o gerenciador de registros trata essa linha como uma nova mensagem de registro. Caso contrário, ele anexa essa linha à mensagem de log atual. Esse comportamento garante que o componente do gerenciador de registros não divida mensagens que se estendem por várias linhas, como rastreamentos de pilha.

`periodicUploadIntervalSec`

(Opcional) O período em segundos no qual o componente do gerenciador de registros verifica se há novos arquivos de log a serem carregados.

Padrão: `300` (5 minutos)

Mínimo: `0.000001` (1 microssegundo)

Example Exemplo: atualização da mesclagem de configurações

O exemplo de configuração a seguir especifica o upload de registros do sistema e de `com.example.HelloWorld` componentes para o CloudWatch Logs.

```
{
  "logsUploaderConfiguration": {
    "systemLogsConfiguration": {
      "uploadToCloudWatch": "true",
      "minimumLogLevel": "INFO",
      "diskSpaceLimit": "10",
      "diskSpaceLimitUnit": "MB",
      "deleteLogFileAfterCloudUpload": "false"
    },
    "componentLogsConfiguration": [
      {
        "componentName": "com.example.HelloWorld",
        "minimumLogLevel": "INFO",
        "logFileDirectoryPath": "/greengrass/v2/logs",
        "logFileRegex": "com.example.HelloWorld\\w*.log",
        "diskSpaceLimit": "20",
        "diskSpaceLimitUnit": "MB",
        "deleteLogFileAfterCloudUpload": "false"
      }
    ]
  },
```

```
  "periodicUploadIntervalSec": "300"
}
```

Uso

O componente do gerenciador de registros é carregado nos seguintes grupos de registros e fluxos de registros.

2.1.0 and later

Nome do grupo de registros

```
/aws/greengrass/componentType/region/componentName
```

O nome do grupo de registros usa as seguintes variáveis:

- `componentType`— O tipo do componente, que pode ser um dos seguintes:
  - `GreengrassSystemComponent`— Esse grupo de registros inclui registros do núcleo e dos componentes do plug-in, que são executados na mesma JVM do núcleo do Greengrass. O componente faz parte do núcleo do Greengrass.
  - `UserComponent`— Esse grupo de registros inclui registros de componentes genéricos, componentes Lambda e outros aplicativos no dispositivo. O componente não faz parte do núcleo do Greengrass.

  Para ter mais informações, consulte Tipos de componentes.
- `region`— A AWS região que o dispositivo principal usa.
- `componentName`— O nome do componente. Para registros do sistema, esse valor é`System`.

Nome do fluxo de log

```
/date/thing/thingName
```

O nome do fluxo de log usa as seguintes variáveis:

- `date`— A data do registro, como`2020/12/15`. O componente do gerenciador de registros usa o `yyyy/MM/dd` formato.
- `thingName`— O nome do dispositivo principal.

> **Note**
>
> Se o nome de uma coisa contiver dois pontos (`:`), o gerenciador de registros substituirá os dois pontos por um sinal de adição (). +

2.0.x

Nome do grupo de registros

```
/aws/greengrass/componentType/region/componentName
```

O nome do grupo de registros usa as seguintes variáveis:

- `componentType`— O tipo do componente, que pode ser um dos seguintes:
  - `GreengrassSystemComponent`— O componente faz parte do núcleo do Greengrass.
  - `UserComponent`— O componente não faz parte do núcleo do Greengrass. O gerenciador de registros usa esse tipo para componentes do Greengrass e outros aplicativos no dispositivo.
- `region`— A AWS região que o dispositivo principal usa.
- `componentName`— O nome do componente. Para registros do sistema, esse valor é`System`.

Nome do fluxo de log

```
/date/deploymentTargets/thingName
```

O nome do fluxo de log usa as seguintes variáveis:

- `date`— A data do registro, como`2020/12/15`. O componente do gerenciador de registros usa o `yyyy/MM/dd` formato.
- `deploymentTargets`— As coisas cujas implantações incluem o componente. O componente do gerenciador de registros separa cada destino por uma barra. Se o componente for executado no dispositivo principal como resultado de uma implantação local, esse valor será`LOCAL_DEPLOYMENT`.

  Considere um exemplo em que você tem um dispositivo principal chamado`MyGreengrassCore`, e o dispositivo principal tem duas implantações:

  - Uma implantação que tem como alvo o dispositivo principal,`MyGreengrassCore`.
  - Uma implantação que tem como alvo um grupo de coisas chamado`MyGreengrassCoreGroup`, que contém o dispositivo principal.

  Os `deploymentTargets` para este dispositivo principal são`thing/MyGreengrassCore/thinggroup/MyGreengrassCoreGroup`.
- `thingName`— O nome do dispositivo principal.

Formatos para entradas de registro.

O núcleo do Greengrass grava arquivos de log no formato string ou JSON. Para registros do sistema, você controla o formato definindo o `format` campo da `logging` entrada. Você pode encontrar a `logging` entrada no arquivo de configuração do componente do núcleo do Greengrass. Para obter mais informações, consulte Configuração do núcleo do Greengrass.

O formato do texto é de formato livre e aceita qualquer sequência de caracteres. A seguinte mensagem do serviço de status da frota é um exemplo de registro em formato de string:

```
2023-03-26T18:18:27.271Z [INFO] (pool-1-thread-2)
com.aws.greengrass.status.FleetStatusService: fss-status-update-published.
Status update published to FSS. {trigger=CADENCE, serviceName=FleetStatusService,
currentState=RUNNING}
```

Você deve usar o formato JSON se quiser visualizar os registros com o comando logs do Greengrass CLI ou interagir com os registros programaticamente. O exemplo a seguir descreve a forma JSON:

```
{
   "loggerName": <string>,
   "level": <"DEBUG" | "INFO" | "ERROR" | "TRACE" | "WARN">,
   "eventType": <string, optional>,
   "cause": <string, optional>,
   "contexts": {},
   "thread": <string>,
   "message": <string>,
   "timestamp": <epoch time> # Needs to be epoch time
}
```

Para controlar a saída dos registros do seu componente, você pode usar a opção `minimumLogLevel` de configuração. Para usar essa opção, seu componente deve gravar suas

entradas de registro no formato JSON. Você deve usar o mesmo formato do arquivo de log do sistema.

Arquivo de log local

Esse componente usa o mesmo arquivo de log do componente do núcleo do Greengrass.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

Para ver os registros desse componente

- Execute o comando a seguir no dispositivo principal para visualizar o arquivo de log desse componente em tempo real. Substitua */greengrass/v2* ou *C:\greengrass\v2* pelo caminho para a pasta AWS IoT Greengrass raiz.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

Changelog

A tabela a seguir descreve as alterações em cada versão do componente.

Version (Versão)	Alterações
2.3.7	Versão atualizada para a versão 2.12.0 do Greengrass nucleus.

Version (Versão)	Alterações
2.3.6	Correções de erros e melhorias • Ajusta os níveis de log para determinados erros.
2.3.5	Melhorias Melhora a velocidade de upload dos registros. Versão atualizada para a versão 2.11.0 do Greengrass nucleus.
2.3.4	Correções de erros e melhorias • Adiciona suporte para definir o `periodicUploadIntervalSec` parâmetro para valores fracionários. O mínimo é de 1 microssegundo. • Corrige um problema em que o gerenciador de registros não respeita os CloudWatch `putLogEvents` limites.
2.3.3	Versão atualizada para a versão 2.10.0 do Greengrass nucleus.
2.3.2	Correções de erros e melhorias • Melhora o gerenciamento do espaço para que os arquivos de log não sejam excluídos antes de serem carregados. • Corrige problemas com o gerenciamento de cache. • Correções e melhorias adicionais de pequenos bugs.
2.3.1	Correções de erros e melhorias • Corrige um problema em que grupos de arquivos de destino com vários arquivos de log ativos carregam entradas duplicadas para. CloudWatch • Correções e melhorias adicionais de pequenos bugs.

Version (Versão)	Alterações
2.3.0	**Note** Recomendamos que você atualize para o Greengrass nucleus 2.9.1 ao atualizar para o log manager 2.3.0. Novos atributos Reduz os atrasos no registro processando e carregando diretamente os arquivos de log ativos, em vez de esperar que novos arquivos sejam alternados. Correções de erros e melhorias • Melhora o suporte à rotação de registros ao girar arquivos com um nome exclusivo. • Correções e melhorias adicionais de pequenos bugs.
2.2.8	Versão atualizada para a versão 2.9.0 do Greengrass nucleus.
2.2.7	Versão atualizada para a versão 2.8.0 do Greengrass nucleus.
2.2.6	Versão atualizada para a versão 2.7.0 do Greengrass nucleus.
2.2.5	Versão atualizada para a versão 2.6.0 do Greengrass nucleus.
2.2.4	Correções de erros e melhorias • Melhora a estabilidade ao lidar com configurações inválidas. • Pequenas correções e melhorias adicionais.

Version (Versão)	Alterações
2.2.3	Correções de erros e melhorias • Melhora a estabilidade em determinados cenários em que o component e é reiniciado ou encontra erros. • Corrige problemas em que mensagens de log grandes e arquivos de log grandes não são carregados em determinados cenários. • Corrige problemas com a forma como esse componente lida com as atualizações de redefinição de configuração. • Corrige um problema em que um valor de `null` `diskSpaceLimit` configuração impedia a implantação do componente.
2.2.2	Correções de erros e melhorias • Adiciona suporte para mensagens de log maiores que 256 kilobytes. O componente gerenciador de registros divide essas grandes mensagens de log em várias mensagens com o mesmo timestamp do evento de log.
2.2.1	Versão atualizada para a versão 2.5.0 do Greengrass nucleus.
2.2.0	Novo recurso • Adiciona o parâmetro `componentLogsConfigurationMap` de configuração para oferecer suporte a um formato de mapa para configurações de log de componentes. Cada `componentName` objeto no mapa define a configuração de log para um componente ou aplicativ o.
2.1.3	Versão atualizada para a versão 2.4.0 do Greengrass nucleus.
2.1.2	Versão atualizada para a versão 2.3.0 do Greengrass nucleus.
2.1.1	Correções de erros e melhorias • Corrige um problema em que a configuração do log do sistema não foi atualizada em alguns casos.

Version (Versão)	Alterações
2.1.0	Correções de erros e melhorias • Use padrões para `logFileDirectoryPath` e `logFileRegex` que funcionem para componentes do Greengrass que imprimem na saída padrão (stdout) e no erro padrão (stderr). • Direcione corretamente o tráfego por meio de um proxy de rede configurado ao fazer o upload dos registros para o CloudWatch Logs. • Manipule corretamente os caracteres de dois pontos (`:`) nos nomes dos fluxos de log. CloudWatch Os nomes dos fluxos de registro de registros não oferecem suporte a dois pontos. • Simplifique os nomes do fluxo de log removendo os nomes dos grupos de coisas do fluxo de log. • Remova uma mensagem de registro de erros que é impressa durante o comportamento normal.
2.0.x	Versão inicial.

Componentes de aprendizado de máquina

AWS IoT Greengrassfornece os seguintes componentes de aprendizado de máquina que você pode implantar em dispositivos compatíveis para realizar inferência de aprendizado de máquina usando modelos treinados na Amazon SageMaker ou com seus próprios modelos pré-treinados armazenados no Amazon S3.

AWSfornece as seguintes categorias de componentes de aprendizado de máquina:

- Componente de modelo — contém modelos de aprendizado de máquina como artefatos do Greengrass.
- Componente de tempo de execução — contém o script que instala a estrutura de aprendizado de máquina e suas dependências no dispositivo principal do Greengrass.
- Componente de inferência — contém o código de inferência e inclui dependências de componentes para instalar a estrutura de aprendizado de máquina e baixar modelos de aprendizado de máquina pré-treinados.

Você pode usar o código de inferência de amostra e os modelos pré-treinados nos componentes de aprendizado AWS de máquina fornecidos para realizar a classificação de imagens e a detecção de objetos usando DLR e Lite. TensorFlow Para realizar inferências personalizadas de aprendizado de máquina com seus próprios modelos armazenados no Amazon S3 ou para usar uma estrutura de aprendizado de máquina diferente, você pode usar as receitas desses componentes públicos como modelos para criar componentes personalizados de aprendizado de máquina. Para ter mais informações, consulte Personalize seus componentes de aprendizado de máquina.

AWS IoT Greengrasstambém inclui um componente AWS fornecido para gerenciar a instalação e o ciclo de vida do agente SageMaker Edge Manager nos dispositivos principais do Greengrass. Com o SageMaker Edge Manager, você pode usar modelos SageMaker compilados pelo Amazon Neo diretamente no seu dispositivo principal. Para ter mais informações, consulte Use o Amazon SageMaker Edge Manager nos dispositivos principais do Greengrass.

A tabela a seguir lista os componentes de aprendizado de máquina que estão disponíveis noAWS IoT Greengrass.

> **Note**
>
> Vários componentes AWS fornecidos dependem de versões secundárias específicas do núcleo do Greengrass. Por causa dessa dependência, você precisa atualizar esses componentes ao atualizar o núcleo do Greengrass para uma nova versão secundária. Para obter informações sobre as versões específicas do núcleo das quais cada componente depende, consulte o tópico do componente correspondente. Para obter mais informações sobre a atualização do núcleo, consulteAtualize o software AWS IoT Greengrass principal (OTA).

Quando um componente tem um tipo de componente genérico e Lambda, a versão atual do componente é do tipo genérico e uma versão anterior do componente é do tipo Lambda.

Componente	Descrição	Tipo de componente	SO com suporte	Código aberto
Lookout for Vision Edge Agent	Implanta o tempo de execução	Genérico	Linux	Não

Componente	Descrição	Tipo de componente	SO com suporte	Código aberto
	do Amazon Lookout for Vision no dispositivo principal do Greengrass, para que você possa usar a visão computacional para encontrar defeitos em produtos industriais.			
SageMaker Gerente de borda	Implanta o agente Amazon SageMaker Edge Manager no dispositivo principal do Greengrass.	Genérico	Linux, Windows	Não

Componente	Descrição	Tipo de componente	SO com suporte	Código aberto
Classificação de imagens DLR	Componente de inferência que usa o repositório de modelos de classificação de imagem DLR e o componente de tempo de execução do DLR como dependências para instalar o DLR, baixar exemplos de modelos de classificação de imagens e realizar inferência de classificação de imagens em dispositivos compatíveis.	Genérico	Linux, Windows	Não

Componente	Descrição	Tipo de componente	SO com suporte	Código aberto
Detecção de objetos DLR	Componente de inferência que usa o repositório de modelos de detecção de objetos DLR e o componente de tempo de execução do DLR como dependências para instalar o DLR, baixar modelos de detecção de objetos de amostra e realizar inferência de detecção de objetos em dispositivos compatíveis.	Genérico	Linux, Windows	Não

Componente	Descrição	Tipo de componente	SO com suporte	Código aberto
Armazenamento de modelos de classificação de imagens DLR	Componente de modelo que contém amostras de ResNet -50 modelos de classificação de imagens como artefatos do Greengrass.	Genérico	Linux, Windows	Não
Armazenamento de modelos de detecção de objetos DLR	Componente de modelo que contém exemplos de modelos de detecção de objetos YOLOv3 como artefatos do Greengrass.	Genérico	Linux, Windows	Não

Componente	Descrição	Tipo de componente	SO com suporte	Código aberto
Tempo de execução do DLR	Componente de tempo de execução que contém um script de instalação usado para instalar o DLR e suas dependências no dispositivo principal do Greengrass.	Genérico	Linux, Windows	Não

Componente	Descrição	Tipo de componente	SO com suporte	Código aberto
TensorFlow Classificação de imagens Lite	Componente de inferência que usa o TensorFlow repositório de modelos de classificação de imagem TensorFlow Lite e o componente de tempo de execução Lite como dependências para instalar o TensorFlow Lite, baixar exemplos de modelos de classificação de imagens e realizar inferência de classificação de imagens em dispositivos compatíveis.	Genérico	Linux, Windows	Não

Componente	Descrição	Tipo de componente	SO com suporte	Código aberto
TensorFlow Detecção leve de objetos	Componente de inferência que usa o TensorFlow repositório de modelos de detecção de objetos TensorFlow Lite e o componente de tempo de execução Lite como dependências para instalar o TensorFlow Lite, baixar modelos de detecção de objetos de amostra e realizar inferência de detecção de objetos em dispositivos compatíveis.	Genérico	Linux, Windows	Não

Componente	Descrição	Tipo de componente	SO com suporte	Código aberto
TensorFlow Loja de modelos de classificação de imagens Lite	Componente de modelo que contém um modelo MobileNet v1 de amostra como um artefato do Greengrass.	Genérico	Linux, Windows	Não
TensorFlow Loja de modelos de detecção de objetos Lite	Componente de modelo que contém um MobileNet modelo de amostra de detecção de disparo único (SSD) como um artefato do Greengrass.	Genérico	Linux, Windows	Não

Componente	Descrição	Tipo de componente	SO com suporte	Código aberto
TensorFlow Tempo de execução leve	Componente de tempo de execução que contém um script de instalação usado para instalar o TensorFlow Lite e suas dependências no dispositivo principal do Greengrass.	Genérico	Linux, Windows	Não

Lookout for Vision Edge Agent

O componente Lookout for Vision Edge Agent `aws.iot.lookoutvision.EdgeAgent` () instala um servidor de tempo de execução local do Amazon Lookout for Vision, que usa visão computacional para encontrar defeitos visuais em produtos industriais.

Para usar esse componente, crie e implante componentes do modelo de aprendizado de máquina do Lookout for Vision. Esses modelos de aprendizado de máquina preveem a presença de anomalias nas imagens ao encontrar padrões nas imagens que você usa para treinar o modelo. Em seguida, você pode desenvolver e implantar componentes personalizados do Greengrass, chamados de componentes do aplicativo cliente, que fornecem imagens e fluxos de vídeo a esse componente de tempo de execução para detectar anomalias usando os modelos de aprendizado de máquina.

Você pode usar a API do Lookout for Vision Edge Agent para interagir com esse componente a partir de outros componentes do Greengrass. Essa API é implementada usando o gRPC, que é um protocolo para fazer chamadas de procedimentos remotos. Para obter mais informações, consulte Como escrever um componente de aplicativo cliente e a referência da API do Lookout for Vision Edge Agent no Guia do desenvolvedor do Amazon Lookout for Vision.

Para obter mais informações sobre como usar esse componente, consulte o seguinte:

- Amazon Loout Lookout for Vision
- O que é o Amazon Lookout for Vision? no Guia do desenvolvedor do Amazon Lookout for Vision
- Criação de um modelo do Lookout for Vision no Guia do desenvolvedor do Amazon Lookout for Vision.
- Usando um modelo Lookout for Vision em um dispositivo de ponta no Amazon Lookout for Vision Developer Guide.

> **Note**
>
> O componente Lookout for Vision Edge Agent está disponível somente no Regiões da AWS seguinte:
>
> - Leste dos EUA (Ohio)
> - Leste dos EUA (N. da Virgínia)
> - Oeste dos EUA (Oregon)
> - Europa (Frankfurt)
> - Europa (Irlanda)
> - Ásia-Pacífico (Tóquio)
> - Ásia-Pacífico (Seul)

Tópicos

Versões

Esse componente tem as seguintes versões:

- 1,2.x
- 1.1.x
- 1,0.x
- 0,1x

Tipo

Este componente é um componente genérico (`aws.greengrass.generic`). O núcleo do Greengrass executa os scripts do ciclo de vida do componente.

Para ter mais informações, consulte Tipos de componentes.

Sistema operacional

Esse componente pode ser instalado somente nos dispositivos principais do Linux.

Requisitos

Esse componente tem os seguintes requisitos:

- O dispositivo principal do Greengrass deve usar uma arquitetura Armv8 (AArch64) ou x86_64.
- Se você usa a versão 1.0.0 ou posterior desse componente, o Python 3.8 ou o Python 3.9, `pip` inclusive, instalado no dispositivo principal do Greengrass.

  Se você usa a versão 0.1.x desse componente, o Python 3.7, inclusive, `pip` está instalado no dispositivo principal do Greengrass.

  > **Important**
  >
  > O dispositivo deve ter uma dessas versões exatas do Python. Esse componente não é compatível com versões posteriores do Python.

- Para usar a inferência de unidade de processamento gráfico (GPU), o dispositivo principal deve atender aos seguintes requisitos. A inferência de GPU é opcional na versão 1.1.0 e posterior desse componente.

- Uma unidade de processamento gráfico (GPU) compatível com CUDA. Para obter mais informações, consulte Verificar se você tem uma GPU compatível com CUDA na documentação do kit de ferramentas CUDA.
- cuDNN, CUDA e TensorRT instalados no dispositivo principal do Greengrass.
  - Nos dispositivos NVIDIA Jetson, como o Jetson Nano ou o Jetson Xavier, o cuDNN, o CUDA e o TensorRT vêm instalados com a NVIDIA. JetPack Você não precisa fazer nenhuma alteração. Esse componente suporta JetPack 4.4, JetPack4.5, JetPack 4.5.1 e JetPack4.6.1.

    > **Important**
    >
    > Você deve instalar uma dessas versões do JetPack e não outra versão. O serviço Lookout for Vision compila modelos de visão computacional para JetPack essas plataformas.

  - Em dispositivos x86 com uma GPU que tenha a microarquitetura NVIDIA Ampere (ou a capacidade de computação da GPU seja 8.0), faça o seguinte:
    - Instale o cuDNN seguindo as instruções no Guia de instalação do cuDNN da NVIDIA.
    - Instale o CUDA versão 11.2 seguindo as instruções no Guia de instalação do NVIDIA CUDA para Linux.
    - Instale o TensorRT versão 8.2.0 seguindo as instruções na documentação do NVIDIA TensorRT.
  - Em dispositivos x86 com uma GPU que tenha uma arquitetura NVIDIA anterior à Ampere (ou a capacidade de computação da GPU seja menor que 8,0), faça o seguinte:
    - Instale o cuDNN seguindo as instruções no Guia de instalação do cuDNN da NVIDIA.
    - Instale o CUDA versão 10.2 seguindo as instruções no Guia de instalação do NVIDIA CUDA para Linux.
    - Instale o TensorRT versão 7.1.3 ou posterior, mas anterior à versão 8.0.0, seguindo as instruções na documentação do NVIDIA TensorRT.
- O usuário do sistema que executa esse componente deve ser membro do grupo do sistema que tem acesso à GPU no dispositivo. O nome desse grupo difere de acordo com o sistema operacional. Consulte a documentação do sistema operacional e da GPU para determinar o nome desse grupo de sistemas.

Por exemplo, em dispositivos NVIDIA Jetson, o nome desse grupo é `video`, e você pode executar o comando a seguir para adicionar um usuário do sistema a esse grupo. Substitua *`ggc_user`* *`pelo`* nome do usuário a ser adicionado.

```
sudo usermod -aG video ggc_user
```

Dependências

Esse componente não tem nenhuma dependência.

Configuração

Esse componente fornece os seguintes parâmetros de configuração que você pode personalizar ao implantar o componente.

`Socket`

(Opcional) O soquete de arquivo em que o Edge Agent opera. Os componentes do modelo Lookout for Vision usam esse soquete de arquivo para se comunicar com o Edge Agent. Se você alterar esse parâmetro, deverá especificar o mesmo valor ao implantar os componentes do modelo Lookout for Vision.

Padrão: `unix:///tmp/aws.iot.lookoutvision.EdgeAgent.sock`

Arquivo de log local

Esse componente usa o seguinte arquivo de log.

```
/greengrass/v2/logs/aws.iot.lookoutvision.EdgeAgent.log
```

Para ver os registros desse componente

- Execute o comando a seguir no dispositivo principal para visualizar o arquivo de log desse componente em tempo real. */greengrass/v2*Substitua pelo caminho para a pasta AWS IoT Greengrass raiz.

  ```
  sudo tail -f /greengrass/v2/logs/aws.iot.lookoutvision.EdgeAgent.log
  ```

Changelog

A tabela a seguir descreve as alterações em cada versão do componente.

Version (Versão)	Alterações
1.2.0	Melhorias e correções de erros gerais.
1.1.9	Melhorias e correções de erros gerais.
1.1.8	Melhorias e correções de erros gerais.
1.1.7	Novos atributos • Instala o `opencv-python-headless` pacote no ambiente virtual do Lookout for Vision Edge Agent. Correções de erros e melhorias • Melhora o cálculo da pontuação de confiança. • Redimensiona a máscara do modelo de mapa de calor para o tamanho original do arquivo. • Melhorias e correções de erros gerais.
1.1.6	Novos atributos Novos valores adicionados ao `DetectAnomalies` resultado. • `anomaly_score` — O número entre 0,0 e 1,0 que indica o quão anômala é uma imagem. • `anomaly_threshold` — Limite definido durante o treinamento do modelo que determina o limite entre uma imagem anômala e uma imagem normal. Melhorias e correções de erros gerais.
1.1.4	Novos atributos Foi adicionado suporte ao OpenCV para redimensionamento de imagens quando disponível. O agente Edge usa o Pillow quando o OpenCV não está disponível.

Version (Versão)	Alterações
	Correções de erros e melhorias Melhorias e correções de erros gerais.
1.1.3	Melhorias e correções de erros gerais.
1.1.1	Melhorias e correções de erros gerais.
1.1.0	Novos atributos • Adiciona suporte para modelos de segmentação de imagens, que identificam anomalias nas imagens. • Adiciona suporte para inferência de CPU, para que você possa usar os modelos Lookout for Vision em dispositivos principais sem uma GPU. Correções de erros e melhorias • Melhorias e correções de erros gerais.
1.0.0	Essa versão do componente Lookout for Vision Edge Agent requer uma versão do Python diferente da versão 0.1.x. Se você quiser atualizar da v0.1.x para a v1.x, você deve atualizar a instalação do Python no dispositivo principal. Correções de erros e melhorias • Melhorias e correções de erros gerais.
0.1.37	Melhorias e correções de erros gerais.
0.1.36	Versão inicial.

SageMaker Gerente de borda

> **Important**
>
> SageMaker O Edge Manager será descontinuado em 26 de abril de 2024. Para obter mais informações sobre como continuar implantando seus modelos em dispositivos de ponta, consulte Fim da vida útil do SageMaker Edge Manager.

O componente Amazon SageMaker Edge Manager (`aws.greengrass.SageMakerEdgeManager`) instala o binário do agente do SageMaker Edge Manager.

SageMaker O Edge Manager fornece gerenciamento de modelos para dispositivos periféricos para que você possa otimizar, proteger, monitorar e manter modelos de aprendizado de máquina em frotas de dispositivos periféricos. O componente SageMaker Edge Manager instala e gerencia o ciclo de vida do agente do SageMaker Edge Manager em seu dispositivo principal. Você também pode usar o SageMaker Edge Manager para empacotar e usar modelos SageMaker compilados pelo NEO como componentes do modelo nos dispositivos principais do Greengrass. Para obter mais informações sobre como usar o agente SageMaker Edge Manager em seu dispositivo principal, consulteUse o Amazon SageMaker Edge Manager nos dispositivos principais do Greengrass.

SageMaker O componente Edge Manager v1.3.x instala o binário do agente Edge Manager v1.20220822.836f3023. Para obter mais informações sobre as versões binárias do agente Edge Manager, consulte Agente do Edge Manager.

> **Note**
>
> O componente SageMaker Edge Manager está disponível somente no seguinte Regiões da AWS:
>
> - Leste dos EUA (Ohio)
> - Leste dos EUA (N. da Virgínia)
> - Oeste dos EUA (Oregon)
> - UE (Frankfurt)
> - UE (Irlanda)
> - Ásia-Pacífico (Tóquio)

Tópicos

- Arquivo de log local
- Changelog

Versões

Esse componente tem as seguintes versões:

- 1.3.x
- 1,2.x
- 1.1.x
- 1,0.x

Tipo

Este componente é um componente genérico (`aws.greengrass.generic`). O núcleo do Greengrass executa os scripts do ciclo de vida do componente.

Para ter mais informações, consulte Tipos de componentes.

Sistema operacional

Esse componente pode ser instalado em dispositivos principais que executam os seguintes sistemas operacionais:

- Linux
- Windows

Requisitos

Esse componente tem os seguintes requisitos:

- Um dispositivo principal do Greengrass executado no Amazon Linux 2, uma plataforma Linux baseada em Debian (x86_64 ou Armv8) ou Windows (x86_64). Se você não tiver uma, consulte Tutorial: Conceitos básicos do AWS IoT Greengrass V2.
- Python 3.6 ou posterior, inclusive `pip` para sua versão do Python, instalada em seu dispositivo principal.
- A função do dispositivo Greengrass foi configurada com o seguinte:

- Uma relação de confiança que permite `credentials.iot.amazonaws.com` e assume `sagemaker.amazonaws.com` a função, conforme mostrado no exemplo de política do IAM a seguir.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.iot.amazonaws.com"
       },
      "Action": "sts:AssumeRole"
    },
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "sagemaker.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

- A política gerenciada AmazonSageMakerEdgeDeviceFleetPolicydo IAM.
- A `s3:PutObject` ação, conforme mostrado no exemplo de política do IAM a seguir.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:PutObject"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

- Um bucket do Amazon S3 criado no mesmo dispositivo central do Greengrass Conta da AWS e no Região da AWS mesmo dispositivo. SageMaker O Edge Manager requer um bucket S3 para criar uma frota de dispositivos de ponta e armazenar dados de amostra da execução de inferência em seu dispositivo. Para obter informações sobre a criação de buckets do S3, consulte Introdução ao Amazon S3.
- Uma frota de dispositivos de SageMaker ponta que usa o mesmo alias de AWS IoT função do seu dispositivo principal do Greengrass. Para ter mais informações, consulte Crie uma frota de dispositivos de ponta.
- Seu dispositivo principal do Greengrass foi registrado como um dispositivo de ponta em sua frota de dispositivos SageMaker Edge. O nome do dispositivo de borda deve corresponder ao AWS IoT nome do dispositivo principal. Para ter mais informações, consulte Registre seu dispositivo principal do Greengrass.

Endpoints e portas

Esse componente deve ser capaz de realizar solicitações de saída para os seguintes endpoints e portas, além dos endpoints e portas necessários para a operação básica. Para ter mais informações, consulte Permitir o tráfego de dispositivos por meio de um proxy ou firewall.

Endpoint	Porta	Obrigatório	Descrição
`edge.sagemaker.`*`region`*`.amazonaws.com`	443	Sim	Verifique o status de registro do dispositivo e envie métricas para SageMaker.
`*.s3.amazonaws.com`	443	Sim	Faça upload dos dados de captura para o bucket

Endpoint	Porta	Obrigatório	Descrição
			do S3 que você especificar. Você pode * substituir pelo nome de cada bucket em que você carrega dados.

Dependências

Quando você implanta um componente, AWS IoT Greengrass também implanta versões compatíveis de suas dependências. Isso significa que você deve atender aos requisitos do componente e de todas as suas dependências para implantá-lo com êxito. Esta seção lista as dependências das versões lançadas desse componente e as restrições de versão semântica que definem as versões dos componentes para cada dependência. Você também pode visualizar as dependências de cada versão do componente no AWS IoT Greengrass console. Na página de detalhes do componente, procure a lista de dependências.

1.3.5 and 1.3.6

A tabela a seguir lista as dependências das versões 1.3.5 e 1.3.6 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.13.0	Flexível
Serviço de troca de tokens	>=0,0,0	Rígido

1.3.4

A tabela a seguir lista as dependências da versão 1.3.4 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.12.0	Flexível
Serviço de troca de tokens	>=0,0,0	Rígido

1.3.3

A tabela a seguir lista as dependências da versão 1.3.3 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.11.0	Flexível
Serviço de troca de tokens	>=0,0,0	Rígido

1.3.2

A tabela a seguir lista as dependências da versão 1.3.2 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.10.0	Flexível
Serviço de troca de tokens	>=0,0,0	Rígido

1.3.1

A tabela a seguir lista as dependências da versão 1.3.1 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.9.0	Flexível
Serviço de troca de tokens	>=0,0,0	Rígido

1.1.1 - 1.3.0

A tabela a seguir lista as dependências das versões 1.1.1 a 1.3.0 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.8.0	Flexível
Serviço de troca de tokens	>=0,0,0	Rígido

1.1.0

A tabela a seguir lista as dependências da versão 1.1.0 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.6.0	Flexível
Serviço de troca de tokens	>=0,0,0	Rígido

1.0.3

A tabela a seguir lista as dependências da versão 1.0.3 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.5.0	Flexível
Serviço de troca de tokens	>=0,0,0	Rígido

1.0.1 and 1.0.2

A tabela a seguir lista as dependências das versões 1.0.1 e 1.0.2 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.4.0	Flexível

Dependência	Versões compatíveis	Tipo de dependência
Serviço de troca de tokens	>=0,0,0	Rígido

1.0.0

A tabela a seguir lista as dependências da versão 1.0.0 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.3.0	Flexível
Serviço de troca de tokens	>=0,0,0	Rígido

Para obter mais informações sobre dependências de componentes, consulte a referência da receita do componente.

Configuração

Esse componente fornece os seguintes parâmetros de configuração que você pode personalizar ao implantar o componente.

> **Note**
>
> Esta seção descreve os parâmetros de configuração que você define no componente. Para obter mais informações sobre a configuração correspondente do SageMaker Edge Manager, consulte o Edge Manager Agent no Amazon SageMaker Developer Guide.

`DeviceFleetName`

O nome da frota de dispositivos do SageMaker Edge Manager que contém seu dispositivo principal do Greengrass.

Você deve especificar um valor para esse parâmetro na atualização de configuração ao implantar esse componente.

`BucketName`

O nome do bucket do S3 para o qual você carrega os dados de inferência capturados. O nome do bucket deve conter a string`sagemaker`.

Se você `CaptureDataDestination` definir como`Cloud`, ou se `CaptureDataPeriodicUpload` definir como`true`, deverá especificar um valor para esse parâmetro na atualização de configuração ao implantar esse componente.

> **Note**
>
> Capturar dados é um SageMaker recurso que você usa para carregar entradas de inferência, resultados de inferência e dados de inferência adicionais em um bucket do S3 ou em um diretório local para análise futura. Para obter mais informações sobre o uso de dados de captura com o SageMaker Edge Manager, consulte Gerenciar modelo no Amazon SageMaker Developer Guide.

`CaptureDataBatchSize`

(Opcional) O tamanho de um lote de solicitações de dados de captura que o agente processa. Esse valor deve ser menor que o tamanho do buffer especificado em`CaptureDataBufferSize`. Recomendamos que você não exceda a metade do tamanho do buffer.

O agente processa um lote de solicitações quando o número de solicitações no buffer atinge o `CaptureDataBatchSize` número ou quando o `CaptureDataPushPeriodSeconds` intervalo termina, o que ocorrer primeiro.

Padrão: `10`

`CaptureDataBufferSize`

(Opcional) O número máximo de solicitações de dados de captura armazenadas no buffer.

Padrão: `30`

`CaptureDataDestination`

(Opcional) O destino em que você armazena os dados capturados. Esse parâmetro pode ter os seguintes valores:

- `Cloud`— Carrega os dados capturados no bucket do S3 que você especifica em. `BucketName`
- `Disk`—Grava os dados capturados no diretório de trabalho do componente.

Se você especificar`Disk`, também poderá optar por carregar periodicamente os dados capturados em seu bucket do S3 configurando como`CaptureDataPeriodicUpload`. `true`

Padrão: `Cloud`

`CaptureDataPeriodicUpload`

(Opcional) Valor da string que especifica se os dados capturados devem ser carregados periodicamente. Os valores compatíveis são `true` e `false`.

Defina esse parâmetro como `true` se você `CaptureDataDestination` definir`Disk`, e você também deseja que o agente carregue periodicamente os dados capturados em seu bucket do S3.

Padrão: `false`

`CaptureDataPeriodicUploadPeriodSeconds`

(Opcional) O intervalo em segundos no qual o agente do SageMaker Edge Manager carrega os dados capturados no bucket do S3. Use esse parâmetro se você `CaptureDataPeriodicUpload` definir como`true`.

Padrão: `8`

`CaptureDataPushPeriodSeconds`

(Opcional) O intervalo em segundos no qual o agente do SageMaker Edge Manager processa um lote de solicitações de dados de captura do buffer.

O agente processa um lote de solicitações quando o número de solicitações no buffer atinge o `CaptureDataBatchSize` número ou quando o `CaptureDataPushPeriodSeconds` intervalo termina, o que ocorrer primeiro.

Padrão: `4`

`CaptureDataBase64EmbedLimit`

(Opcional) O tamanho máximo em bytes dos dados capturados que o agente do SageMaker Edge Manager carrega.

Padrão: `3072`

`FolderPrefix`

(Opcional) O nome da pasta na qual o agente grava os dados capturados. Se você `CaptureDataDestination` definir como`Disk`, o agente criará a pasta no diretório especificado

porCaptureDataDiskPath. Se você CaptureDataDestination definir comoCloud, ou se definir comotrue, CaptureDataPeriodicUpload o agente criará a pasta no seu bucket do S3.

Padrão: sme-capture

CaptureDataDiskPath

Esse recurso está disponível nas versões v1.1.0 e posteriores do componente SageMaker Edge Manager.

(Opcional) O caminho para a pasta na qual o agente cria a pasta de dados capturada. Se você CaptureDataDestination definir comoDisk, o agente criará a pasta de dados capturados nesse diretório. Se você não especificar esse valor, o agente criará a pasta de dados capturados no diretório de trabalho do componente. Use o FolderPrefix parâmetro para especificar o nome da pasta de dados capturada.

Padrão: */greengrass/v2*/work/aws.greengrass.SageMakerEdgeManager/capture

LocalDataRootPath

Esse recurso está disponível nas versões v1.2.0 e posteriores do componente SageMaker Edge Manager.

(Opcional) O caminho em que esse componente armazena os seguintes dados no dispositivo principal:

- O banco de dados local para dados de tempo de execução quando você define como DbEnabletrue.
- SageMaker Modelos neocompilados que esse componente baixa automaticamente quando você define comoDeploymentEnable. true

Padrão: */greengrass/v2*/work/aws.greengrass.SageMakerEdgeManager

DbEnable

(Opcional) Você pode ativar esse componente para armazenar dados de tempo de execução em um banco de dados local para preservar os dados, caso o componente falhe ou o dispositivo perca energia.

Esse banco de dados requer 5 MB de armazenamento no sistema de arquivos do dispositivo principal.

Padrão: `false`

`DeploymentEnable`

Esse recurso está disponível nas versões v1.2.0 e posteriores do componente SageMaker Edge Manager.

(Opcional) Você pode habilitar esse componente para recuperar automaticamente modelos SageMaker compilados pelo Neo a partir dos quais você carrega para o Amazon S3. Depois de fazer o upload de um novo modelo para o Amazon S3, use o SageMaker Studio ou a SageMaker API para implantar o novo modelo nesse dispositivo principal. Ao habilitar esse recurso, você pode implantar novos modelos nos dispositivos principais sem precisar criar uma AWS IoT Greengrass implantação.

> **Important**
>
> Para usar esse recurso, você deve `DbEnable` definir como`true`. Esse recurso usa o banco de dados local para rastrear modelos que ele recupera Nuvem AWS do.

Padrão: `false`

`DeploymentPollInterval`

Esse recurso está disponível nas versões v1.2.0 e posteriores do componente SageMaker Edge Manager.

(Opcional) A quantidade de tempo (em minutos) entre o qual esse componente verifica se há novos modelos para baixar. Essa opção se aplica quando você define como `DeploymentEnable`true.

Padrão: `1440` (1 dia)

`DLRBackendOptions`

Esse recurso está disponível nas versões v1.2.0 e posteriores do componente SageMaker Edge Manager.

(Opcional) Os sinalizadores de tempo de execução do DLR a serem definidos no tempo de execução do DLR que esse componente usa. Você pode definir o seguinte sinalizador:

- `TVM_TENSORRT_CACHE_DIR`— Ative o cache do modelo TensorRT. Especifique um caminho absoluto para uma pasta existente que tenha permissões de leitura/gravação.

- `TVM_TENSORRT_CACHE_DISK_SIZE_MB`— Atribui o limite superior da pasta de cache do modelo TensorRT. Quando o tamanho do diretório ultrapassa esse limite, os mecanismos em cache menos usados são excluídos. O valor padrão é 512 MB.

Por exemplo, você pode definir esse parâmetro com o valor a seguir para ativar o cache do modelo TensorRT e limitar o tamanho do cache a 800 MB.

```
TVM_TENSORRT_CACHE_DIR=/data/secured_folder/trt/cache;
 TVM_TENSORRT_CACHE_DISK_SIZE_MB=800
```

`SagemakerEdgeLogVerbose`

(Opcional) Valor da string que especifica se o registro de depuração deve ser ativado. Os valores compatíveis são `true` e `false`.

Padrão: `false`

`UnixSocketName`

(Opcional) A localização do descritor do arquivo de soquete do SageMaker Edge Manager no dispositivo principal.

Padrão: `/tmp/aws.greengrass.SageMakerEdgeManager.sock`

Example Exemplo: atualização da mesclagem de configurações

O exemplo de configuração a seguir especifica que o dispositivo principal faz parte do *`MyEdgeDeviceFleet`*e que o agente grava os dados de captura no dispositivo e em um bucket do S3. Essa configuração também permite o registro de depuração.

```
{
    "DeviceFleetName": "MyEdgeDeviceFleet",
    "BucketName": "DOC-EXAMPLE-BUCKET",
    "CaptureDataDestination": "Disk",
    "CaptureDataPeriodicUpload": "true",
    "SagemakerEdgeLogVerbose": "true"
}
```

Arquivo de log local

Esse componente usa o seguinte arquivo de log.

Linux

```
/greengrass/v2/logs/aws.greengrass.SageMakerEdgeManager.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.SageMakerEdgeManager.log
```

Para ver os registros desse componente

- Execute o comando a seguir no dispositivo principal para visualizar o arquivo de log desse componente em tempo real. Substitua */greengrass/v2* ou *C:\greengrass\v2* pelo caminho para a pasta AWS IoT Greengrass raiz.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.greengrass.SageMakerEdgeManager.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\aws.greengrass.SageMakerEdgeManager.log -Tail
   10 -Wait
  ```

Changelog

A tabela a seguir descreve as alterações em cada versão do componente.

Version (Versão)	Alterações
1.3.6	Versão atualizada para a versão 2.12.5 do Greengrass nucleus.
1.3.5	Versão atualizada para a versão 2.12.0 do Greengrass nucleus.
1.3.4	Versão atualizada para a versão 2.11.0 do Greengrass nucleus.
1.3.3	Versão atualizada para a versão 2.10.0 do Greengrass nucleus.
1.3.2	Versão atualizada para a versão 2.9.0 do Greengrass nucleus.

Version (Versão)	Alterações
1.3.1	Versão atualizada para a versão 2.8.0 do Greengrass nucleus.
1.3.0	Novos atributos • Adiciona suporte ao gerenciamento do tamanho do disco de cache TensorRT. • Adiciona o `TVM_TENSORRT_CACHE_DISK_SIZE_MB` sinalizador opcional ao BackendOptions parâmetro DLR para definir o limite de tamanho para modelos em cache no disco. Melhorias • Fornece maior simultaneidade de previsão. Isso ajuda a obter um melhor uso dos mecanismos aceleradores de dispositivos, como GPUs.
1.2.0	Novos atributos • Adiciona suporte a esse componente para recuperar automaticamente os modelos SageMaker compilados pelo NEO que você carrega para o Amazon S3. Ao habilitar esse recurso, você pode implantar novos modelos nos dispositivos principais sem precisar criar uma AWS IoT Greengrass implantação. • Adiciona suporte a um banco de dados de backup que esse component e usa para preservar os dados de tempo de execução, caso o componente falhe ou o dispositivo perca energia. • Adiciona suporte para você configurar sinalizadores de tempo de execução do DLR ao configurar esse componente.
1.1.1	Versão atualizada para a versão 2.7.0 do Greengrass nucleus.

Version (Versão)	Alterações
1.1.0	Novos atributos • Adiciona suporte para os principais dispositivos do Greengrass que executam o Amazon Linux 2. • Adiciona o novo parâmetro `CaptureDataDiskPath` de configuração. Você pode usar esse parâmetro para especificar o caminho da pasta de dados capturada no seu dispositivo. Correções de erros e melhorias • Versão atualizada para a versão 2.5.0 do Greengrass nucleus.
1.0.3	Versão atualizada para a versão 2.4.0 do Greengrass nucleus.
1.0.2	Correções de erros e melhorias Atualiza o script de instalação no ciclo de vida do componente. Seus dispositivos principais agora devem ter o Python 3.6 ou posterior, inclusive `pip` para sua versão do Python, instalado no dispositivo antes de você implantar esse componente.
1.0.1	Versão atualizada para a versão 2.3.0 do Greengrass nucleus.
1.0.0	Versão inicial.

Classificação de imagens DLR

O componente de classificação de imagem DLR (`aws.greengrass.DLRImageClassification`) contém exemplos de código de inferência para realizar inferência de classificação de imagens usando modelos Deep Learning Runtime e resnet-50. Esse componente usa a variante Armazenamento de modelos de classificação de imagens DLR e os Tempo de execução do DLR componentes como dependências para baixar o DLR e os modelos de amostra.

Para usar esse componente de inferência com um modelo DLR personalizado, crie uma versão personalizada do componente de armazenamento de modelos dependente. Para usar seu próprio código de inferência personalizado, você pode usar a receita desse componente como modelo para criar um componente de inferência personalizado.

Tópicos

Versões

Esse componente tem as seguintes versões:

- 2.1.x
- 2.0.x

Tipo

Este componente é um componente genérico (`aws.greengrass.generic`). O núcleo do Greengrass executa os scripts do ciclo de vida do componente.

Para ter mais informações, consulte Tipos de componentes.

Sistema operacional

Esse componente pode ser instalado em dispositivos principais que executam os seguintes sistemas operacionais:

- Linux
- Windows

Requisitos

Esse componente tem os seguintes requisitos:

- Nos principais dispositivos do Greengrass que executam o Amazon Linux 2 ou o Ubuntu 18.04, a GNU C Library (glibc) versão 2.27 ou posterior está instalada no dispositivo.

- Em dispositivos ARMv7L, como o Raspberry Pi, dependências do OpenCV-Python instaladas no dispositivo. Execute o comando a seguir para instalar as dependências.

  ```
  sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
   libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
  ```

- Os dispositivos Raspberry Pi que executam o Raspberry Pi OS Bullseye devem atender aos seguintes requisitos:

  - NumPy 1.22.4 ou posterior instalado no dispositivo. O Raspberry Pi OS Bullseye inclui uma versão anterior do NumPy, então você pode executar o seguinte comando para atualizar NumPy o dispositivo.

    ```
    pip3 install --upgrade numpy
    ```

  - A pilha de câmeras antiga ativada no dispositivo. O Raspberry Pi OS Bullseye inclui uma nova pilha de câmeras que é ativada por padrão e não é compatível, portanto, você deve habilitar a pilha de câmeras antiga.

    Para habilitar a pilha de câmeras antiga

    1. Execute o comando a seguir para abrir a ferramenta de configuração do Raspberry Pi.

       ```
       sudo raspi-config
       ```

    2. Selecione Opções de interface.
    3. Selecione Câmera antiga para ativar a pilha de câmeras antigas.
    4. Reinicie o Raspberry Pi.

Dependências

Quando você implanta um componente, AWS IoT Greengrass também implanta versões compatíveis de suas dependências. Isso significa que você deve atender aos requisitos do componente e de todas as suas dependências para implantá-lo com êxito. Esta seção lista as dependências das versões lançadas desse componente e as restrições de versão semântica que definem as versões dos componentes para cada dependência. Você também pode visualizar as dependências de cada versão do componente no AWS IoT Greengrass console. Na página de detalhes do componente, procure a lista de dependências.

2.1.13 and 2.1.14

A tabela a seguir lista as dependências das versões 2.1.13 e 2.1.14 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.13.0	Flexível
Armazenamento de modelos de classificação de imagens DLR	~2.1.0	Rígido
DLR	~1.6.0	Rígido

2.1.12

A tabela a seguir lista as dependências da versão 2.1.12 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.12.0	Flexível
Armazenamento de modelos de classificação de imagens DLR	~2.1.0	Rígido
DLR	~1.6.0	Rígido

2.1.11

A tabela a seguir lista as dependências da versão 2.1.11 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.11.0	Flexível

Dependência	Versões compatíveis	Tipo de dependência
Armazenamento de modelos de classificação de imagens DLR	~2.1.0	Rígido
DLR	~1.6.0	Rígido

2.1.10

A tabela a seguir lista as dependências da versão 2.1.10 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.10.0	Flexível
Armazenamento de modelos de classificação de imagens DLR	~2.1.0	Rígido
DLR	~1.6.0	Rígido

2.1.9

A tabela a seguir lista as dependências da versão 2.1.9 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.9.0	Flexível
Armazenamento de modelos de classificação de imagens DLR	~2.1.0	Rígido
DLR	~1.6.0	Rígido

2.1.8

A tabela a seguir lista as dependências da versão 2.1.8 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.8.0	Flexível
Armazenamento de modelos de classificação de imagens DLR	~2.1.0	Rígido
DLR	~1.6.0	Rígido

2.1.7

A tabela a seguir lista as dependências da versão 2.1.7 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.7.0	Flexível
Armazenamento de modelos de classificação de imagens DLR	~2.1.0	Rígido
DLR	~1.6.0	Rígido

2.1.6

A tabela a seguir lista as dependências da versão 2.1.6 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.6.0	Flexível

Dependência	Versões compatíveis	Tipo de dependência
Armazenamento de modelos de classificação de imagens DLR	~2.1.0	Rígido
DLR	~1.6.0	Rígido

2.1.4 - 2.1.5

A tabela a seguir lista as dependências das versões 2.1.4 a 2.1.5 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.5.0	Flexível
Armazenamento de modelos de classificação de imagens DLR	~2.1.0	Rígido
DLR	~1.6.0	Rígido

2.1.3

A tabela a seguir lista as dependências da versão 2.1.3 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.4.0	Flexível
Armazenamento de modelos de classificação de imagens DLR	~2.1.0	Rígido
DLR	~1.6.0	Rígido

2.1.2

A tabela a seguir lista as dependências da versão 2.1.2 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.3.0	Flexível
Armazenamento de modelos de classificação de imagens DLR	~2.1.0	Rígido
DLR	~1.6.0	Rígido

2.1.1

A tabela a seguir lista as dependências da versão 2.1.1 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.2.0	Flexível
Armazenamento de modelos de classificação de imagens DLR	~2.1.0	Rígido
DLR	~1.6.0	Rígido

2.0.x

A tabela a seguir lista as dependências da versão 2.0.x desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	~2.0.0	Flexível

Dependência	Versões compatíveis	Tipo de dependência
Armazenamento de modelos de classificação de imagens DLR	~2.0.0	Rígido
DLR	~1.3.0	Flexível

Configuração

Esse componente fornece os seguintes parâmetros de configuração que você pode personalizar ao implantar o componente.

2.1.x

`accessControl`

(Opcional) O objeto que contém a política de autorização que permite que o componente publique mensagens no tópico de notificações padrão.

Padrão:

```
{
   "aws.greengrass.ipc.mqttproxy": {
      "aws.greengrass.DLRImageClassification:mqttproxy:1": {
         "policyDescription": "Allows access to publish via topic ml/dlr/image-
classification.",
         "operations": [
            "aws.greengrass#PublishToIoTCore"
         ],
         "resources": [
            "ml/dlr/image-classification"
         ]
      }
   }
}
```

`PublishResultsOnTopic`

(Opcional) O tópico no qual você deseja publicar os resultados da inferência. Se você modificar esse valor, também deverá modificar o valor de `resources` no `accessControl` parâmetro para corresponder ao nome do tópico personalizado.

Padrão: `ml/dlr/image-classification`

`Accelerator`

O acelerador que você deseja usar. Os valores compatíveis são `cpu` e `gpu`.

Os modelos de amostra no componente do modelo dependente oferecem suporte somente à aceleração da CPU. Para usar a aceleração de GPU com um modelo personalizado diferente, crie um componente de modelo personalizado para substituir o componente de modelo público.

Padrão: `cpu`

`ImageDirectory`

(Opcional) O caminho da pasta no dispositivo em que os componentes de inferência leem imagens. Você pode modificar esse valor para qualquer local em seu dispositivo ao qual tenha acesso de leitura/gravação.

Padrão: */greengrass/v2*`/packages/artifacts-unarchived/`*component-name*`/image_classification/sample_images/`

> **Note**
>
> Se você definir o valor de `UseCamera` para `true`, esse parâmetro de configuração será ignorado.

`ImageName`

(Opcional) O nome da imagem que o componente de inferência usa como entrada para uma previsão de criação. O componente procura a imagem na pasta especificada em`ImageDirectory`. Por padrão, o componente usa a imagem de amostra no diretório de imagens padrão. AWS IoT Greengrass suporta os seguintes formatos de imagem: `jpeg` `jpg`png,, npy e.

Padrão: `cat.jpeg`

> **Note**
>
> Se você definir o valor de `UseCamera` para`true`, esse parâmetro de configuração será ignorado.

`InferenceInterval`

(Opcional) O tempo em segundos entre cada previsão feita pelo código de inferência. O código de inferência de amostra é executado indefinidamente e repete suas previsões no intervalo de tempo especificado. Por exemplo, você pode alterar isso para um intervalo menor se quiser usar imagens tiradas por uma câmera para previsão em tempo real.

Padrão: `3600`

`ModelResourceKey`

(Opcional) Os modelos usados no componente de modelo público dependente. Modifique esse parâmetro somente se você substituir o componente do modelo público por um componente personalizado.

Padrão:

```
{
    "armv7l": "DLR-resnet50-armv7l-cpu-ImageClassification",
    "aarch64": "DLR-resnet50-aarch64-cpu-ImageClassification",
    "x86_64": "DLR-resnet50-x86_64-cpu-ImageClassification",
    "windows": "DLR-resnet50-win-cpu-ImageClassification"
}
```

`UseCamera`

(Opcional) Valor da string que define se as imagens de uma câmera conectada ao dispositivo principal do Greengrass devem ser usadas. Os valores compatíveis são `true` e `false`.

Quando você define esse valor como`true`, o código de inferência de amostra acessa a câmera no seu dispositivo e executa a inferência localmente na imagem capturada. Os valores dos `ImageDirectory` parâmetros `ImageName` e são ignorados. Certifique-se de que o

usuário que está executando esse componente tenha acesso de leitura/gravação ao local em que a câmera armazena as imagens capturadas.

Padrão: `false`

> **Note**
>
> Quando você visualiza a receita desse componente, o parâmetro de `UseCamera` configuração não aparece na configuração padrão. No entanto, você pode modificar o valor desse parâmetro em uma atualização de mesclagem de configuração ao implantar o componente.
> Ao definir como `UseCameratrue`, você também deve criar um link simbólico para permitir que o componente de inferência acesse sua câmera a partir do ambiente virtual criado pelo componente de tempo de execução. Para obter mais informações sobre o uso de uma câmera com os componentes de inferência de amostra, consulteAtualizar configurações de componentes.

2.0.x

`MLRootPath`

(Opcional) O caminho da pasta nos dispositivos principais do Linux em que os componentes de inferência leem imagens e gravam resultados de inferência. Você pode modificar esse valor em qualquer local em seu dispositivo no qual o usuário que está executando esse componente tenha acesso de leitura/gravação.

Padrão: `/greengrass/v2/work/variant.DLR/greengrass_ml`

Padrão: `/greengrass/v2/work/variant.TensorFlowLite/greengrass_ml`

`Accelerator`

O acelerador que você deseja usar. Os valores compatíveis são `cpu` e `gpu`.

Os modelos de amostra no componente do modelo dependente oferecem suporte somente à aceleração da CPU. Para usar a aceleração de GPU com um modelo personalizado diferente, crie um componente de modelo personalizado para substituir o componente de modelo público.

Padrão: `cpu`

`ImageName`

(Opcional) O nome da imagem que o componente de inferência usa como entrada para uma previsão de criação. O componente procura a imagem na pasta especificada em`ImageDirectory`. O local padrão é*`MLRootPath`*`/images`. AWS IoT Greengrass suporta os seguintes formatos de imagem: `jpeg` `jpg`png,, `npy` e.

Padrão: `cat.jpeg`

`InferenceInterval`

(Opcional) O tempo em segundos entre cada previsão feita pelo código de inferência. O código de inferência de amostra é executado indefinidamente e repete suas previsões no intervalo de tempo especificado. Por exemplo, você pode alterar isso para um intervalo menor se quiser usar imagens tiradas por uma câmera para previsão em tempo real.

Padrão: `3600`

`ModelResourceKey`

(Opcional) Os modelos usados no componente de modelo público dependente. Modifique esse parâmetro somente se você substituir o componente do modelo público por um componente personalizado.

Padrão:

```
armv7l: "DLR-resnet50-armv7l-cpu-ImageClassification"
x86_64: "DLR-resnet50-x86_64-cpu-ImageClassification"
```

Arquivo de log local

Esse componente usa o seguinte arquivo de log.

Linux

```
/greengrass/v2/logs/aws.greengrass.DLRImageClassification.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.DLRImageClassification.log
```

Para ver os registros desse componente

- Execute o comando a seguir no dispositivo principal para visualizar o arquivo de log desse componente em tempo real. Substitua */greengrass/v2* ou *C:\greengrass\v2* pelo caminho para a pasta AWS IoT Greengrass raiz.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.greengrass.DLRImageClassification.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\aws.greengrass.DLRImageClassification.log -
  Tail 10 -Wait
  ```

Changelog

A tabela a seguir descreve as alterações em cada versão do componente.

Version (Versão)	Alterações
2.1.14	Versão atualizada para a versão 2.12.5 do Greengrass nucleus.
2.1.13	Versão atualizada para a versão 2.12.0 do Greengrass nucleus.
2.1.12	Versão atualizada para a versão 2.11.0 do Greengrass nucleus.
2.1.11	Versão atualizada para a versão 2.10.0 do Greengrass nucleus.
2.1.10	Versão atualizada para a versão 2.9.0 do Greengrass nucleus.
2.1.9	Versão atualizada para a versão 2.8.0 do Greengrass nucleus.
2.1.8	Versão atualizada para a versão 2.7.0 do Greengrass nucleus.
2.1.7	Versão atualizada para a versão 2.6.0 do Greengrass nucleus.
2.1.6	Versão atualizada para a versão 2.5.0 do Greengrass nucleus.
2.1.5	Componente lançado em todos Regiões da AWS.

Version (Versão)	Alterações
2.1.4	Versão atualizada para a versão 2.4.0 do Greengrass nucleus. Esta versão não está disponível na Europa (Londres) (`eu-west-2`).
2.1.3	Versão atualizada para a versão 2.3.0 do Greengrass nucleus.
2.1.2	Versão atualizada para a versão 2.2.0 do Greengrass nucleus.
2.1.1	Novos atributos • Use o Deep Learning Runtime v1.6.0. • Adicione suporte para classificação de imagens de amostra nas plataformas Armv8 (AArch64). Isso amplia o suporte ao aprendizado de máquina para os principais dispositivos do Greengrass que executam o NVIDIA Jetson, como o Jetson Nano. • Ative a integração da câmera para inferência de amostras. Use o novo parâmetro de `UseCamera` configuração para permitir que o código de inferência de amostra acesse a câmera em seu dispositivo principal do Greengrass e execute a inferência localmente na imagem capturada. • Adicione suporte para publicação de resultados de inferência no Nuvem AWS. Use o novo parâmetro de `PublishResultsOnTopic` configuração para especificar o tópico no qual você deseja publicar os resultados. • Adicione o novo parâmetro de `ImageDirectory` configuração que permite especificar um diretório personalizado para a imagem na qual você deseja realizar a inferência. Correções de erros e melhorias • Grave os resultados da inferência no arquivo de log do componente em vez de em um arquivo de inferência separado. • Use o módulo de registro do software AWS IoT Greengrass Core para registrar a saída do componente. • Use o AWS IoT Device SDK para ler a configuração do componente e aplicar as alterações na configuração.
2.0.4	Versão inicial.

Detecção de objetos DLR

O componente de detecção de objetos DLR (`aws.greengrass.DLRObjectDetection`) contém exemplos de código de inferência para realizar inferência de detecção de objetos usando o Deep Learning Runtime e amostras de modelos pré-treinados. Esse componente usa a variante Armazenamento de modelos de detecção de objetos DLR e os Tempo de execução do DLR componentes como dependências para baixar o DLR e os modelos de amostra.

Para usar esse componente de inferência com um modelo DLR personalizado, crie uma versão personalizada do componente de armazenamento de modelos dependente. Para usar seu próprio código de inferência personalizado, você pode usar a receita desse componente como modelo para criar um componente de inferência personalizado.

Tópicos

Versões

Esse componente tem as seguintes versões:

- 2.1.x
- 2.0.x

Tipo

Este componente é um componente genérico (`aws.greengrass.generic`). O núcleo do Greengrass executa os scripts do ciclo de vida do componente.

Para ter mais informações, consulte Tipos de componentes.

Sistema operacional

Esse componente pode ser instalado em dispositivos principais que executam os seguintes sistemas operacionais:

- Linux
- Windows

Requisitos

Esse componente tem os seguintes requisitos:

- Nos principais dispositivos do Greengrass que executam o Amazon Linux 2 ou o Ubuntu 18.04, a GNU C Library (glibc) versão 2.27 ou posterior está instalada no dispositivo.
- Em dispositivos ARMv7L, como o Raspberry Pi, dependências do OpenCV-Python instaladas no dispositivo. Execute o comando a seguir para instalar as dependências.

  ```
  sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
   libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
  ```

- Os dispositivos Raspberry Pi que executam o Raspberry Pi OS Bullseye devem atender aos seguintes requisitos:
  - NumPy 1.22.4 ou posterior instalado no dispositivo. O Raspberry Pi OS Bullseye inclui uma versão anterior do NumPy, então você pode executar o seguinte comando para atualizar NumPy o dispositivo.

    ```
    pip3 install --upgrade numpy
    ```

  - A pilha de câmeras antiga ativada no dispositivo. O Raspberry Pi OS Bullseye inclui uma nova pilha de câmeras que é ativada por padrão e não é compatível, portanto, você deve habilitar a pilha de câmeras antiga.

    Para habilitar a pilha de câmeras antiga

    1. Execute o comando a seguir para abrir a ferramenta de configuração do Raspberry Pi.

       ```
       sudo raspi-config
       ```

    2. Selecione Opções de interface.

3. Selecione Câmera antiga para ativar a pilha de câmeras antigas.
4. Reinicie o Raspberry Pi.

Dependências

Quando você implanta um componente, AWS IoT Greengrass também implanta versões compatíveis de suas dependências. Isso significa que você deve atender aos requisitos do componente e de todas as suas dependências para implantá-lo com êxito. Esta seção lista as dependências das versões lançadas desse componente e as restrições de versão semântica que definem as versões dos componentes para cada dependência. Você também pode visualizar as dependências de cada versão do componente no AWS IoT Greengrass console. Na página de detalhes do componente, procure a lista de dependências.

2.1.13 and 2.1.14

A tabela a seguir lista as dependências das versões 2.1.13 e 2.1.14 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.13.0	Flexível
Armazenamento de modelos de detecção de objetos DLR	~2.1.0	Rígido
DLR	~1.6.0	Rígido

2.1.12

A tabela a seguir lista as dependências da versão 2.1.12 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.12.0	Flexível
Armazenamento de modelos de detecção de objetos DLR	~2.1.0	Rígido
DLR	~1.6.0	Rígido

2.1.11

A tabela a seguir lista as dependências da versão 2.1.11 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.11.0	Flexível
Armazenamento de modelos de detecção de objetos DLR	~2.1.0	Rígido
DLR	~1.6.0	Rígido

2.1.10

A tabela a seguir lista as dependências da versão 2.1.10 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.10.0	Flexível
Armazenamento de modelos de detecção de objetos DLR	~2.1.0	Rígido
DLR	~1.6.0	Rígido

2.1.9

A tabela a seguir lista as dependências da versão 2.1.9 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.9.0	Flexível
Armazenamento de modelos de detecção de objetos DLR	~2.1.0	Rígido
DLR	~1.6.0	Rígido

2.1.8

A tabela a seguir lista as dependências da versão 2.1.8 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.8.0	Flexível
Armazenamento de modelos de detecção de objetos DLR	~2.1.0	Rígido
DLR	~1.6.0	Rígido

2.1.7

A tabela a seguir lista as dependências da versão 2.1.7 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.7.0	Flexível
Armazenamento de modelos de detecção de objetos DLR	~2.1.0	Rígido
DLR	~1.6.0	Rígido

2.1.6

A tabela a seguir lista as dependências da versão 2.1.6 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.6.0	Flexível
Armazenamento de modelos de detecção de objetos DLR	~2.1.0	Rígido
DLR	~1.6.0	Rígido

2.1.4 - 2.1.5

A tabela a seguir lista as dependências das versões 2.1.4 a 2.1.5 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.5.0	Flexível
Armazenamento de modelos de detecção de objetos DLR	~2.1.0	Rígido
DLR	~1.6.0	Rígido

2.1.3

A tabela a seguir lista as dependências da versão 2.1.3 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.4.0	Flexível
Armazenamento de modelos de detecção de objetos DLR	~2.1.0	Rígido
DLR	~1.6.0	Rígido

2.1.2

A tabela a seguir lista as dependências da versão 2.1.2 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.3.0	Flexível
Armazenamento de modelos de detecção de objetos DLR	~2.1.0	Rígido
DLR	~1.6.0	Rígido

2.1.1

A tabela a seguir lista as dependências da versão 2.1.1 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.2.0	Flexível
Armazenamento de modelos de detecção de objetos DLR	~2.1.0	Rígido
DLR	~1.6.0	Rígido

2.0.x

A tabela a seguir lista as dependências da versão 2.0.x desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	~2.0.0	Flexível
Armazenamento de modelos de detecção de objetos DLR	~2.0.0	Rígido
DLR	~1.3.0	Flexível

Configuração

Esse componente fornece os seguintes parâmetros de configuração que você pode personalizar ao implantar o componente.

2.1.x

`accessControl`

(Opcional) O objeto que contém a política de autorização que permite que o componente publique mensagens no tópico de notificações padrão.

Padrão:

```
{
   "aws.greengrass.ipc.mqttproxy": {
      "aws.greengrass.DLRObjectDetection:mqttproxy:1": {
         "policyDescription": "Allows access to publish via topic ml/dlr/object-
detection.",
         "operations": [
            "aws.greengrass#PublishToIoTCore"
         ],
         "resources": [
            "ml/dlr/object-detection"
         ]
      }
   }
}
```

`PublishResultsOnTopic`

(Opcional) O tópico no qual você deseja publicar os resultados da inferência. Se você modificar esse valor, também deverá modificar o valor de `resources` no `accessControl` parâmetro para corresponder ao nome do tópico personalizado.

Padrão: `ml/dlr/object-detection`

`Accelerator`

O acelerador que você deseja usar. Os valores compatíveis são `cpu` e `gpu`.

Os modelos de amostra no componente do modelo dependente oferecem suporte somente à aceleração da CPU. Para usar a aceleração de GPU com um modelo personalizado diferente, crie um componente de modelo personalizado para substituir o componente de modelo público.

Padrão: `cpu`

`ImageDirectory`

(Opcional) O caminho da pasta no dispositivo em que os componentes de inferência leem imagens. Você pode modificar esse valor para qualquer local em seu dispositivo ao qual tenha acesso de leitura/gravação.

Padrão: */greengrass/v2*`/packages/artifacts-unarchived/`*component-name*`/object_detection/sample_images/`

> **Note**
>
> Se você definir o valor de `UseCamera` para `true`, esse parâmetro de configuração será ignorado.

`ImageName`

(Opcional) O nome da imagem que o componente de inferência usa como entrada para uma previsão de criação. O componente procura a imagem na pasta especificada em `ImageDirectory`. Por padrão, o componente usa a imagem de amostra no diretório de imagens padrão. AWS IoT Greengrass suporta os seguintes formatos de imagem: `jpeg` `jpg` `png`, `npy` e.

Padrão: `objects.jpg`

> **Note**
>
> Se você definir o valor de `UseCamera` para `true`, esse parâmetro de configuração será ignorado.

`InferenceInterval`

(Opcional) O tempo em segundos entre cada previsão feita pelo código de inferência. O código de inferência de amostra é executado indefinidamente e repete suas previsões no intervalo de tempo especificado. Por exemplo, você pode alterar isso para um intervalo menor se quiser usar imagens tiradas por uma câmera para previsão em tempo real.

Padrão: `3600`

`ModelResourceKey`

(Opcional) Os modelos usados no componente de modelo público dependente. Modifique esse parâmetro somente se você substituir o componente do modelo público por um componente personalizado.

Padrão:

```
{
```

```
    "armv7l": "DLR-yolo3-armv7l-cpu-ObjectDetection",
    "aarch64": "DLR-yolo3-aarch64-gpu-ObjectDetection",
    "x86_64": "DLR-yolo3-x86_64-cpu-ObjectDetection",
    "windows": "DLR-resnet50-win-cpu-ObjectDetection"
}
```

`UseCamera`

(Opcional) Valor da string que define se as imagens de uma câmera conectada ao dispositivo principal do Greengrass devem ser usadas. Os valores compatíveis são `true` e `false`.

Quando você define esse valor como`true`, o código de inferência de amostra acessa a câmera no seu dispositivo e executa a inferência localmente na imagem capturada. Os valores dos `ImageDirectory` parâmetros `ImageName` e são ignorados. Certifique-se de que o usuário que está executando esse componente tenha acesso de leitura/gravação ao local em que a câmera armazena as imagens capturadas.

Padrão: `false`

> **Note**
>
> Quando você visualiza a receita desse componente, o parâmetro de `UseCamera` configuração não aparece na configuração padrão. No entanto, você pode modificar o valor desse parâmetro em uma atualização de mesclagem de configuração ao implantar o componente.
> Ao definir como `UseCameratrue`, você também deve criar um link simbólico para permitir que o componente de inferência acesse sua câmera a partir do ambiente virtual criado pelo componente de tempo de execução. Para obter mais informações sobre o uso de uma câmera com os componentes de inferência de amostra, consulteAtualizar configurações de componentes.

2.0.x

`MLRootPath`

(Opcional) O caminho da pasta nos dispositivos principais do Linux em que os componentes de inferência leem imagens e gravam resultados de inferência. Você pode modificar esse valor em qualquer local em seu dispositivo no qual o usuário que está executando esse componente tenha acesso de leitura/gravação.

Padrão: */greengrass/v2*/work/variant.DLR/greengrass_ml

Padrão: */greengrass/v2*/work/variant.TensorFlowLite/greengrass_ml

`Accelerator`

Não modifique. Atualmente, o único valor suportado para o acelerador é porque `cpu` os modelos nos componentes do modelo dependente são compilados somente para o acelerador de CPU.

`ImageName`

(Opcional) O nome da imagem que o componente de inferência usa como entrada para uma previsão de criação. O componente procura a imagem na pasta especificada em`ImageDirectory`. O local padrão é*MLRootPath*`/images`. AWS IoT Greengrass suporta os seguintes formatos de imagem: `jpeg` `jpg`png,, `npy` e.

Padrão: `objects.jpg`

`InferenceInterval`

(Opcional) O tempo em segundos entre cada previsão feita pelo código de inferência. O código de inferência de amostra é executado indefinidamente e repete suas previsões no intervalo de tempo especificado. Por exemplo, você pode alterar isso para um intervalo menor se quiser usar imagens tiradas por uma câmera para previsão em tempo real.

Padrão: `3600`

`ModelResourceKey`

(Opcional) Os modelos usados no componente de modelo público dependente. Modifique esse parâmetro somente se você substituir o componente do modelo público por um componente personalizado.

Padrão:

```
{
    armv7l: "DLR-yolo3-armv7l-cpu-ObjectDetection",
    x86_64: "DLR-yolo3-x86_64-cpu-ObjectDetection"
}
```

Arquivo de log local

Esse componente usa o seguinte arquivo de log.

Linux

```
/greengrass/v2/logs/aws.greengrass.DLRObjectDetection.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.DLRObjectDetection.log
```

Para ver os registros desse componente

- Execute o comando a seguir no dispositivo principal para visualizar o arquivo de log desse componente em tempo real. Substitua */greengrass/v2* ou *C:\greengrass\v2* pelo caminho para a pasta AWS IoT Greengrass raiz.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.greengrass.DLRObjectDetection.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\aws.greengrass.DLRObjectDetection.log -Tail 10
   -Wait
  ```

Changelog

A tabela a seguir descreve as alterações em cada versão do componente.

Version (Versão)	Alterações
2.1.14	Versão atualizada para a versão 2.12.5 do Greengrass nucleus.
2.1.13	Versão atualizada para a versão 2.12.0 do Greengrass nucleus.
2.1.12	Versão atualizada para a versão 2.11.0 do Greengrass nucleus.
2.1.11	Versão atualizada para a versão 2.10.0 do Greengrass nucleus.
2.1.10	Versão atualizada para a versão 2.9.0 do Greengrass nucleus.

Version (Versão)	Alterações
2.1.9	Versão atualizada para a versão 2.8.0 do Greengrass nucleus.
2.1.8	Versão atualizada para a versão 2.7.0 do Greengrass nucleus.
2.1.7	Versão atualizada para a versão 2.6.0 do Greengrass nucleus.
2.1.6	Versão atualizada para a versão 2.5.0 do Greengrass nucleus.
2.1.5	Componente lançado em todos Regiões da AWS.
2.1.4	Versão atualizada para a versão 2.4.0 do Greengrass nucleus. Esta versão não está disponível na Europa (Londres) (`eu-west-2`).
2.1.3	Versão atualizada para a versão 2.3.0 do Greengrass nucleus.
2.1.2	Correções de erros e melhorias • Corrige um problema de escala de imagem que resultou em caixas delimitadoras imprecisas nos resultados de inferência de detecção de objetos DLR da amostra.

Version (Versão)	Alterações
2.1.1	Novos atributos • Use o Deep Learning Runtime v1.6.0. • Adicione suporte para detecção de objetos de amostra nas plataform as Armv8 (AArch64). Isso amplia o suporte ao aprendizado de máquina para os principais dispositivos do Greengrass que executam o NVIDIA Jetson, como o Jetson Nano. • Ative a integração da câmera para inferência de amostras. Use o novo parâmetro de `UseCamera` configuração para permitir que o código de inferência de amostra acesse a câmera em seu dispositivo principal do Greengrass e execute a inferência localmente na imagem capturada. • Adicione suporte para publicação de resultados de inferência no Nuvem AWS. Use o novo parâmetro de `PublishResultsOnTopic` configuração para especificar o tópico no qual você deseja publicar os resultados. • Adicione o novo parâmetro de `ImageDirectory` configuração que permite especificar um diretório personalizado para a imagem na qual você deseja realizar a inferência. Correções de erros e melhorias • Grave os resultados da inferência no arquivo de log do componente em vez de em um arquivo de inferência separado. • Use o módulo de registro do software AWS IoT Greengrass Core para registrar a saída do componente. • Use o AWS IoT Device SDK para ler a configuração do componente e aplicar as alterações na configuração.
2.0.4	Versão inicial.

Armazenamento de modelos de classificação de imagens DLR

O armazenamento de modelos de classificação de imagens DLR é um componente de modelo de aprendizado de máquina que contém modelos pré-treinados de ResNet -50 como artefatos do Greengrass. Os modelos pré-treinados usados neste componente são obtidos do GluonCV Model Zoo e compilados usando o SageMaker Neo Deep Learning Runtime.

O componente de inferência de classificação de imagem DLR usa esse componente como uma dependência para a fonte do modelo. Para usar um modelo DLR personalizado, crie uma versão personalizada desse componente do modelo e inclua seu modelo personalizado como um artefato do componente. Você pode usar a receita desse componente como modelo para criar componentes de modelo personalizados.

> **Note**
>
> O nome do componente de armazenamento de modelos de classificação de imagem DLR varia de acordo com sua versão. O nome do componente para a versão 2.1.x e versões posteriores é. `variant.DLR.ImageClassification.ModelStore` O nome do componente para a versão 2.0.x é. `variant.ImageClassification.ModelStore`

Tópicos

Versões

Esse componente tem as seguintes versões:

- 2,1x () `variant.DLR.ImageClassification.ModelStore`
- 2,0x () `variant.ImageClassification.ModelStore`

Tipo

Este componente é um componente genérico (`aws.greengrass.generic`). O núcleo do Greengrass executa os scripts do ciclo de vida do componente.

Para ter mais informações, consulte Tipos de componentes.

Sistema operacional

Esse componente pode ser instalado em dispositivos principais que executam os seguintes sistemas operacionais:

- Linux
- Windows

Requisitos

Esse componente tem os seguintes requisitos:

- Nos principais dispositivos do Greengrass que executam o Amazon Linux 2 ou o Ubuntu 18.04, a GNU C Library (glibc) versão 2.27 ou posterior está instalada no dispositivo.
- Em dispositivos ARMv7L, como o Raspberry Pi, dependências do OpenCV-Python instaladas no dispositivo. Execute o comando a seguir para instalar as dependências.

  ```
  sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
   libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
  ```
- Os dispositivos Raspberry Pi que executam o Raspberry Pi OS Bullseye devem atender aos seguintes requisitos:
  - NumPy 1.22.4 ou posterior instalado no dispositivo. O Raspberry Pi OS Bullseye inclui uma versão anterior do NumPy, então você pode executar o seguinte comando para atualizar NumPy o dispositivo.

    ```
    pip3 install --upgrade numpy
    ```
  - A pilha de câmeras antiga ativada no dispositivo. O Raspberry Pi OS Bullseye inclui uma nova pilha de câmeras que é ativada por padrão e não é compatível, portanto, você deve habilitar a pilha de câmeras antiga.

    Para habilitar a pilha de câmeras antiga

    1. Execute o comando a seguir para abrir a ferramenta de configuração do Raspberry Pi.

       ```
       sudo raspi-config
       ```

2. Selecione Opções de interface.
3. Selecione Câmera antiga para ativar a pilha de câmeras antigas.
4. Reinicie o Raspberry Pi.

Dependências

Quando você implanta um componente, AWS IoT Greengrass também implanta versões compatíveis de suas dependências. Isso significa que você deve atender aos requisitos do componente e de todas as suas dependências para implantá-lo com êxito. Esta seção lista as dependências das versões lançadas desse componente e as restrições de versão semântica que definem as versões dos componentes para cada dependência. Você também pode visualizar as dependências de cada versão do componente no AWS IoT Greengrass console. Na página de detalhes do componente, procure a lista de dependências.

2.1.12 - 2.1.14

A tabela a seguir lista as dependências das versões 2.1.12 e 2.1.13 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.13.0	Flexível

2.1.11

A tabela a seguir lista as dependências da versão 2.1.11 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.12.0	Flexível

2.1.10

A tabela a seguir lista as dependências da versão 2.1.10 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.11.0	Flexível

2.1.9

A tabela a seguir lista as dependências da versão 2.1.9 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.10.0	Flexível

2.1.8

A tabela a seguir lista as dependências da versão 2.1.8 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.9.0	Flexível

2.1.7

A tabela a seguir lista as dependências da versão 2.1.7 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.8.0	Flexível

2.1.6

A tabela a seguir lista as dependências da versão 2.1.6 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.7.0	Flexível

2.1.5

A tabela a seguir lista as dependências da versão 2.1.5 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.6.0	Flexível

2.1.4

A tabela a seguir lista as dependências da versão 2.1.4 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.5.0	Flexível

2.1.3

A tabela a seguir lista as dependências da versão 2.1.3 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.4.0	Flexível

2.1.2

A tabela a seguir lista as dependências da versão 2.1.2 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.3.0	Flexível

2.1.1

A tabela a seguir lista as dependências da versão 2.1.1 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.2.0	Flexível

2.0.x

A tabela a seguir lista as dependências da versão 2.0.x desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	~2.0.0	Flexível

Configuração

Esse componente não tem nenhum parâmetro de configuração.

Arquivo de log local

Esse componente não gera registros.

Changelog

A tabela a seguir descreve as alterações em cada versão do componente.

Version (Versão)	Alterações
2.1.13	Versão atualizada para a versão 2.12.5 do Greengrass nucleus.
2.1.12	Versão atualizada para a versão 2.12.0 do Greengrass nucleus.
2.1.11	Versão atualizada para a versão 2.11.0 do Greengrass nucleus.
2.1.10	Versão atualizada para a versão 2.10.0 do Greengrass nucleus.
2.1.9	Versão atualizada para a versão 2.9.0 do Greengrass nucleus.
2.1.8	Versão atualizada para a versão 2.8.0 do Greengrass nucleus.
2.1.7	Versão atualizada para a versão 2.7.0 do Greengrass nucleus.
2.1.6	Versão atualizada para a versão 2.6.0 do Greengrass nucleus.
2.1.5	Novos atributos • Adiciona exemplos de modelos de classificação de imagens para dispositivos principais do Windows.

Version (Versão)	Alterações
	• Versão atualizada para a versão 2.5.0 do Greengrass nucleus.
2.1.4	Versão atualizada para a versão 2.4.0 do Greengrass nucleus.
2.1.3	Versão atualizada para a versão 2.3.0 do Greengrass nucleus.
2.1.2	Versão atualizada para a versão 2.2.0 do Greengrass nucleus.
2.1.1	Novos atributos • Adicione um modelo de classificação de imagem de amostra ResNet -50 para plataformas Armv8 (AArch64). Isso amplia o suporte ao aprendizado de máquina para os principais dispositivos do Greengrass que executam o NVIDIA Jetson, como o Jetson Nano.
2.0.4	Versão inicial.

Armazenamento de modelos de detecção de objetos DLR

O repositório de modelos de detecção de objetos DLR é um componente de modelo de aprendizado de máquina que contém modelos YOLOv3 pré-treinados como artefatos do Greengrass. Os modelos de amostra usados neste componente são obtidos do GluonCV Model Zoo e compilados usando o SageMaker Neo Deep Learning Runtime.

O componente de inferência de detecção de objetos DLR usa esse componente como uma dependência para a fonte do modelo. Para usar um modelo DLR personalizado, crie uma versão personalizada desse componente do modelo e inclua seu modelo personalizado como um artefato do componente. Você pode usar a receita desse componente como modelo para criar componentes de modelo personalizados.

> **Note**
>
> O nome do componente de armazenamento de modelos de detecção de objetos DLR varia de acordo com sua versão. O nome do componente para a versão 2.1.x e versões posteriores é. `variant.DLR.ObjectDetection.ModelStore` O nome do componente para a versão 2.0.x é. `variant.ObjectDetection.ModelStore`

Tópicos

Versões

Esse componente tem as seguintes versões:

- 2.1.x
- 2.0.x

Tipo

Este componente é um componente genérico (`aws.greengrass.generic`). O núcleo do Greengrass executa os scripts do ciclo de vida do componente.

Para ter mais informações, consulte Tipos de componentes.

Sistema operacional

Esse componente pode ser instalado em dispositivos principais que executam os seguintes sistemas operacionais:

- Linux
- Windows

Requisitos

Esse componente tem os seguintes requisitos:

- Nos principais dispositivos do Greengrass que executam o Amazon Linux 2 ou o Ubuntu 18.04, a GNU C Library (glibc) versão 2.27 ou posterior está instalada no dispositivo.
- Em dispositivos ARMv7L, como o Raspberry Pi, dependências do OpenCV-Python instaladas no dispositivo. Execute o comando a seguir para instalar as dependências.

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
 libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- Os dispositivos Raspberry Pi que executam o Raspberry Pi OS Bullseye devem atender aos seguintes requisitos:
  - NumPy 1.22.4 ou posterior instalado no dispositivo. O Raspberry Pi OS Bullseye inclui uma versão anterior do NumPy, então você pode executar o seguinte comando para atualizar NumPy o dispositivo.

    ```
    pip3 install --upgrade numpy
    ```

  - A pilha de câmeras antiga ativada no dispositivo. O Raspberry Pi OS Bullseye inclui uma nova pilha de câmeras que é ativada por padrão e não é compatível, portanto, você deve habilitar a pilha de câmeras antiga.

    Para habilitar a pilha de câmeras antiga

    1. Execute o comando a seguir para abrir a ferramenta de configuração do Raspberry Pi.

       ```
       sudo raspi-config
       ```

    2. Selecione Opções de interface.
    3. Selecione Câmera antiga para ativar a pilha de câmeras antigas.
    4. Reinicie o Raspberry Pi.

Dependências

Quando você implanta um componente, AWS IoT Greengrass também implanta versões compatíveis de suas dependências. Isso significa que você deve atender aos requisitos do componente e de todas as suas dependências para implantá-lo com êxito. Esta seção lista as dependências das versões lançadas desse componente e as restrições de versão semântica que definem as versões do componente para cada dependência. Você também pode visualizar as dependências de cada

versão do componente no AWS IoT Greengrass console. Na página de detalhes do componente, procure a lista de dependências.

2.1.13 and 2.1.14

A tabela a seguir lista as dependências das versões 2.1.13 e 2.1.14 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.13.0	Flexível

2.1.12

A tabela a seguir lista as dependências da versão 2.1.12 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.12.0	Flexível

2.1.11

A tabela a seguir lista as dependências da versão 2.1.11 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.11.0	Flexível

2.1.10

A tabela a seguir lista as dependências da versão 2.1.10 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.10.0	Flexível

2.1.9

A tabela a seguir lista as dependências da versão 2.1.9 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.9.0	Flexível

2.1.8

A tabela a seguir lista as dependências da versão 2.1.8 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.8.0	Flexível

2.1.7

A tabela a seguir lista as dependências da versão 2.1.7 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.7.0	Flexível

2.1.5 and 2.1.6

A tabela a seguir lista as dependências das versões 2.1.5 e 2.1.6 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.6.0	Flexível

2.1.4

A tabela a seguir lista as dependências da versão 2.1.4 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.5.0	Flexível

2.1.3

A tabela a seguir lista as dependências da versão 2.1.3 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.4.0	Flexível

2.1.2

A tabela a seguir lista as dependências da versão 2.1.2 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.3.0	Flexível

2.1.1

A tabela a seguir lista as dependências da versão 2.1.1 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.2.0	Flexível

2.0.x

A tabela a seguir lista as dependências da versão 2.0.x desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	~2.0.0	Flexível

Configuração

Esse componente não tem nenhum parâmetro de configuração.

Arquivo de log local

Esse componente não gera registros.

Changelog

A tabela a seguir descreve as alterações em cada versão do componente.

Version (Versão)	Alterações
2.1.14	Versão atualizada para a versão 2.12.5 do Greengrass nucleus.
2.1.13	Versão atualizada para a versão 2.12.0 do Greengrass nucleus.
2.1.12	Versão atualizada para a versão 2.11.0 do Greengrass nucleus.
2.1.11	Versão atualizada para a versão 2.10.0 do Greengrass nucleus.
2.1.10	Versão atualizada para a versão 2.9.0 do Greengrass nucleus.
2.1.9	Versão atualizada para a versão 2.8.0 do Greengrass nucleus.
2.1.8	Versão atualizada para a versão 2.7.0 do Greengrass nucleus.
2.1.7	Versão atualizada para a versão 2.6.0 do Greengrass nucleus.
2.1.6	Adiciona um modelo de CPU para corrigir um problema em dispositivos Armv8 (AArch64).
2.1.5	Novos atributos • Adiciona modelos de detecção de objetos de amostra para dispositivos principais do Windows. Correções de erros e melhorias • Versão atualizada para a versão 2.5.0 do Greengrass nucleus.
2.1.4	Versão atualizada para a versão 2.4.0 do Greengrass nucleus.

Version (Versão)	Alterações
2.1.3	Versão atualizada para a versão 2.3.0 do Greengrass nucleus.
2.1.2	Versão atualizada para a versão 2.2.0 do Greengrass nucleus.
2.1.1	Novos atributos • Adicione uma amostra do modelo de detecção de objetos YOLOv3 para plataformas Armv8 (AArch64). Isso amplia o suporte ao aprendizado de máquina para os principais dispositivos do Greengrass que executam o NVIDIA Jetson, como o Jetson Nano.
2.0.4	Versão inicial.

Tempo de execução do DLR

O componente de tempo de execução do DLR (`variant.DLR`) contém um script que instala o Deep Learning Runtime (DLR) e suas dependências em um ambiente virtual no seu dispositivo. Os Detecção de objetos DLR componentes Classificação de imagens DLR e usam esse componente como uma dependência para instalar o DLR. A versão 1.6.x do componente instala o DLR v1.6.0 e a versão 1.3.x do componente instala o DLR v1.3.0.

Para usar um tempo de execução diferente, você pode usar a receita desse componente como modelo para criar um componente de aprendizado de máquina personalizado.

Tópicos

- Versões
- Tipo
- Sistema operacional
- Requisitos
- Dependências
- Configuração
- Uso
- Arquivo de log local
- Changelog

Versões

Esse componente tem as seguintes versões:

- 1.6.x
- 1.3.x

Tipo

Este componente é um componente genérico (`aws.greengrass.generic`). O núcleo do Greengrass executa os scripts do ciclo de vida do componente.

Para ter mais informações, consulte Tipos de componentes.

Sistema operacional

Esse componente pode ser instalado em dispositivos principais que executam os seguintes sistemas operacionais:

- Linux
- Windows

Requisitos

Esse componente tem os seguintes requisitos:

- Nos principais dispositivos do Greengrass que executam o Amazon Linux 2 ou o Ubuntu 18.04, a GNU C Library (glibc) versão 2.27 ou posterior está instalada no dispositivo.
- Em dispositivos ARMv7L, como o Raspberry Pi, dependências do OpenCV-Python instaladas no dispositivo. Execute o comando a seguir para instalar as dependências.

  ```
  sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
   libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
  ```

- Os dispositivos Raspberry Pi que executam o Raspberry Pi OS Bullseye devem atender aos seguintes requisitos:
  - NumPy 1.22.4 ou posterior instalado no dispositivo. O Raspberry Pi OS Bullseye inclui uma versão anterior do NumPy, então você pode executar o seguinte comando para atualizar NumPy o dispositivo.

```
pip3 install --upgrade numpy
```

- A pilha de câmeras antiga ativada no dispositivo. O Raspberry Pi OS Bullseye inclui uma nova pilha de câmeras que é ativada por padrão e não é compatível, portanto, você deve habilitar a pilha de câmeras antiga.

  Para habilitar a pilha de câmeras antiga

  1. Execute o comando a seguir para abrir a ferramenta de configuração do Raspberry Pi.

     ```
     sudo raspi-config
     ```

  2. Selecione Opções de interface.
  3. Selecione Câmera antiga para ativar a pilha de câmeras antigas.
  4. Reinicie o Raspberry Pi.

Endpoints e portas

Por padrão, esse componente usa um script de instalação para instalar pacotes usando os `pip` comandos `apt` `yum`,`brew`,, e, dependendo da plataforma usada pelo dispositivo principal. Esse componente deve ser capaz de realizar solicitações de saída para vários índices e repositórios de pacotes para executar o script do instalador. Para permitir o tráfego de saída desse componente por meio de um proxy ou firewall, você deve identificar os endpoints dos índices e repositórios de pacotes aos quais seu dispositivo principal se conecta para instalar.

Considere o seguinte ao identificar os endpoints necessários para o script de instalação desse componente:

- Os endpoints dependem da plataforma do dispositivo principal. Por exemplo, um dispositivo principal que executa o Ubuntu usa `apt` em vez de `yum` ou`brew`. Além disso, dispositivos que usam o mesmo índice de pacotes podem ter listas de fontes diferentes e, portanto, podem recuperar pacotes de repositórios diferentes.
- Os endpoints podem ser diferentes entre vários dispositivos que usam o mesmo índice de pacotes, porque cada dispositivo tem suas próprias listas de origem que definem onde recuperar os pacotes.

- Os endpoints podem mudar com o tempo. Cada índice de pacotes fornece os URLs dos repositórios nos quais você baixa pacotes, e o proprietário de um pacote pode alterar os URLs fornecidos pelo índice de pacotes.

Para obter mais informações sobre as dependências que esse componente instala e como desabilitar o script do instalador, consulte o parâmetro de UseInstallerconfiguração.

Para obter mais informações sobre terminais e portas necessários para a operação básica, consultePermitir o tráfego de dispositivos por meio de um proxy ou firewall.

Dependências

Quando você implanta um componente, AWS IoT Greengrass também implanta versões compatíveis de suas dependências. Isso significa que você deve atender aos requisitos do componente e de todas as suas dependências para implantá-lo com êxito. Esta seção lista as dependências das versões lançadas desse componente e as restrições de versão semântica que definem as versões do componente para cada dependência. Você também pode visualizar as dependências de cada versão do componente no AWS IoT Greengrass console. Na página de detalhes do componente, procure a lista de dependências.

1.6.11 - 1.6.16

A tabela a seguir lista as dependências das versões 1.6.11 a 1.6.16 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <3.0.0	Flexível

1.6.10

A tabela a seguir lista as dependências da versão 1.6.10 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.9.0	Flexível

1.6.9

A tabela a seguir lista as dependências da versão 1.6.9 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.8.0	Flexível

1.6.8

A tabela a seguir lista as dependências da versão 1.6.8 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.7.0	Flexível

1.6.6 and 1.6.7

A tabela a seguir lista as dependências das versões 1.6.6 e 1.6.7 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.6.0	Flexível

1.6.4 and 1.6.5

A tabela a seguir lista as dependências das versões 1.6.4 e 1.6.5 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.5.0	Flexível

1.6.3

A tabela a seguir lista as dependências da versão 1.6.3 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.4.0	Flexível

1.6.2

A tabela a seguir lista as dependências da versão 1.6.2 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.3.0	Flexível

1.6.1

A tabela a seguir lista as dependências da versão 1.6.1 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.2.0	Flexível

1.3.x

A tabela a seguir lista as dependências da versão 1.3.x desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	~2.0.0	Flexível

Para obter mais informações sobre dependências de componentes, consulte a referência da receita do componente.

Configuração

Esse componente fornece os seguintes parâmetros de configuração que você pode personalizar ao implantar o componente.

`MLRootPath`

(Opcional) O caminho da pasta nos dispositivos principais do Linux em que os componentes de inferência leem imagens e gravam resultados de inferência. Você pode modificar esse valor em qualquer local em seu dispositivo no qual o usuário que está executando esse componente tenha acesso de leitura/gravação.

Padrão: *`/greengrass/v2`*`/work/variant.DLR/greengrass_ml`

`WindowsMLRootPath`

Esse recurso está disponível na versão 1.6.6 e posterior desse componente.

(Opcional) O caminho da pasta no dispositivo principal do Windows em que os componentes de inferência leem imagens e gravam resultados de inferência. Você pode modificar esse valor em qualquer local em seu dispositivo no qual o usuário que está executando esse componente tenha acesso de leitura/gravação.

Padrão: *`C:\greengrass\v2`*`\\work\\variant.DLR\\greengrass_ml`

`UseInstaller`

(Opcional) Valor da string que define se o script do instalador deve ser usado nesse componente para instalar o DLR e suas dependências. Os valores compatíveis são `true` e `false`.

Defina esse valor como `false` se você quiser usar um script personalizado para a instalação do DLR ou se quiser incluir dependências de tempo de execução em uma imagem Linux pré-criada. Para usar esse componente com os componentes AWS de inferência DLR fornecidos, instale as bibliotecas a seguir, incluindo quaisquer dependências, e disponibilize-as para o usuário do sistema, como, por exemplo`ggc_user`, que executa os componentes de ML.

- Python 3.7 ou posterior, inclusive `pip` para sua versão do Python.
- Tempo de execução de aprendizado profundo v1.6.0
- NumPy.
- OpenCV-Python.
- AWS IoT Device SDK v2 para Python.
- AWS Python de tempo de execução comum (CRT).
- Picamera (somente para dispositivos Raspberry Pi).
- `awscam`módulo (para AWS DeepLens dispositivos).

- libGL (para dispositivos Linux)

Padrão: `true`

Uso

Use esse componente com o parâmetro `UseInstaller` de configuração definido `true` para instalar o DLR e suas dependências em seu dispositivo. O componente configura um ambiente virtual em seu dispositivo que inclui o OpenCV NumPy e as bibliotecas necessárias para o DLR.

> **Note**
>
> O script do instalador neste componente também instala as versões mais recentes das bibliotecas adicionais do sistema que são necessárias para configurar o ambiente virtual em seu dispositivo e usar a estrutura de aprendizado de máquina instalada. Isso pode atualizar as bibliotecas do sistema existentes em seu dispositivo. Consulte a tabela a seguir para ver a lista de bibliotecas que esse componente instala para cada sistema operacional compatível. Se você quiser personalizar esse processo de instalação, defina o parâmetro de `UseInstaller` configuração como `false` e desenvolva seu próprio script de instalação.

Plataforma	Bibliotecas instaladas no sistema do dispositivo	Bibliotecas instaladas no ambiente virtual
Armv7l	`build-essential` , `cmake`, `ca-certificates` , `git`	`setuptools` , `wheel`
Amazon Linux 2	`mesa-libGL`	Nenhum
Ubuntu	`wget`	Nenhum

Quando você implanta seu componente de inferência, esse componente de tempo de execução primeiro verifica se seu dispositivo já tem o DLR e suas dependências instaladas e, se não, ele os instala para você.

Arquivo de log local

Esse componente usa o seguinte arquivo de log.

Linux

```
/greengrass/v2/logs/variant.DLR.log
```

Windows

```
C:\greengrass\v2\logs\variant.DLR.log
```

Para visualizar os registros desse componente

- Execute o comando a seguir no dispositivo principal para visualizar o arquivo de log desse componente em tempo real. Substitua */greengrass/v2* ou *C:\greengrass\v2* pelo caminho para a pasta AWS IoT Greengrass raiz.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/variant.DLR.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\variant.DLR.log -Tail 10 -Wait
  ```

Changelog

A tabela a seguir descreve as alterações em cada versão do componente.

Version (Versão)	Alterações
1.6.16	Versão atualizada para o Greengrass nucleus versão 2.12.5.
1.6.12	Correções de erros e melhorias • Corrige o script de instalação para usuários do sistema operacional Windows.
1.6.11	Versão atualizada para a versão 2.9.0 do Greengrass nucleus.
1.6.10	Versão atualizada para a versão 2.8.0 do Greengrass nucleus.

Version (Versão)	Alterações
1.6.9	Versão atualizada para a versão 2.7.0 do Greengrass nucleus.
1.6.8	Versão atualizada para a versão 2.6.0 do Greengrass nucleus.
1.6.7	Correções de erros e melhorias • Atualiza o script de `UseInstaller` instalação para instalar o LibGL, que não está disponível por padrão em determinadas plataformas Linux. • Atualiza o script de `UseInstaller` instalação para sempre usar o Python 3.9 no ambiente virtual desse componente. Essa alteração ajuda a garantir a compatibilidade com outras bibliotecas.
1.6.6	Novos atributos • Adiciona suporte para dispositivos principais que executam o Windows. • Adiciona o novo parâmetro de `WindowsMLRootPath` configuração que você pode usar para configurar a pasta de resultados de inferência nos dispositivos principais do Windows.
1.6.5	Novos atributos • Adiciona o novo parâmetro de `UseInstaller` configuração que você pode usar para desativar o script de instalação nesse componente.
1.6.4	Versão atualizada para a versão 2.4.0 do Greengrass nucleus.
1.6.3	Versão atualizada para a versão 2.3.0 do Greengrass nucleus.
1.6.2	Versão atualizada para a versão 2.2.0 do Greengrass nucleus.

Version (Versão)	Alterações
1.6.1	Novos atributos • Instale o Deep Learning Runtime v1.6.0 e suas dependências. • Adicione suporte para instalação de DLR em plataformas Armv8 (AArch64). Isso amplia o suporte ao aprendizado de máquina para os principais dispositivos do Greengrass que executam o NVIDIA Jetson, como o Jetson Nano. Correções de erros e melhorias • Instale o AWS IoT Device SDK no ambiente virtual para ler a configura ção do componente e aplicar as alterações de configuração. • Correções e melhorias adicionais de pequenos bugs.
1.3.2	Versão inicial. Instala o DLR v1.3.0.

TensorFlow Classificação de imagens Lite

O componente de classificação de imagem TensorFlow Lite (`aws.greengrass.TensorFlowLiteImageClassification`) contém código de inferência de amostra para realizar inferência de classificação de imagem usando o tempo de execução TensorFlow Lite e uma amostra de modelo quantizado MobileNet 1.0 pré-treinado. Esse componente usa a variante TensorFlow Loja de modelos de classificação de imagens Lite e os TensorFlow Tempo de execução leve componentes como dependências para baixar o tempo de execução do TensorFlow Lite e o modelo de amostra.

Para usar esse componente de inferência com um modelo TensorFlow Lite personalizado, crie uma versão personalizada do componente de armazenamento de modelos dependente. Para usar seu próprio código de inferência personalizado, você pode usar a receita desse componente como modelo para criar um componente de inferência personalizado.

Tópicos

- Versões
- Tipo
- Sistema operacional
- Requisitos

- Dependências
- Configuração
- Arquivo de log local
- Changelog

Versões

Esse componente tem as seguintes versões:

- 2.1.x

Tipo

Este componente é um componente genérico (`aws.greengrass.generic`). O núcleo do Greengrass executa os scripts do ciclo de vida do componente.

Para ter mais informações, consulte Tipos de componentes.

Sistema operacional

Esse componente pode ser instalado em dispositivos principais que executam os seguintes sistemas operacionais:

- Linux
- Windows

Requisitos

Esse componente tem os seguintes requisitos:

- Nos principais dispositivos do Greengrass que executam o Amazon Linux 2 ou o Ubuntu 18.04, a GNU C Library (glibc) versão 2.27 ou posterior está instalada no dispositivo.
- Em dispositivos ARMv7L, como o Raspberry Pi, dependências do OpenCV-Python instaladas no dispositivo. Execute o comando a seguir para instalar as dependências.

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
 libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- Os dispositivos Raspberry Pi que executam o Raspberry Pi OS Bullseye devem atender aos seguintes requisitos:
  - NumPy 1.22.4 ou posterior instalado no dispositivo. O Raspberry Pi OS Bullseye inclui uma versão anterior do NumPy, então você pode executar o seguinte comando para atualizar NumPy o dispositivo.

    ```
    pip3 install --upgrade numpy
    ```

  - A pilha de câmeras antiga ativada no dispositivo. O Raspberry Pi OS Bullseye inclui uma nova pilha de câmeras que é ativada por padrão e não é compatível, portanto, você deve habilitar a pilha de câmeras antiga.

    Para habilitar a pilha de câmeras antiga

    1. Execute o comando a seguir para abrir a ferramenta de configuração do Raspberry Pi.

       ```
       sudo raspi-config
       ```

    2. Selecione Opções de interface.
    3. Selecione Câmera antiga para ativar a pilha de câmeras antigas.
    4. Reinicie o Raspberry Pi.

Dependências

Quando você implanta um componente, AWS IoT Greengrass também implanta versões compatíveis de suas dependências. Isso significa que você deve atender aos requisitos do componente e de todas as suas dependências para implantá-lo com êxito. Esta seção lista as dependências das versões lançadas desse componente e as restrições de versão semântica que definem as versões do componente para cada dependência. Você também pode visualizar as dependências de cada versão do componente no AWS IoT Greengrass console. Na página de detalhes do componente, procure a lista de dependências.

2.1.11 and 2.1.12

A tabela a seguir lista as dependências das versões 2.1.11 e 2.1.12 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.13.0	Flexível
TensorFlow Loja de modelos de classificação de imagens Lite	>=2.1.0 <2.2.0	Rígido
TensorFlow Leve	>=2,5.0 <2,6,0	Rígido

2.1.10

A tabela a seguir lista as dependências da versão 2.1.10 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.12.0	Flexível
TensorFlow Loja de modelos de classificação de imagens Lite	>=2.1.0 <2.2.0	Rígido
TensorFlow Leve	>=2,5.0 <2,6,0	Rígido

2.1.9

A tabela a seguir lista as dependências da versão 2.1.9 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.11.0	Flexível
TensorFlow Loja de modelos de classificação de imagens Lite	>=2.1.0 <2.2.0	Rígido
TensorFlow Leve	>=2,5.0 <2,6,0	Rígido

2.1.8

A tabela a seguir lista as dependências da versão 2.1.8 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.10.0	Flexível
TensorFlow Loja de modelos de classificação de imagens Lite	>=2.1.0 <2.2.0	Rígido
TensorFlow Leve	>=2,5.0 <2,6,0	Rígido

2.1.7

A tabela a seguir lista as dependências da versão 2.1.7 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.9.0	Flexível
TensorFlow Loja de modelos de classificação de imagens Lite	>=2.1.0 <2.2.0	Rígido
TensorFlow Leve	>=2,5.0 <2,6,0	Rígido

2.1.6

A tabela a seguir lista as dependências da versão 2.1.6 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.8.0	Flexível

Dependência	Versões compatíveis	Tipo de dependência
TensorFlow Loja de modelos de classificação de imagens Lite	>=2.1.0 <2.2.0	Rígido
TensorFlow Leve	>=2,5.0 <2,6,0	Rígido

2.1.5

A tabela a seguir lista as dependências da versão 2.1.5 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.7.0	Flexível
TensorFlow Loja de modelos de classificação de imagens Lite	>=2.1.0 <2.2.0	Rígido
TensorFlow Leve	>=2,5.0 <2,6,0	Rígido

2.1.4

A tabela a seguir lista as dependências da versão 2.1.4 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.6.0	Flexível
TensorFlow Loja de modelos de classificação de imagens Lite	>=2.1.0 <2.2.0	Rígido
TensorFlow Leve	>=2,5.0 <2,6,0	Rígido

2.1.3

A tabela a seguir lista as dependências da versão 2.1.3 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.5.0	Flexível
TensorFlow Loja de modelos de classificação de imagens Lite	>=2.1.0 <2.2.0	Rígido
TensorFlow Leve	>=2,5.0 <2,6,0	Rígido

2.1.2

A tabela a seguir lista as dependências da versão 2.1.2 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.4.0	Flexível
TensorFlow Loja de modelos de classificação de imagens Lite	>=2.1.0 <2.2.0	Rígido
TensorFlow Leve	>=2,5.0 <2,6,0	Rígido

2.1.1

A tabela a seguir lista as dependências da versão 2.1.1 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.3.0	Flexível

Dependência	Versões compatíveis	Tipo de dependência
TensorFlow Loja de modelos de classificação de imagens Lite	>=2.1.0 <2.2.0	Rígido
TensorFlow Leve	>=2,5.0 <2,6,0	Rígido

2.1.0

A tabela a seguir lista as dependências da versão 2.1.0 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.2.0	Flexível
TensorFlow Loja de modelos de classificação de imagens Lite	>=2.1.0 <2.2.0	Rígido
TensorFlow Leve	>=2,5.0 <2,6,0	Rígido

Configuração

Esse componente fornece os seguintes parâmetros de configuração que você pode personalizar ao implantar o componente.

`accessControl`

(Opcional) O objeto que contém a política de autorização que permite que o componente publique mensagens no tópico de notificações padrão.

Padrão:

```
{
   "aws.greengrass.ipc.mqttproxy": {
      "aws.greengrass.TensorFlowLiteImageClassification:mqttproxy:1": {
         "policyDescription": "Allows access to publish via topic ml/tflite/image-
classification.",
```

```
            "operations": [
               "aws.greengrass#PublishToIoTCore"
            ],
            "resources": [
               "ml/tflite/image-classification"
            ]
         }
      }
   }
```

`PublishResultsOnTopic`

(Opcional) O tópico no qual você deseja publicar os resultados da inferência. Se você modificar esse valor, também deverá modificar o valor de `resources` no `accessControl` parâmetro para corresponder ao nome do tópico personalizado.

Padrão: `ml/tflite/image-classification`

`Accelerator`

O acelerador que você deseja usar. Os valores compatíveis são `cpu` e `gpu`.

Os modelos de amostra no componente do modelo dependente oferecem suporte somente à aceleração da CPU. Para usar a aceleração de GPU com um modelo personalizado diferente, crie um componente de modelo personalizado para substituir o componente de modelo público.

Padrão: `cpu`

`ImageDirectory`

(Opcional) O caminho da pasta no dispositivo em que os componentes de inferência leem imagens. Você pode modificar esse valor para qualquer local em seu dispositivo ao qual tenha acesso de leitura/gravação.

Padrão: *`/greengrass/v2`*`/packages/artifacts-unarchived/`*`component-name`*`/image_classification/sample_images/`

> **Note**
>
> Se você definir o valor de `UseCamera` para `true`, esse parâmetro de configuração será ignorado.

`ImageName`

(Opcional) O nome da imagem que o componente de inferência usa como entrada para uma previsão de criação. O componente procura a imagem na pasta especificada em`ImageDirectory`. Por padrão, o componente usa a imagem de amostra no diretório de imagens padrão. AWS IoT Greengrass suporta os seguintes formatos de imagem: `jpeg` `jpg`png,, npy e.

Padrão: `cat.jpeg`

> **Note**
>
> Se você definir o valor de `UseCamera` para`true`, esse parâmetro de configuração será ignorado.

`InferenceInterval`

(Opcional) O tempo em segundos entre cada previsão feita pelo código de inferência. O código de inferência de amostra é executado indefinidamente e repete suas previsões no intervalo de tempo especificado. Por exemplo, você pode alterar isso para um intervalo menor se quiser usar imagens tiradas por uma câmera para previsão em tempo real.

Padrão: `3600`

`ModelResourceKey`

(Opcional) Os modelos usados no componente de modelo público dependente. Modifique esse parâmetro somente se você substituir o componente do modelo público por um componente personalizado.

Padrão:

```
{
    "model": "TensorFlowLite-Mobilenet"
}
```

`UseCamera`

(Opcional) Valor da string que define se as imagens de uma câmera conectada ao dispositivo principal do Greengrass devem ser usadas. Os valores compatíveis são `true` e `false`.

Quando você define esse valor como`true`, o código de inferência de amostra acessa a câmera no seu dispositivo e executa a inferência localmente na imagem capturada. Os valores dos `ImageDirectory` parâmetros `ImageName` e são ignorados. Certifique-se de que o usuário que está executando esse componente tenha acesso de leitura/gravação ao local onde a câmera armazena as imagens capturadas.

Padrão: `false`

> **Note**
>
> Quando você visualiza a receita desse componente, o parâmetro de `UseCamera` configuração não aparece na configuração padrão. No entanto, você pode modificar o valor desse parâmetro em uma atualização de mesclagem de configuração ao implantar o componente.
> Ao definir como `UseCameratrue`, você também deve criar um link simbólico para permitir que o componente de inferência acesse sua câmera a partir do ambiente virtual criado pelo componente de tempo de execução. Para obter mais informações sobre o uso de uma câmera com os componentes de inferência de amostra, consulteAtualizar configurações de componentes.

Arquivo de log local

Esse componente usa o seguinte arquivo de log.

Linux

```
/greengrass/v2/logs/aws.greengrass.TensorFlowLiteImageClassification.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.TensorFlowLiteImageClassification.log
```

Para ver os registros desse componente

- Execute o comando a seguir no dispositivo principal para visualizar o arquivo de log desse componente em tempo real. Substitua */greengrass/v2* ou *C:\greengrass\v2* pelo caminho para a pasta AWS IoT Greengrass raiz.

Linux

```
sudo tail -f /greengrass/v2/logs/
aws.greengrass.TensorFlowLiteImageClassification.log
```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs
\aws.greengrass.TensorFlowLiteImageClassification.log -Tail 10 -Wait
```

Changelog

A tabela a seguir descreve as alterações em cada versão do componente.

Version (Versão)	Alterações
2.1.12	Versão atualizada para a versão 2.12.5 do Greengrass nucleus.
2.1.11	Versão atualizada para a versão 2.12.0 do Greengrass nucleus.
2.1.10	Versão atualizada para a versão 2.11.0 do Greengrass nucleus.
2.1.9	Versão atualizada para a versão 2.10.0 do Greengrass nucleus.
2.1.8	Versão atualizada para a versão 2.9.0 do Greengrass nucleus.
2.1.7	Versão atualizada para a versão 2.8.0 do Greengrass nucleus.
2.1.6	Versão atualizada para a versão 2.7.0 do Greengrass nucleus.
2.1.5	Versão atualizada para a versão 2.6.0 do Greengrass nucleus.
2.1.4	Versão atualizada para a versão 2.5.0 do Greengrass nucleus.
2.1.3	Versão atualizada para a versão 2.4.0 do Greengrass nucleus.
2.1.2	Versão atualizada para a versão 2.3.0 do Greengrass nucleus.
2.1.1	Versão atualizada para a versão 2.2.0 do Greengrass nucleus.

Version (Versão)	Alterações
2.1.0	Versão inicial.

TensorFlow Detecção leve de objetos

O componente de detecção de objetos TensorFlow Lite (`aws.greengrass.TensorFlowLiteObjectDetection`) contém um código de inferência de amostra para realizar a inferência de detecção de objetos usando o TensorFlow Lite e um modelo pré-treinado de Single Shot Detection (SSD) 1.0. MobileNet Esse componente usa a variante TensorFlow Loja de modelos de detecção de objetos Lite e os TensorFlow Tempo de execução leve componentes como dependências para baixar o TensorFlow Lite e o modelo de amostra.

Para usar esse componente de inferência com um modelo TensorFlow Lite personalizado, você pode criar uma versão personalizada do componente de armazenamento de modelos dependente. Para usar seu próprio código de inferência personalizado, use a receita desse componente como modelo para criar um componente de inferência personalizado.

Tópicos

- Versões
- Tipo
- Sistema operacional
- Requisitos
- Dependências
- Configuração
- Arquivo de log local
- Changelog

Versões

Esse componente tem as seguintes versões:

- 2.1.x

Tipo

Este componente é um componente genérico (`aws.greengrass.generic`). O núcleo do Greengrass executa os scripts do ciclo de vida do componente.

Para ter mais informações, consulte Tipos de componentes.

Sistema operacional

Esse componente pode ser instalado em dispositivos principais que executam os seguintes sistemas operacionais:

- Linux
- Windows

Requisitos

Esse componente tem os seguintes requisitos:

- Nos principais dispositivos do Greengrass que executam o Amazon Linux 2 ou o Ubuntu 18.04, a GNU C Library (glibc) versão 2.27 ou posterior está instalada no dispositivo.
- Em dispositivos ARMv7L, como o Raspberry Pi, dependências do OpenCV-Python instaladas no dispositivo. Execute o comando a seguir para instalar as dependências.

  ```
  sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
   libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
  ```

- Os dispositivos Raspberry Pi que executam o Raspberry Pi OS Bullseye devem atender aos seguintes requisitos:
  - NumPy 1.22.4 ou posterior instalado no dispositivo. O Raspberry Pi OS Bullseye inclui uma versão anterior do NumPy, então você pode executar o seguinte comando para atualizar NumPy o dispositivo.

    ```
    pip3 install --upgrade numpy
    ```

  - A pilha de câmeras antiga ativada no dispositivo. O Raspberry Pi OS Bullseye inclui uma nova pilha de câmeras que é ativada por padrão e não é compatível, portanto, você deve habilitar a pilha de câmeras antiga.

Para habilitar a pilha de câmeras antiga

1. Execute o comando a seguir para abrir a ferramenta de configuração do Raspberry Pi.

   ```
   sudo raspi-config
   ```

2. Selecione Opções de interface.
3. Selecione Câmera antiga para ativar a pilha de câmeras antigas.
4. Reinicie o Raspberry Pi.

Dependências

Quando você implanta um componente, AWS IoT Greengrass também implanta versões compatíveis de suas dependências. Isso significa que você deve atender aos requisitos do componente e de todas as suas dependências para implantá-lo com êxito. Esta seção lista as dependências das versões lançadas desse componente e as restrições de versão semântica que definem as versões do componente para cada dependência. Você também pode visualizar as dependências de cada versão do componente no AWS IoT Greengrass console. Na página de detalhes do componente, procure a lista de dependências.

2.1.11 and 2.1.12

A tabela a seguir lista as dependências das versões 2.1.11 e 2.1.12 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.13.0	Flexível
TensorFlow Loja de modelos de classificação de imagens Lite	>=2.1.0 <2.2.0	Rígido
TensorFlow Leve	>=2,5.0 <2,6,0	Rígido

2.1.10

A tabela a seguir lista as dependências da versão 2.1.10 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.12.0	Flexível
TensorFlow Loja de modelos de classificação de imagens Lite	>=2.1.0 <2.2.0	Rígido
TensorFlow Leve	>=2,5.0 <2,6,0	Rígido

2.1.9

A tabela a seguir lista as dependências da versão 2.1.9 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.11.0	Flexível
TensorFlow Loja de modelos de classificação de imagens Lite	>=2.1.0 <2.2.0	Rígido
TensorFlow Leve	>=2,5.0 <2,6,0	Rígido

2.1.8

A tabela a seguir lista as dependências da versão 2.1.8 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.10.0	Flexível
TensorFlow Loja de modelos de classificação de imagens Lite	>=2.1.0 <2.2.0	Rígido
TensorFlow Leve	>=2,5.0 <2,6,0	Rígido

2.1.7

A tabela a seguir lista as dependências da versão 2.1.7 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.9.0	Flexível
TensorFlow Loja de modelos de classificação de imagens Lite	>=2.1.0 <2.2.0	Rígido
TensorFlow Leve	>=2,5.0 <2,6,0	Rígido

2.1.6

A tabela a seguir lista as dependências da versão 2.1.6 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.8.0	Flexível
TensorFlow Loja de modelos de classificação de imagens Lite	>=2.1.0 <2.2.0	Rígido
TensorFlow Leve	>=2,5.0 <2,6,0	Rígido

2.1.5

A tabela a seguir lista as dependências da versão 2.1.5 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.7.0	Flexível

Dependência	Versões compatíveis	Tipo de dependência
TensorFlow Loja de modelos de classificação de imagens Lite	>=2.1.0 <2.2.0	Rígido
TensorFlow Leve	>=2,5.0 <2,6,0	Rígido

2.1.4

A tabela a seguir lista as dependências da versão 2.1.4 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.6.0	Flexível
TensorFlow Loja de modelos de classificação de imagens Lite	>=2.1.0 <2.2.0	Rígido
TensorFlow Leve	>=2,5.0 <2,6,0	Rígido

2.1.3

A tabela a seguir lista as dependências da versão 2.1.3 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.5.0	Flexível
TensorFlow Loja de modelos de classificação de imagens Lite	>=2.1.0 <2.2.0	Rígido
TensorFlow Leve	>=2,5.0 <2,6,0	Rígido

2.1.2

A tabela a seguir lista as dependências da versão 2.1.2 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.4.0	Flexível
TensorFlow Loja de modelos de classificação de imagens Lite	>=2.1.0 <2.2.0	Rígido
TensorFlow Leve	>=2,5.0 <2,6,0	Rígido

2.1.1

A tabela a seguir lista as dependências da versão 2.1.1 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.3.0	Flexível
TensorFlow Loja de modelos de classificação de imagens Lite	>=2.1.0 <2.2.0	Rígido
TensorFlow Leve	>=2,5.0 <2,6,0	Rígido

2.1.0

A tabela a seguir lista as dependências da versão 2.1.0 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.2.0	Flexível

Dependência	Versões compatíveis	Tipo de dependência
TensorFlow Loja de modelos de classificação de imagens Lite	>=2.1.0 <2.2.0	Rígido
TensorFlow Leve	>=2,5.0 <2,6,0	Rígido

Configuração

Esse componente fornece os seguintes parâmetros de configuração que você pode personalizar ao implantar o componente.

`accessControl`

(Opcional) O objeto que contém a política de autorização que permite que o componente publique mensagens no tópico de notificações padrão.

Padrão:

```
{
   "aws.greengrass.ipc.mqttproxy": {
      "aws.greengrass.TensorFlowLiteObjectDetection:mqttproxy:1": {
         "policyDescription": "Allows access to publish via topic ml/tflite/object-
detection.",
         "operations": [
            "aws.greengrass#PublishToIoTCore"
         ],
         "resources": [
            "ml/tflite/object-detection"
         ]
      }
   }
}
```

`PublishResultsOnTopic`

(Opcional) O tópico no qual você deseja publicar os resultados da inferência. Se você modificar esse valor, também deverá modificar o valor de `resources` no `accessControl` parâmetro para corresponder ao nome do tópico personalizado.

Padrão: `ml/tflite/object-detection`

`Accelerator`

O acelerador que você deseja usar. Os valores compatíveis são `cpu` e `gpu`.

Os modelos de amostra no componente do modelo dependente oferecem suporte somente à aceleração da CPU. Para usar a aceleração de GPU com um modelo personalizado diferente, crie um componente de modelo personalizado para substituir o componente de modelo público.

Padrão: `cpu`

`ImageDirectory`

(Opcional) O caminho da pasta no dispositivo em que os componentes de inferência leem imagens. Você pode modificar esse valor para qualquer local em seu dispositivo ao qual tenha acesso de leitura/gravação.

Padrão: *`/greengrass/v2`*`/packages/artifacts-unarchived/`*`component-name`*`/object_detection/sample_images/`

> **Note**
>
> Se você definir o valor de `UseCamera` para `true`, esse parâmetro de configuração será ignorado.

`ImageName`

(Opcional) O nome da imagem que o componente de inferência usa como entrada para uma previsão de criação. O componente procura a imagem na pasta especificada em `ImageDirectory`. Por padrão, o componente usa a imagem de amostra no diretório de imagens padrão. AWS IoT Greengrass suporta os seguintes formatos de imagem: `jpeg` `jpg`png,, `npy` e.

Padrão: `objects.jpg`

> **Note**
>
> Se você definir o valor de `UseCamera` para `true`, esse parâmetro de configuração será ignorado.

`InferenceInterval`

(Opcional) O tempo em segundos entre cada previsão feita pelo código de inferência. O código de inferência de amostra é executado indefinidamente e repete suas previsões no intervalo de tempo especificado. Por exemplo, você pode alterar isso para um intervalo menor se quiser usar imagens tiradas por uma câmera para previsão em tempo real.

Padrão: `3600`

`ModelResourceKey`

(Opcional) Os modelos usados no componente de modelo público dependente. Modifique esse parâmetro somente se você substituir o componente do modelo público por um componente personalizado.

Padrão:

```
{
    "model": "TensorFlowLite-SSD"
}
```

`UseCamera`

(Opcional) Valor da string que define se as imagens de uma câmera conectada ao dispositivo principal do Greengrass devem ser usadas. Os valores compatíveis são `true` e `false`.

Quando você define esse valor como`true`, o código de inferência de amostra acessa a câmera no seu dispositivo e executa a inferência localmente na imagem capturada. Os valores dos `ImageDirectory` parâmetros `ImageName` e são ignorados. Certifique-se de que o usuário que está executando esse componente tenha acesso de leitura/gravação ao local onde a câmera armazena as imagens capturadas.

Padrão: `false`

> **Note**
>
> Quando você visualiza a receita desse componente, o parâmetro de `UseCamera` configuração não aparece na configuração padrão. No entanto, você pode modificar o valor desse parâmetro em uma atualização de mesclagem de configuração ao implantar o componente.

Ao definir como `UseCameratrue`, você também deve criar um link simbólico para permitir que o componente de inferência acesse sua câmera a partir do ambiente virtual criado pelo componente de tempo de execução. Para obter mais informações sobre como usar uma câmera com os componentes de inferência de amostra, consulteAtualizar configurações de componentes.

Note

Quando você visualiza a receita desse componente, o parâmetro de `UseCamera` configuração não aparece na configuração padrão. No entanto, você pode modificar o valor desse parâmetro em uma atualização de mesclagem de configuração ao implantar o componente.
Ao definir como `UseCameratrue`, você também deve criar um link simbólico para permitir que o componente de inferência acesse sua câmera a partir do ambiente virtual criado pelo componente de tempo de execução. Para obter mais informações sobre como usar uma câmera com os componentes de inferência de amostra, consulteAtualizar configurações de componentes.

Arquivo de log local

Esse componente usa o seguinte arquivo de log.

Linux

```
/greengrass/v2/logs/aws.greengrass.TensorFlowLiteObjectDetection.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.TensorFlowLiteObjectDetection.log
```

Para ver os registros desse componente

- Execute o comando a seguir no dispositivo principal para visualizar o arquivo de log desse componente em tempo real. Substitua */greengrass/v2* ou *C:\greengrass\v2* pelo caminho para a pasta AWS IoT Greengrass raiz.

Linux

```
sudo tail -f /greengrass/v2/logs/
aws.greengrass.TensorFlowLiteObjectDetection.log
```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs
\aws.greengrass.TensorFlowLiteObjectDetection.log -Tail 10 -Wait
```

Changelog

A tabela a seguir descreve as alterações em cada versão do componente.

Version (Versão)	Alterações
2.1.12	Versão atualizada para a versão 2.12.5 do Greengrass nucleus.
2.1.11	Versão atualizada para a versão 2.12.0 do Greengrass nucleus.
2.1.10	Versão atualizada para a versão 2.11.0 do Greengrass nucleus.
2.1.9	Versão atualizada para a versão 2.10.0 do Greengrass nucleus.
2.1.8	Versão atualizada para a versão 2.9.0 do Greengrass nucleus.
2.1.7	Versão atualizada para a versão 2.8.0 do Greengrass nucleus.
2.1.6	Versão atualizada para a versão 2.7.0 do Greengrass nucleus.
2.1.5	Versão atualizada para a versão 2.6.0 do Greengrass nucleus.
2.1.4	Versão atualizada para a versão 2.5.0 do Greengrass nucleus.
2.1.3	Versão atualizada para a versão 2.4.0 do Greengrass nucleus.
2.1.2	Versão atualizada para a versão 2.3.0 do Greengrass nucleus.

Version (Versão)	Alterações
2.1.1	Correções de erros e melhorias • Corrige um problema de escala de imagem que resultou em caixas delimitadoras imprecisas nos resultados de inferência de detecção de objetos TensorFlow Lite de amostra.
2.1.0	Versão inicial.

TensorFlow Loja de modelos de classificação de imagens Lite

O repositório de modelos de classificação de imagens TensorFlow Lite (`variant.TensorFlowLite.ImageClassification.ModelStore`) é um componente do modelo de aprendizado de máquina que contém um modelo MobileNet v1 pré-treinado como um artefato do Greengrass. O modelo de amostra usado nesse componente é obtido do TensorFlowHub e implementado usando o TensorFlow Lite.

O componente de TensorFlow Classificação de imagens Lite inferência usa esse componente como uma dependência para a fonte do modelo. Para usar um modelo TensorFlow Lite personalizado, crie uma versão personalizada desse componente de modelo e inclua seu modelo personalizado como um artefato de componente. Você pode usar a receita desse componente como modelo para criar componentes de modelo personalizados.

Tópicos

- Versões
- Tipo
- Sistema operacional
- Requisitos
- Dependências
- Configuração
- Arquivo de log local
- Changelog

Versões

Esse componente tem as seguintes versões:

- 2.1.x

Tipo

Este componente é um componente genérico (`aws.greengrass.generic`). O núcleo do Greengrass executa os scripts do ciclo de vida do componente.

Para ter mais informações, consulte Tipos de componentes.

Sistema operacional

Esse componente pode ser instalado em dispositivos principais que executam os seguintes sistemas operacionais:

- Linux
- Windows

Requisitos

Esse componente tem os seguintes requisitos:

- Nos principais dispositivos do Greengrass que executam o Amazon Linux 2 ou o Ubuntu 18.04, a GNU C Library (glibc) versão 2.27 ou posterior está instalada no dispositivo.
- Em dispositivos ARMv7L, como o Raspberry Pi, dependências do OpenCV-Python instaladas no dispositivo. Execute o comando a seguir para instalar as dependências.

  ```
  sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
   libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
  ```

- Os dispositivos Raspberry Pi que executam o Raspberry Pi OS Bullseye devem atender aos seguintes requisitos:
  - NumPy 1.22.4 ou posterior instalado no dispositivo. O Raspberry Pi OS Bullseye inclui uma versão anterior do NumPy, então você pode executar o seguinte comando para atualizar NumPy o dispositivo.

```
pip3 install --upgrade numpy
```

- A pilha de câmeras antiga ativada no dispositivo. O Raspberry Pi OS Bullseye inclui uma nova pilha de câmeras que é ativada por padrão e não é compatível, portanto, você deve habilitar a pilha de câmeras antiga.

  Para habilitar a pilha de câmeras antiga

  1. Execute o comando a seguir para abrir a ferramenta de configuração do Raspberry Pi.

     ```
     sudo raspi-config
     ```

  2. Selecione Opções de interface.
  3. Selecione Câmera antiga para ativar a pilha de câmeras antigas.
  4. Reinicie o Raspberry Pi.

Dependências

Quando você implanta um componente, AWS IoT Greengrass também implanta versões compatíveis de suas dependências. Isso significa que você deve atender aos requisitos do componente e de todas as suas dependências para implantá-lo com êxito. Esta seção lista as dependências das versões lançadas desse componente e as restrições de versão semântica que definem as versões do componente para cada dependência. Você também pode visualizar as dependências de cada versão do componente no AWS IoT Greengrass console. Na página de detalhes do componente, procure a lista de dependências.

2.1.11 and 2.1.12

A tabela a seguir lista as dependências das versões 2.1.11 e 2.1.12 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.13.0	Flexível

2.1.10

A tabela a seguir lista as dependências da versão 2.1.10 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.12.0	Flexível

2.1.9

A tabela a seguir lista as dependências da versão 2.1.9 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.11.0	Flexível

2.1.8

A tabela a seguir lista as dependências da versão 2.1.8 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.10.0	Flexível

2.1.7

A tabela a seguir lista as dependências da versão 2.1.7 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.9.0	Flexível

2.1.6

A tabela a seguir lista as dependências da versão 2.1.6 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.8.0	Flexível

2.1.5

A tabela a seguir lista as dependências da versão 2.1.5 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.7.0	Flexível

2.1.4

A tabela a seguir lista as dependências da versão 2.1.4 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.6.0	Flexível

2.1.3

A tabela a seguir lista as dependências da versão 2.1.3 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.5.0	Flexível

2.1.2

A tabela a seguir lista as dependências da versão 2.1.2 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.4.0	Flexível

2.1.1

A tabela a seguir lista as dependências da versão 2.1.1 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.3.0	Flexível

2.1.0

A tabela a seguir lista as dependências da versão 2.1.0 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.2.0	Flexível

Configuração

Esse componente não tem nenhum parâmetro de configuração.

Arquivo de log local

Esse componente não gera registros.

Changelog

A tabela a seguir descreve as alterações em cada versão do componente.

Version (Versão)	Alterações
2.1.12	Versão atualizada para a versão 2.12.5 do Greengrass nucleus.
2.1.11	Versão atualizada para a versão 2.12.0 do Greengrass nucleus.
2.1.10	Versão atualizada para a versão 2.11.0 do Greengrass nucleus.
2.1.9	Versão atualizada para a versão 2.10.0 do Greengrass nucleus.
2.1.8	Versão atualizada para a versão 2.9.0 do Greengrass nucleus.
2.1.7	Versão atualizada para a versão 2.8.0 do Greengrass nucleus.

Version (Versão)	Alterações
2.1.6	Versão atualizada para a versão 2.7.0 do Greengrass nucleus.
2.1.5	Versão atualizada para a versão 2.6.0 do Greengrass nucleus.
2.1.4	Versão atualizada para a versão 2.5.0 do Greengrass nucleus.
2.1.3	Versão atualizada para a versão 2.4.0 do Greengrass nucleus.
2.1.2	Versão atualizada para a versão 2.3.0 do Greengrass nucleus.
2.1.1	Versão atualizada para a versão 2.2.0 do Greengrass nucleus.
2.1.0	Versão inicial.

TensorFlow Loja de modelos de detecção de objetos Lite

O repositório de modelos de detecção de objetos TensorFlow Lite (`variant.TensorFlowLite.ObjectDetection.ModelStore`) é um componente de modelo de aprendizado de máquina que contém um modelo pré-treinado de detecção de disparo único (SSD) como um MobileNet artefato do Greengrass. O modelo de amostra usado nesse componente é obtido do TensorFlow Hub e implementado usando o TensorFlow Lite.

O componente de inferência de detecção de objetos TensorFlow Lite usa esse componente como uma dependência para a fonte do modelo. Para usar um modelo TensorFlow Lite personalizado, crie uma versão personalizada desse componente de modelo e inclua seu modelo personalizado como um artefato de componente. Você pode usar a receita desse componente como modelo para criar componentes de modelo personalizados.

Tópicos

- Versões
- Tipo
- Sistema operacional
- Requisitos
- Dependências
- Configuração

- Arquivo de log local
- Changelog

Versões

Esse componente tem as seguintes versões:

- 2.1.x

Tipo

Este componente é um componente genérico (`aws.greengrass.generic`). O núcleo do Greengrass executa os scripts do ciclo de vida do componente.

Para ter mais informações, consulte Tipos de componentes.

Sistema operacional

Esse componente pode ser instalado em dispositivos principais que executam os seguintes sistemas operacionais:

- Linux
- Windows

Requisitos

Esse componente tem os seguintes requisitos:

- Nos principais dispositivos do Greengrass que executam o Amazon Linux 2 ou o Ubuntu 18.04, a GNU C Library (glibc) versão 2.27 ou posterior está instalada no dispositivo.
- Em dispositivos ARMv7L, como o Raspberry Pi, dependências do OpenCV-Python instaladas no dispositivo. Execute o comando a seguir para instalar as dependências.

  ```
  sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
   libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
  ```

- Os dispositivos Raspberry Pi que executam o Raspberry Pi OS Bullseye devem atender aos seguintes requisitos:

- NumPy 1.22.4 ou posterior instalado no dispositivo. O Raspberry Pi OS Bullseye inclui uma versão anterior do NumPy, então você pode executar o seguinte comando para atualizar NumPy o dispositivo.

  ```
  pip3 install --upgrade numpy
  ```

- A pilha de câmeras antiga ativada no dispositivo. O Raspberry Pi OS Bullseye inclui uma nova pilha de câmeras que é ativada por padrão e não é compatível, portanto, você deve habilitar a pilha de câmeras antiga.

  Para habilitar a pilha de câmeras antiga

  1. Execute o comando a seguir para abrir a ferramenta de configuração do Raspberry Pi.

     ```
     sudo raspi-config
     ```

  2. Selecione Opções de interface.
  3. Selecione Câmera antiga para ativar a pilha de câmeras antigas.
  4. Reinicie o Raspberry Pi.

Dependências

Quando você implanta um componente, AWS IoT Greengrass também implanta versões compatíveis de suas dependências. Isso significa que você deve atender aos requisitos do componente e de todas as suas dependências para implantá-lo com êxito. Esta seção lista as dependências das versões lançadas desse componente e as restrições de versão semântica que definem as versões do componente para cada dependência. Você também pode visualizar as dependências de cada versão do componente no AWS IoT Greengrass console. Na página de detalhes do componente, procure a lista de dependências.

2.1.11 and 2.1.12

A tabela a seguir lista as dependências das versões 2.1.11 e 2.1.12 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.13.0	Flexível

2.1.10

A tabela a seguir lista as dependências da versão 2.1.10 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.12.0	Flexível

2.1.9

A tabela a seguir lista as dependências da versão 2.1.9 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.11.0	Flexível

2.1.8

A tabela a seguir lista as dependências da versão 2.1.8 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.10.0	Flexível

2.1.7

A tabela a seguir lista as dependências da versão 2.1.7 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.9.0	Flexível

2.1.6

A tabela a seguir lista as dependências da versão 2.1.6 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.8.0	Flexível

2.1.5

A tabela a seguir lista as dependências da versão 2.1.5 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.7.0	Flexível

2.1.4

A tabela a seguir lista as dependências da versão 2.1.4 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.6.0	Flexível

2.1.3

A tabela a seguir lista as dependências da versão 2.1.3 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.5.0	Flexível

2.1.2

A tabela a seguir lista as dependências da versão 2.1.2 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.4.0	Flexível

2.1.1

A tabela a seguir lista as dependências da versão 2.1.1 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.3.0	Flexível

2.1.0

A tabela a seguir lista as dependências da versão 2.1.0 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.2.0	Flexível

Configuração

Esse componente não tem nenhum parâmetro de configuração.

Arquivo de log local

Esse componente não gera registros.

Changelog

A tabela a seguir descreve as alterações em cada versão do componente.

Version (Versão)	Alterações
2.1.12	Versão atualizada para a versão 2.12.5 do Greengrass nucleus.
2.1.11	Versão atualizada para a versão 2.12.0 do Greengrass nucleus.
2.1.10	Versão atualizada para a versão 2.11.0 do Greengrass nucleus.
2.1.9	Versão atualizada para a versão 2.10.0 do Greengrass nucleus.
2.1.8	Versão atualizada para a versão 2.9.0 do Greengrass nucleus.

Version (Versão)	Alterações
2.1.7	Versão atualizada para a versão 2.8.0 do Greengrass nucleus.
2.1.6	Versão atualizada para a versão 2.7.0 do Greengrass nucleus.
2.1.5	Versão atualizada para a versão 2.6.0 do Greengrass nucleus.
2.1.4	Versão atualizada para a versão 2.5.0 do Greengrass nucleus.
2.1.3	Versão atualizada para a versão 2.4.0 do Greengrass nucleus.
2.1.2	Versão atualizada para a versão 2.3.0 do Greengrass nucleus.
2.1.1	Versão atualizada para a versão 2.2.0 do Greengrass nucleus.
2.1.0	Versão inicial.

TensorFlow Tempo de execução leve

O componente de tempo de execução do TensorFlow Lite (`variant.TensorFlowLite`) contém um script que instala o TensorFlow Lite versão 2.5.0 e suas dependências em um ambiente virtual em seu dispositivo. A classificação de imagem TensorFlow TensorFlow Lite e o componente de detecção de objetos Lite usam esse componente de tempo de execução como uma dependência para instalar o TensorFlow Lite.

> **Note**
>
> TensorFlow O componente Lite Runtime v2.5.6 e versões posteriores reinstalam as instalações existentes do TensorFlow Lite Runtime e suas dependências. Essa reinstalação ajuda a garantir que o dispositivo principal execute versões compatíveis do TensorFlow Lite e de suas dependências.

Para usar um tempo de execução diferente, você pode usar a receita desse componente como modelo para criar um componente de aprendizado de máquina personalizado.

Tópicos

- Versões

Versões

Esse componente tem as seguintes versões:

- 2,5.x

Tipo

Este componente é um componente genérico (`aws.greengrass.generic`). O núcleo do Greengrass executa os scripts do ciclo de vida do componente.

Para ter mais informações, consulte Tipos de componentes.

Sistema operacional

Esse componente pode ser instalado em dispositivos principais que executam os seguintes sistemas operacionais:

- Linux
- Windows

Requisitos

Esse componente tem os seguintes requisitos:

- Nos principais dispositivos do Greengrass que executam o Amazon Linux 2 ou o Ubuntu 18.04, a GNU C Library (glibc) versão 2.27 ou posterior está instalada no dispositivo.

- Em dispositivos ARMv7L, como o Raspberry Pi, dependências do OpenCV-Python instaladas no dispositivo. Execute o comando a seguir para instalar as dependências.

  ```
  sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
   libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
  ```

- Os dispositivos Raspberry Pi que executam o Raspberry Pi OS Bullseye devem atender aos seguintes requisitos:
  - NumPy 1.22.4 ou posterior instalado no dispositivo. O Raspberry Pi OS Bullseye inclui uma versão anterior do NumPy, então você pode executar o seguinte comando para atualizar NumPy o dispositivo.

    ```
    pip3 install --upgrade numpy
    ```

  - A pilha de câmeras antiga ativada no dispositivo. O Raspberry Pi OS Bullseye inclui uma nova pilha de câmeras que é ativada por padrão e não é compatível, portanto, você deve habilitar a pilha de câmeras antiga.

    Para habilitar a pilha de câmeras antiga

    1. Execute o comando a seguir para abrir a ferramenta de configuração do Raspberry Pi.

       ```
       sudo raspi-config
       ```

    2. Selecione Opções de interface.
    3. Selecione Câmera antiga para ativar a pilha de câmeras antigas.
    4. Reinicie o Raspberry Pi.

Endpoints e portas

Por padrão, esse componente usa um script de instalação para instalar pacotes usando os `pip` comandos `apt` `yum`,`brew`,, e, dependendo da plataforma usada pelo dispositivo principal. Esse componente deve ser capaz de realizar solicitações de saída para vários índices e repositórios de pacotes para executar o script do instalador. Para permitir o tráfego de saída desse componente por meio de um proxy ou firewall, você deve identificar os endpoints dos índices e repositórios de pacotes aos quais seu dispositivo principal se conecta para instalar.

Considere o seguinte ao identificar os endpoints necessários para o script de instalação desse componente:

- Os endpoints dependem da plataforma do dispositivo principal. Por exemplo, um dispositivo principal que executa o Ubuntu usa `apt` em vez de `yum` ou `brew`. Além disso, dispositivos que usam o mesmo índice de pacotes podem ter listas de fontes diferentes e, portanto, podem recuperar pacotes de repositórios diferentes.
- Os endpoints podem ser diferentes entre vários dispositivos que usam o mesmo índice de pacotes, porque cada dispositivo tem suas próprias listas de origem que definem onde recuperar os pacotes.
- Os endpoints podem mudar com o tempo. Cada índice de pacotes fornece os URLs dos repositórios nos quais você baixa pacotes, e o proprietário de um pacote pode alterar os URLs fornecidos pelo índice de pacotes.

Para obter mais informações sobre as dependências que esse componente instala e como desabilitar o script do instalador, consulte o parâmetro de UseInstallerconfiguração.

Para obter mais informações sobre terminais e portas necessários para a operação básica, consultePermitir o tráfego de dispositivos por meio de um proxy ou firewall.

Dependências

Quando você implanta um componente, AWS IoT Greengrass também implanta versões compatíveis de suas dependências. Isso significa que você deve atender aos requisitos do componente e de todas as suas dependências para implantá-lo com êxito. Esta seção lista as dependências das versões lançadas desse componente e as restrições de versão semântica que definem as versões dos componentes para cada dependência. Você também pode visualizar as dependências de cada versão do componente no AWS IoT Greengrass console. Na página de detalhes do componente, procure a lista de dependências.

2.5.14 and 2.5.15

A tabela a seguir lista as dependências das versões 2.5.14 e 2.5.15 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.13.0	Flexível

2.5.13

A tabela a seguir lista as dependências da versão 2.5.13 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.12.0	Flexível

2.5.12

A tabela a seguir lista as dependências da versão 2.5.12 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.11.0	Flexível

2.5.11

A tabela a seguir lista as dependências da versão 2.5.11 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.10.0	Flexível

2.5.10

A tabela a seguir lista as dependências da versão 2.5.10 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.9.0	Flexível

2.5.9

A tabela a seguir lista as dependências da versão 2.5.9 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.8.0	Flexível

2.5.8

A tabela a seguir lista as dependências da versão 2.5.8 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.7.0	Flexível

2.5.5 - 2.5.7

A tabela a seguir lista as dependências das versões 2.5.5 a 2.5.7 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.6.0	Flexível

2.5.3 and 2.5.4

A tabela a seguir lista as dependências das versões 2.5.3 e 2.5.4 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.5.0	Flexível

2.5.2

A tabela a seguir lista as dependências da versão 2.5.2 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.4.0	Flexível

2.5.1

A tabela a seguir lista as dependências da versão 2.5.1 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.3.0	Flexível

2.5.0

A tabela a seguir lista as dependências da versão 2.5.0 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.2.0	Flexível

Para obter mais informações sobre dependências de componentes, consulte a referência da receita do componente.

Configuração

Esse componente fornece os seguintes parâmetros de configuração que você pode personalizar ao implantar o componente.

`MLRootPath`

(Opcional) O caminho da pasta nos dispositivos principais do Linux em que os componentes de inferência leem imagens e gravam resultados de inferência. Você pode modificar esse valor em qualquer local em seu dispositivo no qual o usuário que está executando esse componente tenha acesso de leitura/gravação.

Padrão: `/greengrass/v2/work/variant.TensorFlowLite/greengrass_ml`

`WindowsMLRootPath`

Esse recurso está disponível na versão 1.6.6 e posterior desse componente.

(Opcional) O caminho da pasta no dispositivo principal do Windows em que os componentes de inferência leem imagens e gravam resultados de inferência. Você pode modificar esse valor em qualquer local em seu dispositivo no qual o usuário que está executando esse componente tenha acesso de leitura/gravação.

Padrão: `C:\greengrass\v2\\work\\variant.DLR\\greengrass_ml`

`UseInstaller`

(Opcional) Valor da string que define se o script do instalador deve ser usado nesse componente para instalar o TensorFlow Lite e suas dependências. Os valores compatíveis são `true` e `false`.

Defina esse valor como `false` se você quiser usar um script personalizado para a instalação do TensorFlow Lite ou se quiser incluir dependências de tempo de execução em uma imagem Linux pré-criada. Para usar esse componente com os componentes AWS de inferência TensorFlow Lite fornecidos, instale as bibliotecas a seguir, incluindo quaisquer dependências, e disponibilize-as para o usuário do sistema, como, por exemplo`gc_user`, que executa os componentes de ML.

- Python 3.8 ou posterior, inclusive `pip` para sua versão do Python
- TensorFlow Lite v2.5.0
- NumPy
- OpenCV-Python
- AWS IoT Device SDK v2 para Python
- AWS Python de tempo de execução comum (CRT)
- Picamera (para dispositivos Raspberry Pi)
- `awscam`módulo (para AWS DeepLens dispositivos)
- libGL (para dispositivos Linux)

Padrão: `true`

Uso

Use esse componente com o parâmetro `UseInstaller` de configuração definido `true` para instalar o TensorFlow Lite e suas dependências em seu dispositivo. O componente configura um ambiente virtual em seu dispositivo que inclui o OpenCV NumPy e as bibliotecas necessárias para o Lite. TensorFlow

> **Note**
>
> O script do instalador neste componente também instala as versões mais recentes das bibliotecas adicionais do sistema que são necessárias para configurar o ambiente virtual em seu dispositivo e usar a estrutura de aprendizado de máquina instalada. Isso pode atualizar as bibliotecas do sistema existentes em seu dispositivo. Consulte a tabela a seguir para ver a lista de bibliotecas que esse componente instala para cada sistema operacional

compatível. Se você quiser personalizar esse processo de instalação, defina o parâmetro de `UseInstaller` configuração como `false` e desenvolva seu próprio script de instalação.

Plataforma	Bibliotecas instaladas no sistema do dispositivo	Bibliotecas instaladas no ambiente virtual
Armv7l	`build-essential` , `cmake`, `ca-certificates` , `git`	`setuptools` , `wheel`
Amazon Linux 2	`mesa-libGL`	Nenhum
Ubuntu	`wget`	Nenhum

Quando você implanta seu componente de inferência, esse componente de tempo de execução primeiro verifica se seu dispositivo já tem o TensorFlow Lite e suas dependências instaladas. Caso contrário, o componente de tempo de execução os instala para você.

Arquivo de log local

Esse componente usa o seguinte arquivo de log.

Linux

```
/greengrass/v2/logs/variant.TensorFlowLite.log
```

Windows

```
C:\greengrass\v2\logs\variant.TensorFlowLite.log
```

Para visualizar os registros desse componente

- Execute o comando a seguir no dispositivo principal para visualizar o arquivo de log desse componente em tempo real. Substitua */greengrass/v2* ou *C:\greengrass\v2* pelo caminho para a pasta AWS IoT Greengrass raiz.

Linux

```
sudo tail -f /greengrass/v2/logs/variant.TensorFlowLite.log
```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\variant.TensorFlowLite.log -Tail 10 -Wait
```

Changelog

A tabela a seguir descreve as alterações em cada versão do componente.

Version (Versão)	Alterações
2.5.15	Versão atualizada para a versão 2.12.5 do Greengrass nucleus.
2.5.14	Versão atualizada para a versão 2.12.0 do Greengrass nucleus.
2.5.13	Versão atualizada para a versão 2.11.0 do Greengrass nucleus.
2.5.12	Versão atualizada para a versão 2.10.0 do Greengrass nucleus.
2.5.11	Versão atualizada para a versão 2.9.0 do Greengrass nucleus.
2.5.10	Versão atualizada para a versão 2.8.0 do Greengrass nucleus.
2.5.9	Versão atualizada para a versão 2.7.0 do Greengrass nucleus.
2.5.8	Versão atualizada para a versão 2.6.0 do Greengrass nucleus.
2.5.7	Correções de erros e melhorias • Atualiza o script de `UseInstaller` instalação para instalar o LibGL, que não está disponível por padrão em determinadas plataformas Linux. • Atualiza o script de `UseInstaller` instalação para sempre usar o Python 3.9 no ambiente virtual desse componente. Essa alteração ajuda a garantir a compatibilidade com outras bibliotecas.

Version (Versão)	Alterações
2.5.6	Correções de erros e melhorias • Atualiza esse componente para instalar o patch mais recente do TensorFlow Lite 2.5.0 (`tflite-runtime-2.5.0.post1`), para que você possa usar esse componente com o Python 3.9. Se esse componente não conseguir instalar esse patch, ele será instalado `tflite-runtime-2.5.0` em vez disso. • Atualiza esse componente para reinstalar as instalações existentes do TensorFlow Lite e suas dependências. Essa alteração ajuda a garantir que o dispositivo principal execute versões compatíveis do TensorFlow Lite e de suas dependências.
2.5.5	Novos atributos • Adiciona suporte para dispositivos principais que executam o Windows. • Adiciona o novo parâmetro de `WindowsMLRootPath` configuração que você pode usar para configurar a pasta de resultados de inferência nos dispositivos principais do Windows.
2.5.4	Novos atributos • Adiciona o novo parâmetro de `UseInstaller` configuração que permite desativar o script de instalação nesse componente.
2.5.3	Versão atualizada para a versão 2.4.0 do Greengrass nucleus.
2.5.2	Versão atualizada para a versão 2.3.0 do Greengrass nucleus.
2.5.1	Versão atualizada para a versão 2.2.0 do Greengrass nucleus.
2.5.0	Versão inicial.

Adaptador de protocolo Modbus-RTU

O componente do adaptador de protocolo Modbus-RTU (`aws.greengrass.Modbus`) pesquisa informações de dispositivos Modbus RTU locais.

Para solicitar informações de um dispositivo Modbus RTU local com esse componente, publique uma mensagem no tópico em que esse componente se inscreve. Na mensagem, especifique a solicitação Modbus RTU a ser enviada para um dispositivo. Em seguida, esse componente publica uma resposta que contém o resultado da solicitação do Modbus RTU.

> **Note**
>
> Esse componente fornece funcionalidade semelhante ao conector do adaptador de protocolo Modbus RTU na AWS IoT Greengrass V1. Para obter mais informações, consulte Conector do adaptador de protocolo Modbus RTU no Guia do desenvolvedor AWS IoT Greengrass V1.

Tópicos

Versões

Esse componente tem as seguintes versões:

- 2.1.x
- 2.0.x

Tipo

Esse componente é um componente Lambda () `aws.greengrass.lambda`. O núcleo do Greengrass executa a função Lambda desse componente usando o componente lançador Lambda.

Para ter mais informações, consulte Tipos de componentes.

Sistema operacional

Esse componente pode ser instalado somente nos dispositivos principais do Linux.

Requisitos

Esse componente tem os seguintes requisitos:

- Seu dispositivo principal deve atender aos requisitos para executar as funções do Lambda. Se você quiser que o dispositivo principal execute funções Lambda em contêineres, o dispositivo deve atender aos requisitos para fazer isso. Para ter mais informações, consulte Requisitos da função do Lambda.
- Python versão 3.7 instalado no dispositivo de núcleo e adicionado à variável de ambiente PATH.
- Uma conexão física entre o dispositivo AWS IoT Greengrass principal e os dispositivos Modbus. O dispositivo principal deve estar fisicamente conectado à rede Modbus RTU por meio de uma porta serial, como uma porta USB.
- Para receber dados de saída desse componente, você deve mesclar a seguinte atualização de configuração para o componente antigo do roteador de assinatura (`aws.greengrass.LegacySubscriptionRouter`) ao implantar esse componente. Essa configuração especifica o tópico em que esse componente publica respostas.

  Legacy subscription router v2.1.x

  ```
  {
    "subscriptions": {
      "aws-greengrass-modbus": {
        "id": "aws-greengrass-modbus",
        "source": "component:aws.greengrass.Modbus",
        "subject": "modbus/adapter/response",
        "target": "cloud"
      }
    }
  }
  ```

Legacy subscription router v2.0.x

```
{
  "subscriptions": {
    "aws-greengrass-modbus": {
      "id": "aws-greengrass-modbus",
      "source": "arn:aws:lambda:region:aws:function:aws-greengrass-
modbus:version",
      "subject": "modbus/adapter/response",
      "target": "cloud"
    }
  }
}
```

- Substitua a *região* pela Região da AWS que você usa.
- *Substitua a versão pela versão da função Lambda que esse componente executa.* Para encontrar a versão da função Lambda, você deve visualizar a receita da versão desse componente que você deseja implantar. Abra a página de detalhes desse componente no AWS IoT Greengrass console e procure o par de valores-chave da função Lambda. Esse par de valores-chave contém o nome e a versão da função Lambda.

> **Important**
>
> Você deve atualizar a versão da função Lambda no roteador de assinatura legado sempre que implantar esse componente. Isso garante que você use a versão correta da função Lambda para a versão do componente que você implanta.

Para ter mais informações, consulte Criar implantações.

- O adaptador de protocolo Modbus-RTU é suportado para ser executado em uma VPC.

Dependências

Quando você implanta um componente, AWS IoT Greengrass também implanta versões compatíveis de suas dependências. Isso significa que você deve atender aos requisitos do componente e de todas as suas dependências para implantá-lo com êxito. Esta seção lista as dependências das versões lançadas desse componente e as restrições de versão semântica que definem as versões dos componentes para cada dependência. Você também pode visualizar as dependências de cada

versão do componente no AWS IoT Greengrass console. Na página de detalhes do componente, procure a lista de dependências.

2.1.8

A tabela a seguir lista as dependências da versão 2.1.8 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.13.0	Rígido
Lançador Lambda	^2.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	^2.0.0	Flexível
Serviço de troca de tokens	^2.0.0	Rígido

2.1.7

A tabela a seguir lista as dependências da versão 2.1.7 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.12.0	Rígido
Lançador Lambda	^2.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	^2.0.0	Flexível
Serviço de troca de tokens	^2.0.0	Rígido

2.1.6

A tabela a seguir lista as dependências da versão 2.1.6 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.11.0	Rígido
Lançador Lambda	^2.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	^2.0.0	Flexível
Serviço de troca de tokens	^2.0.0	Rígido

2.1.4 and 2.1.5

A tabela a seguir lista as dependências das versões 2.1.4 e 2.1.5 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.10.0	Rígido
Lançador Lambda	^2.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	^2.0.0	Flexível
Serviço de troca de tokens	^2.0.0	Rígido

2.1.3

A tabela a seguir lista as dependências da versão 2.1.3 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.9.0	Rígido
Lançador Lambda	^2.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	^2.0.0	Flexível

Dependência	Versões compatíveis	Tipo de dependência
Serviço de troca de tokens	^2.0.0	Rígido

2.1.2

A tabela a seguir lista as dependências da versão 2.1.2 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.8.0	Rígido
Lançador Lambda	^2.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	^2.0.0	Flexível
Serviço de troca de tokens	^2.0.0	Rígido

2.1.1

A tabela a seguir lista as dependências da versão 2.1.1 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.7.0	Rígido
Lançador Lambda	^2.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	^2.0.0	Flexível
Serviço de troca de tokens	^2.0.0	Rígido

2.0.8 and 2.1.0

A tabela a seguir lista as dependências das versões 2.0.8 e 2.1.0 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.6.0	Rígido
Lançador Lambda	^2.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	^2.0.0	Flexível
Serviço de troca de tokens	^2.0.0	Rígido

2.0.7

A tabela a seguir lista as dependências da versão 2.0.7 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.5.0	Rígido
Lançador Lambda	^2.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	^2.0.0	Flexível
Serviço de troca de tokens	^2.0.0	Rígido

2.0.6

A tabela a seguir lista as dependências da versão 2.0.6 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.4.0	Rígido
Lançador Lambda	^2.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	^2.0.0	Flexível

Dependência	Versões compatíveis	Tipo de dependência
Serviço de troca de tokens	^2.0.0	Rígido

2.0.5

A tabela a seguir lista as dependências da versão 2.0.5 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.3.0	Rígido
Lançador Lambda	^2.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	^2.0.0	Flexível
Serviço de troca de tokens	^2.0.0	Rígido

2.0.4

A tabela a seguir lista as dependências da versão 2.0.4 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.2.0	Rígido
Lançador Lambda	^2.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	^2.0.0	Flexível
Serviço de troca de tokens	^2.0.0	Rígido

2.0.3

A tabela a seguir lista as dependências da versão 2.0.3 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.3 <2.1.0	Rígido
Lançador Lambda	>=1.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	>=1.0.0	Flexível
Serviço de troca de tokens	>=1.0.0	Rígido

Para obter mais informações sobre dependências de componentes, consulte a referência da receita do componente.

Configuração

Esse componente fornece os seguintes parâmetros de configuração que você pode personalizar ao implantar o componente.

> **Note**
>
> A configuração padrão desse componente inclui parâmetros da função Lambda. Recomendamos que você edite somente os parâmetros a seguir para configurar esse componente em seus dispositivos.

v2.1.x

`lambdaParams`

Um objeto que contém os parâmetros da função Lambda desse componente. Esse objeto contém as seguintes informações:

`EnvironmentVariables`

Um objeto que contém os parâmetros da função Lambda. Esse objeto contém as seguintes informações:

`ModbusLocalPort`

O caminho absoluto para a porta serial física do Modbus no dispositivo principal, como `/dev/ttyS2`.

Para executar esse componente em um contêiner, você deve definir esse caminho como um dispositivo do sistema (em `containerParams.devices`) que o componente possa acessar. Esse componente é executado em um contêiner por padrão.

> **Note**
>
> Esse componente deve ter acesso de leitura/gravação ao dispositivo.

`ModbusBaudRate`

(Opcional) Um valor de string que especifica a taxa de transmissão para comunicação serial com dispositivos Modbus TCP locais.

Padrão: `9600`

`ModbusByteSize`

(Opcional) Um valor de string que especifica o tamanho de um byte na comunicação serial com dispositivos Modbus TCP locais. Escolha `5`,`6`,`7`, ou `8` bits.

Padrão: `8`

`ModbusParity`

(Opcional) O modo de paridade a ser usado para verificar a integridade dos dados na comunicação serial com dispositivos Modbus TCP locais.

- `E`— Verifique a integridade dos dados com paridade uniforme.
- `0`— Verifique a integridade dos dados com paridade ímpar.
- `N`— Não verifique a integridade dos dados.

Padrão: `N`

`ModbusStopBits`

(Opcional) Um valor de string que especifica o número de bits que indicam o fim de um byte na comunicação serial com dispositivos Modbus TCP locais.

Padrão: 1

`containerMode`

(Opcional) O modo de conteinerização desse componente. Escolha uma das seguintes opções:

- `GreengrassContainer`— O componente é executado em um ambiente de execução isolado dentro do AWS IoT Greengrass contêiner.

  Se você especificar essa opção, deverá especificar um dispositivo do sistema (in`containerParams.devices`) para dar ao contêiner acesso ao dispositivo Modbus.
- `NoContainer`— O componente não é executado em um ambiente de execução isolado.

Padrão: `GreengrassContainer`

`containerParams`

(Opcional) Um objeto que contém os parâmetros do contêiner desse componente. O componente usa esses parâmetros se você especificar `GreengrassContainer` para`containerMode`.

Esse objeto contém as seguintes informações:

`memorySize`

(Opcional) A quantidade de memória (em kilobytes) a ser alocada para o componente.

O padrão é 512 MB (525.312 KB).

`devices`

(Opcional) Um objeto que especifica os dispositivos do sistema que o componente pode acessar em um contêiner.

> **Important**
>
> Para executar esse componente em um contêiner, você deve especificar o dispositivo do sistema que você configura na variável de `ModbusLocalPort` ambiente.

Esse objeto contém as seguintes informações:

`0`— Este é um índice de matriz como uma string.

Um objeto que contém as seguintes informações:

`path`

O caminho para o dispositivo do sistema no dispositivo principal. Isso deve ter o mesmo valor que o valor que você configurou`ModbusLocalPort`.

`permission`

(Opcional) A permissão para acessar o dispositivo do sistema a partir do contêiner. Esse valor deve ser`rw`, o que especifica que o componente tem acesso de leitura/gravação ao dispositivo do sistema.

Padrão: `rw`

`addGroupOwner`

(Opcional) Adicionar ou não o grupo do sistema que executa o componente como proprietário do dispositivo do sistema.

Padrão: `true`

`pubsubTopics`

(Opcional) Um objeto que contém os tópicos em que o componente se inscreve para receber mensagens. Você pode especificar cada tópico e se o componente se inscreve nos tópicos do MQTT AWS IoT Core ou nos tópicos locais de publicação/assinatura.

Esse objeto contém as seguintes informações:

`0`— Este é um índice de matriz como uma string.

Um objeto que contém as seguintes informações:

`type`

(Opcional) O tipo de mensagem de publicação/assinatura que esse componente usa para assinar mensagens. Escolha uma das seguintes opções:

- `PUB_SUB` – Assine mensagens locais de publicar/assinar. Se você escolher essa opção, o tópico não poderá conter curingas MQTT. Para obter mais informações sobre como enviar mensagens do componente personalizado ao especificar essa opção, consulte[Publique/assine mensagens locais](#).

- `IOT_CORE`— Assine as mensagens do AWS IoT Core MQTT. Se você escolher essa opção, o tópico poderá conter curingas do MQTT. Para obter mais informações sobre como enviar mensagens de componentes personalizados ao especificar essa opção, consulte[Publique/assine mensagens MQTT AWS IoT Core](#).

Padrão: `PUB_SUB`

`topic`

(Opcional) O tópico no qual o componente se inscreve para receber mensagens. Se você especificar `IotCore` para`type`, poderá usar curingas MQTT (+e#) neste tópico.

Example Exemplo: atualização da mesclagem de configuração (modo contêiner)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "ModbusLocalPort": "/dev/ttyS2"
    }
  },
  "containerMode": "GreengrassContainer",
  "containerParams": {
    "devices": {
      "0": {
        "path": "/dev/ttyS2",
        "permission": "rw",
        "addGroupOwner": true
      }
    }
  }
}
```

Example Exemplo: atualização de mesclagem de configuração (sem modo de contêiner)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "ModbusLocalPort": "/dev/ttyS2"
    }
  },
  "containerMode": "NoContainer"
}
```

v2.0.x

`lambdaParams`

Um objeto que contém os parâmetros da função Lambda desse componente. Esse objeto contém as seguintes informações:

`EnvironmentVariables`

Um objeto que contém os parâmetros da função Lambda. Esse objeto contém as seguintes informações:

`ModbusLocalPort`

O caminho absoluto para a porta serial física do Modbus no dispositivo principal, como `/dev/ttyS2`.

Para executar esse componente em um contêiner, você deve definir esse caminho como um dispositivo do sistema (em `containerParams.devices`) que o componente possa acessar. Esse componente é executado em um contêiner por padrão.

> **Note**
>
> Esse componente deve ter acesso de leitura/gravação ao dispositivo.

`containerMode`

(Opcional) O modo de conteinerização desse componente. Escolha uma das seguintes opções:

- `GreengrassContainer` — O componente é executado em um ambiente de execução isolado dentro do AWS IoT Greengrass contêiner.

  Se você especificar essa opção, deverá especificar um dispositivo do sistema (in `containerParams.devices`) para dar ao contêiner acesso ao dispositivo Modbus.
- `NoContainer` — O componente não é executado em um ambiente de execução isolado.

Padrão: `GreengrassContainer`

`containerParams`

(Opcional) Um objeto que contém os parâmetros do contêiner desse componente. O componente usa esses parâmetros se você especificar `GreengrassContainer` para`containerMode`.

Esse objeto contém as seguintes informações:

`memorySize`

(Opcional) A quantidade de memória (em kilobytes) a ser alocada para o componente.

O padrão é 512 MB (525.312 KB).

`devices`

(Opcional) Um objeto que especifica os dispositivos do sistema que o componente pode acessar em um contêiner.

> **Important**
>
> Para executar esse componente em um contêiner, você deve especificar o dispositivo do sistema que você configura na variável de `ModbusLocalPort` ambiente.

Esse objeto contém as seguintes informações:

`0`— Este é um índice de matriz como uma string.

Um objeto que contém as seguintes informações:

`path`

O caminho para o dispositivo do sistema no dispositivo principal. Isso deve ter o mesmo valor que o valor que você configurou`ModbusLocalPort`.

`permission`

(Opcional) A permissão para acessar o dispositivo do sistema a partir do contêiner. Esse valor deve ser`rw`, o que especifica que o componente tem acesso de leitura/gravação ao dispositivo do sistema.

Padrão: `rw`

`addGroupOwner`

(Opcional) Adicionar ou não o grupo do sistema que executa o componente como proprietário do dispositivo do sistema.

Padrão: `true`

`pubsubTopics`

(Opcional) Um objeto que contém os tópicos em que o componente se inscreve para receber mensagens. Você pode especificar cada tópico e se o componente se inscreve nos tópicos do MQTT AWS IoT Core ou nos tópicos locais de publicação/assinatura.

Esse objeto contém as seguintes informações:

`0`— Este é um índice de matriz como uma string.

Um objeto que contém as seguintes informações:

`type`

(Opcional) O tipo de mensagem de publicação/assinatura que esse componente usa para assinar mensagens. Escolha uma das seguintes opções:

- `PUB_SUB` – Assine mensagens locais de publicar/assinar. Se você escolher essa opção, o tópico não poderá conter curingas MQTT. Para obter mais informações sobre como enviar mensagens do componente personalizado ao especificar essa opção, consultePublique/assine mensagens locais.
- `IOT_CORE`— Assine as mensagens do AWS IoT Core MQTT. Se você escolher essa opção, o tópico poderá conter curingas do MQTT. Para obter mais informações sobre como enviar mensagens de componentes personalizados ao especificar essa opção, consultePublique/assine mensagens MQTT AWS IoT Core.

Padrão: `PUB_SUB`

`topic`

(Opcional) O tópico no qual o componente se inscreve para receber mensagens. Se você especificar `IotCore` para`type`, poderá usar curingas MQTT (`+`e`#`) neste tópico.

Example Exemplo: atualização da mesclagem de configuração (modo contêiner)

```
{
```

```
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "ModbusLocalPort": "/dev/ttyS2"
    }
  },
  "containerMode": "GreengrassContainer",
  "containerParams": {
    "devices": {
      "0": {
        "path": "/dev/ttyS2",
        "permission": "rw",
        "addGroupOwner": true
      }
    }
  }
}
```

Example Exemplo: atualização de mesclagem de configuração (sem modo de contêiner)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "ModbusLocalPort": "/dev/ttyS2"
    }
  },
  "containerMode": "NoContainer"
}
```

Dados de entrada

Esse componente aceita os parâmetros de solicitação Modbus RTU no tópico a seguir e envia a solicitação Modbus RTU para o dispositivo. Por padrão, esse componente assina mensagens locais de publicação/assinatura. Para obter mais informações sobre como publicar mensagens nesse componente a partir de seus componentes personalizados, consulte[Publique/assine mensagens locais](#).

Tópico padrão (publicação/assinatura local): `modbus/adapter/request`

A mensagem aceita as seguintes propriedades. As mensagens de entrada devem estar no formato JSON.

`request`

Os parâmetros para o envio da solicitação Modbus RTU.

O formato da mensagem de solicitação depende do tipo de solicitação Modbus RTU que ela representa. As propriedades a seguir são obrigatórias para todas as solicitações.

Tipo: `object` que contém as seguintes informações:

`operation`

O nome da operação a ser executada. Por exemplo, especifique `ReadCoilsRequest` a leitura de bobinas em um dispositivo Modbus RTU. Para obter mais informações sobre as operações suportadas, consulteRespostas e solicitações Modbus RTU.

Tipo: `string`

`device`

O dispositivo de destino da solicitação.

Esse valor deve ser um número inteiro entre `0` e. `247`

Tipo: `integer`

Os outros parâmetros a ser incluídos na solicitação dependem da operação. Esse componente manipula a verificação de redundância cíclica (CRC) para verificar as solicitações de dados para você.

> **Note**
>
> Se sua solicitação incluir uma `address` propriedade, você deverá especificar seu valor como um número inteiro. Por exemplo, `"address": 1`.

`id`

Um ID arbitrário para a solicitação. Use essa propriedade para mapear uma solicitação de entrada para uma resposta de saída. Quando você especifica essa propriedade, o componente define a `id` propriedade no objeto de resposta com esse valor.

Tipo: `string`

Example Exemplo de entrada: solicitação de bobinas de leitura

```
{
  "request": {
    "operation": "ReadCoilsRequest",
    "device": 1,
    "address": 1,
    "count": 1
  },
  "id": "MyRequest"
}
```

Dados de saída

Por padrão, esse componente publica respostas como dados de saída no seguinte tópico do MQTT. Você deve especificar esse tópico conforme a `subject` configuração do componente antigo do roteador de assinatura. Para obter mais informações sobre como assinar mensagens sobre esse tópico em seus componentes personalizados, consultePublique/assine mensagens MQTT AWS IoT Core.

Tópico padrão (AWS IoT Core MQTT): `modbus/adapter/response`

O formato da mensagem de resposta depende da operação da solicitação e do status da resposta. Para ver exemplos, consulte Exemplos de solicitações e respostas.

Cada resposta inclui as seguintes propriedades:

`response`

A resposta do dispositivo Modbus RTU.

Tipo: `object` que contém as seguintes informações:

`status`

O status da solicitação. O status pode ser um dos valores a seguir:

- `Success`— A solicitação era válida, o componente enviou a solicitação para a rede Modbus RTU e a rede Modbus RTU retornou uma resposta.
- `Exception`— A solicitação era válida, o componente enviou a solicitação para a rede Modbus RTU e a rede Modbus RTU retornou uma exceção. Para ter mais informações, consulte Status da resposta: Exceção.

- No `Response`— A solicitação era inválida e o componente detectou o erro antes de enviar a solicitação para a rede Modbus RTU. Para ter mais informações, consulte Status de resposta: Sem resposta.

`operation`

A operação solicitada pelo componente.

`device`

O dispositivo para o qual o componente enviou a solicitação.

`payload`

A resposta do dispositivo Modbus RTU. Se `status` for`No Response`, esse objeto contém somente uma `error` propriedade com a descrição do erro (por exemplo,`[Input/Output] No Response received from the remote unit`).

`id`

O ID da solicitação, que você pode usar para identificar qual resposta corresponde a qual solicitação.

> **Note**
>
> Uma resposta para uma operação de gravação é simplesmente um eco da solicitação. Embora as respostas escritas não incluam informações significativas, é uma boa prática verificar o status da resposta para ver se a solicitação foi bem-sucedida ou não.

Example Exemplo de resultado: sucesso

```
{
  "response" : {
    "status" : "success",
    "device": 1,
    "operation": "ReadCoilsRequest",
    "payload": {
      "function_code": 1,
      "bits": [1]
    }
  },
  "id" : "MyRequest"
```

```
}
```

Example Exemplo de resultado: falha

```
{
  "response" : {
    "status" : "fail",
    "error_message": "Internal Error",
    "error": "Exception",
    "device": 1,
    "operation": "ReadCoilsRequest",
    "payload": {
      "function_code": 129,
      "exception_code": 2
    }
  },
  "id" : "MyRequest"
}
```

Para obter mais exemplos, consulte Exemplos de solicitações e respostas.

Respostas e solicitações Modbus RTU

Esse conector aceita parâmetros de solicitação Modbus RTU como dados de entrada e publica respostas como dados de saída.

As operações comuns a seguir têm suporte.

Nome da operação na solicitação	Código da função em resposta
ReadCoilsRequest	01
ReadDiscreteInputsRequest	02
ReadHoldingRegistersRequest	03
ReadInputRegistersRequest	04
WriteSingleCoilRequest	05
WriteSingleRegisterRequest	06

Nome da operação na solicitação	Código da função em resposta
WriteMultipleCoilsRequest	15
WriteMultipleRegistersRequest	16
MaskWriteRegisterRequest	22
ReadWriteMultipleRegistersRequest	23

Exemplos de solicitações e respostas

Veja a seguir exemplos de solicitações e respostas para operações com suporte.

Bobinas de leitura

Exemplo de solicitação:

```
{
  "request": {
    "operation": "ReadCoilsRequest",
    "device": 1,
    "address": 1,
    "count": 1
  },
  "id": "TestRequest"
}
```

Exemplo de resposta:

```
{
  "response": {
    "status": "success",
    "device": 1,
    "operation": "ReadCoilsRequest",
    "payload": {
      "function_code": 1,
      "bits": [1]
    }
  },
  "id" : "TestRequest"
```

```
}
```

Leia entradas discretas

Exemplo de solicitação:

```
{
  "request": {
    "operation": "ReadDiscreteInputsRequest",
    "device": 1,
    "address": 1,
    "count": 1
  },
  "id": "TestRequest"
}
```

Exemplo de resposta:

```
{
  "response": {
    "status": "success",
    "device": 1,
    "operation": "ReadDiscreteInputsRequest",
    "payload": {
      "function_code": 2,
      "bits": [1]
    }
  },
  "id" : "TestRequest"
}
```

Leia os registros de retenção

Exemplo de solicitação:

```
{
  "request": {
    "operation": "ReadHoldingRegistersRequest",
    "device": 1,
    "address": 1,
    "count": 1
  },
  "id": "TestRequest"
```

```
}
```

Exemplo de resposta:

```
{
  "response": {
    "status": "success",
    "device": 1,
    "operation": "ReadHoldingRegistersRequest",
    "payload": {
      "function_code": 3,
      "registers": [20,30]
    }
  },
  "id" : "TestRequest"
}
```

Leia os registros de entrada

Exemplo de solicitação:

```
{
  "request": {
    "operation": "ReadInputRegistersRequest",
    "device": 1,
    "address": 1,
    "count": 1
  },
  "id": "TestRequest"
}
```

Escreva uma única bobina

Exemplo de solicitação:

```
{
  "request": {
    "operation": "WriteSingleCoilRequest",
    "device": 1,
    "address": 1,
    "value": 1
  },
  "id": "TestRequest"
```

```
}
```

Exemplo de resposta:

```
{
  "response": {
    "status": "success",
    "device": 1,
    "operation": "WriteSingleCoilRequest",
    "payload": {
      "function_code": 5,
      "address": 1,
      "value": true
    }
  },
  "id" : "TestRequest"
}
```

Escreva um único registro

Exemplo de solicitação:

```
{
  "request": {
    "operation": "WriteSingleRegisterRequest",
    "device": 1,
    "address": 1,
    "value": 1
  },
  "id": "TestRequest"
}
```

Escreva várias bobinas

Exemplo de solicitação:

```
{
  "request": {
    "operation": "WriteMultipleCoilsRequest",
    "device": 1,
    "address": 1,
    "values": [1,0,0,1]
  },
```

```
  "id": "TestRequest"
}
```

Exemplo de resposta:

```
{
  "response": {
    "status": "success",
    "device": 1,
    "operation": "WriteMultipleCoilsRequest",
    "payload": {
      "function_code": 15,
      "address": 1,
      "count": 4
    }
  },
  "id" : "TestRequest"
}
```

Escreva vários registros

Exemplo de solicitação:

```
{
  "request": {
    "operation": "WriteMultipleRegistersRequest",
    "device": 1,
    "address": 1,
    "values": [20,30,10]
  },
  "id": "TestRequest"
}
```

Exemplo de resposta:

```
{
  "response": {
    "status": "success",
    "device": 1,
    "operation": "WriteMultipleRegistersRequest",
    "payload": {
      "function_code": 23,
```

```
      "address": 1,
      "count": 3
    }
  },
  "id" : "TestRequest"
}
```

Registro de gravação de máscaras

Exemplo de solicitação:

```
{
  "request": {
    "operation": "MaskWriteRegisterRequest",
    "device": 1,
    "address": 1,
    "and_mask": 175,
    "or_mask": 1
  },
  "id": "TestRequest"
}
```

Exemplo de resposta:

```
{
  "response": {
    "status": "success",
    "device": 1,
    "operation": "MaskWriteRegisterRequest",
    "payload": {
      "function_code": 22,
      "and_mask": 0,
      "or_mask": 8
    }
  },
  "id" : "TestRequest"
}
```

Ler, gravar vários registros

Exemplo de solicitação:

```
{
```

```
  "request": {
    "operation": "ReadWriteMultipleRegistersRequest",
    "device": 1,
    "read_address": 1,
    "read_count": 2,
    "write_address": 3,
    "write_registers": [20,30,40]
  },
  "id": "TestRequest"
}
```

Exemplo de resposta:

```
{
  "response": {
    "status": "success",
    "device": 1,
    "operation": "ReadWriteMultipleRegistersRequest",
    "payload": {
      "function_code": 23,
      "registers": [10,20,10,20]
    }
  },
  "id" : "TestRequest"
}
```

> **Note**
>
> A resposta inclui os registros que o componente lê.

Status da resposta: Exceção

As exceções pode ocorrer quando o formato da solicitação é válido, mas a solicitação não é concluída com êxito. Nesse caso, a resposta contém as seguintes informações:

- A `status` é definida como `Exception`.
- O código da função `function_code` é igual ao código da função da solicitação + 128.
- O `exception_code` contém o código da exceção. Para obter mais informações, consulte os códigos de exceção Modbus.

Exemplo:

```
{
  "response": {
    "status": "fail",
    "error_message": "Internal Error",
    "error": "Exception",
    "device": 1,
    "operation": "ReadCoilsRequest",
    "payload": {
      "function_code": 129,
      "exception_code": 2
    }
  },
  "id": "TestRequest"
}
```

Status de resposta: Sem resposta

Esse conector executa verificações de validação na solicitação Modbus. Por exemplo, ele verifica se há formatos inválidos e campos ausentes. Se a validação falhar, o conector não enviará a solicitação. Em vez disso, ele retornará uma resposta com as seguintes informações:

- A `status` é definida como `No Response`.
- O `error` contém o motivo do erro.
- O `error_message` contém a mensagem do erro.

Exemplos:

```
{
  "response": {
    "status": "fail",
    "error_message": "Invalid address field. Expected <type 'int'>, got <type 'str'>",
    "error": "No Response",
    "device": 1,
    "operation": "ReadCoilsRequest",
    "payload": {
      "error": "Invalid address field. Expected Expected <type 'int'>, got <type
 'str'>"
    }
  },
```

```
  "id": "TestRequest"
}
```

Se a solicitação tem como destino um dispositivo inexistente, ou se a rede Modbus RTU não está funcionando, você pode obter um `ModbusIOException`, que usa o formato Sem resposta.

```
{
  "response": {
    "status": "fail",
    "error_message": "[Input/Output] No Response received from the remote unit",
    "error": "No Response",
    "device": 1,
    "operation": "ReadCoilsRequest",
    "payload": {
      "error": "[Input/Output] No Response received from the remote unit"
    }
  },
  "id": "TestRequest"
}
```

Arquivo de log local

Esse componente usa o seguinte arquivo de log.

```
/greengrass/v2/logs/aws.greengrass.Modbus.log
```

Para ver os registros desse componente

- Execute o comando a seguir no dispositivo principal para visualizar o arquivo de log desse componente em tempo real. */greengrass/v2*Substitua pelo caminho para a pasta AWS IoT Greengrass raiz.

  ```
  sudo tail -f /greengrass/v2/logs/aws.greengrass.Modbus.log
  ```

Licenças

Esse componente inclui o seguinte software/licenciamento de terceiros:

- Licença pymodbus /BSD
- Licença pyserial/BSD

Esse componente é lançado de acordo com o Contrato de Licença de Software Principal do Greengrass.

Changelog

A tabela a seguir descreve as alterações em cada versão do componente.

Version (Versão)	Alterações
2.1.8	Versão atualizada para a versão 2.12.0 do Greengrass nucleus.
2.1.7	Versão atualizada para a versão 2.11.0 do Greengrass nucleus.
2.1.6	Versão atualizada para a versão 2.10.0 do Greengrass nucleus.
2.1.5	Correções de erros e melhorias • Corrige um problema com a `ReadDiscreteInput` operação.
2.1.4	Versão atualizada para a versão 2.9.0 do Greengrass nucleus.
2.1.3	Versão atualizada para a versão 2.8.0 do Greengrass nucleus.
2.1.2	Versão atualizada para a versão 2.7.0 do Greengrass nucleus.
2.1.1	Versão atualizada para a versão 2.6.0 do Greengrass nucleus.
2.1.0	Novos atributos • Adiciona as `ModbusStopBits` opções `ModbusBaudRate` `ModbusByteSize` ,`ModbusParity` , e que você pode especificar para configurar a comunicação serial com dispositivos Modbus RTU.
2.0.8	Versão atualizada para a versão 2.5.0 do Greengrass nucleus.
2.0.7	Versão atualizada para a versão 2.4.0 do Greengrass nucleus.
2.0.6	Versão atualizada para a versão 2.3.0 do Greengrass nucleus.
2.0.5	Versão atualizada para a versão 2.2.0 do Greengrass nucleus.
2.0.4	Versão atualizada para a versão 2.1.0 do Greengrass nucleus.

Version (Versão)	Alterações
2.0.3	Versão inicial.

Ponte MQTT

O componente de ponte MQTT (`aws.greengrass.clientdevices.mqtt.Bridge`) retransmite mensagens MQTT entre dispositivos cliente, publicação/assinatura local do Greengrass e. AWS IoT Core Você pode usar esse componente para agir em mensagens MQTT de dispositivos cliente em componentes personalizados e sincronizar dispositivos cliente com o. Nuvem AWS

> **Note**
>
> Os dispositivos cliente são dispositivos IoT locais que se conectam a um dispositivo central do Greengrass para enviar mensagens e dados MQTT para processamento. Para ter mais informações, consulte Interaja com dispositivos IoT locais.

Você pode usar esse componente para retransmitir mensagens entre os seguintes agentes de mensagens:

- MQTT local — O agente MQTT local manipula mensagens entre dispositivos cliente e um dispositivo principal.
- Publicação/assinatura local — O agente de mensagens local do Greengrass manipula mensagens entre componentes em um dispositivo principal. Para obter mais informações sobre como interagir com essas mensagens nos componentes do Greengrass, consulte. Publique/assine mensagens locais
- AWS IoT Core — O corretor AWS IoT Core MQTT lida com mensagens entre dispositivos Nuvem AWS e destinos de IoT. Para obter mais informações sobre como interagir com essas mensagens nos componentes do Greengrass, consulte. Publique/assine mensagens MQTT AWS IoT Core

  > **Note**
  >
  > A ponte MQTT usa QoS 1 para publicar e assinar AWS IoT Core, mesmo quando um dispositivo cliente usa QoS 0 para publicar e assinar o broker MQTT local. Como resultado, você pode observar latência adicional ao retransmitir mensagens MQTT de dispositivos clientes no agente MQTT local para o. AWS IoT Core Para obter mais informações sobre

a configuração do MQTT nos dispositivos principais, consulteDefina os tempos limite do MQTT e as configurações de cache.

Tópicos

Versões

Esse componente tem as seguintes versões:

- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

Tipo

Este componente é um componente de plug-in (`aws.greengrass.plugin`). O núcleo do Greengrass executa esse componente na mesma Java Virtual Machine (JVM) do núcleo. O núcleo reinicia quando você altera a versão desse componente no dispositivo principal.

Esse componente usa o mesmo arquivo de log do núcleo do Greengrass. Para ter mais informações, consulte Monitore AWS IoT Greengrass os registros.

Para ter mais informações, consulte Tipos de componentes.

Sistema operacional

Esse componente pode ser instalado em dispositivos principais que executam os seguintes sistemas operacionais:

- Linux
- Windows

Requisitos

Esse componente tem os seguintes requisitos:

- Se você configurar o componente intermediário MQTT do dispositivo principal para usar uma porta diferente da porta padrão 8883, deverá usar a ponte MQTT v2.1.0 ou posterior. Configure-o para se conectar na porta em que o broker opera.
- O componente de ponte MQTT tem suporte para ser executado em uma VPC.

Dependências

Quando você implanta um componente, AWS IoT Greengrass também implanta versões compatíveis de suas dependências. Isso significa que você deve atender aos requisitos do componente e de todas as suas dependências para implantá-lo com êxito. Esta seção lista as dependências das versões lançadas desse componente e as restrições de versão semântica que definem as versões dos componentes para cada dependência. Você também pode visualizar as dependências de cada versão do componente no AWS IoT Greengrass console. Na página de detalhes do componente, procure a lista de dependências.

2.3.2

A tabela a seguir lista as dependências da versão 2.3.2 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Autenticação do dispositivo cliente	>=2.2.0 <2.6.0	Rígido

2.3.0 and 2.3.1

A tabela a seguir lista as dependências das versões 2.3.0 e 2.3.1 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Autenticação do dispositivo cliente	>=2.2.0 <2.5.0	Rígido

2.2.5 and 2.2.6

A tabela a seguir lista as dependências das versões 2.2.5 e 2.2.6 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Autenticação do dispositivo cliente	>=2.2.0 <2.5.0	Rígido

2.2.3 and 2.2.4

A tabela a seguir lista as dependências das versões 2.2.3 e 2.2.4 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Autenticação do dispositivo cliente	>=2.2.0 <2.4.0	Rígido

2.2.0 – 2.2.2

A tabela a seguir lista as dependências das versões 2.2.0 a 2.2.2 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Autenticação do dispositivo cliente	>=2.2.0 <2.3.0	Rígido

2.1.1

A tabela a seguir lista as dependências da versão 2.1.1 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Autenticação do dispositivo cliente	>=2.0.0 <2.2.0	Rígido

2.0.0 to 2.1.0

A tabela a seguir lista as dependências das versões 2.0.0 a 2.1.0 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Autenticação do dispositivo cliente	>=2.0.0 <2.1.0	Rígido

Para obter mais informações sobre dependências de componentes, consulte a referência da receita do componente.

Configuração

Esse componente fornece os seguintes parâmetros de configuração que você pode personalizar ao implantar o componente.

2.3.0 – 2.3.2

`mqttTopicMapping`

Os mapeamentos de tópicos que você deseja unir. Esse componente assina mensagens no tópico de origem e publica as mensagens recebidas no tópico de destino. Cada mapeamento de tópicos define o tópico, o tipo de origem e o tipo de destino.

Esse objeto contém as seguintes informações:

`topicMappingNameKey`

O nome desse mapeamento de tópicos. Substitua *`topicMappingNameKey`* por um nome que ajude a identificar esse mapeamento de tópicos.

Esse objeto contém as seguintes informações:

`topic`

O tópico ou filtro de tópicos para fazer a ponte entre os corretores de origem e de destino.

Você pode usar os curingas de tópico + e # MQTT para retransmitir mensagens em todos os tópicos que correspondam a um filtro de tópico. Para obter mais informações, consulte os tópicos do MQTT no Guia do AWS IoT Core desenvolvedor.

> **Note**
>
> Para usar os curingas de tópico do MQTT com o corretor de `Pubsub` origem, você deve usar a versão v2.6.0 ou posterior do componente Greengrass nucleus.

`targetTopicPrefix`

O prefixo a ser adicionado ao tópico de destino quando esse componente retransmite a mensagem.

`source`

O agente de mensagens de origem. Escolha uma das seguintes opções:

- `LocalMqtt`— O agente MQTT local onde os dispositivos do cliente se comunicam.
- `Pubsub`— O agente local de publicação/assinatura de mensagens do Greengrass.
- `IotCore`— O agente de mensagens AWS IoT Core MQTT.

> **Note**
>
> A ponte MQTT usa QoS 1 para publicar e assinar AWS IoT Core, mesmo quando um dispositivo cliente usa QoS 0 para publicar e assinar o broker MQTT local. Como resultado, você pode observar latência adicional ao retransmitir mensagens MQTT de dispositivos clientes no agente MQTT local para o. AWS IoT Core Para obter mais informações sobre a configuração do MQTT nos dispositivos principais, consulteDefina os tempos limite do MQTT e as configurações de cache.

`source` e `target` deve ser diferente.

`target`

O agente de mensagens de destino. Escolha uma das seguintes opções:

- `LocalMqtt` — O agente MQTT local onde os dispositivos do cliente se comunicam.
- `Pubsub` — O agente local de publicação/assinatura de mensagens do Greengrass.
- `IotCore` — O agente de mensagens AWS IoT Core MQTT.

> **Note**
>
> A ponte MQTT usa QoS 1 para publicar e assinar AWS IoT Core, mesmo quando um dispositivo cliente usa QoS 0 para publicar e assinar o broker MQTT local. Como resultado, você pode observar latência adicional ao retransmitir mensagens MQTT de dispositivos clientes no agente MQTT local para o. AWS IoT Core Para obter mais informações sobre a configuração do MQTT nos dispositivos principais, consulte Defina os tempos limite do MQTT e as configurações de cache.

`source` e `target` deve ser diferente.

mqtt5 RouteOptions

(Opcional) Fornece opções para configurar mapeamentos de tópicos para conectar mensagens do tópico de origem ao tópico de destino.

Esse objeto contém as seguintes informações:

`mqtt5 RouteOptionsNameKey`

O nome das opções de rota para um mapeamento de tópicos. Substitua *`mqtt5 RouteOptionsNameKey`* pela *`topicMappingNamechave`* correspondente definida no `mqttTopicMapping` campo.

Esse objeto contém as seguintes informações:

Sem local

(Opcional) Quando ativada, a ponte não encaminha mensagens sobre um tópico publicado pela própria ponte. Use isso para evitar loops, da seguinte forma:

```
{
```

```
    "mqtt5RouteOptions": {
        "toIoTCore": {
            "noLocal": true
        }
    },
    "mqttTopicMapping": {
        "toIoTCore": {
            "topic": "device",
            "source": "LocalMqtt",
            "target": "IotCore"
        },
        "toLocal": {
            "topic": "device",
            "source": "IotCore",
            "target": "LocalMqtt"
        }
    }
}
```

`noLocal`só é compatível com rotas em `source` que `LocalMqtt` o.

Padrão: False

retainAsPublished

(Opcional) Quando ativadas, as mensagens encaminhadas pela ponte têm o mesmo `retain` sinalizador das mensagens publicadas no agente para essa rota.

`retainAsPublished`só é compatível com rotas em `source` que `LocalMqtt` o.

Padrão: False

mqtt

(Opcional) Configurações do protocolo MQTT para comunicação com o corretor local.

versão

(Opcional) A versão do protocolo MQTT usada pela ponte para se comunicar com o corretor local. Deve ser igual à versão do MQTT selecionada na configuração do núcleo.

Escolha uma das seguintes opções:

- `mqtt3`
- `mqtt5`

Você deve implantar um agente MQTT quando o `target` campo `source` ou do `mqttTopicMapping` objeto estiver definido como. `LocalMqtt` Se você escolher a `mqtt5` opção, deverá usar Corretora MQTT 5 (EMQX) o.

Padrão: `mqtt3`

ackTimeoutSeconds

(Opcional) Intervalo de tempo para aguardar os pacotes PUBACK, SUBACK ou UNSUBACK antes de falhar na operação.

Padrão: 60

connAckTimeoutSrta

(Opcional) Intervalo de tempo para esperar por um pacote CONNACK antes de desligar a conexão.

Padrão: 20000 (20 segundos)

pingTimeoutMs

(Opcional) A quantidade de tempo em milissegundos que a ponte espera para receber uma mensagem PINGACK do agente local. Se a espera exceder o tempo limite, a ponte será fechada e reabrirá a conexão MQTT. Esse valor deve ser menor que`keepAliveTimeoutSeconds`.

Padrão: 30000 (30 segundos)

keepAliveTimeoutSegundos

(Opcional) A quantidade de tempo em segundos entre cada mensagem PING que a ponte envia para manter a conexão MQTT ativa. Esse valor deve ser maior que`pingTimeoutMs`.

Padrão: 60

maxReconnectDelaySrta

(Opcional) O tempo máximo em segundos para o MQTT se reconectar.

Padrão: 30000 (30 segundos)

minReconnectDelaySrta

(Opcional) O tempo mínimo em segundos para o MQTT se reconectar.

Receba o máximo

(Opcional) O número máximo de pacotes de QoS1 não reconhecidos que a ponte pode enviar.

Padrão: 100

maximumPacketSize

O número máximo de bytes que o cliente aceitará para um pacote MQTT.

Padrão: nulo (sem limite)

sessionExpiryInterval

(Opcional) A quantidade de tempo em segundos que você pode solicitar para que uma sessão dure entre a ponte e a corretora local.

Padrão: 4294967295 (a sessão nunca expira)

`brokerUri`

(Opcional) O URI do broker MQTT local. Você deve especificar esse parâmetro se configurar o agente MQTT para usar uma porta diferente da porta padrão 8883. Use o seguinte formato e substitua a *`porta pela porta`* em que o corretor MQTT opera:`ssl://localhost:`*`port`*.

Padrão: `ssl://localhost:8883`

startupTimeoutSeconds

(Opcional) O tempo máximo em segundos para o componente iniciar. O estado do componente muda para `BROKEN` se ele exceder esse tempo limite.

Padrão: `120`

Example Exemplo: atualização da mesclagem de configurações

O exemplo de atualização de configuração a seguir especifica o seguinte:

- Retransmita mensagens de dispositivos clientes para AWS IoT Core tópicos que correspondam ao filtro de `clients/+/hello/world` tópicos.
- Retransmita mensagens de dispositivos cliente para publicação/assinatura local em tópicos que correspondam ao filtro de tópicos e adicione o `events/input/` prefixo ao `clients/+/detections` tópico de destino. O tópico de destino resultante corresponde ao filtro de `events/input/clients/+/detections` tópicos.

- Retransmita mensagens de dispositivos cliente para AWS IoT Core tópicos que correspondam ao filtro de tópicos e adicione o `$aws/rules/StatusUpdateRule/` prefixo ao `clients/+/status` tópico de destino. Este exemplo retransmite essas mensagens diretamente para uma AWS IoT regra chamada `StatusUpdateRule` para reduzir custos usando o Basic Ingest.

```
{
  "mqttTopicMapping": {
    "ClientDeviceHelloWorld": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    },
    "ClientDeviceEvents": {
      "topic": "clients/+/detections",
      "targetTopicPrefix": "events/input/",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ClientDeviceCloudStatusUpdate": {
      "topic": "clients/+/status",
      "targetTopicPrefix": "$aws/rules/StatusUpdateRule/",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

Example Exemplo: Configurando o MQTT 5

O exemplo de configuração a seguir atualiza o seguinte:

- Permite que a ponte use o protocolo MQTT 5 com o broker local.
- Configura a configuração de retenção de MQTT como publicada para o mapeamento de `ClientDeviceHelloWorld` tópicos.

```
{
  "mqttTopicMapping": {
    "ClientDeviceHelloWorld": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
```

```
      "target": "IotCore"
    }
  },
  "mqtt5RouteOptions": {
    "ClientDeviceHelloWorld": {
      "retainAsPublished": true
    }
  },
  "mqtt": {
    "version": "mqtt5"
  }
}
```

2.2.6

`mqttTopicMapping`

Os mapeamentos de tópicos que você deseja unir. Esse componente assina mensagens no tópico de origem e publica as mensagens recebidas no tópico de destino. Cada mapeamento de tópicos define o tópico, o tipo de origem e o tipo de destino.

Esse objeto contém as seguintes informações:

`topicMappingNameKey`

O nome desse mapeamento de tópicos. Substitua *`topicMappingNameKey`* por um nome que ajude a identificar esse mapeamento de tópicos.

Esse objeto contém as seguintes informações:

`topic`

O tópico ou filtro de tópicos para fazer a ponte entre os corretores de origem e de destino.

Você pode usar os curingas de tópico + e # MQTT para retransmitir mensagens em todos os tópicos que correspondam a um filtro de tópico. Para obter mais informações, consulte os tópicos do MQTT no Guia do AWS IoT Core desenvolvedor.

> **Note**
>
> Para usar os curingas de tópico do MQTT com o corretor de `Pubsub` origem, você deve usar a versão v2.6.0 ou posterior do componente Greengrass nucleus.

`targetTopicPrefix`

O prefixo a ser adicionado ao tópico de destino quando esse componente retransmite a mensagem.

`source`

O agente de mensagens de origem. Escolha uma das seguintes opções:

- `LocalMqtt`— O agente MQTT local onde os dispositivos do cliente se comunicam.
- `Pubsub`— O agente local de publicação/assinatura de mensagens do Greengrass.
- `IotCore`— O agente de mensagens AWS IoT Core MQTT.

> **Note**
>
> A ponte MQTT usa QoS 1 para publicar e assinar AWS IoT Core, mesmo quando um dispositivo cliente usa QoS 0 para publicar e assinar o broker MQTT local. Como resultado, você pode observar latência adicional ao retransmitir mensagens MQTT de dispositivos clientes no agente MQTT local para o. AWS IoT Core Para obter mais informações sobre a configuração do MQTT nos dispositivos principais, consulteDefina os tempos limite do MQTT e as configurações de cache.

`source`e `target` deve ser diferente.

`target`

O agente de mensagens de destino. Escolha uma das seguintes opções:

- `LocalMqtt`— O agente MQTT local onde os dispositivos do cliente se comunicam.
- `Pubsub`— O agente local de publicação/assinatura de mensagens do Greengrass.
- `IotCore`— O agente de mensagens AWS IoT Core MQTT.

> **Note**
>
> A ponte MQTT usa QoS 1 para publicar e assinar AWS IoT Core, mesmo quando um dispositivo cliente usa QoS 0 para publicar e assinar o broker MQTT local. Como resultado, você pode observar latência adicional ao retransmitir mensagens MQTT de dispositivos clientes no agente MQTT local para o. AWS IoT Core Para obter mais informações sobre a configuração do MQTT nos dispositivos principais, consulteDefina os tempos limite do MQTT e as configurações de cache.

`sourcee target` deve ser diferente.

`brokerUri`

(Opcional) O URI do broker MQTT local. Você deve especificar esse parâmetro se configurar o agente MQTT para usar uma porta diferente da porta padrão 8883. Use o seguinte formato e substitua a *`porta pela porta`* em que o corretor MQTT opera:`ssl://localhost:`*`port`*.

Padrão: `ssl://localhost:8883`

startupTimeoutSeconds

(Opcional) O tempo máximo em segundos para o componente iniciar. O estado do componente muda para `BROKEN` se ele exceder esse tempo limite.

Padrão: `120`

Example Exemplo: atualização da mesclagem de configurações

O exemplo de atualização de configuração a seguir especifica o seguinte:

- Retransmita mensagens de dispositivos clientes para AWS IoT Core tópicos que correspondam ao filtro de `clients/+/hello/world` tópicos.
- Retransmita mensagens de dispositivos cliente para publicação/assinatura local em tópicos que correspondam ao filtro de tópicos e adicione o `events/input/` prefixo ao `clients/+/detections` tópico de destino. O tópico de destino resultante corresponde ao filtro de `events/input/clients/+/detections` tópicos.
- Retransmita mensagens de dispositivos cliente para AWS IoT Core tópicos que correspondam ao filtro de tópicos e adicione o `$aws/rules/StatusUpdateRule/` prefixo ao `clients/+/`

`status` tópico de destino. Este exemplo retransmite essas mensagens diretamente para uma AWS IoT regra chamada `StatusUpdateRule` para reduzir custos usando o Basic Ingest.

```
{
  "mqttTopicMapping": {
    "ClientDeviceHelloWorld": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    },
    "ClientDeviceEvents": {
      "topic": "clients/+/detections",
      "targetTopicPrefix": "events/input/",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ClientDeviceCloudStatusUpdate": {
      "topic": "clients/+/status",
      "targetTopicPrefix": "$aws/rules/StatusUpdateRule/",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

2.2.0 - 2.2.5

`mqttTopicMapping`

Os mapeamentos de tópicos que você deseja unir. Esse componente assina mensagens no tópico de origem e publica as mensagens recebidas no tópico de destino. Cada mapeamento de tópicos define o tópico, o tipo de origem e o tipo de destino.

Esse objeto contém as seguintes informações:

`topicMappingNameKey`

O nome desse mapeamento de tópicos. Substitua *`topicMappingNameKey`* por um nome que ajude a identificar esse mapeamento de tópicos.

Esse objeto contém as seguintes informações:

`topic`

O tópico ou filtro de tópicos para fazer a ponte entre os corretores de origem e de destino.

Você pode usar os curingas de tópico + e # MQTT para retransmitir mensagens em todos os tópicos que correspondam a um filtro de tópico. Para obter mais informações, consulte os tópicos do MQTT no Guia do AWS IoT Core desenvolvedor.

> **Note**
>
> Para usar os curingas de tópico do MQTT com o corretor de `Pubsub` origem, você deve usar a versão v2.6.0 ou posterior do componente Greengrass nucleus.

`targetTopicPrefix`

O prefixo a ser adicionado ao tópico de destino quando esse componente retransmite a mensagem.

`source`

O agente de mensagens de origem. Escolha uma das seguintes opções:

- `LocalMqtt`— O agente MQTT local onde os dispositivos do cliente se comunicam.
- `Pubsub`— O agente local de publicação/assinatura de mensagens do Greengrass.
- `IotCore`— O agente de mensagens AWS IoT Core MQTT.

> **Note**
>
> A ponte MQTT usa QoS 1 para publicar e assinar AWS IoT Core, mesmo quando um dispositivo cliente usa QoS 0 para publicar e assinar o broker MQTT local. Como resultado, você pode observar latência adicional ao retransmitir mensagens MQTT de dispositivos clientes no agente MQTT local para o. AWS IoT Core Para obter mais informações sobre a configuração do MQTT nos dispositivos principais, consulteDefina os tempos limite do MQTT e as configurações de cache.

`source`e `target` deve ser diferente.

`target`

O agente de mensagens de destino. Escolha uma das seguintes opções:

- `LocalMqtt`— O agente MQTT local onde os dispositivos do cliente se comunicam.
- `Pubsub`— O agente local de publicação/assinatura de mensagens do Greengrass.
- `IotCore`— O agente de mensagens AWS IoT Core MQTT.

> **Note**
>
> A ponte MQTT usa QoS 1 para publicar e assinar AWS IoT Core, mesmo quando um dispositivo cliente usa QoS 0 para publicar e assinar o broker MQTT local. Como resultado, você pode observar latência adicional ao retransmitir mensagens MQTT de dispositivos clientes no agente MQTT local para o. AWS IoT Core Para obter mais informações sobre a configuração do MQTT nos dispositivos principais, consulteDefina os tempos limite do MQTT e as configurações de cache.

`source`e `target` deve ser diferente.

`brokerUri`

(Opcional) O URI do broker MQTT local. Você deve especificar esse parâmetro se configurar o agente MQTT para usar uma porta diferente da porta padrão 8883. Use o seguinte formato e substitua a *`porta pela porta`* em que o corretor MQTT opera:`ssl://localhost:`*`port`*.

Padrão: `ssl://localhost:8883`

Example Exemplo: atualização da mesclagem de configurações

O exemplo de atualização de configuração a seguir especifica o seguinte:

- Retransmita mensagens de dispositivos clientes para AWS IoT Core tópicos que correspondam ao filtro de `clients/+/hello/world` tópicos.
- Retransmita mensagens de dispositivos cliente para publicação/assinatura local em tópicos que correspondam ao filtro de tópicos e adicione o `events/input/` prefixo ao `clients/+/detections` tópico de destino. O tópico de destino resultante corresponde ao filtro de `events/input/clients/+/detections` tópicos.

- Retransmita mensagens de dispositivos cliente para AWS IoT Core tópicos que correspondam ao filtro de tópicos e adicione o `$aws/rules/StatusUpdateRule/` prefixo ao `clients/+/status` tópico de destino. Este exemplo retransmite essas mensagens diretamente para uma AWS IoT regra chamada `StatusUpdateRule` para reduzir custos usando o Basic Ingest.

```
{
  "mqttTopicMapping": {
    "ClientDeviceHelloWorld": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    },
    "ClientDeviceEvents": {
      "topic": "clients/+/detections",
      "targetTopicPrefix": "events/input/",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ClientDeviceCloudStatusUpdate": {
      "topic": "clients/+/status",
      "targetTopicPrefix": "$aws/rules/StatusUpdateRule/",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

2.1.x

`mqttTopicMapping`

Os mapeamentos de tópicos que você deseja unir. Esse componente assina mensagens no tópico de origem e publica as mensagens recebidas no tópico de destino. Cada mapeamento de tópicos define o tópico, o tipo de origem e o tipo de destino.

Esse objeto contém as seguintes informações:

`topicMappingNameKey`

O nome desse mapeamento de tópicos. Substitua *`topicMappingNameKey`* por um nome que ajude a identificar esse mapeamento de tópicos.

Esse objeto contém as seguintes informações:

`topic`

O tópico ou filtro de tópicos para fazer a ponte entre os corretores de origem e de destino.

Se você especificar o corretor de `IotCore` origem `LocalMqtt` ou de origem, poderá usar os curingas de tópico + e # MQTT para retransmitir mensagens em todos os tópicos que correspondam a um filtro de tópico. Para obter mais informações, consulte os tópicos do MQTT no Guia do AWS IoT Core desenvolvedor.

`source`

O agente de mensagens de origem. Escolha uma das seguintes opções:

- `LocalMqtt`— O agente MQTT local onde os dispositivos do cliente se comunicam.
- `Pubsub`— O agente local de publicação/assinatura de mensagens do Greengrass.
- `IotCore`— O agente de mensagens AWS IoT Core MQTT.

> **Note**
>
> A ponte MQTT usa QoS 1 para publicar e assinar AWS IoT Core, mesmo quando um dispositivo cliente usa QoS 0 para publicar e assinar o broker MQTT local. Como resultado, você pode observar latência adicional ao retransmitir mensagens MQTT de dispositivos clientes no agente MQTT local para o. AWS IoT Core Para obter mais informações sobre a configuração do MQTT nos dispositivos principais, consulteDefina os tempos limite do MQTT e as configurações de cache.

`source`e `target` deve ser diferente.

`target`

O agente de mensagens de destino. Escolha uma das seguintes opções:

- `LocalMqtt`— O agente MQTT local onde os dispositivos do cliente se comunicam.
- `Pubsub`— O agente local de publicação/assinatura de mensagens do Greengrass.
- `IotCore`— O agente de mensagens AWS IoT Core MQTT.

> **Note**
>
> A ponte MQTT usa QoS 1 para publicar e assinar AWS IoT Core, mesmo quando um dispositivo cliente usa QoS 0 para publicar e assinar o broker MQTT local. Como resultado, você pode observar latência adicional ao retransmitir mensagens MQTT de dispositivos clientes no agente MQTT local para o. AWS IoT Core Para obter mais informações sobre a configuração do MQTT nos dispositivos principais, consulte Defina os tempos limite do MQTT e as configurações de cache.

`source` e `target` deve ser diferente.

`brokerUri`

(Opcional) O URI do broker MQTT local. Você deve especificar esse parâmetro se configurar o agente MQTT para usar uma porta diferente da porta padrão 8883. Use o seguinte formato e substitua a *`porta pela porta`* em que o corretor MQTT opera:`ssl://localhost:`*`port`*.

Padrão: `ssl://localhost:8883`

Example Exemplo: atualização da mesclagem de configurações

O exemplo de atualização de configuração a seguir especifica a retransmissão de mensagens de dispositivos cliente para AWS IoT Core tópicos no `clients/MyClientDevice1/hello/world` e. `clients/MyClientDevice2/hello/world`

```
{
  "mqttTopicMapping": {
    "ClientDevice1HelloWorld": {
      "topic": "clients/MyClientDevice1/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    },
    "ClientDevice2HelloWorld": {
      "topic": "clients/MyClientDevice2/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
```

```
}
```

2.0.x

`mqttTopicMapping`

Os mapeamentos de tópicos que você deseja unir. Esse componente assina mensagens no tópico de origem e publica as mensagens recebidas no tópico de destino. Cada mapeamento de tópicos define o tópico, o tipo de origem e o tipo de destino.

Esse objeto contém as seguintes informações:

`topicMappingNameKey`

O nome desse mapeamento de tópicos. Substitua *`topicMappingNameKey`* por um nome que ajude a identificar esse mapeamento de tópicos.

Esse objeto contém as seguintes informações:

`topic`

O tópico ou filtro de tópicos para fazer a ponte entre os corretores de origem e de destino.

Se você especificar o corretor de `IotCore` origem `LocalMqtt` ou de origem, poderá usar os curingas de tópico + e # MQTT para retransmitir mensagens em todos os tópicos que correspondam a um filtro de tópico. Para obter mais informações, consulte os tópicos do MQTT no Guia do AWS IoT Core desenvolvedor.

`source`

O agente de mensagens de origem. Escolha uma das seguintes opções:

- `LocalMqtt`— O agente MQTT local onde os dispositivos do cliente se comunicam.
- `Pubsub`— O agente local de publicação/assinatura de mensagens do Greengrass.
- `IotCore`— O agente de mensagens AWS IoT Core MQTT.

> **Note**
>
> A ponte MQTT usa QoS 1 para publicar e assinar AWS IoT Core, mesmo quando um dispositivo cliente usa QoS 0 para publicar e assinar o broker MQTT local. Como resultado, você pode observar latência adicional ao retransmitir mensagens MQTT de dispositivos clientes no agente MQTT local

para o. AWS IoT Core Para obter mais informações sobre a configuração do MQTT nos dispositivos principais, consulteDefina os tempos limite do MQTT e as configurações de cache.

`source`e `target` deve ser diferente.

`target`

O agente de mensagens de destino. Escolha uma das seguintes opções:

- `LocalMqtt`— O agente MQTT local onde os dispositivos do cliente se comunicam.
- `Pubsub`— O agente local de publicação/assinatura de mensagens do Greengrass.
- `IotCore`— O agente de mensagens AWS IoT Core MQTT.

> **Note**
>
> A ponte MQTT usa QoS 1 para publicar e assinar AWS IoT Core, mesmo quando um dispositivo cliente usa QoS 0 para publicar e assinar o broker MQTT local. Como resultado, você pode observar latência adicional ao retransmitir mensagens MQTT de dispositivos clientes no agente MQTT local para o. AWS IoT Core Para obter mais informações sobre a configuração do MQTT nos dispositivos principais, consulteDefina os tempos limite do MQTT e as configurações de cache.

`source`e `target` deve ser diferente.

Example Exemplo: atualização da mesclagem de configurações

O exemplo de atualização de configuração a seguir especifica a retransmissão de mensagens de dispositivos cliente para AWS IoT Core tópicos no `clients/MyClientDevice1/hello/world` e. `clients/MyClientDevice2/hello/world`

```
{
  "mqttTopicMapping": {
    "ClientDevice1HelloWorld": {
      "topic": "clients/MyClientDevice1/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    },
```

```
    "ClientDevice2HelloWorld": {
      "topic": "clients/MyClientDevice2/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

Arquivo de log local

Esse componente usa o mesmo arquivo de log do componente do núcleo do Greengrass.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

Para ver os registros desse componente

- Execute o comando a seguir no dispositivo principal para visualizar o arquivo de log desse componente em tempo real. Substitua */greengrass/v2* ou *C:\greengrass\v2* pelo caminho para a pasta AWS IoT Greengrass raiz.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

Changelog

A tabela a seguir descreve as alterações em cada versão do componente.

Version (Versão)	Alterações
2.3.2	Versão atualizada para a versão 2.5.0 da autenticação do dispositivo cliente.
2.3.1	Correções de erros e melhorias Corrige um problema em que o cliente MQTT local entra em um loop de desconexão.
2.3.0	Novos atributos Adiciona suporte ao MQTT5 para fazer a ponte entre fontes MQTT locais AWS IoT Core e fontes MQTT.
2.2.6	Novos atributos Adiciona uma nova opção `startupTimeoutSeconds` de configuração.
2.2.5	Versão atualizada para a versão 2.4.0 da autenticação do dispositivo cliente.
2.2.4	Versão atualizada para a versão 2.3.0 de autenticação de dispositivo cliente Greengrass.
2.2.3	Esta versão contém correções de erros e melhorias.
2.2.2	Correções de erros e melhorias • Ajustes de registro.
2.2.1	Correções de erros e melhorias Corrige problemas que podem resultar na falha da assinatura do MQTT bridge nos tópicos do MQTT.
2.2.0	Novos atributos • Adiciona suporte para curingas de tópicos do MQTT (#e+) quando você especifica publicação/assinatura local como agente de mensagens de origem.

Version (Versão)	Alterações
	Esse recurso requer a versão v2.6.0 ou posterior do componente do núcleo do Greengrass. • Adiciona a `targetTopicPrefix` opção, que você pode especificar para configurar a ponte MQTT para adicionar um prefixo ao tópico de destino ao retransmitir uma mensagem.
2.1.1	Correções de erros e melhorias • Corrige problemas com a forma como esse componente lida com as atualizações de redefinição de configuração. • Reduz a frequência de desconexões do cliente MQTT quando os certificados são alternados.
2.1.0	Novos atributos • Adiciona o `brokerUri` parâmetro, que permite usar uma porta de agente MQTT não padrão.
2.0.1	Esta versão inclui correções de erros e melhorias.
2.0.0	Versão inicial.

Corretor MQTT 3.1.1 (Moquette)

O componente de agente Moquette MQTT (`aws.greengrass.clientdevices.mqtt.Moquette`) manipula mensagens MQTT entre dispositivos cliente e um dispositivo principal do Greengrass. Esse componente fornece uma versão modificada do broker Moquette MQTT. Implante esse broker MQTT para executar um broker MQTT leve. Para obter mais informações sobre como escolher um corretor MQTT, consulteEscolha um corretor MQTT.

Esse corretor implementa o protocolo MQTT 3.1.1. Inclui suporte para QoS 0, QoS 1, mensagens retidas de QoS 2, mensagens de última vontade e sessões persistentes.

> **Note**
>
> Os dispositivos cliente são dispositivos IoT locais que se conectam a um dispositivo central do Greengrass para enviar mensagens e dados MQTT para processamento. Para ter mais informações, consulte Interaja com dispositivos IoT locais.

Tópicos

Versões

Esse componente tem as seguintes versões:

- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

Tipo

Este componente é um componente de plug-in (`aws.greengrass.plugin`). O núcleo do Greengrass executa esse componente na mesma Java Virtual Machine (JVM) do núcleo. O núcleo reinicia quando você altera a versão desse componente no dispositivo principal.

Esse componente usa o mesmo arquivo de log do núcleo do Greengrass. Para ter mais informações, consulte Monitore AWS IoT Greengrass os registros.

Para ter mais informações, consulte Tipos de componentes.

Sistema operacional

Esse componente pode ser instalado em dispositivos principais que executam os seguintes sistemas operacionais:

- Linux
- Windows

Requisitos

Esse componente tem os seguintes requisitos:

- O dispositivo principal deve ser capaz de aceitar conexões na porta em que o agente MQTT opera. Esse componente executa o agente MQTT na porta 8883 por padrão. Você pode especificar uma porta diferente ao configurar esse componente.

  Se você especificar uma porta diferente e usar o componente de ponte MQTT para retransmitir mensagens MQTT para outros corretores, deverá usar a ponte MQTT v2.1.0 ou posterior. Configure-o para usar a porta em que o agente MQTT opera.

  Se você especificar uma porta diferente e usar o componente detector de IP para gerenciar os endpoints do broker MQTT, deverá usar o detector de IP v2.1.0 ou posterior. Configure-o para relatar a porta em que o agente MQTT opera.
- O componente de corretor Moquette MQTT tem suporte para ser executado em uma VPC.

Dependências

Quando você implanta um componente, AWS IoT Greengrass também implanta versões compatíveis de suas dependências. Isso significa que você deve atender aos requisitos do componente e de todas as suas dependências para implantá-lo com êxito. Esta seção lista as dependências das versões lançadas desse componente e as restrições de versão semântica que definem as versões do componente para cada dependência. Você também pode visualizar as dependências de cada versão do componente no AWS IoT Greengrass console. Na página de detalhes do componente, procure a lista de dependências.

2.3.7

A tabela a seguir lista as dependências da versão 2.3.7 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Autenticação do dispositivo cliente	>=2.2.0 <2.6.0	Rígido

2.3.2 – 2.3.6

A tabela a seguir lista as dependências das versões 2.3.2 a 2.3.6 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Autenticação do dispositivo cliente	>=2.2.0 <2.5.0	Rígido

2.3.0 and 2.3.1

A tabela a seguir lista as dependências das versões 2.3.0 e 2.3.1 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Autenticação do dispositivo cliente	>=2.2.0 <2.4.0	Rígido

2.2.0

A tabela a seguir lista as dependências da versão 2.2.0 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Autenticação do dispositivo cliente	>=2.2.0 <2.3.0	Rígido

2.1.0

A tabela a seguir lista as dependências da versão 2.1.0 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Autenticação do dispositivo cliente	>=2.0.0 <2.2.0	Rígido

2.0.0 - 2.0.2

A tabela a seguir lista as dependências das versões 2.0.0 a 2.0.2 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Autenticação do dispositivo cliente	>=2.0.0 <2.1.0	Rígido

Para obter mais informações sobre dependências de componentes, consulte a referência da receita do componente.

Configuração

Esse componente fornece os seguintes parâmetros de configuração que você pode personalizar ao implantar o componente.

`moquette`

(Opcional) A configuração do broker Moquette MQTT a ser usada. Você pode configurar um subconjunto das opções de configuração do Moqeutte neste componente. Para obter mais informações, consulte os comentários embutidos no arquivo de configuração do Moquette.

Esse objeto contém as seguintes informações:

`ssl_port`

(Opcional) A porta em que o corretor MQTT opera.

> **Note**
>
> Se você especificar uma porta diferente e usar o componente de ponte MQTT para retransmitir mensagens MQTT para outros corretores, deverá usar a ponte MQTT v2.1.0 ou posterior. Configure-o para usar a porta em que o agente MQTT opera.

Se você especificar uma porta diferente e usar o componente detector de IP para gerenciar os endpoints do broker MQTT, deverá usar o detector de IP v2.1.0 ou posterior. Configure-o para relatar a porta em que o agente MQTT opera.

Padrão: `8883`

`host`

(Opcional) A interface na qual o corretor MQTT se vincula. Por exemplo, você pode alterar esse parâmetro para que o broker MQTT se vincule somente a uma rede local específica.

Padrão: `0.0.0.0` (vincula a todas as interfaces de rede)

startupTimeoutSeconds

(Opcional) O tempo máximo em segundos para o componente iniciar. O estado do componente muda para `BROKEN` se ele exceder esse tempo limite.

Padrão: `120`

Example Exemplo: atualização da mesclagem de configurações

O exemplo de configuração a seguir especifica a operação do agente MQTT na porta 443.

```
{
  "moquette": {
    "ssl_port": "443"
  }
}
```

Arquivo de log local

Esse componente usa o mesmo arquivo de log do componente do núcleo do Greengrass.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

Para ver os registros desse componente

- Execute o comando a seguir no dispositivo principal para visualizar o arquivo de log desse componente em tempo real. Substitua */greengrass/v2* ou *C:\greengrass\v2* pelo caminho para a pasta AWS IoT Greengrass raiz.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

Changelog

A tabela a seguir descreve as alterações em cada versão do componente.

Version (Versão)	Alterações
2.3.7	Versão atualizada para a versão 2.5.0 da autenticação do dispositivo cliente.
2.3.6	Correções de erros e melhorias • Melhorias e correções de erros gerais.
2.3.5	Correções de erros e melhorias • Moquette atualizado para a versão 0.17.
2.3.4	Correções de erros e melhorias • Corrige um problema em que os clientes podem enfrentar erros de sessão inválidos ao enviar ou receber mensagens, devido a IDs de clientes duplicados. Esse problema fez com que a sessão do cliente fosse encerrada.
2.3.3	Novos atributos Adiciona uma nova opção `startupTimeoutSeconds` de configuração.

Version (Versão)	Alterações
2.3.2	Versão atualizada para a versão 2.4.0 da autenticação do dispositivo cliente.
2.3.1	Correções de erros e melhorias • Corrige uma condição de corrida em que os clientes podem ser desconectados após tentarem se reconectar, devido a uma sessão inválida.
2.3.0	Adiciona suporte para cadeias de certificados.
2.2.0	Versão atualizada para a versão 2.2.0 da autenticação do dispositivo cliente.
2.1.0	Correções de erros e melhorias • Atualiza esse componente para usar a versão 0.16 do Moquette, que melhora o desempenho e inclui várias outras melhorias. • Corrige um problema em que o certificado do servidor MQTT local gira com mais frequência do que o pretendido em determinados cenários. Para aplicar essa correção, você também deve usar a versão 2.1.0 ou posterior do componente de autenticação do dispositivo cliente.
2.0.2	Correções de erros e melhorias • Aumenta o tamanho máximo da mensagem MQTT de 8.092 bytes para 128 kilobytes. O limite efetivo de carga útil da mensagem MQTT é um pouco menor, porque o limite de tamanho da mensagem inclui cabeçalhos de mensagens. • Adiciona suporte para valores inteiros no `ssl_port` parâmetro.
2.0.1	Versão atualizada para a versão 2.4.0 do Greengrass nucleus.
2.0.0	Versão inicial.

Corretora MQTT 5 (EMQX)

O componente broker EMQX MQTT (`aws.greengrass.clientdevices.mqtt.EMQX`) manipula mensagens MQTT entre dispositivos cliente e um dispositivo principal do Greengrass. Esse

componente fornece uma versão modificada do broker EMQX MQTT 5.0. Implante esse agente MQTT para usar os recursos do MQTT 5 na comunicação entre dispositivos cliente e um dispositivo principal. Para obter mais informações sobre como escolher um corretor MQTT, consulteEscolha um corretor MQTT.

Esse corretor implementa o protocolo MQTT 5.0. Ele inclui suporte para intervalos de expiração de sessões e mensagens, propriedades do usuário, assinaturas compartilhadas, aliases de tópicos e muito mais. O MQTT 5 é compatível com versões anteriores do MQTT 3.1.1, portanto, se você executar o broker Moquette MQTT 3.1.1, poderá substituí-lo pelo broker EMQX MQTT 5, e os dispositivos clientes poderão continuar se conectando e operando normalmente.

> **Note**
>
> Os dispositivos cliente são dispositivos IoT locais que se conectam a um dispositivo principal do Greengrass para enviar mensagens e dados MQTT para processamento. Para ter mais informações, consulte Interaja com dispositivos IoT locais.

Tópicos

Versões

Esse componente tem as seguintes versões:

- 2.0.x
- 1,2.x

- 1.1.x
- 1,0.x

Tipo

Este componente é um componente genérico (`aws.greengrass.generic`). O núcleo do Greengrass executa os scripts do ciclo de vida do componente.

Para ter mais informações, consulte Tipos de componentes.

Sistema operacional

Esse componente pode ser instalado em dispositivos principais que executam os seguintes sistemas operacionais:

- Linux
- Windows

Requisitos

Esse componente tem os seguintes requisitos:

- O dispositivo principal deve ser capaz de aceitar conexões na porta em que o agente MQTT opera. Esse componente executa o agente MQTT na porta 8883 por padrão. Você pode especificar uma porta diferente ao configurar esse componente.

  Se você especificar uma porta diferente e usar o componente de ponte MQTT para retransmitir mensagens MQTT para outros corretores, deverá usar a ponte MQTT v2.1.0 ou posterior. Configure-o para usar a porta em que o agente MQTT opera.

  Se você especificar uma porta diferente e usar o componente detector de IP para gerenciar os endpoints do broker MQTT, deverá usar o detector de IP v2.1.0 ou posterior. Configure-o para relatar a porta em que o agente MQTT opera.
- Nos dispositivos principais do Linux, o Docker é instalado e configurado no dispositivo principal:
  - Docker Engine 1.9.1 ou posterior instalado no dispositivo principal do Greengrass. A versão 20.10 é a versão mais recente verificada para funcionar com o software AWS IoT Greengrass Core. Você deve instalar o Docker diretamente no dispositivo principal antes de implantar componentes que executam contêineres do Docker.

  - O daemon do Docker foi iniciado e executado no dispositivo principal antes de você implantar esse componente.
  - O usuário do sistema que executa esse componente deve ter permissões de root ou administrador. Como alternativa, você pode executar esse componente como usuário do sistema no `docker` grupo e configurar a `requiresPrivileges` opção desse componente `false` para executar o broker EQMX MQTT sem privilégios.
- O componente de agente EMQX MQTT tem suporte para ser executado em uma VPC.
- O componente de corretor EMQX MQTT não é suportado na plataforma. `armv7`

Dependências

Quando você implanta um componente, AWS IoT Greengrass também implanta versões compatíveis de suas dependências. Isso significa que você deve atender aos requisitos do componente e de todas as suas dependências para implantá-lo com êxito. Esta seção lista as dependências das versões lançadas desse componente e as restrições de versão semântica que definem as versões dos componentes para cada dependência. Você também pode visualizar as dependências de cada versão do componente no AWS IoT Greengrass console. Na página de detalhes do componente, procure a lista de dependências.

2.0.1

A tabela a seguir lista as dependências da versão 2.0.1 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Autenticação do dispositivo cliente	>=2.2.0 <2.6.0	Rígido

2.0.0

A tabela a seguir lista as dependências da versão 2.0.0 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Autenticação do dispositivo cliente	>=2.2.0 <2.5.0	Rígido

1.2.2 – 1.2.3

A tabela a seguir lista as dependências das versões 1.2.2 a 1.2.3 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Autenticação do dispositivo cliente	>=2.2.0 <2.5.0	Rígido

1.2.0 and 1.2.1

A tabela a seguir lista as dependências das versões 1.2.0 e 1.2.1 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Autenticação do dispositivo cliente	>=2.2.0 <2.4.0	Rígido

1.0.0 and 1.1.0

A tabela a seguir lista as dependências das versões 1.0.0 e 1.1.0 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Autenticação do dispositivo cliente	>=2.2.0 <2.3.0	Rígido

Para obter mais informações sobre dependências de componentes, consulte a referência da receita do componente.

Configuração

2.0.0 - 2.0.1

Esse componente fornece os seguintes parâmetros de configuração que você pode personalizar ao implantar o componente.

> **Important**
>
> Se você usa a versão 2 do componente MQTT 5 broker (EMQX), você deve atualizar seu arquivo de configuração. Os arquivos de configuração da versão 1 não funcionam com a versão 2.

Configuração do EMQX

(Opcional) A configuração do broker EMQX MQTT a ser usada. Você pode definir as opções de configuração do EMQX neste componente.

Quando você usa o corretor EMQX, o Greengrass usa uma configuração padrão. Essa configuração é usada, a menos que você a modifique usando esse campo.

A modificação das seguintes configurações faz com que o componente do agente EMQX seja reiniciado. Outras alterações de configuração se aplicam sem reiniciar o componente.

- `emqxConfig/cluster`
- `emqxConfig/node`
- `emqxConfig/rpc`

> **Note**
>
> `aws.greengrass.clientdevices.mqtt.EMQX`permite que você configure opções sensíveis à segurança. Isso inclui configurações de TLS, autenticação e provedores de autorização. Recomendamos a configuração padrão que usa a autenticação TLS mútua e o provedor de autenticação de dispositivos do cliente Greengrass.

Example Exemplo: configuração padrão

O exemplo a seguir mostra os padrões definidos para o broker MQTT 5 (EMQX). Você pode substituir essas configurações usando a `emqxConfig` configuração.

```
{
  "authorization": {
    "no_match": "deny",
    "sources": []
```

```
  },
  "node": {
    "cookie": "<placeholder>"
  },
  "listeners": {
     "ssl": {
       "default": {
         "ssl_options": {
           "keyfile": "{work:path}\\data\\key.pem",
           "certfile": "{work:path}\\data\\cert.pem",
           "cacertfile": null,
           "verify": "verify_peer",
           "versions": ["tlsv1.3", "tlsv1.2"],
           "fail_if_no_peer_cert": true
         }
       }
     },
     "tcp": {
       "default": {
         "enabled": false
       }
     },
     "ws": {
       "default": {
         "enabled": false
       }
     },
     "wss": {
       "default": {
         "enabled": false
       }
     }
  },
  "plugins": {
    "states": [{"name_vsn": "gg-1.0.0", "enable": true}],
    "install_dir": "plugins"
  }
}
```

Modo de autenticação

(Opcional) Define o provedor de autorização para o corretor. Pode ser um dos valores a seguir:

- `enabled`— (Padrão) Use o provedor de autenticação e autorização do Greengrass.
- `bypass_on_failure`— Use o provedor de autenticação do Greengrass e, em seguida, use qualquer provedor de autenticação restante na cadeia de provedores do EMQX se o Greengrass negar a autenticação ou a autorização.
- `bypass`— O provedor Greengrass está desativado. A autenticação e a autorização são gerenciadas pela cadeia de fornecedores do EMQX.

`requiresPrivilege`

(Opcional) Nos dispositivos principais do Linux, você pode especificar a execução do broker EMQX MQTT sem privilégios de root ou administrador. Se você definir essa opção como`false`, o usuário do sistema que executa esse componente deverá ser membro do `docker` grupo.

Padrão: `true`

`startupTimeoutSeconds`

(Opcional) O tempo máximo em segundos para o broker EMQX MQTT iniciar. O estado do componente muda para `BROKEN` se ele exceder esse tempo limite.

Padrão: `90`

`ipcTimeoutSeconds`

(Opcional) O tempo máximo em segundos para o componente esperar até que o núcleo do Greengrass responda às solicitações de comunicação entre processos (IPC). Aumente esse número se esse componente relatar erros de tempo limite ao verificar se um dispositivo cliente está autorizado.

Padrão: `5`

`crtLogLevel`

(Opcional) O nível de registro da biblioteca AWS Common Runtime (CRT).

O padrão é o nível de log (in) do broker EMQX MQTT. `log.level` `emqx`

`restartIdentifier`

(Opcional) Configure essa opção para reiniciar o agente EMQX MQTT. Quando esse valor de configuração é alterado, esse componente reinicia o agente MQTT. Você pode usar essa opção para forçar a desconexão dos dispositivos cliente.

`dockerOptions`

(Opcional) Configure essa opção somente em sistemas operacionais Linux para adicionar parâmetros à linha de comando do Docker. Por exemplo, para mapear portas adicionais, use o parâmetro `-p` Docker:

```
"-p 1883:1883"
```

Example Exemplo: atualização de um arquivo de configuração v1.x para v2.x

O exemplo a seguir mostra as alterações necessárias para atualizar um arquivo de configuração v1.x para a versão 2.x.

O arquivo de configuração da versão 1.x:

```
{
    "emqx": {
        "listener.ssl.external": "443",
        "listener.ssl.external.max_connections": "1024000",
        "listener.ssl.external.max_conn_rate": "500",
        "listener.ssl.external.rate_limit": "50KB,5s",
        "listener.ssl.external.handshake_timeout": "15s",
        "log.level": "warning"
    },
    "mergeConfigurationFiles": {
        "etc/plugins/aws_greengrass_emqx_auth.conf": "auth_mode=enabled\n
 use_greengrass_managed_certificates=true\n"
    }
}
```

O arquivo de configuração equivalente para v2:

```
{
    "emqxConfig": {
        "listeners": {
            "ssl": {
                "default": {
                   "bind": "8883",
                   "max_connections": "1024000",
                   "max_conn_rate": "500",
                   "handshake_timeout": "15s"
```

```
                }
            }
        },
        "log": {
            "console": {
              "enable": true,
              "level": "warning"
            }
        }
    },
    "authMode": "enabled"
}
```

Não há equivalente à entrada `listener.ssl.external.rate_limit` de configuração. A opção de `use_greengrass_managed_certificates` configuração foi removida.

Example Exemplo: definir uma nova porta para o corretor

O exemplo a seguir altera a porta em que o agente MQTT opera da 8883 padrão para a porta 1234. Se você estiver usando Linux, inclua o `dockerOptions` campo.

```
{
  "emqxConfig": {
    "listeners": {
      "ssl": {
        "default": {
          "bind": 1234
        }
      }
    }
  },
  "dockerOptions": "-p 1234:1234"
}
```

Example Exemplo: ajuste o nível de registro do corretor MQTT

O exemplo a seguir altera o nível de log do broker MQTT para`debug`. Você pode escolher entre os seguintes níveis de registro:

- `debug`
- `info`
- `notice`

- `warning`
- `error`
- `critical`
- `alert`
- `emergency`

O nível de registro padrão é`warning`.

```
{
  "emqxConfig": {
    "log": {
      "console": {
         "level": "debug"
      }
    }
  }
}
```

Example Exemplo: Ativar o painel do EMQX

O exemplo a seguir ativa o painel do EMQX para que você possa monitorar e gerenciar seu corretor. Se você estiver usando Linux, inclua o `dockerOptions` campo.

```
{
  "emqxConfig": {
    "dashboard": {
      "listeners": {
        "http": {
          "bind": 18083
        }
      }
    }
  },
  "dockerOptions": "-p 18083:18083"
}
```

1.0.0 - 1.2.2

Esse componente fornece os seguintes parâmetros de configuração que você pode personalizar ao implantar o componente.

`emqx`

(Opcional) A configuração do broker EMQX MQTT a ser usada. Você pode configurar um subconjunto de opções de configuração do EMQX neste componente.

Esse objeto contém as seguintes informações:

`listener.ssl.external`

(Opcional) A porta em que o corretor MQTT opera.

> **Note**
>
> Se você especificar uma porta diferente e usar o componente de ponte MQTT para retransmitir mensagens MQTT para outros corretores, deverá usar a ponte MQTT v2.1.0 ou posterior. Configure-o para usar a porta em que o agente MQTT opera. Se você especificar uma porta diferente e usar o componente detector de IP para gerenciar os endpoints do broker MQTT, deverá usar o detector de IP v2.1.0 ou posterior. Configure-o para relatar a porta em que o agente MQTT opera.

Padrão: `8883`

`listener.ssl.external.max_connections`

(Opcional) O número máximo de conexões simultâneas que o agente MQTT suporta.

Padrão: `1024000`

`listener.ssl.external.max_conn_rate`

(Opcional) O número máximo de novas conexões por segundo que o agente MQTT pode receber.

Padrão: `500`

`listener.ssl.external.rate_limit`

(Opcional) O limite de largura de banda para todas as conexões com o agente MQTT. Especifique a largura de banda e a duração dessa largura de banda separadas por uma vírgula (`,`) no seguinte formato:. `bandwidth,duration` Por exemplo, você pode especificar `50KB,5s` para limitar o agente MQTT a 50 kilobytes (KB) de dados a cada 5 segundos.

`listener.ssl.external.handshake_timeout`

(Opcional) A quantidade de tempo que o agente MQTT espera para concluir a autenticação de uma nova conexão.

Padrão: `15s`

`mqtt.max_packet_size`

(Opcional) O tamanho máximo de uma mensagem MQTT.

Padrão: `268435455` (256 MB menos 1)

`log.level`

(Opcional) O nível de log do broker MQTT. Escolha uma das seguintes opções:

- `debug`
- `info`
- `notice`
- `warning`
- `error`
- `critical`
- `alert`
- `emergency`

O nível de registro padrão é`warning`.

`requiresPrivilege`

(Opcional) Nos dispositivos principais do Linux, você pode especificar a execução do broker EMQX MQTT sem privilégios de root ou administrador. Se você definir essa opção como`false`, o usuário do sistema que executa esse componente deverá ser membro do `docker` grupo.

Padrão: `true`

`startupTimeoutSeconds`

(Opcional) O tempo máximo em segundos para o broker EMQX MQTT iniciar. O estado do componente muda para `BROKEN` se ele exceder esse tempo limite.

Padrão: `90`

`ipcTimeoutSeconds`

(Opcional) O tempo máximo em segundos para o componente esperar até que o núcleo do Greengrass responda às solicitações de comunicação entre processos (IPC). Aumente esse número se esse componente relatar erros de tempo limite ao verificar se um dispositivo cliente está autorizado.

Padrão: `5`

`crtLogLevel`

(Opcional) O nível de registro da biblioteca AWS Common Runtime (CRT).

O padrão é o nível de log (in) do broker EMQX MQTT. `log.level` `emqx`

`restartIdentifier`

(Opcional) Configure essa opção para reiniciar o agente EMQX MQTT. Quando esse valor de configuração é alterado, esse componente reinicia o agente MQTT. Você pode usar essa opção para forçar a desconexão dos dispositivos cliente.

`dockerOptions`

(Opcional) Configure essa opção somente em sistemas operacionais Linux para adicionar parâmetros à linha de comando do Docker. Por exemplo, para mapear portas adicionais, use o parâmetro `-p` Docker:

```
"-p 1883:1883"
```

`mergeConfigurationFiles`

(Opcional) Configure essa opção para adicionar ou substituir os padrões nos arquivos de configuração do EMQX especificados. Para obter informações sobre os arquivos de configuração e seus formatos, consulte Configuração na documentação do EMQX 4.0. Os valores que você especifica são anexados ao arquivo de configuração.

O exemplo a seguir atualiza o `etc/emqx.conf` arquivo.

```
"mergeConfigurationFiles": {
    "etc/emqx.conf": "broker.sys_interval=30s\nbroker.sys_heartbeat=10s"
```

```
},
```

Além dos arquivos de configuração suportados pelo EMQX, o Greengrass suporta um arquivo que configura o plug-in de autenticação do Greengrass para EMQX chamado. `etc/plugins/aws_greengrass_emqx_auth.conf` Há duas opções suportadas, `auth_mode` `use_greengrass_managed_certificates` e. Para usar outro provedor de autenticação, defina a `auth_mode` opção como uma das seguintes:

- `enabled`— (Padrão) Use o provedor de autenticação e autorização do Greengrass.
- `bypass_on_failure`— Use o provedor de autenticação do Greengrass e, em seguida, use qualquer provedor de autenticação restante na cadeia de provedores do EMQX se o Greengrass negar a autenticação ou a autorização.
- `bypass`— O provedor Greengrass está desativado. A autenticação e a autorização são então tratadas pela cadeia de fornecedores do EMQX.

Se `use_greengrass_managed_certificates` for`true`, essa opção indica que o Greengrass gerencia os certificados TLS do broker. Se`false`, isso indica que você fornece os certificados por meio de outra fonte.

O exemplo a seguir atualiza os padrões no arquivo de `etc/plugins/aws_greengrass_emqx_auth.conf` configuração.

```
"mergeConfigurationFiles": {
    "etc/plugins/aws_greengrass_emqx_auth.conf": "auth_mode=enabled\n
 use_greengrass_managed_certificates=true\n"
  },
```

> **Note**
>
> `aws.greengrass.clientdevices.mqtt.EMQX`permite que você configure opções sensíveis à segurança. Isso inclui configurações de TLS, autenticação e provedores de autorização. A configuração recomendada é a configuração padrão que usa a autenticação TLS mútua e o provedor Greengrass Client Device Auth.

`replaceConfigurationFiles`

(Opcional) Configure essa opção para substituir os arquivos de configuração do EMQX especificados. Os valores que você especifica substituem todo o arquivo de configuração

existente. Você não pode especificar o `etc/emqx.conf` arquivo nesta seção. Você deve usar `mergeConfigurationFile` para modificar`etc/emqx.conf`.

Example Exemplo: atualização da mesclagem de configurações

O exemplo de configuração a seguir especifica a operação do agente MQTT na porta 443.

```
{
  "emqx": {
    "listener.ssl.external": "443",
    "listener.ssl.external.max_connections": "1024000",
    "listener.ssl.external.max_conn_rate": "500",
    "listener.ssl.external.rate_limit": "50KB,5s",
    "listener.ssl.external.handshake_timeout": "15s",
    "log.level": "warning"
  },
  "requiresPrivilege": "true",
  "startupTimeoutSeconds": "90",
  "ipcTimeoutSeconds": "5"
}
```

Arquivo de log local

Esse componente usa o seguinte arquivo de log.

Linux

```
/greengrass/v2/logs/aws.greengrass.clientdevices.mqtt.EMQX.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.clientdevices.mqtt.EMQX.log
```

Para ver os registros desse componente

- Execute o comando a seguir no dispositivo principal para visualizar o arquivo de log desse componente em tempo real. Substitua */greengrass/v2* ou *C:\greengrass\v2* pelo caminho para a pasta AWS IoT Greengrass raiz.

Linux

```
sudo tail -f /greengrass/v2/logs/aws.greengrass.clientdevices.mqtt.EMQX.log
```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\aws.greengrass.clientdevices.mqtt.EMQX.log -
Tail 10 -Wait
```

Licenças

Nos sistemas operacionais Windows, esse software inclui código distribuído de acordo com os Termos de Licença de Software da Microsoft - Microsoft Visual Studio Community 2022. Ao baixar este software, você concorda com os termos de licença desse código.

Esse componente é lançado de acordo com o Contrato de Licença de Software Principal do Greengrass.

Changelog

A tabela a seguir descreve as alterações em cada versão do componente.

v2.x

Version (Versão)	Alterações
2.0.1	Versão atualizada para a versão 2.5.0 da autenticação do dispositivo cliente.
2.0.0	Essa versão do broker MQTT 5 (EMQX) espera parâmetros de configura ção diferentes da versão 1.x. Se você usar uma configuração não padrão para a versão 1.x, deverá atualizar a configuração do componente para 2.x. Para ter mais informações, consulte Configuração. Novos atributos • Atualiza o corretor MQTT para o EMQX 5.1.1.

Version (Versão)	Alterações
	• Permite alterações na configuração do agente sem reiniciar o componente. Atualizações • Adiciona um novo campo de `emqxConfig` configuração que substitui os campos de `replaceConfigurationFiles` configuração `emqxmergeConfigurationFiles` , e.

v1.x

Version (Versão)	Alterações
1.2.3	Correções de erros e melhorias • Corrige um problema em que os clientes não conseguiam interagir com o EMQX após a autenticação anterior, desconectando e reautenticando o cliente.
1.2.2	Versão atualizada para a versão 2.4.0 da autenticação do dispositivo cliente.
1.2.1	Correções de erros e melhorias • Corrige um problema em que o componente não inicializa no Windows se o Visual C++ Redistributable ainda não estiver presente. • Atualiza o EMQX para a versão 4.4.14.
1.2.0	Adiciona suporte para cadeias de certificados.
1.1.0	Novos atributos • Adiciona suporte às configurações do EMQX, incluindo opções de corretor e plug-ins. Correções de erros e melhorias • Atualiza o EMQX para a versão 4.4.9.

Version (Versão)	Alterações
1.0.1	Corrige um problema durante o handshake TLS que resulta na falha de conexão de alguns clientes MQTT.
1.0.0	Versão inicial.

Emissor de telemetria Nucleus

O componente emissor de telemetria do núcleo (`aws.greengrass.telemetry.NucleusEmitter`) reúne dados de telemetria de integridade do sistema e os publica continuamente em um tópico local e em um tópico do MQTT. AWS IoT Core Esse componente permite que você reúna a telemetria do sistema em tempo real em seus dispositivos principais do Greengrass. Para obter informações sobre o agente de telemetria do Greengrass que publica dados de telemetria do sistema na Amazon, consulte. EventBridge Colete dados de telemetria de integridade do sistema a partir dos dispositivos principais AWS IoT Greengrass

Por padrão, o componente emissor de telemetria nucleus publica dados de telemetria a cada 60 segundos no seguinte tópico local de publicação/assinatura.

```
$local/greengrass/telemetry
```

O componente emissor de telemetria nucleus não é publicado em um tópico do AWS IoT Core MQTT por padrão. Você pode configurar esse componente para publicar em um tópico do AWS IoT Core MQTT ao implantá-lo. O uso de um tópico do MQTT para publicar dados no Nuvem AWS está sujeito a AWS IoT Corepreços.

AWS IoT Greengrassfornece vários componentes da comunidade para ajudá-lo a analisar e visualizar dados de telemetria localmente em seu dispositivo principal usando o InfluxDB e o Grafana. Esses componentes usam dados de telemetria do componente emissor do núcleo. Para obter mais informações, consulte o README do componente editor do InfluxDB.

Tópicos

- Versões
- Tipo

- Sistema operacional
- Dependências
- Configuração
- Dados de saída
- Uso
- Arquivo de log local
- Changelog

Versões

Esse componente tem as seguintes versões:

- 1,0.x

Tipo

Este componente é um componente de plug-in (`aws.greengrass.plugin`). O núcleo do Greengrass executa esse componente na mesma Java Virtual Machine (JVM) do núcleo. O núcleo reinicia quando você altera a versão desse componente no dispositivo principal.

Esse componente usa o mesmo arquivo de log do núcleo do Greengrass. Para ter mais informações, consulte Monitore AWS IoT Greengrass os registros.

Para ter mais informações, consulte Tipos de componentes.

Sistema operacional

Esse componente pode ser instalado em dispositivos principais que executam os seguintes sistemas operacionais:

- Linux
- Windows

Dependências

Quando você implanta um componente, AWS IoT Greengrass também implanta versões compatíveis de suas dependências. Isso significa que você deve atender aos requisitos do componente e de

todas as suas dependências para implantá-lo com êxito. Esta seção lista as dependências das versões lançadas desse componente e as restrições de versão semântica que definem as versões dos componentes para cada dependência. Você também pode visualizar as dependências de cada versão do componente no AWS IoT Greengrassconsole. Na página de detalhes do componente, procure a lista de dependências.

1.0.8

A tabela a seguir lista as dependências da versão 1.0.8 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.4.0 <2.13.0	Rígido

1.0.7

A tabela a seguir lista as dependências da versão 1.0.7 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.4.0 <2.12.0	Rígido

1.0.6

A tabela a seguir lista as dependências da versão 1.0.6 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.4.0 <2.11.0	Rígido

1.0.5

A tabela a seguir lista as dependências da versão 1.0.5 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.4.0 <2.10.0	Rígido

1.0.4

A tabela a seguir lista as dependências da versão 1.0.4 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.4.0 <2.9.0	Rígido

1.0.3

A tabela a seguir lista as dependências da versão 1.0.3 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.4.0 <2.8.0	Rígido

1.0.2

A tabela a seguir lista as dependências da versão 1.0.2 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.4.0 <2.7.0	Rígido

1.0.1

A tabela a seguir lista as dependências da versão 1.0.1 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.4.0 <2.6.0	Rígido

1.0.0

A tabela a seguir lista as dependências da versão 1.0.0 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.4.0 <2.5.0	Rígido

Para obter mais informações sobre dependências de componentes, consulte a referência da receita do componente.

Configuração

Esse componente fornece os seguintes parâmetros de configuração que você pode personalizar ao implantar o componente.

`pubSubPublish`

(Opcional) Define se os dados de telemetria devem ser publicados no `$local/greengrass/telemetry` tópico. Os valores compatíveis são `true` e `false`.

Padrão: `true`

`mqttTopic`

(Opcional) O tópico do AWS IoT Core MQTT no qual esse componente publica dados de telemetria.

Defina esse valor para o tópico do AWS IoT Core MQTT no qual você deseja publicar dados de telemetria. Quando esse valor está vazio, o emissor do núcleo não publica dados de telemetria no. Nuvem AWS

> **Note**
>
> O uso de um tópico do MQTT para publicar dados no Nuvem AWS está sujeito a AWS IoT Corepreços.

Padrão: `""`

`telemetryPublishIntervalMs`

(Opcional) A quantidade de tempo (em milissegundos) entre o qual o componente publica dados de telemetria. Se você definir esse valor abaixo do valor mínimo suportado, o componente usará o valor mínimo em vez disso.

> **Note**
>
> Intervalos de publicação mais baixos resultam em maior uso da CPU em seu dispositivo principal. Recomendamos que você comece com o intervalo de publicação padrão e o ajuste com base no uso da CPU do seu dispositivo.

Mínimo: `500`

Padrão: `60000`

Example Exemplo: atualização da mesclagem de configurações

O exemplo a seguir mostra um exemplo de atualização de mesclagem de configurações que permite publicar dados de telemetria a cada 5 segundos no `$local/greengrass/telemetry` tópico e no `greengrass/myTelemetry` AWS IoT Core tópico MQTT.

```
{
  "pubSubPublish": "true",
  "mqttTopic": "greengrass/myTelemetry",
  "telemetryPublishIntervalMs": 5000
}
```

Dados de saída

Esse componente publica métricas de telemetria como uma matriz JSON no tópico a seguir.

Tópico local: `$local/greengrass/telemetry`

Opcionalmente, você também pode optar por publicar métricas de telemetria em um AWS IoT Core tópico do MQTT. Para obter mais informações sobre tópicos, consulte os tópicos do MQTT no Guia do AWS IoT Core desenvolvedor.

Example Exemplo de dados

```
[
  {
    "A": "Average",
    "N": "CpuUsage",
    "NS": "SystemMetrics",
    "TS": 1627597331445,
```

```
    "U": "Percent",
    "V": 26.21981271562346
  },
  {
    "A": "Count",
    "N": "TotalNumberOfFDs",
    "NS": "SystemMetrics",
    "TS": 1627597331445,
    "U": "Count",
    "V": 7316
  },
  {
    "A": "Count",
    "N": "SystemMemUsage",
    "NS": "SystemMetrics",
    "TS": 1627597331445,
    "U": "Megabytes",
    "V": 10098
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsStarting",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
    "U": "Count",
    "V": 0
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsInstalled",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
    "U": "Count",
    "V": 0
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsStateless",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
    "U": "Count",
    "V": 0
  },
  {
```

```
    "A": "Count",
    "N": "NumberOfComponentsStopping",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
    "U": "Count",
    "V": 0
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsBroken",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
    "U": "Count",
    "V": 0
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsRunning",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
    "U": "Count",
    "V": 7
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsErrored",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
    "U": "Count",
    "V": 0
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsNew",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
    "U": "Count",
    "V": 0
  },
  {
    "A": "Count",
    "N": "NumberOfComponentsFinished",
    "NS": "GreengrassComponents",
    "TS": 1627597331446,
```

```
    "U": "Count",
    "V": 2
  }
]
```

A matriz de saída contém uma lista de métricas que têm as seguintes propriedades:

`A`

O tipo de agregação da métrica.

Para a `CpuUsage` métrica, essa propriedade é definida como `Average` porque o valor publicado da métrica é o valor médio de uso da CPU desde o último evento de publicação.

Para todas as outras métricas, o emissor do núcleo não agrega o valor da métrica e essa propriedade é definida como. `Count`

`N`

O nome da métrica.

`NS`

O namespace métrico.

`TS`

A data e hora de quando os dados foram coletados.

`U`

A unidade do valor da métrica.

`V`

O valor da métrica do .

O emissor do núcleo publica as seguintes métricas:

Nome	Descrição	
Sistema		
`SystemMemUsage`	A quantidade de memória atualmente em uso por todos	

Nome	Descrição	
	os aplicativos no dispositivo de núcleo do Greengrass, incluindo o sistema operacional.	
`CpuUsage`	A quantidade de CPU atualmente em uso por todos os aplicativos no dispositivo de núcleo do Greengrass, incluindo o sistema operacional.	
`TotalNumberOfFDs`	O número de descritores de arquivo armazenados pelo sistema operacional do dispositivo de núcleo do Greengrass. Um descritor de arquivo identifica exclusivamente um arquivo aberto.	
Núcleo Greengrass		
`NumberOfComponentsRunning`	O número de componentes que estão sendo executados no dispositivo principal do Greengrass.	
`NumberOfComponentsErrored`	O número de componentes que estão em estado de erro no dispositivo principal do Greengrass.	
`NumberOfComponentsInstalled`	O número de componentes que estão instalados no dispositivo principal do Greengrass.	

Nome	Descrição
`NumberOfComponentsStarting`	O número de componentes que estão começando no dispositivo principal do Greengrass.
`NumberOfComponentsNew`	O número de componentes que são novos no dispositivo principal do Greengrass.
`NumberOfComponentsStopping`	O número de componentes que estão parando no dispositivo principal do Greengrass.
`NumberOfComponentsFinished`	O número de componentes finalizados no dispositivo principal do Greengrass.
`NumberOfComponentsBroken`	O número de componentes que estão quebrados no dispositivo principal do Greengrass.
`NumberOfComponentsStateless`	O número de componentes sem estado no dispositivo principal do Greengrass.

Uso

Para usar os dados de telemetria de integridade do sistema, você pode criar componentes personalizados que se inscrevem nos tópicos nos quais o emissor de núcleo publica os dados de telemetria e reagem a esses dados conforme necessário. Como o componente emissor de núcleo oferece a opção de publicar dados de telemetria em um tópico local, você pode se inscrever nesse tópico e usar os dados publicados para agir localmente em seu dispositivo principal. O dispositivo principal pode então reagir aos dados de telemetria mesmo quando tem conectividade limitada com a nuvem.

Por exemplo, você pode configurar um componente que escuta dados de telemetria no `$local/greengrass/telemetry` tópico e enviar os dados para o componente gerenciador de stream para transmitir seus dados para o. Nuvem AWS Para obter mais informações sobre a criação desse componente, consulte Publique/assine mensagens locais Crie componentes personalizados que usam o gerenciador de fluxo e.

Arquivo de log local

Esse componente usa o mesmo arquivo de log do componente do núcleo do Greengrass.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

Para ver os registros desse componente

- Execute o comando a seguir no dispositivo principal para visualizar o arquivo de log desse componente em tempo real. Substitua */greengrass/v2* ou *C:\greengrass\v2* pelo caminho para a pasta AWS IoT Greengrass raiz.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

Changelog

A tabela a seguir descreve as alterações em cada versão do componente.

Version (Versão)	Alterações
1.0.8	Versão atualizada para a versão 2.12.0 do Greengrass nucleus.
1.0.7	Versão atualizada para a versão 2.11.0 do Greengrass nucleus.
1.0.6	Versão atualizada para a versão 2.10.0 do Greengrass nucleus.
1.0.5	Versão atualizada para a versão 2.9.0 do Greengrass nucleus.
1.0.4	Versão atualizada para a versão 2.8.0 do Greengrass nucleus.
1.0.3	Versão atualizada para a versão 2.7.0 do Greengrass nucleus.
1.0.2	Versão atualizada para a versão 2.6.0 do Greengrass nucleus.
1.0.1	Versão atualizada para a versão 2.5.0 do Greengrass nucleus.
1.0.0	Versão inicial.

Fornecedor PKCS #11

O componente provedor PKCS #11 (`aws.greengrass.crypto.Pkcs11Provider`) permite que você configure o software AWS IoT Greengrass Core para usar um módulo de segurança de hardware (HSM) por meio da interface PKCS #11. Esse componente permite que você armazene com segurança arquivos de certificado e chave privada para que eles não sejam expostos ou duplicados no software. Para ter mais informações, consulte Integração de segurança de hardware.

Para provisionar um dispositivo principal do Greengrass que armazena seu certificado e chave privada em um HSM, você deve especificar esse componente como um plug-in de provisionamento ao instalar o software Core. AWS IoT Greengrass Para ter mais informações, consulte Instale o software AWS IoT Greengrass principal com provisionamento manual de recursos.

AWS IoT Greengrassfornece esse componente como arquivo JAR que você pode baixar para especificar como um plug-in de provisionamento durante a instalação. Você pode baixar a versão mais recente do arquivo JAR do componente na seguinte URL: https://d2s8p88vqu9w66.cloudfront.net/releases/Pkcs11Provider/aws.greengrass.crypto.Pkcs11Provider-latest.jar.

Tópicos

Versões

Esse componente tem as seguintes versões:

- 2.0.x

Tipo

Este componente é um componente de plug-in (`aws.greengrass.plugin`). O núcleo do Greengrass executa esse componente na mesma Java Virtual Machine (JVM) do núcleo. O núcleo reinicia quando você altera a versão desse componente no dispositivo principal.

Esse componente usa o mesmo arquivo de log do núcleo do Greengrass. Para ter mais informações, consulte Monitore AWS IoT Greengrass os registros.

Para ter mais informações, consulte Tipos de componentes.

Sistema operacional

Esse componente pode ser instalado somente nos dispositivos principais do Linux.

Requisitos

Esse componente tem os seguintes requisitos:

- Um módulo de segurança de hardware que suporta o esquema de assinatura PKCS #1 v1.5 e chaves RSA com tamanho de chave RSA-2048 (ou maior) ou chaves ECC.

> **Note**
>
> Para usar um módulo de segurança de hardware com chaves ECC, você deve usar o Greengrass nucleus v2.5.6 ou posterior.
> Para usar um módulo de segurança de hardware e um gerenciador secreto, você deve usar um módulo de segurança de hardware com chaves RSA.

- Uma biblioteca do provedor PKCS #11 que o software AWS IoT Greengrass Core pode carregar em tempo de execução (usando libdl) para invocar as funções do PKCS #11. A biblioteca do provedor PKCS #11 deve implementar as seguintes operações da API PKCS #11:
  - `C_Initialize`
  - `C_Finalize`
  - `C_GetSlotList`
  - `C_GetSlotInfo`
  - `C_GetTokenInfo`
  - `C_OpenSession`
  - `C_GetSessionInfo`
  - `C_CloseSession`
  - `C_Login`
  - `C_Logout`
  - `C_GetAttributeValue`
  - `C_FindObjectsInit`
  - `C_FindObjects`
  - `C_FindObjectsFinal`
  - `C_DecryptInit`
  - `C_Decrypt`
  - `C_DecryptUpdate`
  - `C_DecryptFinal`
  - `C_SignInit`
  - `C_Sign`
  - `C_SignUpdate`

  - `C_SignFinal`
  - `C_GetMechanismList`
  - `C_GetMechanismInfo`
  - `C_GetInfo`
  - `C_GetFunctionList`
- O módulo de hardware deve ser solucionado pelo rótulo do slot, conforme definido na especificação PKCS#11.
- Você deve armazenar a chave privada e o certificado no HSM no mesmo slot, e eles devem usar o mesmo rótulo de objeto e ID de objeto, se o HSM suportar IDs de objeto.
- O certificado e a chave privada devem ser resolvidos por rótulos de objetos.
- A chave privada deve ter as seguintes permissões:
  - `sign`
  - `decrypt`
- (Opcional) Para usar o componente gerenciador de segredos, você deve usar a versão 2.1.0 ou posterior, e a chave privada deve ter as seguintes permissões:
  - `unwrap`
  - `wrap`
- (Opcional) Se você estiver usando a biblioteca TPM2 e executando o núcleo do Greengrass como um serviço, deverá fornecer uma variável de ambiente com a localização da loja PKCS #11. O exemplo a seguir é um arquivo de serviço systemd com a variável de ambiente necessária:

```
[Unit]
Description=Greengrass Core
After=network.target

[Service]
Type=simple
PIDFile=/var/run/greengrass.pid
Environment=TPM2_PKCS11_STORE=/path/to/store/directory
RemainAfterExit=no
Restart=on-failure
RestartSec=10
ExecStart=/bin/sh /greengrass/v2/alts/current/distro/bin/loader

[Install]
```

```
WantedBy=multi-user.target
```

Dependências

Quando você implanta um componente, AWS IoT Greengrass também implanta versões compatíveis de suas dependências. Isso significa que você deve atender aos requisitos do componente e de todas as suas dependências para implantá-lo com êxito. Esta seção lista as dependências das versões lançadas desse componente e as restrições de versão semântica que definem as versões dos componentes para cada dependência. Você também pode visualizar as dependências de cada versão do componente no AWS IoT Greengrassconsole. Na página de detalhes do componente, procure a lista de dependências.

2.0.7

A tabela a seguir lista as dependências da versão 2.0.7 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2,5,3 <2,13,0	Flexível

2.0.6

A tabela a seguir lista as dependências da versão 2.0.6 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2,5,3 <2,12,0	Flexível

2.0.5

A tabela a seguir lista as dependências da versão 2.0.5 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.5.3 <2.11.0	Flexível

2.0.4

A tabela a seguir lista as dependências da versão 2.0.4 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.5.3 <2.10.0	Flexível

2.0.3

A tabela a seguir lista as dependências da versão 2.0.3 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2,5.3 <2,9,0	Flexível

2.0.2

A tabela a seguir lista as dependências da versão 2.0.2 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2,5.3 <2,8,0	Flexível

2.0.1

A tabela a seguir lista as dependências da versão 2.0.1 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2,5,3 <2,7,0	Flexível

2.0.0

A tabela a seguir lista as dependências da versão 2.0.0 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2,5,3 <2,6,0	Flexível

Para obter mais informações sobre dependências de componentes, consulte a referência da receita do componente.

Configuração

Esse componente fornece os seguintes parâmetros de configuração que você pode personalizar ao implantar o componente.

`name`

Um nome para a configuração PKCS #11.

`library`

O caminho absoluto do arquivo para a biblioteca da implementação do PKCS #11 que o software AWS IoT Greengrass Core pode carregar com libdl.

`slot`

O ID do slot que contém a chave privada e o certificado do dispositivo. Esse valor é diferente do índice do slot ou do rótulo do slot.

`userPin`

O PIN do usuário a ser usado para acessar o slot.

Example Exemplo: atualização da mesclagem de configurações

```
{
  "name": "softhsm_pkcs11",
  "library": "/usr/lib/softhsm/libsofthsm2.so",
  "slot": 1,
  "userPin": "1234"
}
```

Arquivo de log local

Esse componente usa o mesmo arquivo de log do componente do núcleo do Greengrass.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

Para ver os registros desse componente

- Execute o comando a seguir no dispositivo principal para visualizar o arquivo de log desse componente em tempo real. Substitua */greengrass/v2* ou *C:\greengrass\v2* pelo caminho para a pasta AWS IoT Greengrass raiz.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

Changelog

A tabela a seguir descreve as alterações em cada versão do componente.

Version (Versão)	Alterações
2.0.7	Versão atualizada para a versão 2.12.0 do Greengrass nucleus.
2.0.6	Versão atualizada para a versão 2.11.0 do Greengrass nucleus.
2.0.5	Versão atualizada para a versão 2.10.0 do Greengrass nucleus.
2.0.4	Versão atualizada para a versão 2.9.0 do Greengrass nucleus.
2.0.3	Versão atualizada para a versão 2.8.0 do Greengrass nucleus.

Version (Versão)	Alterações
2.0.2	Versão atualizada para a versão 2.7.0 do Greengrass nucleus.
2.0.1	Versão atualizada para a versão 2.6.0 do Greengrass nucleus.
2.0.0	Versão inicial.

Gerente secreto

O componente gerenciador de segredos (`aws.greengrass.SecretManager`) implanta segredos nos dispositivos AWS Secrets Manager principais do Greengrass. Use esse componente para usar com segurança credenciais, como senhas, em componentes personalizados em seus dispositivos principais do Greengrass. Para obter mais informações sobre o Secrets Manager, consulte O que é o AWS Secrets Manager? no Guia do usuário do AWS Secrets Manager .

Para acessar os segredos desse componente em seus componentes personalizados do Greengrass, use a operação GetSecretValue no. AWS IoT Device SDK Para obter mais informações, consulte Use o AWS IoT Device SDK para se comunicar com o núcleo do Greengrass, outros componentes e AWS IoT Core e Recuperar valores secretos.

Esse componente criptografa segredos no dispositivo principal para manter suas credenciais e senhas seguras até que você precise usá-las. Ele usa a chave privada do dispositivo principal para criptografar e descriptografar segredos.

Tópicos

- Versões
- Tipo
- Sistema operacional
- Requisitos
- Dependências
- Configuração
- Arquivo de log local
- Changelog

Versões

Esse componente tem as seguintes versões:

- 2.1.x
- 2.0.x

Tipo

Este componente é um componente de plug-in (`aws.greengrass.plugin`). O núcleo do Greengrass executa esse componente na mesma Java Virtual Machine (JVM) do núcleo. O núcleo reinicia quando você altera a versão desse componente no dispositivo principal.

Esse componente usa o mesmo arquivo de log do núcleo do Greengrass. Para ter mais informações, consulte Monitore AWS IoT Greengrass os registros.

Para ter mais informações, consulte Tipos de componentes.

Sistema operacional

Esse componente pode ser instalado em dispositivos principais que executam os seguintes sistemas operacionais:

- Linux
- Windows

Requisitos

Esse componente tem os seguintes requisitos:

- A função de dispositivo do Greengrass deve permitir a `secretsmanager:GetSecretValue` ação, conforme mostrado no exemplo de política do IAM a seguir.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "secretsmanager:GetSecretValue"
```

```
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:secretsmanager:region:123456789012:secret:MySecret"
      ]
    }
  ]
}
```

> **Note**
>
> Se você usar uma AWS Key Management Service chave gerenciada pelo cliente para criptografar segredos, a função do dispositivo também deverá permitir a ação. `kms:Decrypt`

Para obter mais informações sobre as políticas do IAM para o Secrets Manager, consulte o seguinte no Guia AWS Secrets Manager do usuário:

- Autenticação e controle de acesso para AWS Secrets Manager
- Ações, recursos e chaves de contexto que você pode usar em uma política do IAM ou política secreta para AWS Secrets Manager

- Os componentes personalizados devem definir uma política de autorização que permita `aws.greengrass#GetSecretValue` obter segredos que você armazena com esse componente. Nessa política de autorização, você pode restringir o acesso dos componentes a segredos específicos. Para obter mais informações, consulte Autorização IPC do gerenciador secreto.
- (Opcional) Se você armazenar a chave privada e o certificado do dispositivo principal em um módulo de segurança de hardware (HSM), o HSM deve suportar chaves RSA, a chave privada deve ter a `unwrap` permissão e a chave pública deve ter a permissão. `wrap`

Endpoints e portas

Esse componente deve ser capaz de realizar solicitações de saída para os seguintes endpoints e portas, além dos endpoints e portas necessários para a operação básica. Para ter mais informações, consulte Permitir o tráfego de dispositivos por meio de um proxy ou firewall.

Endpoint	Porta	Obrigatório	Descrição
`secretsmanager. region.amazonaws.com`	443	Sim	Baixe segredos para o dispositivo principal.

Dependências

Quando você implanta um componente, AWS IoT Greengrass também implanta versões compatíveis de suas dependências. Isso significa que você deve atender aos requisitos do componente e de todas as suas dependências para implantá-lo com êxito. Esta seção lista as dependências das versões lançadas desse componente e as restrições de versão semântica que definem as versões do componente para cada dependência. Você também pode visualizar as dependências de cada versão do componente no AWS IoT Greengrass console. Na página de detalhes do componente, procure a lista de dependências.

2.1.7 – 2.1.8

A tabela a seguir lista as dependências das versões 2.1.7 e 2.1.8 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2,5.0 <2,13.0	Flexível

2.1.6

A tabela a seguir lista as dependências da versão 2.1.6 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2,5.0 <2,12.0	Flexível

2.1.5

A tabela a seguir lista as dependências da versão 2.1.5 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2,5.0 <2,11.0	Flexível

2.1.4

A tabela a seguir lista as dependências da versão 2.1.4 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.5.0 <2.10.0	Flexível

2.1.3

A tabela a seguir lista as dependências da versão 2.1.3 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2,5.0 <2,9,0	Flexível

2.1.2

A tabela a seguir lista as dependências da versão 2.1.2 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2,5.0 <2,8,0	Flexível

2.1.1

A tabela a seguir lista as dependências da versão 2.1.1 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2,5.0 <2,7,0	Flexível

2.1.0

A tabela a seguir lista as dependências da versão 2.1.0 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2,5.0 <2,6,0	Flexível

2.0.9

A tabela a seguir lista as dependências da versão 2.0.9 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.5.0	Flexível

2.0.8

A tabela a seguir lista as dependências da versão 2.0.8 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.4.0	Flexível

2.0.7

A tabela a seguir lista as dependências da versão 2.0.7 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.3.0	Flexível

2.0.6

A tabela a seguir lista as dependências da versão 2.0.6 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.2.0	Flexível

2.0.4 and 2.0.5

A tabela a seguir lista as dependências das versões 2.0.4 e 2.0.5 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.3 <2.1.0	Flexível

Para obter mais informações sobre dependências de componentes, consulte a referência da receita do componente.

Configuração

Esse componente fornece os seguintes parâmetros de configuração que você pode personalizar ao implantar o componente.

`cloudSecrets`

Uma lista de segredos do Secrets Manager para implantar no dispositivo principal. Você pode especificar rótulos para definir quais versões de cada segredo devem ser implantadas. Se você não especificar uma versão, esse componente implanta a versão com o rótulo `AWSCURRENT` de teste anexado. Para obter mais informações, consulte Etiquetas de preparação no Guia do AWS Secrets Manager usuário.

O componente gerenciador de segredos armazena os segredos em cache localmente. Se o valor secreto mudar no Secrets Manager, esse componente não recuperará automaticamente o novo valor. Para atualizar a cópia local, atribua um novo rótulo ao segredo e configure esse componente para recuperar o segredo identificado pelo novo rótulo.

Cada objeto contém as seguintes informações:

`arn`

O ARN do segredo a ser implantado. O ARN do segredo pode ser um ARN completo ou parcial. Recomendamos que você especifique um ARN completo em vez de um ARN parcial. Para obter mais informações, consulte Encontrando um segredo em um ARN parcial. Veja a seguir um exemplo de um ARN completo e um ARN parcial:

- ARN completo: `arn:aws:secretsmanager:us-east-2:111122223333:secret:SecretName-abcdef`
- ARN parcial: `arn:aws:secretsmanager:us-east-2:111122223333:secret:SecretName`

`labels`

(Opcional) Uma lista de rótulos para identificar as versões do segredo a serem implantadas no dispositivo principal.

Cada rótulo deve ser uma sequência de caracteres.

Example Exemplo: atualização da mesclagem de configurações

```
{
  "cloudSecrets": [
    {
      "arn": "arn:aws:secretsmanager:us-west-2:123456789012:secret:MyGreengrassSecret-
abcdef"
    }
  ]
}
```

Arquivo de log local

Esse componente usa o mesmo arquivo de log do componente do núcleo do Greengrass.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

Para ver os registros desse componente

- Execute o comando a seguir no dispositivo principal para visualizar o arquivo de log desse componente em tempo real. Substitua */greengrass/v2* ou *C:\greengrass\v2* pelo caminho para a pasta AWS IoT Greengrass raiz.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

Changelog

A tabela a seguir descreve as alterações em cada versão do componente.

Version (Versão)	Alterações
2.1.8	Correções de erros e melhorias Corrige um problema em que o gerente secreto não aceita um pagamento parcial.
2.1.7	Versão atualizada para a versão 2.12.0 do Greengrass nucleus.
2.1.6	Versão atualizada para a versão 2.11.0 do Greengrass nucleus.
2.1.5	Versão atualizada para a versão 2.10.0 do Greengrass nucleus.
2.1.4	Correções de erros e melhorias Corrige um problema em que segredos em cache eram removidos quando o gerenciador de segredos era implantado e o núcleo do Greengrass era reiniciado. Versão atualizada para a versão 2.9.0 do Greengrass nucleus.

Version (Versão)	Alterações
2.1.3	Versão atualizada para a versão 2.8.0 do Greengrass nucleus.
2.1.2	Versão atualizada para a versão 2.7.0 do Greengrass nucleus.
2.1.1	Versão atualizada para a versão 2.6.0 do Greengrass nucleus.
2.1.0	Novos atributos • Adiciona suporte para integração de segurança de hardware. O componente gerenciador de segredos pode criptografar e descripto grafar segredos usando uma chave privada que você armazena em um módulo de segurança de hardware (HSM). Para ter mais informações, consulte Integração de segurança de hardware. Correções de erros e melhorias • Versão atualizada para a versão 2.5.0 do Greengrass nucleus.
2.0.9	Versão atualizada para a versão 2.4.0 do Greengrass nucleus.
2.0.8	Versão atualizada para a versão 2.3.0 do Greengrass nucleus.
2.0.7	Versão atualizada para a versão 2.2.0 do Greengrass nucleus.
2.0.6	Versão atualizada para a versão 2.1.0 do Greengrass nucleus.
2.0.5	Melhorias • Adicione suporte para regiões e AWS GovCloud (US) regiões AWS da China.
2.0.4	Versão inicial.

Tunelamento seguro

Com o `aws.greengrass.SecureTunneling` componente, você pode estabelecer uma comunicação bidirecional segura com um dispositivo central do Greengrass localizado atrás de firewalls restritos.

Por exemplo, imagine que você tenha um dispositivo central do Greengrass protegido por um firewall que proíba todas as conexões de entrada. O tunelamento seguro usa o MQTT para transferir um token de acesso ao dispositivo e, em seguida, usa WebSockets para fazer uma conexão SSH com o dispositivo por meio do firewall. Com esse túnel AWS IoT gerenciado, você pode abrir a conexão SSH necessária para o seu dispositivo. Para obter mais informações sobre como usar o tunelamento AWS IoT seguro para se conectar a dispositivos remotos, consulte tunelamento AWS IoT seguro no Guia do desenvolvedor.AWS IoT

Esse componente se inscreve no agente de mensagens AWS IoT Core MQTT sobre o `$aws/things/greengrass-core-device/tunnels/notify` tópico para receber notificações de tunelamento seguro.

Tópicos

Versões

Esse componente tem as seguintes versões:

- 1,0.x

Tipo

Este componente é um componente genérico (`aws.greengrass.generic`). O núcleo do Greengrass executa os scripts do ciclo de vida do componente.

Para ter mais informações, consulte Tipos de componentes.

Sistema operacional

Esse componente pode ser instalado somente nos dispositivos principais do Linux.

Arquiteturas:

- Armv 71
- Armv8 (AArch64)
- x86_64

Requisitos

Esse componente tem os seguintes requisitos:

- Mínimo de 32 MB de espaço em disco disponível para o componente de tunelamento seguro. Esse requisito não inclui o software principal do Greengrass ou outros componentes executados no mesmo dispositivo.
- Mínimo de 16 MB de RAM disponível para o componente de tunelamento seguro. Esse requisito não inclui o software principal do Greengrass ou outros componentes executados no mesmo dispositivo. Para ter mais informações, consulte Controle a alocação de memória com opções de JVM.
- A versão 2.25 ou superior da Biblioteca GNU C (glibc) com um kernel Linux 3.2 ou superior é necessária para o componente de tunelamento seguro versão 1.0.12 e superior. As versões do sistema operacional e das bibliotecas após a data de fim da vida útil do suporte de longo prazo não são suportadas. Você deve usar um sistema operacional e bibliotecas com suporte de longo prazo.
- Tanto o sistema operacional quanto o Java Runtime devem ser instalados em 64 bits.
- Python 3.5 ou posterior instalado no dispositivo principal do Greengrass e adicionado à variável de ambiente PATH.
- `libcrypto.so.1.1`instalado no dispositivo principal do Greengrass e adicionado à variável de ambiente PATH.
- Abra o tráfego de saída na porta 443 no dispositivo principal do Greengrass.
- Ative o suporte para o serviço de comunicação que você deseja usar para se comunicar com o dispositivo principal do Greengrass. Por exemplo, para abrir uma conexão SSH com o dispositivo, você deve ativar o SSH nesse dispositivo.

Endpoints e portas

Esse componente deve ser capaz de realizar solicitações de saída para os seguintes endpoints e portas, além dos endpoints e portas necessários para a operação básica. Para ter mais informações, consulte Permitir o tráfego de dispositivos por meio de um proxy ou firewall.

Endpoint	Porta	Obrigatório	Descrição
`data.tunneling.iot. region.amazonaws.com`	443	Sim	Estabeleça túneis seguros.

Dependências

Quando você implanta um componente, AWS IoT Greengrass também implanta versões compatíveis de suas dependências. Isso significa que você deve atender aos requisitos do componente e de todas as suas dependências para implantá-lo com êxito. Esta seção lista as dependências das versões lançadas desse componente e as restrições de versão semântica que definem as versões dos componentes para cada dependência. Você também pode visualizar as dependências de cada versão do componente no AWS IoT Greengrass console. Na página de detalhes do componente, procure a lista de dependências.

1.0.19

A tabela a seguir lista as dependências da versão 1.0.19 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <3.0.0	Flexível

1.0.18

A tabela a seguir lista as dependências da versão 1.0.18 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.13.0	Flexível

1.0.16 – 1.0.17

A tabela a seguir lista as dependências das versões 1.0.16 a 1.0.17 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.12.0	Flexível

1.0.14 – 1.0.15

A tabela a seguir lista as dependências das versões 1.0.14 a 1.0.15 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.11.0	Flexível

1.0.11 – 1.0.13

A tabela a seguir lista as dependências das versões 1.0.11 a 1.0.13 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.10.0	Flexível

1.0.10

A tabela a seguir lista as dependências da versão 1.0.10 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.9.0	Flexível

1.0.9

A tabela a seguir lista as dependências da versão 1.0.9 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.8.0	Flexível

1.0.8

A tabela a seguir lista as dependências da versão 1.0.8 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.7.0	Flexível

1.0.5 - 1.0.7

A tabela a seguir lista as dependências das versões 1.0.5 a 1.0.7 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.6.0	Flexível

1.0.4

A tabela a seguir lista as dependências da versão 1.0.4 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.5.0	Flexível

1.0.3

A tabela a seguir lista as dependências da versão 1.0.3 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.4.0	Flexível

1.0.2

A tabela a seguir lista as dependências da versão 1.0.2 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.3.0	Flexível

1.0.1

A tabela a seguir lista as dependências da versão 1.0.1 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.2.0	Flexível

1.0.0

A tabela a seguir lista as dependências da versão 1.0.0 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.3 <2.1.0	Flexível

Para obter mais informações sobre dependências de componentes, consulte a referência da receita do componente.

Configuração

Esse componente fornece os seguintes parâmetros de configuração que você pode personalizar ao implantar o componente.

`OS_DIST_INFO`

(Opcional) O sistema operacional do seu dispositivo principal. Por padrão, o componente tenta identificar automaticamente o sistema operacional em execução no seu dispositivo principal. Se o componente não iniciar com o valor padrão, use esse valor para especificar o sistema

operacional. Para obter uma lista dos sistemas operacionais compatíveis com esse componente, consulteRequisitos do dispositivo.

Esse valor pode ser um dos seguintes:`auto`,`ubuntu`,`amzn2`,`raspberrypi`.

Padrão: `auto`

`accessControl`

(Opcional) O objeto que contém a política de autorização que permite que o componente assine o tópico de notificações de tunelamento seguro.

> **Note**
>
> Não modifique esse parâmetro de configuração se sua implantação for direcionada a um grupo de coisas. Se sua implantação for direcionada a um dispositivo principal individual e você quiser restringir sua assinatura ao tópico do dispositivo, especifique o nome do dispositivo principal. No `resources` valor da política de autorização do dispositivo, substitua o caractere curinga do tópico MQTT pelo nome do item do dispositivo.

```
{
  "aws.greengrass.ipc.mqttproxy": {
    "aws.iot.SecureTunneling:mqttproxy:1": {
      "policyDescription": "Access to tunnel notification pubsub topic",
      "operations": [
        "aws.greengrass#SubscribeToIoTCore"
      ],
      "resources": [
        "$aws/things/+/tunnels/notify"
      ]
    }
  }
}
```

Example Exemplo: atualização da mesclagem de configurações

O exemplo de configuração a seguir especifica para permitir que esse componente abra túneis seguros em um dispositivo central chamado **`MyGreengrassCore`** que executa o Ubuntu.

```
{
```

```
  "OS_DIST_INFO": "ubuntu",
  "accessControl": {
    "aws.greengrass.ipc.mqttproxy": {
      "aws.iot.SecureTunneling:mqttproxy:1": {
        "policyDescription": "Access to tunnel notification pubsub topic",
        "operations": [
          "aws.greengrass#SubscribeToIoTCore"
        ],
        "resources": [
          "$aws/things/MyGreengrassCore/tunnels/notify"
        ]
      }
    }
  }
}
```

Arquivo de log local

Esse componente usa o seguinte arquivo de log.

```
/greengrass/v2/logs/aws.greengrass.SecureTunneling.log
```

Para visualizar os registros desse componente

- Execute o comando a seguir no dispositivo principal para visualizar o arquivo de log desse componente em tempo real. */greengrass/v2*Substitua pelo caminho para a pasta AWS IoT Greengrass raiz.

  ```
  sudo tail -f /greengrass/v2/logs/aws.greengrass.SecureTunneling.log
  ```

Licenças

Esse componente inclui o seguinte software/licenciamento de terceiros:

- AWS IoT Cliente de dispositivo/Licença Apache 2.0
- AWS IoT Device SDK for Java/Licença Apache 2.0
- Licença gson/Apache 2.0
- log4j /Licença Apache 2.0
- slf4j /Licença Apache 2.0

Uso

Para usar o componente de tunelamento seguro em seu dispositivo, faça o seguinte:

1. Implante o componente de tunelamento seguro em seu dispositivo.
2. Abra o console de AWS IoT. No menu à esquerda, escolha Ações remotas e, em seguida, escolha Túneis seguros.
3. Crie um túnel para o seu dispositivo Greengrass.
4. Faça o download do token de acesso à fonte.
5. Use o proxy local com o token de acesso de origem para se conectar ao seu destino. Para obter mais informações, consulte Como usar o proxy local no Guia do AWS IoT desenvolvedor.

Consulte também

- AWS IoT tunelamento seguro no Guia do desenvolvedor AWS IoT
- Como usar o proxy local no Guia do AWS IoT desenvolvedor

Changelog

A tabela a seguir descreve as alterações em cada versão do componente.

Version (Versão)	Alterações
1.0.19	Correções de erros e melhorias • Atualiza o AWS IoT Device Client subjacente invocado pelo componente da versão 1.8.0 para a versão 1.9.0. • Aumenta o limite de túneis simultâneos para 20 túneis em um nível de componente. • Aumenta o tempo limite padrão do AWS IoT Greengrass Core IPC de 3 segundos para 10 segundos. ⚠ Warning Se você estiver usando o proxy local de tunelamento seguro como cliente de origem do túnel, não atualize seu componente para essa

Version (Versão)	Alterações
	versão até que você também tenha atualizado o proxy local para a versão 3.1.1 ou posterior.
1.0.18	Versão atualizada para a versão 2.12.0 do Greengrass nucleus.
1.0.17	Correções de erros e melhorias • Corrige o problema de limpeza de tópicos que estava impedindo os usuários de criar túneis. Agora, esse componente limpará uma linha quando receber o CloseTunnel sinal ou se o túnel expirar após 12 horas.
1.0.16	Versão atualizada para a versão 2.11.0 do Greengrass nucleus.
1.0.15	Correções de erros e melhorias • Corrige um problema de inicialização para usuários que não têm um diretório inicial no dispositivo. O componente de tunelamento seguro agora é iniciado sem criar um diretório para documentos paralelos.
1.0.14	Versão atualizada para a versão 2.10.0 do Greengrass nucleus.
1.0.13	Correções de erros e melhorias • Corrige um problema em que um processo de cliente órfão impede que mais de um túnel atinja o dispositivo.
1.0.12	Correções de erros e melhorias • Adiciona suporte para x86_64 (AMD64) e ARMv8 (Aarch64) quando executado no sistema operacional Raspberry Pi.
1.0.11	Versão atualizada para a versão 2.9.0 do Greengrass nucleus.
1.0.10	Versão atualizada para a versão 2.8.0 do Greengrass nucleus.
1.0.9	Versão atualizada para a versão 2.7.0 do Greengrass nucleus.
1.0.8	Versão atualizada para a versão 2.6.0 do Greengrass nucleus.

Version (Versão)	Alterações
1.0.7	Correções de erros e melhorias • Corrige um problema em que o componente se desconecta quando você transfere arquivos grandes pelo SCP.
1.0.6	Esta versão contém correções de erros.
1.0.5	Versão atualizada para a versão 2.5.0 do Greengrass nucleus.
1.0.4	Versão atualizada para a versão 2.4.0 do Greengrass nucleus.
1.0.3	Versão atualizada para a versão 2.3.0 do Greengrass nucleus.
1.0.2	Versão atualizada para a versão 2.2.0 do Greengrass nucleus.
1.0.1	Versão atualizada para a versão 2.1.0 do Greengrass nucleus.
1.0.0	Versão inicial.

Gerenciador de sombras

O componente do gerenciador de sombras (`aws.greengrass.ShadowManager`) ativa o serviço paralelo local em seu dispositivo principal. O serviço de sombra local permite que os componentes usem a comunicação entre processos para interagir com as sombras locais. O componente shadow manager gerencia o armazenamento de documentos paralelos locais e também gerencia a sincronização dos estados de sombra locais com o serviço AWS IoT Device Shadow.

Para obter mais informações sobre como os dispositivos principais do Greengrass podem interagir com sombras, consulte. Interaja com as sombras do dispositivo

Tópicos

- Versões
- Tipo
- Sistema operacional
- Requisitos
- Dependências

- Configuração
- Arquivo de log local
- Changelog

Versões

Esse componente tem as seguintes versões:

- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

Tipo

Este componente é um componente de plug-in (`aws.greengrass.plugin`). O núcleo do Greengrass executa esse componente na mesma Java Virtual Machine (JVM) do núcleo. O núcleo reinicia quando você altera a versão desse componente no dispositivo principal.

Esse componente usa o mesmo arquivo de log do núcleo do Greengrass. Para ter mais informações, consulte Monitore AWS IoT Greengrass os registros.

Para ter mais informações, consulte Tipos de componentes.

Sistema operacional

Esse componente pode ser instalado em dispositivos principais que executam os seguintes sistemas operacionais:

- Linux
- Windows

Requisitos

Esse componente tem os seguintes requisitos:

- (Opcional) Para sincronizar sombras com o serviço AWS IoT Device Shadow, a política do dispositivo principal AWS IoT do Greengrass deve permitir as AWS IoT Core seguintes ações de política paralela:
  - `iot:GetThingShadow`
  - `iot:UpdateThingShadow`
  - `iot:DeleteThingShadow`

  Para obter mais informações sobre essas AWS IoT Core políticas, consulte as ações AWS IoT Core políticas no Guia do AWS IoT desenvolvedor.

  Para obter mais informações sobre a AWS IoT política mínima, consulte AWS IoTPolítica mínima para dispositivos AWS IoT Greengrass V2 principais
- O componente do gerenciador de sombras tem suporte para execução em uma VPC.

Dependências

Quando você implanta um componente, AWS IoT Greengrass também implanta versões compatíveis de suas dependências. Isso significa que você deve atender aos requisitos do componente e de todas as suas dependências para implantá-lo com êxito. Esta seção lista as dependências das versões lançadas desse componente e as restrições de versão semântica que definem as versões dos componentes para cada dependência. Você também pode visualizar as dependências de cada versão do componente no AWS IoT Greengrass console. Na página de detalhes do componente, procure a lista de dependências.

2.3.5 – 2.3.8

A tabela a seguir lista as dependências das versões 2.3.5 a 2.3.8 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2,5.0 <2,13.0	Flexível

2.3.3 and 2.3.4

A tabela a seguir lista as dependências das versões 2.3.3 e 2.3.4 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2,5.0 <2,12.0	Flexível

2.3.2

A tabela a seguir lista as dependências da versão 2.3.2 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2,5.0 <2,11.0	Flexível

2.3.0 and 2.3.1

A tabela a seguir lista as dependências das versões 2.3.0 e 2.3.1 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.5.0 <2.10.0	Flexível

2.2.3 and 2.2.4

A tabela a seguir lista as dependências das versões 2.2.3 e 2.2.4 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.2.0 <3.0.0	Flexível

2.2.2

A tabela a seguir lista as dependências da versão 2.2.2 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.2.0 <2.9.0	Flexível

2.2.1

A tabela a seguir lista as dependências da versão 2.2.1 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.2.0 <2.8.0	Flexível

2.1.1 and 2.2.0

A tabela a seguir lista as dependências das versões 2.1.1 e 2.2.0 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.2.0 <2.7.0	Flexível

2.0.5 - 2.1.0

A tabela a seguir lista as dependências das versões 2.0.5 a 2.1.0 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.2.0 <2.6.0	Flexível

2.0.3 and 2.0.4

A tabela a seguir lista as dependências das versões 2.0.3 e 2.0.4 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.2.0 <2.5.0	Flexível

2.0.1 and 2.0.2

A tabela a seguir lista as dependências das versões 2.0.1 e 2.0.2 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.2.0 <2.4.0	Flexível

2.0.0

A tabela a seguir lista as dependências da versão 2.0.0 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.2.0 <2.3.0	Flexível

Para obter mais informações sobre dependências de componentes, consulte a referência da receita do componente.

Configuração

Esse componente fornece os seguintes parâmetros de configuração que você pode personalizar ao implantar o componente.

2.3.x

`strategy`

(Opcional) A estratégia que esse componente usa para sincronizar sombras entre AWS IoT Core e o dispositivo principal.

Esse objeto contém as seguintes informações.

`type`

(Opcional) O tipo de estratégia que esse componente usa para sincronizar sombras entre AWS IoT Core e o dispositivo principal. Escolha uma das seguintes opções:

- `realTime`— Sincronize sombras AWS IoT Core sempre que ocorrer uma atualização de sombras.
- `periodic`— Sincronize sombras AWS IoT Core em um intervalo regular que você especifica com o parâmetro `delay` de configuração.

Padrão: `realTime`

`delay`

(Opcional) O intervalo em segundos com AWS IoT Core o qual esse componente sincroniza sombras quando você especifica a estratégia de `periodic` sincronização.

> **Note**
>
> Esse parâmetro é necessário se você especificar a estratégia de `periodic` sincronização.

`synchronize`

(Opcional) As configurações de sincronização que determinam como as sombras são sincronizadas com o. Nuvem AWS

> **Note**
>
> Você deve criar uma atualização de configuração com essa propriedade para sincronizar sombras com o. Nuvem AWS

Esse objeto contém as seguintes informações.

`coreThing`

(Opcional) O dispositivo principal faz sombras para sincronizar. Esse objeto contém as seguintes informações.

`classic`

(Opcional) Por padrão, o gerenciador de sombras sincroniza o estado local da sombra clássica do seu dispositivo principal com o. Nuvem AWS Se você não quiser sincronizar a sombra clássica do dispositivo, defina-a como`false`.

Padrão: `true`

`namedShadows`

(Opcional) A lista de sombras nomeadas do dispositivo principal a serem sincronizadas. Você deve especificar os nomes exatos das sombras.

> **Warning**
>
> O AWS IoT Greengrass serviço usa a sombra `AWSManagedGreengrassV2Deployment` nomeada para gerenciar implantações direcionadas a dispositivos principais individuais. Essa sombra nomeada é reservada para uso pelo AWS IoT Greengrass serviço. Não atualize nem exclua essa sombra nomeada.

`shadowDocumentsMap`

(Opcional) As sombras adicionais do dispositivo a serem sincronizadas. O uso desse parâmetro de configuração facilita a especificação de documentos de sombra. Recomendamos que você use esse parâmetro em vez do `shadowDocuments` objeto.

> **Note**
>
> Se você especificar um `shadowDocumentsMap` objeto, não deverá especificar um `shadowDocuments` objeto.

Cada objeto contém as seguintes informações:

`thingName`

A configuração de sombra do *`ThingName`* para essa configuração de sombra.

`classic`

(Opcional) Se você não quiser sincronizar a sombra clássica do dispositivo com o `thingName` dispositivo, defina-a como`false`.

`namedShadows`

A lista de sombras nomeadas que você deseja sincronizar. Você deve especificar os nomes exatos das sombras.

`shadowDocuments`

(Opcional) A lista de sombras adicionais do dispositivo a serem sincronizadas. Recomendamos que você use o `shadowDocumentsMap` parâmetro em vez disso.

> **Note**
>
> Se você especificar um `shadowDocuments` objeto, não deverá especificar um `shadowDocumentsMap` objeto.

Cada objeto nessa lista contém as seguintes informações.

`thingName`

O nome do dispositivo para o qual sincronizar sombras.

`classic`

(Opcional) Se você não quiser sincronizar a sombra clássica do dispositivo com o `thingName` dispositivo, defina-a como`false`.

Padrão: `true`

`namedShadows`

(Opcional) A lista de sombras de dispositivos nomeados que você deseja sincronizar. Você deve especificar os nomes exatos das sombras.

`direction`

(Opcional) A direção para sincronizar sombras entre o serviço de sombra local e o. Nuvem AWS Você pode configurar essa opção para reduzir a largura de banda e as conexões com o. Nuvem AWS Escolha uma das seguintes opções:

- `betweenDeviceAndCloud`— Sincronize sombras entre o serviço de sombra local e o. Nuvem AWS
- `deviceToCloud`— Envie atualizações paralelas do serviço paralelo local para o. Nuvem AWS e ignore as atualizações paralelas do Nuvem AWS.
- `cloudToDevice`— Receba atualizações Nuvem AWS paralelas do e não envie atualizações paralelas do serviço paralelo local para Nuvem AWS o.

Padrão: `BETWEEN_DEVICE_AND_CLOUD`

`rateLimits`

(Opcional) As configurações que determinam os limites de taxa para solicitações de serviço paralelo.

Esse objeto contém as seguintes informações.

`maxOutboundSyncUpdatesPerSecond`

(Opcional) O número máximo de solicitações de sincronização por segundo que o dispositivo transmite.

Padrão: 100 solicitações/segundo

`maxTotalLocalRequestsRate`

(Opcional) O número máximo de solicitações IPC locais por segundo enviadas ao dispositivo principal.

Padrão: 200 solicitações/segundo

`maxLocalRequestsPerSecondPerThing`

(Opcional) O número máximo de solicitações locais de IPC por segundo enviadas para cada dispositivo de IoT conectado.

Padrão: 20 solicitações/segundo para cada item

> **Note**
>
> Esses parâmetros de limite de taxa definem o número máximo de solicitações por segundo para o serviço paralelo local. O número máximo de solicitações por segundo para o serviço AWS IoT Device Shadow depende do seu Região da AWS. Para obter mais informações, consulte os limites da API AWS IoT Device Shadow Service no Referência geral da Amazon Web Services.

`shadowDocumentSizeLimitBytes`

(Opcional) O tamanho máximo permitido de cada documento de estado JSON para sombras locais.

Se você aumentar esse valor, também deverá aumentar o limite de recursos do documento de estado JSON para sombras na nuvem. Para obter mais informações, consulte os limites da API AWS IoT Device Shadow Service no Referência geral da Amazon Web Services.

Padrão: 8192 bytes

Máximo: 30720 bytes

Example Exemplo: atualização da mesclagem de configurações

O exemplo a seguir mostra um exemplo de atualização de mesclagem de configurações com todos os parâmetros de configuração disponíveis para o componente shadow manager.

```
{
   "strategy":{
      "type":"periodic",
      "delay":300
   },
   "synchronize":{
      "shadowDocumentsMap":{
         "MyDevice1":{
            "classic":false,
            "namedShadows":[
               "MyShadowA",
               "MyShadowB"
            ]
         },
         "MyDevice2":{
            "classic":true,
            "namedShadows":[]
         }
      },
      "direction":"betweenDeviceAndCloud"
   },
   "rateLimits":{
      "maxOutboundSyncUpdatesPerSecond":100,
      "maxTotalLocalRequestsRate":200,
      "maxLocalRequestsPerSecondPerThing":20
   },
   "shadowDocumentSizeLimitBytes":8192
}
```

2.2.x

`strategy`

(Opcional) A estratégia que esse componente usa para sincronizar sombras entre AWS IoT Core e o dispositivo principal.

Esse objeto contém as seguintes informações.

`type`

(Opcional) O tipo de estratégia que esse componente usa para sincronizar sombras entre AWS IoT Core e o dispositivo principal. Escolha uma das seguintes opções:

- `realTime`— Sincronize sombras AWS IoT Core sempre que ocorrer uma atualização de sombras.
- `periodic`— Sincronize sombras AWS IoT Core em um intervalo regular que você especifica com o parâmetro `delay` de configuração.

Padrão: `realTime`

`delay`

(Opcional) O intervalo em segundos com AWS IoT Core o qual esse componente sincroniza sombras quando você especifica a estratégia de `periodic` sincronização.

> **Note**
>
> Esse parâmetro é necessário se você especificar a estratégia de `periodic` sincronização.

`synchronize`

(Opcional) As configurações de sincronização que determinam como as sombras são sincronizadas com o. Nuvem AWS

> **Note**
>
> Você deve criar uma atualização de configuração com essa propriedade para sincronizar sombras com o. Nuvem AWS

Esse objeto contém as seguintes informações.

`coreThing`

(Opcional) O dispositivo principal faz sombras para sincronizar. Esse objeto contém as seguintes informações.

`classic`

(Opcional) Por padrão, o gerenciador de sombras sincroniza o estado local da sombra clássica do seu dispositivo principal com o. Nuvem AWS Se você não quiser sincronizar a sombra clássica do dispositivo, defina-a como`false`.

Padrão: `true`

`namedShadows`

(Opcional) A lista de sombras nomeadas do dispositivo principal a serem sincronizadas. Você deve especificar os nomes exatos das sombras.

> **Warning**
>
> O AWS IoT Greengrass serviço usa a sombra `AWSManagedGreengrassV2Deployment` nomeada para gerenciar implantações direcionadas a dispositivos principais individuais. Essa sombra nomeada é reservada para uso pelo AWS IoT Greengrass serviço. Não atualize nem exclua essa sombra nomeada.

`shadowDocumentsMap`

(Opcional) As sombras adicionais do dispositivo a serem sincronizadas. O uso desse parâmetro de configuração facilita a especificação de documentos de sombra. Recomendamos que você use esse parâmetro em vez do `shadowDocuments` objeto.

> **Note**
>
> Se você especificar um `shadowDocumentsMap` objeto, não deverá especificar um `shadowDocuments` objeto.

Cada objeto contém as seguintes informações:

`thingName`

A configuração de sombra do *`ThingName`* para essa configuração de sombra.

`classic`

(Opcional) Se você não quiser sincronizar a sombra clássica do dispositivo com o `thingName` dispositivo, defina-a como`false`.

`namedShadows`

A lista de sombras nomeadas que você deseja sincronizar. Você deve especificar os nomes exatos das sombras.

`shadowDocuments`

(Opcional) A lista de sombras adicionais do dispositivo a serem sincronizadas. Recomendamos que você use o `shadowDocumentsMap` parâmetro em vez disso.

> **Note**
>
> Se você especificar um `shadowDocuments` objeto, não deverá especificar um `shadowDocumentsMap` objeto.

Cada objeto nessa lista contém as seguintes informações.

`thingName`

O nome do dispositivo para o qual sincronizar sombras.

`classic`

(Opcional) Se você não quiser sincronizar a sombra clássica do dispositivo com o `thingName` dispositivo, defina-a como`false`.

Padrão: `true`

`namedShadows`

(Opcional) A lista de sombras de dispositivos nomeados que você deseja sincronizar. Você deve especificar os nomes exatos das sombras.

`direction`

(Opcional) A direção para sincronizar sombras entre o serviço de sombra local e o. Nuvem AWS Você pode configurar essa opção para reduzir a largura de banda e as conexões com o. Nuvem AWS Escolha uma das seguintes opções:

- `betweenDeviceAndCloud`— Sincronize sombras entre o serviço de sombra local e o. Nuvem AWS
- `deviceToCloud`— Envie atualizações paralelas do serviço paralelo local para o. Nuvem AWS e ignore as atualizações paralelas do Nuvem AWS.
- `cloudToDevice`— Receba atualizações Nuvem AWS paralelas do e não envie atualizações paralelas do serviço paralelo local para Nuvem AWS o.

Padrão: `BETWEEN_DEVICE_AND_CLOUD`

`rateLimits`

(Opcional) As configurações que determinam os limites de taxa para solicitações de serviço paralelo.

Esse objeto contém as seguintes informações.

`maxOutboundSyncUpdatesPerSecond`

(Opcional) O número máximo de solicitações de sincronização por segundo que o dispositivo transmite.

Padrão: 100 solicitações/segundo

`maxTotalLocalRequestsRate`

(Opcional) O número máximo de solicitações IPC locais por segundo enviadas ao dispositivo principal.

Padrão: 200 solicitações/segundo

`maxLocalRequestsPerSecondPerThing`

(Opcional) O número máximo de solicitações locais de IPC por segundo enviadas para cada dispositivo de IoT conectado.

Padrão: 20 solicitações/segundo para cada item

> **Note**
>
> Esses parâmetros de limite de taxa definem o número máximo de solicitações por segundo para o serviço paralelo local. O número máximo de solicitações por segundo

para o serviço AWS IoT Device Shadow depende do seu Região da AWS. Para obter mais informações, consulte os limites da API AWS IoT Device Shadow Service no Referência geral da Amazon Web Services.

`shadowDocumentSizeLimitBytes`

(Opcional) O tamanho máximo permitido de cada documento de estado JSON para sombras locais.

Se você aumentar esse valor, também deverá aumentar o limite de recursos do documento de estado JSON para sombras na nuvem. Para obter mais informações, consulte os limites da API AWS IoT Device Shadow Service no Referência geral da Amazon Web Services.

Padrão: 8192 bytes

Máximo: 30720 bytes

Example Exemplo: atualização da mesclagem de configurações

O exemplo a seguir mostra um exemplo de atualização de mesclagem de configurações com todos os parâmetros de configuração disponíveis para o componente shadow manager.

```
{
   "strategy":{
      "type":"periodic",
      "delay":300
   },
   "synchronize":{
      "shadowDocumentsMap":{
         "MyDevice1":{
            "classic":false,
            "namedShadows":[
               "MyShadowA",
               "MyShadowB"
            ]
         },
         "MyDevice2":{
            "classic":true,
            "namedShadows":[]
         }
```

```
      },
      "direction":"betweenDeviceAndCloud"
   },
   "rateLimits":{
      "maxOutboundSyncUpdatesPerSecond":100,
      "maxTotalLocalRequestsRate":200,
      "maxLocalRequestsPerSecondPerThing":20
   },
   "shadowDocumentSizeLimitBytes":8192
}
```

2.1.x

`strategy`

(Opcional) A estratégia que esse componente usa para sincronizar sombras entre AWS IoT Core e o dispositivo principal.

Esse objeto contém as seguintes informações.

`type`

(Opcional) O tipo de estratégia que esse componente usa para sincronizar sombras entre AWS IoT Core e o dispositivo principal. Escolha uma das seguintes opções:

- `realTime`— Sincronize sombras AWS IoT Core sempre que ocorrer uma atualização de sombras.
- `periodic`— Sincronize sombras AWS IoT Core em um intervalo regular que você especifica com o parâmetro `delay` de configuração.

Padrão: `realTime`

`delay`

(Opcional) O intervalo em segundos com AWS IoT Core o qual esse componente sincroniza sombras quando você especifica a estratégia de `periodic` sincronização.

> **Note**
>
> Esse parâmetro é necessário se você especificar a estratégia de `periodic` sincronização.

`synchronize`

(Opcional) As configurações de sincronização que determinam como as sombras são sincronizadas com o. Nuvem AWS

> **Note**
>
> Você deve criar uma atualização de configuração com essa propriedade para sincronizar sombras com o. Nuvem AWS

Esse objeto contém as seguintes informações.

`coreThing`

(Opcional) O dispositivo principal faz sombras para sincronizar. Esse objeto contém as seguintes informações.

`classic`

(Opcional) Por padrão, o gerenciador de sombras sincroniza o estado local da sombra clássica do seu dispositivo principal com o. Nuvem AWS Se você não quiser sincronizar a sombra clássica do dispositivo, defina-a como`false`.

Padrão: `true`

`namedShadows`

(Opcional) A lista de sombras nomeadas do dispositivo principal a serem sincronizadas. Você deve especificar os nomes exatos das sombras.

> **Warning**
>
> O AWS IoT Greengrass serviço usa a sombra `AWSManagedGreengrassV2Deployment` nomeada para gerenciar implantações direcionadas a dispositivos principais individuais. Essa sombra nomeada é reservada para uso pelo AWS IoT Greengrass serviço. Não atualize nem exclua essa sombra nomeada.

`shadowDocumentsMap`

(Opcional) As sombras adicionais do dispositivo a serem sincronizadas. O uso desse parâmetro de configuração facilita a especificação de documentos de sombra. Recomendamos que você use esse parâmetro em vez do `shadowDocuments` objeto.

> **Note**
>
> Se você especificar um `shadowDocumentsMap` objeto, não deverá especificar um `shadowDocuments` objeto.

Cada objeto contém as seguintes informações:

`thingName`

A configuração de sombra do *`ThingName`* para essa configuração de sombra.

`classic`

(Opcional) Se você não quiser sincronizar a sombra clássica do dispositivo com o `thingName` dispositivo, defina-a como`false`.

`namedShadows`

A lista de sombras nomeadas que você deseja sincronizar. Você deve especificar os nomes exatos das sombras.

`shadowDocuments`

(Opcional) A lista de sombras adicionais do dispositivo a serem sincronizadas. Recomendamos que você use o `shadowDocumentsMap` parâmetro em vez disso.

> **Note**
>
> Se você especificar um `shadowDocuments` objeto, não deverá especificar um `shadowDocumentsMap` objeto.

Cada objeto nessa lista contém as seguintes informações.

`thingName`

O nome do dispositivo para o qual sincronizar sombras.

`classic`

(Opcional) Se você não quiser sincronizar a sombra clássica do dispositivo com o `thingName` dispositivo, defina-a como`false`.

Padrão: `true`

`namedShadows`

(Opcional) A lista de sombras de dispositivos nomeados que você deseja sincronizar. Você deve especificar os nomes exatos das sombras.

`rateLimits`

(Opcional) As configurações que determinam os limites de taxa para solicitações de serviço paralelo.

Esse objeto contém as seguintes informações.

`maxOutboundSyncUpdatesPerSecond`

(Opcional) O número máximo de solicitações de sincronização por segundo que o dispositivo transmite.

Padrão: 100 solicitações/segundo

`maxTotalLocalRequestsRate`

(Opcional) O número máximo de solicitações IPC locais por segundo enviadas ao dispositivo principal.

Padrão: 200 solicitações/segundo

`maxLocalRequestsPerSecondPerThing`

(Opcional) O número máximo de solicitações locais de IPC por segundo enviadas para cada dispositivo de IoT conectado.

Padrão: 20 solicitações/segundo para cada item

> **Note**
>
> Esses parâmetros de limite de taxa definem o número máximo de solicitações por segundo para o serviço paralelo local. O número máximo de solicitações por segundo

para o serviço AWS IoT Device Shadow depende do seu Região da AWS. Para obter mais informações, consulte os limites da API AWS IoT Device Shadow Service no Referência geral da Amazon Web Services.

`shadowDocumentSizeLimitBytes`

(Opcional) O tamanho máximo permitido de cada documento de estado JSON para sombras locais.

Se você aumentar esse valor, também deverá aumentar o limite de recursos do documento de estado JSON para sombras na nuvem. Para obter mais informações, consulte os limites da API AWS IoT Device Shadow Service no Referência geral da Amazon Web Services.

Padrão: 8192 bytes

Máximo: 30720 bytes

Example Exemplo: atualização da mesclagem de configurações

O exemplo a seguir mostra um exemplo de atualização de mesclagem de configurações com todos os parâmetros de configuração disponíveis para o componente shadow manager.

```
{
   "strategy":{
      "type":"periodic",
      "delay":300
   },
   "synchronize":{
      "shadowDocumentsMap":{
         "MyDevice1":{
            "classic":false,
            "namedShadows":[
               "MyShadowA",
               "MyShadowB"
            ]
         },
         "MyDevice2":{
            "classic":true,
            "namedShadows":[]
         }
```

```
      },
      "direction":"betweenDeviceAndCloud"
   },
   "rateLimits":{
      "maxOutboundSyncUpdatesPerSecond":100,
      "maxTotalLocalRequestsRate":200,
      "maxLocalRequestsPerSecondPerThing":20
   },
   "shadowDocumentSizeLimitBytes":8192
}
```

2.0.x

`synchronize`

(Opcional) As configurações de sincronização que determinam como as sombras são sincronizadas com o. Nuvem AWS

> **Note**
>
> Você deve criar uma atualização de configuração com essa propriedade para sincronizar sombras com o. Nuvem AWS

Esse objeto contém as seguintes informações.

`coreThing`

(Opcional) O dispositivo principal faz sombras para sincronizar. Esse objeto contém as seguintes informações.

`classic`

(Opcional) Por padrão, o gerenciador de sombras sincroniza o estado local da sombra clássica do seu dispositivo principal com o. Nuvem AWS Se você não quiser sincronizar a sombra clássica do dispositivo, defina-a como`false`.

Padrão: `true`

`namedShadows`

(Opcional) A lista de sombras nomeadas do dispositivo principal a serem sincronizadas. Você deve especificar os nomes exatos das sombras.

> **Warning**
>
> O AWS IoT Greengrass serviço usa a sombra `AWSManagedGreengrassV2Deployment` nomeada para gerenciar implantações direcionadas a dispositivos principais individuais. Essa sombra nomeada é reservada para uso pelo AWS IoT Greengrass serviço. Não atualize nem exclua essa sombra nomeada.

`shadowDocumentsMap`

(Opcional) As sombras adicionais do dispositivo a serem sincronizadas. O uso desse parâmetro de configuração facilita a especificação de documentos de sombra. Recomendamos que você use esse parâmetro em vez do `shadowDocuments` objeto.

> **Note**
>
> Se você especificar um `shadowDocumentsMap` objeto, não deverá especificar um `shadowDocuments` objeto.

Cada objeto contém as seguintes informações:

`thingName`

A configuração de sombra do *`ThingName`* para essa configuração de sombra.

`classic`

(Opcional) Se você não quiser sincronizar a sombra clássica do dispositivo com o `thingName` dispositivo, defina-a como`false`.

`namedShadows`

A lista de sombras nomeadas que você deseja sincronizar. Você deve especificar os nomes exatos das sombras.

`shadowDocuments`

(Opcional) A lista de sombras adicionais do dispositivo a serem sincronizadas. Recomendamos que você use o `shadowDocumentsMap` parâmetro em vez disso.

> **Note**
>
> Se você especificar um `shadowDocuments` objeto, não deverá especificar um `shadowDocumentsMap` objeto.

Cada objeto nessa lista contém as seguintes informações.

`thingName`

O nome do dispositivo para o qual sincronizar sombras.

`classic`

(Opcional) Se você não quiser sincronizar a sombra clássica do dispositivo com o `thingName` dispositivo, defina-a como`false`.

Padrão: `true`

`namedShadows`

(Opcional) A lista de sombras de dispositivos nomeados que você deseja sincronizar. Você deve especificar os nomes exatos das sombras.

`rateLimits`

(Opcional) As configurações que determinam os limites de taxa para solicitações de serviço paralelo.

Esse objeto contém as seguintes informações.

`maxOutboundSyncUpdatesPerSecond`

(Opcional) O número máximo de solicitações de sincronização por segundo que o dispositivo transmite.

Padrão: 100 solicitações/segundo

`maxTotalLocalRequestsRate`

(Opcional) O número máximo de solicitações IPC locais por segundo enviadas ao dispositivo principal.

Padrão: 200 solicitações/segundo

`maxLocalRequestsPerSecondPerThing`

(Opcional) O número máximo de solicitações locais de IPC por segundo enviadas para cada dispositivo de IoT conectado.

Padrão: 20 solicitações/segundo para cada item

> **Note**
>
> Esses parâmetros de limite de taxa definem o número máximo de solicitações por segundo para o serviço paralelo local. O número máximo de solicitações por segundo para o serviço AWS IoT Device Shadow depende do seu Região da AWS. Para obter mais informações, consulte os limites da API AWS IoT Device Shadow Service no Referência geral da Amazon Web Services.

`shadowDocumentSizeLimitBytes`

(Opcional) O tamanho máximo permitido de cada documento de estado JSON para sombras locais.

Se você aumentar esse valor, também deverá aumentar o limite de recursos do documento de estado JSON para sombras na nuvem. Para obter mais informações, consulte os limites da API AWS IoT Device Shadow Service no Referência geral da Amazon Web Services.

Padrão: 8192 bytes

Máximo: 30720 bytes

Example Exemplo: atualização da mesclagem de configurações

O exemplo a seguir mostra um exemplo de atualização de mesclagem de configurações com todos os parâmetros de configuração disponíveis para o componente shadow manager.

```
{
  "synchronize": {
    "coreThing": {
      "classic": true,
      "namedShadows": [
```

```
        "MyCoreShadowA",
        "MyCoreShadowB"
      ]
    },
    "shadowDocuments": [
      {
        "thingName": "MyDevice1",
        "classic": false,
        "namedShadows": [
          "MyShadowA",
          "MyShadowB"
        ]
      },
      {
        "thingName": "MyDevice2",
        "classic": true,
        "namedShadows": []
      }
    ]
  },
  "rateLimits": {
    "maxOutboundSyncUpdatesPerSecond": 100,
    "maxTotalLocalRequestsRate": 200,
    "maxLocalRequestsPerSecondPerThing": 20
  },
  "shadowDocumentSizeLimitBytes": 8192
}
```

Arquivo de log local

Esse componente usa o mesmo arquivo de log do componente do núcleo do Greengrass.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

Para ver os registros desse componente

- Execute o comando a seguir no dispositivo principal para visualizar o arquivo de log desse componente em tempo real. Substitua */greengrass/v2* ou *C:\greengrass\v2* pelo caminho para a pasta AWS IoT Greengrass raiz.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
  ```

Changelog

A tabela a seguir descreve as alterações em cada versão do componente.

Version (Versão)	Alterações
2.3.8	Correções de erros e melhorias • Corrige um problema em que o shadow manager cria uma situação de impasse durante a conexão do cliente MQTT.
2.3.7	Correções de erros e melhorias • Corrige um problema em que o Shadow Manager registra periodica mente um `NullPointerException` erro durante a sincronização do Shadow Manager.
2.3.6	Correções de erros e melhorias • Corrige um problema em que as propriedades de sombra que são excluídas por meio de Nuvem AWS atualizações enquanto o dispositi vo está off-line continuam existindo na sombra local após recuperar a conectividade.
2.3.5	Versão atualizada para a versão 2.12.0 do Greengrass nucleus.

Version (Versão)	Alterações
2.3.4	Correções de erros e melhorias • Adiciona suporte para documentos de estado de sombra nulos e vazios.
2.3.3	Versão atualizada para a versão 2.11.0 do Greengrass nucleus.
2.3.2	Correções de erros e melhorias • Corrige um problema em que o gerenciador de sombras entra no `BROKEN` estado quando o banco de dados paralelo local está corrompido. • Versão atualizada para a versão 2.10.0 do Greengrass nucleus.
2.3.1	Correções de erros e melhorias • Corrige uma condição que pode impedir a sincronização das atualizações do Cloud Shadow. • Corrige um problema em que as alterações na configuração de sincronização de sombra nomeada se aplicam somente a uma sombra nomeada.
2.3.0	Correções de erros e melhorias • Corrige um problema que pode impedir a sincronização das sombras quando a chave privada do dispositivo Greengrass é armazenada em um módulo de segurança de hardware.
2.2.4	Correções de erros e melhorias • Corrige um problema em que a validação do tamanho da sombra não era consistente com a nuvem ao atualizar o documento de sombra local. • Corrige um problema em que o gerenciador paralelo para de ouvir as atualizações de configuração se uma implantação executa uma `RESET` nos nós de configuração.
2.2.3	Versão atualizada para a versão 2.9.0 do Greengrass nucleus.
2.2.2	Versão atualizada para a versão 2.8.0 do Greengrass nucleus.
2.2.1	Versão atualizada para a versão 2.7.0 do Greengrass nucleus.

Version (Versão)	Alterações
2.2.0	Novos atributos • Adiciona suporte para o serviço paralelo local na interface local de publicação/assinatura. Agora você pode se comunicar com o agente de mensagens local de publicação/assinatura sobre tópicos paralelos do MQTT para obter, atualizar e excluir sombras no dispositivo principal. Esse recurso permite conectar dispositivos cliente ao serviço paralelo local usando a ponte MQTT para retransmitir mensagens sobre tópicos paralelos entre dispositivos cliente e a interface local de publicação/assinatura. Esse recurso requer a versão v2.6.0 ou posterior do componente do núcleo do Greengrass. Para conectar dispositivos cliente ao serviço paralelo local, você também deve usar a versão 2.2.0 ou posterior do componente de ponte MQTT. • Adiciona a `direction` opção que você pode configurar para personalizar a direção para sincronizar sombras entre o serviço de sombra local e o. Nuvem AWS Você pode configurar essa opção para reduzir a largura de banda e as conexões com o. Nuvem AWS
2.1.1	Correções de erros e melhorias • Corrige um problema em que a profundidade máxima nas `reported` seções `desired` e no documento de estado de sombra do dispositivo JSON era de 4 níveis em vez de 5 níveis. • Versão atualizada para a versão 2.6.0 do Greengrass nucleus.
2.1.0	Novos atributos • Adiciona suporte para intervalos periódicos de sincronização de sombra, para que você possa configurar o dispositivo principal para reduzir o uso e as cobranças da largura de banda.
2.0.6	Esta versão contém correções de erros e melhorias.
2.0.5	Versão atualizada para a versão 2.5.0 do Greengrass nucleus.

Version (Versão)	Alterações
2.0.4	Correções de erros e melhorias • Corrige um problema que fazia com que o gerenciador de sombras excluísse versões recém-criadas de qualquer sombra que tenha sido excluída anteriormente. • Atualiza a operação `DeleteThingShadow` IPC para incrementar a versão de sombra quando chamada.
2.0.3	Versão atualizada para a versão 2.4.0 do Greengrass nucleus.
2.0.2	Correções de erros e melhorias • Corrigido um problema que fazia com que o gerenciador de sombras não reconhecesse a `delta` propriedade ao sincronizar estados de sombra de AWS IoT Core. • Corrigido um problema que às vezes fazia com que as solicitações de sincronização de uma sombra fossem mescladas incorretamente.
2.0.1	Versão atualizada para a versão 2.3.0 do Greengrass nucleus.
2.0.0	Versão inicial.

Amazon SNS

O componente Amazon SNS (`aws.greengrass.SNS`) publica mensagens em um tópico do Amazon Simple Notification Service (Amazon SNS) Service (Amazon SNS). Você pode usar esse componente para enviar eventos dos dispositivos principais do Greengrass para servidores web, endereços de e-mail e outros assinantes de mensagens. Para obter mais informações, consulte O que é o Amazon SNS? no Guia do desenvolvedor do Amazon Simple Notification Service.

Para publicar em um tópico do Amazon SNS com esse componente, publique uma mensagem no tópico em que esse componente se inscreve. Por padrão, esse componente se inscreve no tópico `sns/message` local de publicação/assinatura. Você pode especificar outros tópicos, incluindo tópicos do AWS IoT Core MQTT, ao implantar esse componente.

Em seu componente personalizado, talvez você queira implementar a lógica de filtragem ou formatação para processar mensagens de outras fontes antes de publicá-las nesse componente.

Isso permite que você centralize sua lógica de processamento de mensagens em um único componente.

> **Note**
>
> Esse componente fornece funcionalidade semelhante ao conector Amazon SNS na AWS IoT Greengrass V1. Para obter mais informações, consulte o conector Amazon SNS no Guia do desenvolvedor AWS IoT Greengrass V1.

Tópicos

Versões

Esse componente tem as seguintes versões:

- 2.1.x
- 2.0.x

Tipo

Esse componente é um componente Lambda () `aws.greengrass.lambda`. O núcleo do Greengrass executa a função Lambda desse componente usando o componente lançador Lambda.

Para ter mais informações, consulte Tipos de componentes.

Sistema operacional

Esse componente pode ser instalado somente nos dispositivos principais do Linux.

Requisitos

Esse componente tem os seguintes requisitos:

- Seu dispositivo principal deve atender aos requisitos para executar as funções do Lambda. Se você quiser que o dispositivo principal execute funções Lambda em contêineres, o dispositivo deve atender aos requisitos para fazer isso. Para ter mais informações, consulte Requisitos da função do Lambda.
- Python versão 3.7 instalado no dispositivo de núcleo e adicionado à variável de ambiente PATH.
- Um tópico do Amazon SNS. Para obter mais informações, consulte Criação de um tópico do Amazon SNS no Guia do desenvolvedor do Amazon Simple Notification Service.
- A função de dispositivo do Greengrass deve permitir a `sns:Publish` ação, conforme mostrado no exemplo de política do IAM a seguir.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "sns:Publish"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:sns:region:account-id:topic-name"
      ]
    }
  ]
}
```

  Você pode substituir dinamicamente o tópico padrão na carga da mensagem de entrada desse componente. Se seu aplicativo usa esse recurso, a política do IAM deve incluir todos os tópicos de destino como recursos. Você pode conceder acesso granular ou condicional aos recursos (por exemplo, usando um esquema de nomeação * curinga).

- Para receber dados de saída desse componente, você deve mesclar a seguinte atualização de configuração para o componente antigo do roteador de assinatura (`aws.greengrass.LegacySubscriptionRouter`) ao implantar esse componente. Essa configuração especifica o tópico em que esse componente publica as respostas.

  Legacy subscription router v2.1.x

  ```
  {
    "subscriptions": {
      "aws-greengrass-sns": {
        "id": "aws-greengrass-sns",
        "source": "component:aws.greengrass.SNS",
        "subject": "sns/message/status",
        "target": "cloud"
      }
    }
  }
  ```

  Legacy subscription router v2.0.x

  ```
  {
    "subscriptions": {
      "aws-greengrass-sns": {
        "id": "aws-greengrass-sns",
        "source": "arn:aws:lambda:region:aws:function:aws-greengrass-sns:version",
        "subject": "sns/message/status",
        "target": "cloud"
      }
    }
  }
  ```

  - Substitua a *região* pela Região da AWS que você usa.
  - *Substitua a versão pela versão da função Lambda que esse componente executa.* Para encontrar a versão da função Lambda, você deve visualizar a receita da versão desse componente que você deseja implantar. Abra a página de detalhes desse componente no AWS IoT Greengrass console e procure o par de valores-chave da função Lambda. Esse par de valores-chave contém o nome e a versão da função Lambda.

> **Important**
>
> Você deve atualizar a versão da função Lambda no roteador de assinatura legado sempre que implantar esse componente. Isso garante que você use a versão correta da função Lambda para a versão do componente que você implanta.

Para ter mais informações, consulte Criar implantações.

- O componente Amazon SNS tem suporte para execução em uma VPC. Para implantar esse componente em uma VPC, é necessário o seguinte.
  - O componente Amazon SNS deve ter conectividade com a `sns.region.amazonaws.com` qual tenha o endpoint VPC de. `com.amazonaws.us-east-1.sns`

Endpoints e portas

Esse componente deve ser capaz de realizar solicitações de saída para os seguintes endpoints e portas, além dos endpoints e portas necessários para a operação básica. Para ter mais informações, consulte Permitir o tráfego de dispositivos por meio de um proxy ou firewall.

Endpoint	Porta	Obrigatório	Descrição
`sns.region.amazonaws.com`	443	Sim	Publique mensagens no Amazon SNS.

Dependências

Quando você implanta um componente, AWS IoT Greengrass também implanta versões compatíveis de suas dependências. Isso significa que você deve atender aos requisitos do componente e de todas as suas dependências para implantá-lo com êxito. Esta seção lista as dependências das versões lançadas desse componente e as restrições de versão semântica que definem as versões do componente para cada dependência. Você também pode visualizar as dependências de cada versão do componente no AWS IoT Greengrass console. Na página de detalhes do componente, procure a lista de dependências.

2.1.7

A tabela a seguir lista as dependências da versão 2.1.7 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.13.0	Rígido
Lançador Lambda	^2.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	^2.0.0	Flexível
Serviço de troca de tokens	^2.0.0	Rígido

2.1.6

A tabela a seguir lista as dependências da versão 2.1.6 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.12.0	Rígido
Lançador Lambda	^2.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	^2.0.0	Flexível
Serviço de troca de tokens	^2.0.0	Rígido

2.1.5

A tabela a seguir lista as dependências da versão 2.1.5 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.11.0	Rígido

Dependência	Versões compatíveis	Tipo de dependência
Lançador Lambda	^2.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	^2.0.0	Flexível
Serviço de troca de tokens	^2.0.0	Rígido

2.1.4

A tabela a seguir lista as dependências da versão 2.1.4 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.10.0	Rígido
Lançador Lambda	^2.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	^2.0.0	Flexível
Serviço de troca de tokens	^2.0.0	Rígido

2.1.3

A tabela a seguir lista as dependências da versão 2.1.3 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.9.0	Rígido
Lançador Lambda	^2.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	^2.0.0	Flexível
Serviço de troca de tokens	^2.0.0	Rígido

2.1.2

A tabela a seguir lista as dependências da versão 2.1.2 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.8.0	Rígido
Lançador Lambda	^2.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	^2.0.0	Flexível
Serviço de troca de tokens	^2.0.0	Rígido

2.1.1

A tabela a seguir lista as dependências da versão 2.1.1 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.7.0	Rígido
Lançador Lambda	^2.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	^2.0.0	Flexível
Serviço de troca de tokens	^2.0.0	Rígido

2.0.8 - 2.1.0

A tabela a seguir lista as dependências das versões 2.0.8 e 2.1.0 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.6.0	Rígido

Dependência	Versões compatíveis	Tipo de dependência
Lançador Lambda	^2.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	^2.0.0	Flexível
Serviço de troca de tokens	^2.0.0	Rígido

2.0.7

A tabela a seguir lista as dependências da versão 2.0.7 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.5.0	Rígido
Lançador Lambda	^2.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	^2.0.0	Flexível
Serviço de troca de tokens	^2.0.0	Rígido

2.0.6

A tabela a seguir lista as dependências da versão 2.0.6 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.4.0	Rígido
Lançador Lambda	^2.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	^2.0.0	Flexível
Serviço de troca de tokens	^2.0.0	Rígido

2.0.5

A tabela a seguir lista as dependências da versão 2.0.5 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.3.0	Rígido
Lançador Lambda	^2.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	^2.0.0	Flexível
Serviço de troca de tokens	^2.0.0	Rígido

2.0.4

A tabela a seguir lista as dependências da versão 2.0.4 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.2.0	Rígido
Lançador Lambda	^2.0.0	Rígido
Lambda runtimes (Tempos de execução do Lambda)	^2.0.0	Flexível
Serviço de troca de tokens	^2.0.0	Rígido

2.0.3

A tabela a seguir lista as dependências da versão 2.0.3 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.3 <2.1.0	Rígido
Lançador Lambda	>=1.0.0	Rígido

Dependência	Versões compatíveis	Tipo de dependência
Lambda runtimes (Tempos de execução do Lambda)	>=1.0.0	Flexível
Serviço de troca de tokens	>=1.0.0	Rígido

Para obter mais informações sobre dependências de componentes, consulte a referência da receita do componente.

Configuração

Esse componente fornece os seguintes parâmetros de configuração que você pode personalizar ao implantar o componente.

> **Note**
>
> A configuração padrão desse componente inclui parâmetros da função Lambda. Recomendamos que você edite somente os parâmetros a seguir para configurar esse componente em seus dispositivos.

`lambdaParams`

Um objeto que contém os parâmetros da função Lambda desse componente. Esse objeto contém as seguintes informações:

`EnvironmentVariables`

Um objeto que contém os parâmetros da função Lambda. Esse objeto contém as seguintes informações:

`DEFAULT_SNS_ARN`

O ARN do tópico padrão do Amazon SNS em que esse componente publica mensagens. Você pode substituir o tópico de destino pela `sns_topic_arn` propriedade na carga da mensagem de entrada.

`containerMode`

(Opcional) O modo de conteinerização desse componente. Escolha uma das seguintes opções:

- `NoContainer`— O componente não é executado em um ambiente de execução isolado.
- `GreengrassContainer`— O componente é executado em um ambiente de execução isolado dentro do AWS IoT Greengrass contêiner.

Padrão: `GreengrassContainer`

`containerParams`

(Opcional) Um objeto que contém os parâmetros do contêiner desse componente. O componente usa esses parâmetros se você especificar `GreengrassContainer` para`containerMode`.

Esse objeto contém as seguintes informações:

`memorySize`

(Opcional) A quantidade de memória (em kilobytes) a ser alocada para o componente.

O padrão é 512 MB (525.312 KB).

`pubsubTopics`

(Opcional) Um objeto que contém os tópicos em que o componente se inscreve para receber mensagens. Você pode especificar cada tópico e se o componente se inscreve nos tópicos do MQTT AWS IoT Core ou nos tópicos locais de publicação/assinatura.

Esse objeto contém as seguintes informações:

`0`— Este é um índice de matriz como uma string.

Um objeto que contém as seguintes informações:

`type`

(Opcional) O tipo de mensagem de publicação/assinatura que esse componente usa para assinar mensagens. Escolha uma das seguintes opções:

- `PUB_SUB` – Assine mensagens locais de publicar/assinar. Se você escolher essa opção, o tópico não poderá conter curingas do MQTT. Para obter mais informações sobre como enviar mensagens do componente personalizado ao especificar essa opção, consulte Publique/assine mensagens locais.
- `IOT_CORE`— Assine as mensagens do AWS IoT Core MQTT. Se você escolher essa opção, o tópico poderá conter curingas do MQTT. Para obter mais informações sobre como enviar mensagens de componentes personalizados ao especificar essa opção, consulte Publique/assine mensagens MQTT AWS IoT Core.

Padrão: PUB_SUB

`topic`

(Opcional) O tópico no qual o componente se inscreve para receber mensagens. Se você especificar `IotCore` para`type`, poderá usar curingas MQTT (+e#) neste tópico.

Example Exemplo: atualização da mesclagem de configuração (modo contêiner)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "DEFAULT_SNS_ARN": "arn:aws:sns:us-west-2:123456789012:mytopic"
    }
  },
  "containerMode": "GreengrassContainer"
}
```

Example Exemplo: atualização de mesclagem de configuração (sem modo de contêiner)

```
{
  "lambdaExecutionParameters": {
    "EnvironmentVariables": {
      "DEFAULT_SNS_ARN": "arn:aws:sns:us-west-2:123456789012:mytopic"
    }
  },
  "containerMode": "NoContainer"
}
```

Dados de entrada

Esse componente aceita mensagens sobre o tópico a seguir e publica a mensagem no estado em que se encontra no tópico de destino do Amazon SNS. Por padrão, esse componente assina mensagens locais de publicação/assinatura. Para obter mais informações sobre como publicar mensagens nesse componente a partir de seus componentes personalizados, consulte[Publique/assine mensagens locais](#).

Tópico padrão (publicação/assinatura local): `sns/message`

A mensagem aceita as seguintes propriedades. As mensagens de entrada devem estar no formato JSON.

`request`

As informações sobre a mensagem a ser enviada ao tópico do Amazon SNS.

Tipo: `object` que contém as seguintes informações:

`message`

O conteúdo da mensagem como uma sequência de caracteres.

Para enviar um objeto JSON, serialize-o como uma string e especifique `json` a `message_structure` propriedade.

Tipo: `string`

`subject`

(Opcional) O assunto da mensagem.

Tipo: `string`

O assunto pode ser texto ASCII e até 100 caracteres. Ele deve começar com uma letra, número ou sinal de pontuação. Não pode incluir quebras de linha nem caracteres de controle.

`sns_topic_arn`

(Opcional) O ARN do tópico do Amazon SNS em que esse componente publica a mensagem. Especifique essa propriedade para substituir o tópico padrão do Amazon SNS.

Tipo: `string`

`message_structure`

(Opcional) A estrutura da mensagem. Especifique `json` para enviar uma mensagem JSON que você serializa como uma string na `content` propriedade.

Tipo: `string`

Valores válidos: `json`

`id`

Um ID arbitrário para a solicitação. Use essa propriedade para mapear uma solicitação de entrada para uma resposta de saída. Quando você especifica essa propriedade, o componente define a `id` propriedade no objeto de resposta com esse valor.

Tipo: `string`

> **Note**
>
> O tamanho da mensagem pode ser de no máximo 256 KB.

Example Exemplo de entrada: mensagem de string

```
{
  "request": {
    "subject": "Message subject",
    "message": "Message data",
    "sns_topic_arn": "arn:aws:sns:region:account-id:topic2-name"
  },
  "id": "request123"
}
```

Example Exemplo de entrada: mensagem JSON

```
{
  "request": {
    "subject": "Message subject",
    "message": "{ \"default\": \"Message data\" }",
    "message_structure": "json"
  },
  "id": "request123"
}
```

Dados de saída

Por padrão, esse componente publica respostas como dados de saída no seguinte tópico do MQTT. Você deve especificar esse tópico conforme a `subject` configuração do componente antigo do roteador de assinatura. Para obter mais informações sobre como assinar mensagens sobre esse tópico em seus componentes personalizados, consultePublique/assine mensagens MQTT AWS IoT Core.

Tópico padrão (AWS IoT Core MQTT): `sns/message/status`

Example Exemplo de resultado: sucesso

```
{
```

```
  "response": {
    "sns_message_id": "f80a81bc-f44c-56f2-a0f0-d5af6a727c8a",
    "status": "success"
  },
  "id": "request123"
}
```

Example Exemplo de resultado: falha

```
{
  "response" : {
    "error": "InvalidInputException",
    "error_message": "SNS Topic Arn is invalid",
    "status": "fail"
  },
  "id": "request123"
}
```

Arquivo de log local

Esse componente usa o seguinte arquivo de log.

```
/greengrass/v2/logs/aws.greengrass.SNS.log
```

Para ver os registros desse componente

- Execute o comando a seguir no dispositivo principal para visualizar o arquivo de log desse componente em tempo real. */greengrass/v2*Substitua pelo caminho para a pasta AWS IoT Greengrass raiz.

  ```
  sudo tail -f /greengrass/v2/logs/aws.greengrass.SNS.log
  ```

Licenças

Esse componente inclui o seguinte software/licenciamento de terceiros:

- AWS SDK for Python (Boto3)/Licença Apache 2.0
- botocore/Licença Apache 2.0
- dateutil/Licença PSF

- docutils/Licença BSD, GNU Licença pública geral (GPL), Licença Python Software Foundation, Domínio público
- jmespath/Licença MIT
- s3transfer/Licença Apache 2.0
- urllib3/Licença MIT

Esse componente é lançado de acordo com o Contrato de Licença de Software Principal do Greengrass.

Changelog

A tabela a seguir descreve as alterações em cada versão do componente.

Version (Versão)	Alterações
2.1.7	Versão atualizada para a versão 2.12.0 do Greengrass nucleus.
2.1.6	Versão atualizada para a versão 2.11.0 do Greengrass nucleus.
2.1.5	Versão atualizada para a versão 2.10.0 do Greengrass nucleus.
2.1.4	Versão atualizada para a versão 2.9.0 do Greengrass nucleus.
2.1.3	Versão atualizada para a versão 2.8.0 do Greengrass nucleus.
2.1.2	Versão atualizada para a versão 2.7.0 do Greengrass nucleus.
2.1.1	Versão atualizada para a versão 2.6.0 do Greengrass nucleus.
2.1.0	Novos atributos • Adiciona suporte para configurações de proxy de rede HTTPS. Para obter mais informações, consulte Permita que o dispositivo principal confie em um proxy HTTPS e Conectar-se à porta 443 ou por meio de um proxy de rede.
2.0.8	Versão atualizada para a versão 2.5.0 do Greengrass nucleus.
2.0.7	Versão atualizada para a versão 2.4.0 do Greengrass nucleus.

Version (Versão)	Alterações
2.0.6	Versão atualizada para a versão 2.3.0 do Greengrass nucleus.
2.0.5	Versão atualizada para a versão 2.2.0 do Greengrass nucleus.
2.0.4	Versão atualizada para a versão 2.1.0 do Greengrass nucleus.
2.0.3	Versão inicial.

Gerenciador de fluxo

O componente stream manager (`aws.greengrass.StreamManager`) permite que você processe fluxos de dados a serem transferidos para os dispositivos Nuvem AWS principais do Greengrass.

Para obter mais informações sobre como configurar e usar o gerenciador de fluxo em componentes personalizados, consulteGerencie fluxos de dados nos dispositivos principais do Greengrass.

Tópicos

- Versões
- Tipo
- Sistema operacional
- Requisitos
- Dependências
- Configuração
- Arquivo de log local
- Changelog

Versões

Esse componente tem as seguintes versões:

- 2.1.x
- 2.0.x

> **Note**
>
> Se você usa o gerenciador de stream para exportar dados para a nuvem, não poderá atualizar a versão 2.0.7 do componente stream manager para uma versão entre v2.0.8 e v2.0.11. Se você estiver implantando o gerenciador de fluxo pela primeira vez, é altamente recomendável que você implante a versão mais recente do componente do gerenciador de fluxo.

Tipo

Este componente é um componente genérico (`aws.greengrass.generic`). O núcleo do Greengrass executa os scripts do ciclo de vida do componente.

Para ter mais informações, consulte Tipos de componentes.

Sistema operacional

Esse componente pode ser instalado em dispositivos principais que executam os seguintes sistemas operacionais:

- Linux
- Windows

Requisitos

Esse componente tem os seguintes requisitos:

- A função de troca de tokens deve permitir o acesso aos Nuvem AWS destinos que você usa com o gerenciador de stream. Para obter mais informações, consulte:
  - the section called “Canais do AWS IoT Analytics”
  - the section called “Amazon Kinesis Data Streams”
  - the section called “Propriedades do ativo AWS IoT SiteWise”
  - the section called “Objetos do Amazon S3”
- O componente gerenciador de stream tem suporte para execução em uma VPC. Para implantar esse componente em uma VPC, é necessário o seguinte.

- O componente do gerenciador de fluxo deve ter conectividade com o AWS serviço no qual você publica dados.
  - Amazon S3: `com.amazonaws.region.s3`
  - Amazon Kinesis Data Streams: `com.amazonaws.region.kinesis-streams`
  - AWS IoT SiteWise: `com.amazonaws.region.iotsitewise.data`
- Se você publicar dados no Amazon S3 na `us-east-1` região, esse componente tentará usar o endpoint global do S3 por padrão; no entanto, esse endpoint não está disponível por meio do endpoint da interface VPC do Amazon S3. Para obter mais informações, consulte Restrições e limitações do AWS PrivateLink Amazon S3. Para resolver isso, você pode escolher entre as seguintes opções.
  - Configure o componente do gerenciador de fluxo para usar o endpoint regional do S3 na `us-east-1` região, fornecendo a variável de `AWS_S3_US_EAST_1_REGIONAL_ENDPOINT=regional` ambiente.
  - Crie um endpoint VPC do gateway Amazon S3 em vez de um endpoint VPC com interface do Amazon S3. Os endpoints do gateway S3 oferecem suporte ao acesso ao endpoint global do S3. Para obter mais informações, consulte Criar um endpoint de gateway.

Endpoints e portas

Esse componente deve ser capaz de realizar solicitações de saída para os seguintes endpoints e portas, além dos endpoints e portas necessários para a operação básica. Para ter mais informações, consulte Permitir o tráfego de dispositivos por meio de um proxy ou firewall.

Endpoint	Porta	Obrigatório	Descrição
`iotanalytics. region.amazonaws.com`	443	Não	Obrigatório se você publicar dados em AWS IoT Analytics.
`kinesis.region.amazonaws.com`	443	Não	Obrigatório se você publicar

Endpoint	Porta	Obrigatório	Descrição
			dados no Firehose.
`data.iotsitewise.` *`region`*`.amazonaws.com`	443	Não	Obrigatório se você publicar dados em AWS IoT SiteWise.
`*.s3.amazonaws.com`	443	Não	Obrigatório se você publicar dados em buckets do S3. Você pode `*` substituir pelo nome de cada bucket em que você publica dados.

Dependências

Quando você implanta um componente, AWS IoT Greengrass também implanta versões compatíveis de suas dependências. Isso significa que você deve atender aos requisitos do componente e de todas as suas dependências para implantá-lo com êxito. Esta seção lista as dependências das versões lançadas desse componente e as restrições de versão semântica que definem as versões dos componentes para cada dependência. Você também pode visualizar as dependências de cada versão do componente no AWS IoT Greengrass console. Na página de detalhes do componente, procure a lista de dependências.

2.1.11

A tabela a seguir lista as dependências da versão 2.1.11 a 2.1.10 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.13.0	Flexível
Serviço de troca de tokens	>=0,0,0	Rígido

2.1.9 – 2.1.10

A tabela a seguir lista as dependências das versões 2.1.9 a 2.1.10 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.12.0	Flexível
Serviço de troca de tokens	>=0,0,0	Rígido

2.1.5 – 2.1.8

A tabela a seguir lista as dependências das versões 2.1.5 a 2.1.8 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.11.0	Flexível
Serviço de troca de tokens	>=0,0,0	Rígido

2.1.2 – 2.1.4

A tabela a seguir lista as dependências das versões 2.1.2 a 2.1.4 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.10.0	Flexível

Dependência	Versões compatíveis	Tipo de dependência
Serviço de troca de tokens	>=0,0,0	Rígido

2.1.1

A tabela a seguir lista as dependências da versão 2.1.1 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.9.0	Flexível
Serviço de troca de tokens	>=0,0,0	Rígido

2.1.0

A tabela a seguir lista as dependências da versão 2.1.0 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.8.0	Flexível
Serviço de troca de tokens	>=0,0,0	Rígido

2.0.15

A tabela a seguir lista as dependências da versão 2.0.15 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.7.0	Flexível
Serviço de troca de tokens	>=0,0,0	Rígido

2.0.13 and 2.0.14

A tabela a seguir lista as dependências das versões 2.0.13 e 2.0.14 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.6.0	Flexível
Serviço de troca de tokens	>=0,0,0	Rígido

2.0.11 and 2.0.12

A tabela a seguir lista as dependências das versões 2.0.11 e 2.0.12 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.5.0	Flexível
Serviço de troca de tokens	>=0,0,0	Rígido

2.0.10

A tabela a seguir lista as dependências da versão 2.0.10 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.4.0	Flexível
Serviço de troca de tokens	>=0,0,0	Rígido

2.0.9

A tabela a seguir lista as dependências da versão 2.0.9 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.3.0	Flexível
Serviço de troca de tokens	>=0,0,0	Rígido

2.0.8

A tabela a seguir lista as dependências da versão 2.0.8 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.0 <2.2.0	Flexível
Serviço de troca de tokens	>=0,0,0	Rígido

2.0.7

A tabela a seguir lista as dependências da versão 2.0.7 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.0.3 <2.1.0	Flexível
Serviço de troca de tokens	>=0,0,0	Rígido

Para obter mais informações sobre dependências de componentes, consulte a referência da receita do componente.

Configuração

Esse componente fornece os seguintes parâmetros de configuração que você pode personalizar ao implantar o componente.

`STREAM_MANAGER_STORE_ROOT_DIR`

(Opcional) O caminho absoluto do diretório local usado para armazenar fluxos. Esse valor deve começar com uma barra (por exemplo, `/data`).

Você deve especificar uma pasta existente, e o usuário do sistema que executa o componente do gerenciador de stream deve ter permissões para ler e gravar nessa pasta. Por exemplo, você pode executar os comandos a seguir para criar e configurar uma pasta`/var/greengrass/streams`, que você especifica como a pasta raiz do gerenciador de streams. Esses comandos permitem que o usuário padrão do sistema`aggc_user`,, leia e grave nessa pasta.

```
sudo mkdir /var/greengrass/streams
sudo chown ggc_user /var/greengrass/streams
sudo chmod 700 /var/greengrass/streams
```

Padrão: */greengrass/v2*/work/aws.greengrass.StreamManager

`STREAM_MANAGER_SERVER_PORT`

(Opcional) O número da porta local a ser usada para se comunicar com o gerenciador de fluxo.

Você pode especificar `0` o uso de uma porta disponível aleatória.

Padrão: `8088`

`STREAM_MANAGER_AUTHENTICATE_CLIENT`

(Opcional) Você pode tornar obrigatória a autenticação dos clientes antes de poderem interagir com o gerenciador de stream. O SDK do Stream Manager controla a interação entre clientes e o gerenciador de streams. Esse parâmetro determina quais clientes podem chamar o SDK do Stream Manager para trabalhar com streams. Para obter mais informações, consulte Autenticação do cliente do stream manager.

Se você especificar`true`, o SDK do Stream Manager permite somente componentes do Greengrass como clientes.

Se você especificar`false`, o SDK do Stream Manager permite que todos os processos no dispositivo principal sejam clientes.

Padrão: `true`

`STREAM_MANAGER_EXPORTER_MAX_BANDWIDTH`

(Opcional) A largura de banda máxima média (em kilobits por segundo) que o gerenciador de streaming pode usar para exportar dados.

Padrão: sem limite

`STREAM_MANAGER_EXPORTER_THREAD_POOL_SIZE`

(Opcional) O número máximo de threads ativos que o gerenciador de stream pode usar para exportar dados.

O tamanho ideal depende do hardware, do volume do fluxo e do número planejado de fluxos de exportação. Se a velocidade de exportação for lenta, você poderá ajustar essa configuração para encontrar o tamanho ideal para seu hardware e caso de negócios. A CPU e a memória do

hardware do dispositivo de núcleo são fatores limitantes. Para iniciar, você pode tentar definir esse valor igual ao número de núcleos do processador no dispositivo.

Tenha cuidado para não definir um tamanho superior ao que o seu hardware pode suportar. Cada fluxo consome recursos de hardware, então tente limitar o número de fluxos de exportação em dispositivos restritos.

Padrão: 5 tópicos

`STREAM_MANAGER_EXPORTER_S3_DESTINATION_MULTIPART_UPLOAD_MIN_PART_SIZE_BYTES`

(Opcional) O tamanho mínimo (em bytes) de uma peça em um upload de várias partes para o Amazon S3. O gerenciador de fluxo usa essa configuração e o tamanho do arquivo de entrada para determinar como agrupar dados em lote em uma solicitação PUT de várias partes.

> **Note**
>
> O gerenciador de streams usa a `sizeThresholdForMultipartUploadBytes` propriedade streams para determinar se você deseja exportar para o Amazon S3 como um upload de uma ou várias partes. AWS IoT Greengrass os componentes podem definir esse limite ao criar um stream que exporta para o Amazon S3.

Padrão: `5242880` (5 MB). Esse também é o valor mínimo.

`LOG_LEVEL`

(Opcional) O nível de registro do componente. Escolha entre os seguintes níveis de registro, listados aqui em ordem de nível:

- `TRACE`
- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

Padrão: `INFO`

`JVM_ARGS`

(Opcional) Os argumentos personalizados da Java Virtual Machine a serem transmitidos ao gerenciador de stream na inicialização. Separe vários argumentos por espaços.

Só use esse parâmetro quando precisar substituir as configurações padrão usadas pela JVM. Por exemplo, talvez seja necessário aumentar o tamanho do heap padrão caso você planeje exportar um grande número de fluxos.

Example Exemplo: atualização da mesclagem de configurações

O exemplo de configuração a seguir especifica o uso de uma porta não padrão.

```
{
  "STREAM_MANAGER_SERVER_PORT": "18088"
}
```

Arquivo de log local

Esse componente usa o seguinte arquivo de log.

Linux

```
/greengrass/v2/logs/aws.greengrass.StreamManager.log
```

Windows

```
C:\greengrass\v2\logs\aws.greengrass.StreamManager.log
```

Para ver os registros desse componente

- Execute o comando a seguir no dispositivo principal para visualizar o arquivo de log desse componente em tempo real. Substitua */greengrass/v2* ou *C:\greengrass\v2* pelo caminho para a pasta AWS IoT Greengrass raiz.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.greengrass.StreamManager.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\aws.greengrass.StreamManager.log -Tail 10 -
  Wait
  ```

Changelog

A tabela a seguir descreve as alterações em cada versão do componente.

Version (Versão)	Alterações
2.1.12	Correções de erros e melhorias Atualiza a ordem em que as credenciais são usadas para que as credenciais do Greengrass sejam preferidas AWS para solicitações de serviço.
2.1.11	Versão atualizada para a versão 2.12.0 do Greengrass nucleus.
2.1.10	Correções de erros e melhorias Corrige um problema em que a configuração do proxy HTTPS não confia na cadeia de certificados da autoridade de certificação (CA) do Greengras s.
2.1.9	Versão atualizada para a versão 2.11.0 do Greengrass nucleus.
2.1.8	Correções de erros e melhorias Corrige um problema em que o gerenciador de fluxo repete infinitamente SiteWise as exportações que falham com. `InvalidRequestExce ption`
2.1.7	Correções de erros e melhorias Corrige um problema em que o gerenciador de fluxo não consegue ler a configuração do proxy corretamente.
2.1.6	Correções de erros e melhorias Corrige um problema que poderia causar uma falha na inicialização em determinados processadores ARMv8, incluindo o Jetson Nano.
2.1.5	Versão atualizada para a versão 2.10.0 do Greengrass nucleus.

Version (Versão)	Alterações
2.1.4	Correções de erros e melhorias • Corrige um problema em que as entradas do mesmo ativo de proprieda de com o mesmo carimbo de data/hora em um único lote retornam `ConflictingOperationException` da SiteWise API, o que faz com que o gerenciador de fluxo tente novamente continuamente. • Atualiza o tempo limite de conexão padrão de 3 segundos para 1 minuto.
2.1.3	Correções de erros e melhorias Corrige um problema de inicialização no sistema operacional Windows ao ser executado como usuário SYSTEM.
2.1.2	Correções de erros e melhorias • Corrige um problema no sistema operacional Windows que usa um idioma diferente do inglês. • Versão atualizada para a versão 2.9.0 do Greengrass nucleus.
2.1.1	Versão atualizada para a versão 2.8.0 do Greengrass nucleus.
2.1.0	Novos atributos • Atualiza esse componente para enviar automaticamente métricas de telemetria para a Amazon. EventBridge Para ter mais informações, consulte Colete dados de telemetria de integridade do sistema a partir dos dispositivos principais AWS IoT Greengrass. Esse recurso requer a versão 2.7.0 ou posterior do componente do núcleo do Greengrass. • Versão atualizada para a versão 2.7.0 do Greengrass nucleus.
2.0.15	Versão atualizada para a versão 2.6.0 do Greengrass nucleus.
2.0.14	Esta versão contém correções de erros e melhorias.
2.0.13	Versão atualizada para a versão 2.5.0 do Greengrass nucleus.

Version (Versão)	Alterações
2.0.12	Correções de erros e melhorias Corrige um problema que impedia a atualização do stream manager v2.0.7 para uma versão entre v2.0.8 e v2.0.11. Se você usa o gerenciador de stream para exportar dados para a nuvem, agora você pode atualizar para a v2.0.12.
2.0.11	Versão atualizada para a versão 2.4.0 do Greengrass nucleus.
2.0.10	Versão atualizada para a versão 2.3.0 do Greengrass nucleus.
2.0.9	Versão atualizada para a versão 2.2.0 do Greengrass nucleus.
2.0.8	Versão atualizada para a versão 2.1.0 do Greengrass nucleus.
2.0.7	Versão inicial.

Agente Systems Manager

O componente AWS Systems Manager Agent (`aws.greengrass.SystemsManagerAgent`) instala o Systems Manager Agent, para que você possa gerenciar os dispositivos principais com o Systems Manager. O Systems Manager é um AWS serviço que você pode usar para visualizar e controlar sua infraestrutura AWS, incluindo instâncias do Amazon EC2, servidores e máquinas virtuais (VMs) locais e dispositivos periféricos. O Systems Manager permite que você visualize dados operacionais, automatize tarefas operacionais e mantenha a segurança e a conformidade. Para obter mais informações, consulte O que é AWS Systems Manager? e Sobre o Systems Manager Agent no Guia AWS Systems Manager do Usuário.

As ferramentas e os recursos do Systems Manager são chamados de capacidades. Os dispositivos principais do Greengrass oferecem suporte a todos os recursos do Systems Manager. Para obter mais informações sobre esses recursos e como usar o Systems Manager para gerenciar dispositivos principais, consulte os recursos do Systems Manager no Guia AWS Systems Manager do Usuário.

Tópicos

- Versões
- Tipo

Versões

Esse componente tem as seguintes versões:

- 1.1.x
- 1,0.x

Tipo

Este componente é um componente genérico (`aws.greengrass.generic`). O núcleo do Greengrass executa os scripts do ciclo de vida do componente.

Para ter mais informações, consulte Tipos de componentes.

Sistema operacional

Esse componente pode ser instalado somente nos dispositivos principais do Linux.

Requisitos

Esse componente tem os seguintes requisitos:

- Um dispositivo principal do Greengrass executado em uma plataforma Linux de 64 bits: Armv8 (AArch64) ou x86_64.
- Você deve ter uma função de serviço AWS Identity and Access Management (IAM) que o Systems Manager possa assumir. Essa função deve incluir a política ManagedInstanceCore gerenciada do AmazonSSM ou uma política personalizada que defina permissões equivalentes. Para obter mais informações, consulte Criar uma função de serviço do IAM para dispositivos periféricos no Guia AWS Systems Manager do usuário.

Ao implantar esse componente, você deve especificar o nome dessa função para o parâmetro `SSMRegistrationRole` de configuração.

- A função do dispositivo Greengrass deve permitir as ações `ssm:AddTagsToResource` e. `ssm:RegisterManagedInstance` A função de dispositivo também deve permitir a `iam:PassRole` ação da função de serviço do IAM que atenda ao requisito anterior. O exemplo a seguir da política do IAM concede essas permissões.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "iam:PassRole"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:iam::account-id:role/SSMServiceRole"
      ]
    },
    {
      "Action": [
        "ssm:AddTagsToResource",
        "ssm:RegisterManagedInstance"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

Endpoints e portas

Esse componente deve ser capaz de realizar solicitações de saída para os seguintes endpoints e portas, além dos endpoints e portas necessários para a operação básica. Para ter mais informações, consulte Permitir o tráfego de dispositivos por meio de um proxy ou firewall.

Endpoint	Porta	Obrigatório	Descrição
`ec2messages. region.amazonaws.com`	443	Sim	Comunique-se com o serviço Systems Manager no Nuvem AWS.
`ssm.region.amazonaws.com`	443	Sim	Registre o dispositivo principal como um nó gerenciado do Systems Manager.
`ssmmessages. region.amazonaws.com`	443	Sim	Comunique-se com o Session Manager, um recurso do Systems Manager, no Nuvem AWS.

Para obter mais informações, consulte Referência: ec2messages, ssmmessages e outras chamadas de API no Guia do usuário.AWS Systems Manager

Dependências

Quando você implanta um componente, AWS IoT Greengrass também implanta versões compatíveis de suas dependências. Isso significa que você deve atender aos requisitos do componente e de todas as suas dependências para implantá-lo com êxito. Esta seção lista as dependências das versões lançadas desse componente e as restrições de versão semântica que definem as versões dos componentes para cada dependência. Você também pode visualizar as dependências de cada versão do componente no AWS IoT Greengrass console. Na página de detalhes do componente, procure a lista de dependências.

A tabela a seguir lista as dependências das versões 1.0.0 a 1.2.4 desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Serviço de troca de tokens	^2.0.0	Flexível

Para obter mais informações sobre dependências de componentes, consulte a referência da receita do componente.

Configuração

Esse componente fornece os seguintes parâmetros de configuração que você pode personalizar ao implantar o componente.

`SSMRegistrationRole`

A função de serviço do IAM que o Systems Manager pode assumir e que inclui a política ManagedInstanceCore gerenciada do AmazonSSM ou uma política personalizada que define permissões equivalentes. Para obter mais informações, consulte Criar uma função de serviço do IAM para dispositivos periféricos no Guia AWS Systems Manager do usuário.

`SSMOverrideExistingRegistration`

(Opcional) Se o dispositivo principal já executa o Systems Manager Agent registrado com uma ativação híbrida, você pode substituir o registro existente do Systems Manager Agent do dispositivo. Defina essa opção `true` para registrar o dispositivo principal como um nó gerenciado usando o Systems Manager Agent fornecido por esse componente.

> **Note**
>
> Essa opção se aplica somente aos dispositivos registrados com uma ativação híbrida. Se o dispositivo principal for executado em uma instância do Amazon EC2 com o Systems Manager Agent instalado e uma função de perfil de instância configurada, o ID do nó gerenciado existente da instância do Amazon EC2 começa com. `i-` Quando você instala o componente Systems Manager Agent, o agente do Systems Manager registra um novo nó gerenciado cuja ID começa com `mi-` em vez de`i-`. Em seguida, você pode usar o nó gerenciado cujo ID começa com `mi-` para gerenciar o dispositivo principal com o Systems Manager.

Padrão: `false`

`SSMResourceTags`

(Opcional) As tags a serem adicionadas ao nó gerenciado do Systems Manager que esse componente cria para o dispositivo principal. Você pode usar essas tags para gerenciar grupos de dispositivos principais com o Systems Manager. Por exemplo, você pode executar um comando em todos os dispositivos que tenham uma tag especificada por você.

Especifique uma lista em que cada tag seja um objeto com a `Key` e `Value` a. Por exemplo, o valor a seguir `SSMResourceTags` instrui esse componente a definir a **`Owner`** tag como **`richard-roe`** no nó gerenciado do dispositivo principal.

```
[
  {
    "Key": "Owner",
    "Value": "richard-roe"
  }
]
```

Esse componente ignora essas tags se o nó gerenciado já existir e `SSMOverrideExistingRegistration` existir`false`.

Example Exemplo: atualização da mesclagem de configurações

O exemplo de configuração a seguir especifica o uso de uma função de serviço chamada `SSMServiceRole` para permitir que o dispositivo principal se registre e se comunique com o Systems Manager.

```
{
  "SSMRegistrationRole": "SSMServiceRole",
  "SSMOverrideExistingRegistration": false,
  "SSMResourceTags": [
    {
      "Key": "Owner",
      "Value": "richard-roe"
    },
    {
      "Key": "Team",
      "Value": "solar"
    }
  ]
}
```

Arquivo de log local

O software Systems Manager Agent grava registros em uma pasta fora da pasta raiz do Greengrass. Para obter mais informações, consulte Visualizando os registros do Systems Manager Agent no Guia AWS Systems Manager do Usuário.

O componente Systems Manager Agent usa scripts de shell para instalar, iniciar e interromper o Systems Manager Agent. Você pode encontrar a saída desses scripts no arquivo de log a seguir.

```
/greengrass/v2/logs/aws.greengrass.SystemsManagerAgent.log
```

Para ver os registros desse componente

- Execute o comando a seguir no dispositivo principal para visualizar o arquivo de log desse componente em tempo real. */greengrass/v2*Substitua pelo caminho para a pasta AWS IoT Greengrass raiz.

  ```
  sudo tail -f /greengrass/v2/logs/aws.greengrass.SystemsManagerAgent.log
  ```

Consulte também

- Gerencie os principais dispositivos do Greengrass com AWS Systems Manager
- O que é o AWS Systems Manager? no AWS Systems Manager Guia do usuário
- Sobre o Systems Manager Agent no Guia AWS Systems Manager do Usuário

Changelog

A tabela a seguir descreve as alterações em cada versão do componente.

Version (Versão)	Alterações
1.2.4	Correções de erros e melhorias Atualiza esse componente para obter a versão 3.2.2303.0 do Agente.
1.2.3	Correções de erros e melhorias • Adiciona novas tentativas para a instalação do componente Agent com o snap on Greengrass. • Atualiza a configuração do componente Agente para usar somente a identidade Onprem no Greengrass. • Atualiza esse componente para atualizar o Agente somente quando a versão do Agente instalada não coincide com a versão do componente Greengrass SSM Agent.
1.1.0	Esta versão contém correções de erros e melhorias.
1.0.0	Versão inicial.

Serviço de troca de tokens

O componente do serviço de troca de tokens (`aws.greengrass.TokenExchangeService`) fornece AWS credenciais que você pode usar para interagir com AWS serviços em seus componentes personalizados.

O serviço de troca de tokens executa uma instância de contêiner do Amazon Elastic Container Service (Amazon ECS) como um servidor local. Esse servidor local se conecta ao

provedor de AWS IoT credenciais usando o alias de AWS IoT função que você configura no componente núcleo principal do Greengrass. O componente fornece duas variáveis de ambiente `AWS_CONTAINER_CREDENTIALS_FULL_URI` `AWS_CONTAINER_AUTHORIZATION_TOKEN` e. `AWS_CONTAINER_CREDENTIALS_FULL_URI`define o URI para esse servidor local. Quando um componente cria um cliente AWS SDK, o cliente reconhece essa variável de ambiente do URI e usa o token no `AWS_CONTAINER_AUTHORIZATION_TOKEN` para se conectar ao serviço de troca de tokens e recuperar AWS as credenciais. Isso permite que os dispositivos principais do Greengrass liguem para as operações AWS de serviço. Para obter mais informações sobre como usar esse componente em componentes personalizados, consulteInteraja com AWS os serviços.

> **Important**
>
> Support para adquirir AWS credenciais dessa forma foi adicionado aos AWS SDKs em 13 de julho de 2016. Seu componente deve usar uma versão do AWS SDK criada nessa data ou após essa data. Para obter mais informações, consulte Como usar um AWS SDK compatível no Amazon Elastic Container Service Developer Guide.

Tópicos

Versões

Esse componente tem as seguintes versões:

- 2.0.x

Tipo

Este componente é um componente genérico (`aws.greengrass.generic`). O núcleo do Greengrass executa os scripts do ciclo de vida do componente.

Para ter mais informações, consulte Tipos de componentes.

Sistema operacional

Esse componente pode ser instalado em dispositivos principais que executam os seguintes sistemas operacionais:

- Linux
- Windows

Dependências

Esse componente não tem nenhuma dependência.

Configuração

Esse componente não tem nenhum parâmetro de configuração.

Arquivo de log local

Esse componente usa o mesmo arquivo de log do componente do núcleo do Greengrass.

Linux

```
/greengrass/v2/logs/greengrass.log
```

Windows

```
C:\greengrass\v2\logs\greengrass.log
```

Para ver os registros desse componente

- Execute o comando a seguir no dispositivo principal para visualizar o arquivo de log desse componente em tempo real. Substitua */greengrass/v2* ou *C:\greengrass\v2* pelo caminho para a pasta AWS IoT Greengrass raiz.

Linux

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
```

Changelog

A tabela a seguir descreve as alterações em cada versão do componente.

Version (Versão)	Alterações
2.0.3	Versão inicial.

Coletor IoT OPC-UA SiteWise

O componente coletor IOT SiteWise OPC-UA (`aws.iot.SiteWiseEdgeCollectorOpcua`) permite que os AWS IoT SiteWise gateways coletem dados de servidores OPC-UA locais.

Com esse componente, os AWS IoT SiteWise gateways podem se conectar a vários servidores OPC-UA. Para obter mais informações sobre AWS IoT SiteWise gateways, consulte Usando AWS IoT SiteWise na borda no Guia do AWS IoT SiteWise usuário.

Tópicos

- Versões
- Tipo
- Sistema operacional
- Requisitos
- Dependências
- Configuração
- Dados de entrada

- Dados de saída
- Arquivo de log local
- Solução de problemas e depuração
- Licenças
- Changelog
- Consulte também

Versões

Esse componente tem as seguintes versões:

- 2.4.x
- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

Tipo

Este componente é um componente genérico (`aws.greengrass.generic`). O núcleo do Greengrass executa os scripts do ciclo de vida do componente.

Para ter mais informações, consulte Tipos de componentes.

Sistema operacional

Esse componente pode ser instalado em dispositivos principais que executam os seguintes sistemas operacionais:

- Linux
- Windows

Requisitos

Esse componente tem os seguintes requisitos:

- O dispositivo principal do Greengrass deve ser executado em uma das seguintes plataformas:
  - SO: Ubuntu 18.04 ou posterior

    Arquitetura: x86_64 (AMD64) ou ARMv8 (Aarch64)
  - SO: Red Hat Enterprise Linux (RHEL) 8

    Arquitetura: x86_64 (AMD64) ou ARMv8 (Aarch64)
  - SO: Amazon Linux 2

    Arquitetura: x86_64 (AMD64) ou ARMv8 (Aarch64)
  - SO: Debian 11

    Arquitetura: x86_64 (AMD64) ou ARMv8 (Aarch64)
  - SISTEMA OPERACIONAL: Windows Server 2019 ou posterior

    Arquitetura: x86_64 (AMD64)
- O dispositivo principal do Greengrass deve permitir conectividade de rede de saída com servidores OPC-UA.

Dependências

Quando você implanta um componente, AWS IoT Greengrass também implanta versões compatíveis de suas dependências. Isso significa que você deve atender aos requisitos do componente e de todas as suas dependências para implantá-lo com êxito. Esta seção lista as dependências das versões lançadas desse componente e as restrições de versão semântica que definem as versões dos componentes para cada dependência. Você também pode visualizar as dependências de cada versão do componente no AWS IoT Greengrass console. Na página de detalhes do componente, procure a lista de dependências.

A tabela a seguir lista as dependências de todas as versões desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.3.0 <3.0.0	Rígido
Gerenciador de fluxo	>2.0.10<3.0.0	Rígido
Gerente secreto	>=2.0.8 <3.0.0	Rígido

Para obter mais informações sobre dependências de componentes, consulte a referência da receita do componente.

Configuração

Esse componente não tem nenhum parâmetro de configuração.

Você pode usar o AWS IoT SiteWise console ou a API para configurar o componente coletor IOT SiteWise OPC-UA. Para obter mais informações, consulte Etapa 4: Adicionar fontes de dados - opcional no Guia AWS IoT SiteWise do usuário.

Dados de entrada

Esse componente só aceita dados nos seguintes formatos, todos os outros serão ignorados e descartados. A tabela abaixo mapeia os tipos de dados OPC UA para seus SiteWise equivalentes.

SiteWise tipo de dados	Tipo de dados OPC UA	Descrição

`STRING`	`String`
	`Guid`
	`XmlElement`
		Uma sequência de caracteres de comprimento máximo de 1024 bytes.

`INTEGER`	`SByte`
	`Byte`
	`Int16`
	`UInt16`
	`Int32`
	`UInt32*`
	`Int64*`
		Um inteiro assinado de 32 bits com um intervalo de. `-2,147,483,648 to 2,147,483,647`

`DOUBLE`	`UInt32*` `Int64*`	Um número de ponto flutuante com alcance `-10^100 to 10^100` e precisão `IEEE 754` dupla.

SiteWise tipo de dados	Tipo de dados OPC UA	Descrição

	`Float`
	
	`Double`	

`BOOLEAN`	`Boolean`	`true` ou `false`.

* Para tipos de dados OPC UA `UInt32` `eInt64`, seu tipo de SiteWise dados será `INTEGER` se SiteWise for capaz de representar seu valor, caso contrário, será`DOUBLE`.

Dados de saída

Esse componente grava `BatchPutAssetPropertyValue` mensagens no gerenciador de AWS IoT Greengrass streams. Para obter mais informações, consulte BatchPutAssetPropertyValue na Referência de APIs do AWS IoT SiteWise .

Arquivo de log local

Esse componente usa o seguinte arquivo de log.

Linux

```
/greengrass/v2/logs/aws.iot.SiteWiseEdgeCollectorOpcua.log
```

Windows

```
C:\greengrass\v2\logs\aws.iot.SiteWiseEdgeCollectorOpcua.log
```

Para ver os registros desse componente

- Execute o comando a seguir no dispositivo principal para visualizar o arquivo de log desse componente em tempo real. Substitua */greengrass/v2* ou *C:\greengrass\v2* pelo caminho para a pasta AWS IoT Greengrass raiz.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.iot.SiteWiseEdgeCollectorOpcua.log
  ```

Windows (PowerShell)

```
Get-Content C:\greengrass\v2\logs\aws.iot.SiteWiseEdgeCollectorOpcua.log -Tail
 10 -Wait
```

Solução de problemas e depuração

Esse componente inclui um novo registro de eventos para ajudar os clientes a identificar e corrigir problemas. O arquivo de log é separado do arquivo de log local e é encontrado no seguinte local. Substitua */greengrass/v2* ou *C:\greengrass\v2* pelo caminho para a pasta AWS IoT Greengrass raiz.

Linux

```
/greengrass/v2/work/aws.iot.SiteWiseEdgeCollectorOpcua/logs/
IotSiteWiseOpcUaCollectorEvents.log
```

Windows

```
C:\greengrass\v2\work\aws.iot.SiteWiseEdgeCollectorOpcua\logs
\IotSiteWiseOpcUaCollectorEvents.log
```

Esse registro inclui informações detalhadas e instruções de solução de problemas. As informações de solução de problemas são fornecidas junto com o diagnóstico, com uma descrição de como solucionar o problema e, às vezes, com links para mais informações. As informações de diagnóstico incluem o seguinte:

- Nível de gravidade
- Timestamp
- Informações adicionais específicas do evento

Example Log de exemplo

```
dataSourceConnectionSuccess:
  Summary: Successfully connected to OpcUa server
  Level: INFO
```

```
  Timestamp: '2023-06-15T21:04:16.303Z'
  Description: Successfully connected to the data source.
  AssociatedMetrics:
  - Name: FetchedDataStreams
    Description: The number of fetched data streams for this data source
    Value: 1.0
    Namespace: IoTSiteWise
    Dimensions:
    - Name: SourceName
      Value: SourceName{value=OPC-UA Server}
    - Name: ThingName
      Value: test-core
  AssociatedData:
  - Name: DataSourceTrace
    Description: Name of the data source
    Data:
    - OPC-UA Server
  - Name: EndpointUri
    Description: The endpoint to which the connection was attempted.
    Data:
    - '"opc.tcp://10.0.0.1:1234"'
```

Licenças

Esse componente é lançado de acordo com o Contrato de Licença de Software Principal do Greengrass.

Changelog

A tabela a seguir descreve as alterações em cada versão do componente.

Version (Versão)	Alterações
2.4.2	Correções de erros e melhorias • Corrige problemas durante a descoberta do servidor OPC UA, nos quais um nó pode ser descoberto várias vezes. • Corrige o recurso de instantâneo para garantir que o carimbo de data/hora seja novo para cada ponto de dados do instantâneo.
2.4.1	Correções de erros e melhorias • Corrige problemas relacionados ao suporte de proxy.

Version (Versão)	Alterações
	• Corrige o problema em que a limpeza do thread falhou e causou um bloqueio de dados.
2.4.0	Novos atributos • Adiciona um registro de eventos para facilitar a identificação e a correção de problemas. Correções de erros e melhorias • Corrige um problema com o cliente OPC-UA que causava erros de certificado ao se conectar a um servidor OPC-UA que usa a versão 1.05 da especificação OPC-UA.
2.3.0	Novos atributos • Adiciona suporte para a configuração do proxy HTTP do Greengrass nucleus no Linux. Correções de erros e melhorias • Corrige um problema de segurança (CVE-2019-19135).

2.2.0	Novos atributos
	• Adiciona suporte para instalação do Data Collection Pack na arquitetura Linux ARMv8.
	• Requisitos mínimos para Linux ARMv8:
	  • Memória: 4 GB
	  • CPU: ARM Cortex-A72 ou especificação equivalente
	Correções de erros e melhorias
	• Melhora o registro de métricas no processo de descoberta de nós.
	• Melhora o tratamento de tipos de dados não compatíveis.
	• Melhora o registro de erros do fluxo de dados.

Version (Versão)	Alterações
2.1.3	Novos atributos • Adiciona suporte para Windows Server 2019 ou superior. Correções de erros e melhorias • Melhora as mensagens de erro quando você implanta esse componente em dispositivos não compatíveis.

2.1.1	Novos atributos
	• Adiciona suporte para configurar as seguintes propriedades de assinatura:
	  • DataChangeTrigger- Você pode definir a condição que inicia um alerta de alteração de dados.
	  • QueueSize- A profundidade da fila em um servidor OPC-UA para uma métrica específica em que as notificações de itens monitorados são enfileiradas.
	  • PublishingIntervalMilliseconds- O intervalo (em milissegundos) de um ciclo de publicação especificado quando uma assinatura é criada.
	  • SnapshotFrequencyMilliseconds - Você pode definir a configuração de tempo limite da frequência do instantâneo para garantir que o AWS IoT SiteWise Edge ingira um fluxo constante de dados.
	• Essa versão suporta a ingestão de dados de `BAD` qualidade e filtra dados com base nas seguintes qualidades de dados:
	  • `UNCERTAIN` dados de qualidade
	  • `BAD`dados de qualidade
	Correções de erros e melhorias
	• Melhorias nas métricas do cliente.
	• Corrige a codificação de segurança que às vezes causava problemas ao se conectar a servidores com a criptografia ativada.
	• Corrige um problema em que o grupo de propriedades falhou na atualização.

2.0.3	Correções de erros e melhorias.

Version (Versão)	Alterações
2.0.2	Correções de erros e melhorias na sincronização de prioridade de ativos com o edge.
2.0.1	Versão inicial.

Consulte também

- O que é AWS IoT SiteWise? no Guia do AWS IoT SiteWise usuário.

Simulador de fonte de dados IoT SiteWise OPC-UA

O componente do simulador de fonte de dados IoT SiteWise OPC-UA (`aws.iot.SiteWiseEdgeOpcuaDataSourceSimulator`) inicia um servidor OPC-UA local que gera dados de amostra. Use esse servidor OPC-UA para simular uma fonte de dados lida pelo componente coletor IOT SiteWise OPC-UA em um gateway. AWS IoT SiteWise Em seguida, você pode explorar os AWS IoT SiteWise recursos usando esses dados de amostra. Para obter mais informações sobre AWS IoT SiteWise gateways, consulte Usando AWS IoT SiteWise na borda no Guia do AWS IoT SiteWise usuário.

Tópicos

Versões

Esse componente tem as seguintes versões:

- 1,0.x

Tipo

Este componente é um componente genérico (`aws.greengrass.generic`). O núcleo do Greengrass executa os scripts do ciclo de vida do componente.

Para ter mais informações, consulte Tipos de componentes.

Sistema operacional

Esse componente pode ser instalado em dispositivos principais que executam os seguintes sistemas operacionais:

- Linux
- Windows

Requisitos

Esse componente tem os seguintes requisitos:

- O dispositivo principal do Greengrass deve ser capaz de usar a porta 4840 no host local. O servidor OPC-UA local desse componente é executado nessa porta.

Dependências

Quando você implanta um componente, AWS IoT Greengrass também implanta versões compatíveis de suas dependências. Isso significa que você deve atender aos requisitos do componente e de todas as suas dependências para implantá-lo com êxito. Esta seção lista as dependências das versões lançadas desse componente e as restrições de versão semântica que definem as versões do componente para cada dependência. Você também pode visualizar as dependências de cada versão do componente no AWS IoT Greengrassconsole. Na página de detalhes do componente, procure a lista de dependências.

A tabela a seguir lista as dependências de todas as versões desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.3.0 <3.0.0	Rígido

Para obter mais informações sobre dependências de componentes, consulte a referência da receita do componente.

Configuração

Esse componente não tem nenhum parâmetro de configuração.

Arquivo de log local

Esse componente usa o seguinte arquivo de log.

Linux

```
/greengrass/v2/logs/aws.iot.SiteWiseEdgeOpcuaDataSourceSimulator.log
```

Windows

```
C:\greengrass\v2\logs\aws.iot.SiteWiseEdgeOpcuaDataSourceSimulator.log
```

Para ver os registros desse componente

- Execute o comando a seguir no dispositivo principal para visualizar o arquivo de log desse componente em tempo real. Substitua */greengrass/v2* ou *C:\greengrass\v2* pelo caminho para a pasta AWS IoT Greengrass raiz.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/
  aws.iot.SiteWiseEdgeOpcuaDataSourceSimulator.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs
  \aws.iot.SiteWiseEdgeOpcuaDataSourceSimulator.log -Tail 10 -Wait
  ```

Changelog

A tabela a seguir descreve as alterações em cada versão do componente.

Version (Versão)	Alterações
1.0.0	Versão inicial. Adiciona suporte para Windows Server 2016 ou superior.

Consulte também

- O que éAWS IoT SiteWise? no Guia do AWS IoT SiteWise usuário.

Editora de IoT SiteWise

O componente SiteWise editor de IoT (`aws.iot.SiteWiseEdgePublisher`) permite que os AWS IoT SiteWise gateways exportem dados da borda para o. Nuvem AWS

Para obter mais informações sobre AWS IoT SiteWise gateways, consulte Usando AWS IoT SiteWise na borda no Guia do AWS IoT SiteWise usuário.

Tópicos

Versões

Esse componente tem as seguintes versões:

- 3.1.x
- 3.0.x
- 2.4.x
- 2.3.x
- 2.2.x
- 2.1.x
- 2.0.x

Tipo

Este componente é um componente genérico (`aws.greengrass.generic`). O núcleo do Greengrass executa os scripts do ciclo de vida do componente.

Para ter mais informações, consulte Tipos de componentes.

Sistema operacional

Esse componente pode ser instalado em dispositivos principais que executam os seguintes sistemas operacionais:

- Linux
- Windows

Requisitos

Esse componente tem os seguintes requisitos:

- O dispositivo principal do Greengrass deve ser executado em uma das seguintes plataformas:
  - SO: Ubuntu 18.04 ou posterior

    Arquitetura: x86_64 (AMD64) ou ARMv8 (Aarch64)
  - SO: Red Hat Enterprise Linux (RHEL) 8

    Arquitetura: x86_64 (AMD64) ou ARMv8 (Aarch64)

- SO: Amazon Linux 2

  Arquitetura: x86_64 (AMD64) ou ARMv8 (Aarch64)

- SO: Debian 11

  Arquitetura: x86_64 (AMD64) ou ARMv8 (Aarch64)

- SISTEMA OPERACIONAL: Windows Server 2019 ou posterior

  Arquitetura: x86_64 (AMD64)

- O dispositivo principal do Greengrass deve se conectar à Internet.
- O dispositivo principal do Greengrass deve estar autorizado a realizar a ação`iotsitewise:BatchPutAssetPropertyValue`. Para obter mais informações, consulte Autorizar dispositivos principais a interagir com os AWS serviços.

  Example política de permissões

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

Endpoints e portas

Esse componente deve ser capaz de realizar solicitações de saída para os seguintes endpoints e portas, além dos endpoints e portas necessários para a operação básica. Para ter mais informações, consulte Permitir o tráfego de dispositivos por meio de um proxy ou firewall.

Endpoint	Porta	Obrigatório	Descrição
`data.iotsitewise.` *`region`*`.amazonaws.com`	443	Sim	Publique dados em

Endpoint	Porta	Obrigatório	Descrição
			AWS IoT SiteWise.

Dependências

Quando você implanta um componente, AWS IoT Greengrass também implanta versões compatíveis de suas dependências. Isso significa que você deve atender aos requisitos do componente e de todas as suas dependências para implantá-lo com êxito. Esta seção lista as dependências das versões lançadas desse componente e as restrições de versão semântica que definem as versões do componente para cada dependência. Você também pode visualizar as dependências de cada versão do componente no AWS IoT Greengrass console. Na página de detalhes do componente, procure a lista de dependências.

A tabela a seguir lista as dependências das versões 2.0.x a 2.2.x desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Núcleo Greengrass	>=2.3.0<3.0.0	Rígido
Gerenciador de fluxo	>=2.0.10<3.0.0	Rígido

Para obter mais informações sobre dependências de componentes, consulte a referência da receita do componente.

Configuração

Esse componente não tem nenhum parâmetro de configuração.

Você pode usar o AWS IoT SiteWise console ou a API para configurar o componente SiteWise Editor de IoT. Para obter mais informações, consulte Etapa 3: Configurar o editor - opcional no Guia AWS IoT SiteWise do usuário.

Dados de entrada

Esse componente lê `PutAssetPropertyValueEntry` mensagens do gerenciador de AWS IoT Greengrass fluxo. Para obter mais informações, consulte PutAssetPropertyValueEntrya Referência AWS IoT SiteWise da API.

Arquivo de log local

Esse componente usa o seguinte arquivo de log.

Linux

```
/greengrass/v2/logs/aws.iot.SiteWiseEdgePublisher.log
```

Windows

```
C:\greengrass\v2\logs\aws.iot.SiteWiseEdgePublisher.log
```

Para ver os registros desse componente

- Execute o comando a seguir no dispositivo principal para visualizar o arquivo de log desse componente em tempo real. Substitua */greengrass/v2* ou *C:\greengrass\v2* pelo caminho para a pasta AWS IoT Greengrass raiz.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.iot.SiteWiseEdgePublisher.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\aws.iot.SiteWiseEdgePublisher.log -Tail 10 -
  Wait
  ```

Solução de problemas e depuração

Esse componente inclui um novo registro de eventos para ajudar os clientes a identificar e corrigir problemas. O arquivo de log é separado do arquivo de log local e é encontrado no seguinte local. Substitua */greengrass/v2* ou *C:\greengrass\v2* pelo caminho para a pasta AWS IoT Greengrass raiz.

Linux

```
/greengrass/v2/work/aws.iot.SiteWiseEdgePublisher/logs/
IotSiteWisePublisherEvents.log
```

Windows

```
C:\greengrass\v2\work\aws.iot.SiteWiseEdgePublisher\logs
\IotSiteWisePublisherEvents.log
```

Esse registro inclui informações detalhadas e instruções de solução de problemas. As informações de solução de problemas são fornecidas junto com o diagnóstico, com uma descrição de como solucionar o problema e, às vezes, com links para mais informações. As informações de diagnóstico incluem o seguinte:

- Nível de gravidade
- Timestamp
- Informações adicionais específicas do evento

Example Log de exemplo

```
accountBeingThrottled:
  Summary: Data upload speed slowed due to quota limits
  Level: WARN
  Timestamp: '2023-06-09T21:30:24.654Z'
  Description: The IoT SiteWise Publisher is limited to the "Rate of data points
 ingested"
    quota for a customers account. See the associated documentation and associated
    metric for the number of requests that were limited for more information. Note
    that this may be temporary and not require any change, although if the issue
 continues
    you may need to request an increase for the mentioned quota.
  FurtherInformation:
  - https://docs.aws.amazon.com/iot-sitewise/latest/userguide/quotas.html
  - https://docs.aws.amazon.com/iot-sitewise/latest/userguide/troubleshooting-
gateway.html#gateway-issue-data-streams
  AssociatedMetrics:
  - Name: TotalErrorCount
    Description: The total number of errors of this type that occurred.
```

```
    Value: 327724.0
  AssociatedData:
  - Name: AggregatePropertyAliases
    Description: The aggregated property aliases of the throttled data.
    FileLocation: /greengrass/v2/work/aws.iot.SiteWiseEdgePublisher/./logs/data/
AggregatePropertyAliases_1686346224654.log
```

Licenças

Esse componente é lançado de acordo com o Contrato de Licença de Software Principal do Greengrass.

Changelog

A tabela a seguir descreve as alterações em cada versão do componente.

Version (Versão)	Alterações
3.1.3	Correções de erros e melhorias • Resolveu um problema em que o arquivo de registro de eventos localizado em `/greengrass/v2/work/aws.iot.SiteWiseEdgePublisher/logs/IoTSiteWisePublisherEvents.log` foi criado, mas nenhum evento estava sendo registrado. • Foram adicionadas as seguintes CloudWatch métricas para monitorar a conexão com o broker MQTT: • `IoTSiteWisePublisher.IsConnectedToMqttBroker` • `IoTSiteWisePublisher.NumberOfSubscriptionsToMqttBroker` • `IoTSiteWisePublisher.NumberOfUniqueMqttTopicsReceived` • `IoTSiteWisePublisher.MqttMessageReceivedSuccessCount` • `IoTSiteWisePublisher.MqttReceivedSuccessBytes` Para obter mais informações sobre essas métricas, consulte métricas do AWS IoT Greengrass Version 2 gateway.

Version (Versão)	Alterações
	• Resolveu um problema em que a `BatchCreateJob` API ainda seria chamada mesmo se o upload de um arquivo de parquet para o S3 falhasse.
3.1.2	Correções de erros e melhorias • Corrigido o problema de alto uso da CPU introduzido na versão 3.1.1.
3.1.1	Correções de erros e melhorias • Adiciona registros adicionais que identificam os aliases de dados afetados quando ocorre um erro. • Adiciona a aplicação local dos limites da AWS IoT SiteWise API à idade dos dados ingeridos. • Corrige o problema em que o Publisher mistura os pontos de verificação dos StreamManager streams quando há vários destinos do Amazon S3. • Corrige o gargalo de desempenho na forma como o Publisher lê os StreamManager fluxos.
3.1.0	Novos atributos • Adiciona suporte para publicar dados como arquivos em parquet no Amazon S3. • Adiciona suporte para ingestão em AWS IoT SiteWise buffer.
3.0.0	Correções de erros e melhorias • Corrige problemas relacionados ao suporte de proxy. Novos atributos • Permite o suporte à ingestão de dados de um corretor MQTT.
2.4.1	Correções de erros e melhorias • Habilite o componente para funcionar com o Java Corretto 11 versões 11.0.20.8.1 e posteriores. As versões 2.4.0 e 2.3.3 do component e mostram a mensagem de `"Could not find or load main class"` erro quando usadas com o Java Corretto versão 11.0.20.8.1.

Version (Versão)	Alterações
2.4.0	Novos atributos • Adiciona um novo registro de eventos para facilitar a identificação e a correção de problemas. Correções de erros e melhorias • Melhora a recuperação do checkpoint do Publisher.
2.3.3	Correções de erros e melhorias • Melhora a capacidade de suportar alto rendimento.
2.3.2	Correções de erros e melhorias • Corrige o suporte ao proxy HTTP ao baixar a configuração do Publisher.

2.3.1	Novos atributos
	• Adiciona suporte para instalação do Data Collection Pack na arquitetura Linux ARMv8.
	• Requisitos mínimos para Linux ARMv8:
	  • Memória: 4 GB
	  • CPU: ARM Cortex-A72 ou especificação equivalente

2.2.3	Correções de erros e melhorias • Remove a repetição da exceção genérica que não estava na lista de exceções recuperáveis.
2.2.2	Correções de erros e melhorias • Reintroduz o suporte ao upload de dados AWS IoT SiteWise por meio de um servidor proxy HTTP.

Version (Versão)	Alterações
2.2.1	**Note** Esta versão não oferece suporte à configuração de proxy HTTP. A versão 2.2.2 e superior reintroduz o suporte para esse recurso. Novos atributos • Adiciona suporte a esse componente para alternar a compactação ao fazer upload de dados para o. AWS IoT SiteWise

2.2.0	**Note**
	Esta versão não oferece suporte à configuração de proxy HTTP. A versão 2.2.2 e superior reintroduz o suporte para esse recurso.
	
	Novos atributos
	• Atualiza esse componente para compactar dados antes de enviá-los ao AWS IoT SiteWise serviço.
	  • Na maioria dos casos, essa alteração reduz o uso da largura de banda em 75 por cento em comparação com as versões anteriores desse componente.
	  • Na maioria dos casos, essa alteração aumenta o uso da CPU em até 5%. Em gateways que processam grandes quantidades de dados, essa alteração pode aumentar o uso da CPU em até 15%.
	  • Essa alteração não afeta as taxas AWS IoT SiteWise de serviço ou o uso da cota de serviço.
	  • Adiciona suporte para Windows Server 2019 ou superior.
	Correções de erros e melhorias
	• Corrige um problema que impede que esse componente seja iniciado quando o arquivo do ponto de verificação está corrompido.

Version (Versão)	Alterações
2.1.4	Correções de erros e melhorias • Corrige a compatibilidade com a versão 8 do Java.
2.1.3	**Warning** Essa versão não está mais disponível, exceto nas regiões Leste dos EUA (Ohio), Canadá (Central) e AWS GovCloud (Leste dos EUA). Essa versão do componente requer a versão 11 ou superior do Java para ser executada. As melhorias nesta versão estão disponíveis em versões posteriores desse componente. Correções de erros e melhorias • Melhora as mensagens de erro quando você implanta esse componente em dispositivos não compatíveis. • Atualizações para registrar erros quando os carregamentos de dados falham.
2.1.2	Correções de erros e melhorias • Atualizações para invocar o recurso de exportação de dados expirados assim que os dados expirarem.
2.1.1	Correções de erros e melhorias.
2.1.0	Novos atributos • Adiciona suporte para publicar primeiro os dados mais recentes na nuvem. • Adiciona suporte para não publicar dados expirados na nuvem. • Adiciona suporte para armazenar dados expirados localmente. Correções de erros e melhorias • Reduz a E/S do disco e a latência correspondente.
2.0.2	Correções de erros e melhorias.

Version (Versão)	Alterações
2.0.1	Versão inicial.

Consulte também

- O que é AWS IoT SiteWise? no Guia do AWS IoT SiteWise usuário.

Processador de IoT SiteWise

O componente do SiteWise processador de IoT (`aws.iot.SiteWiseEdgeProcessor`) permite que os AWS IoT SiteWise gateways processem dados na borda.

Com esse componente, os AWS IoT SiteWise gateways podem usar modelos e ativos de ativos para processar dados em dispositivos de gateway. Para obter mais informações sobre AWS IoT SiteWise gateways, consulte Usando AWS IoT SiteWise na borda no Guia do AWS IoT SiteWise usuário.

Tópicos

Versões

Esse componente tem as seguintes versões:

- 3.2.x
- 3.1.x

- 3.0.x
- 2.2.x
- 2.1.x
- 2.0.x

Tipo

Este componente é um componente genérico (`aws.greengrass.generic`). O núcleo do Greengrass executa os scripts do ciclo de vida do componente.

Para ter mais informações, consulte Tipos de componentes.

Sistema operacional

Esse componente pode ser instalado em dispositivos principais que executam os seguintes sistemas operacionais:

- Linux
- Windows

Requisitos

Esse componente tem os seguintes requisitos:

- O dispositivo principal do Greengrass deve ser executado em uma das seguintes plataformas:
  - SISTEMA OPERACIONAL: Ubuntu 20.04 ou 18.04

    Arquitetura: x86_64 (AMD64)
  - SO: Red Hat Enterprise Linux (RHEL) 8

    Arquitetura: x86_64 (AMD64)
  - SO: Amazon Linux 2

    Arquitetura: x86_64 (AMD64)
  - SISTEMA OPERACIONAL: Windows Server 2019 ou posterior

    Arquitetura: x86_64 (AMD64)
- O dispositivo principal do Greengrass deve permitir tráfego de entrada na porta 443.

- O dispositivo principal do Greengrass deve permitir tráfego de saída nas portas 443 e 8883.
- As seguintes portas são reservadas para uso por AWS IoT SiteWise: 80, 443, 3001, 4569, 4572, 8000, 8081, 8082, 8084, 8085, 8086, 8445, 9000, 9500, 11080 e 50010. Usar uma porta reservada para tráfego pode resultar no encerramento de uma conexão.

  > **Note**
  >
  > A porta 8087 é necessária somente para a versão 2.0.15 e posterior desse componente.

- A função de dispositivo do Greengrass deve ter permissões que permitam que você use AWS IoT SiteWise gateways em seus dispositivos. AWS IoT Greengrass V2 Para obter mais informações, consulte Requisitos no Guia AWS IoT SiteWise do usuário.

Endpoints e portas

Esse componente deve ser capaz de realizar solicitações de saída para os seguintes endpoints e portas, além dos endpoints e portas necessários para a operação básica. Para ter mais informações, consulte Permitir o tráfego de dispositivos por meio de um proxy ou firewall.

Endpoint	Porta	Obrigatório	Descrição
`model.iotsitewise.region.amazonaws.com`	443	Sim	Obtenha informações sobre seus AWS IoT SiteWise ativos e modelos de ativos.
`edge.iotsitewise.region.amazonaws.com`	443	Sim	Obtenha informações sobre a configuração do AWS IoT

Endpoint	Porta	Obrigatório	Descrição
			SiteWise gateway do dispositivo principal.
`ecr.region.amazonaws.com`	443	Sim	Baixe imagens do AWS IoT SiteWise Edge Gateway Docker do Amazon Elastic Container Registry.
`iot.region.amazonaws.com`	443	Sim	Obtenha endpoints de dispositivos para seu. Conta da AWS
`sts.region.amazonaws.com`	443	Sim	Obtenha o ID do seu Conta da AWS.

Endpoint	Porta	Obrigatório	Descrição
`monitor.iotsitewise. region.amazonaws.com`	443	Não	Obrigatório se você acessar AWS IoT SiteWise Monitor portais no dispositivo principal.

Dependências

Quando você implanta um componente, AWS IoT Greengrass também implanta versões compatíveis de suas dependências. Isso significa que você deve atender aos requisitos do componente e de todas as suas dependências para implantá-lo com êxito. Esta seção lista as dependências das versões lançadas desse componente e as restrições de versão semântica que definem as versões do componente para cada dependência. Você também pode visualizar as dependências de cada versão do componente no AWS IoT Greengrass console. Na página de detalhes do componente, procure a lista de dependências.

A tabela a seguir lista as dependências das versões 2.0.x a 2.1.x desse componente.

Dependência	Versões compatíveis	Tipo de dependência
Serviço de troca de tokens	>=2.0.3 <3.0.0	Rígido
Gerenciador de fluxo	>=2.0.10 <3.0.0	Rígido
CLI do Greengrass	>=2.3.0 <3.0.0	Rígido

Para obter mais informações sobre dependências de componentes, consulte a referência da receita do componente.

Configuração

Esse componente não tem nenhum parâmetro de configuração.

Arquivo de log local

Esse componente usa o seguinte arquivo de log.

Linux

```
/greengrass/v2/logs/aws.iot.SiteWiseEdgeProcessor.log
```

Windows

```
C:\greengrass\v2\logs\aws.iot.SiteWiseEdgeProcessor.log
```

Para ver os registros desse componente

- Execute o comando a seguir no dispositivo principal para visualizar o arquivo de log desse componente em tempo real. Substitua */greengrass/v2* ou *C:\greengrass\v2* pelo caminho para a pasta AWS IoT Greengrass raiz.

  Linux

  ```
  sudo tail -f /greengrass/v2/logs/aws.iot.SiteWiseEdgeProcessor.log
  ```

  Windows (PowerShell)

  ```
  Get-Content C:\greengrass\v2\logs\aws.iot.SiteWiseEdgeProcessor.log -Tail 10 -
  Wait
  ```

Licenças

Esse componente inclui o seguinte software/licenciamento de terceiros:

- Apache-2.0
- MIT
- Cláusula BSD-2

- Cláusula BSD-3
- CDDL-1.0
- CDDL-1.1
- DISCO
- Zlib
- GPL-3.0 com exceção do GCC
- Domínio público
- Python-2.0
- Unicode-DFS-2015
- Cláusula BSD-1
- OpenSSL
- EPL-1.0
- EPL-2.0
- GPL-2.0 com exceção ClassPath
- MPL-2.0
- CC0-1.0
- JSON

Esse componente é lançado de acordo com o Contrato de Licença de Software Principal do Greengrass.

Changelog

A tabela a seguir descreve as alterações em cada versão do componente.

Version (Versão)	Alterações
3.2.1	Correções de erros e melhorias • Corrige o problema em que as chamadas de AWS IoT SiteWise API não paginam de forma síncrona com o Edge. SiteWise • Corrija o problema de não publicar mais a `MessageRemaining.SiteWise_Edge_Stream` métrica.

Version (Versão)	Alterações

	• Foram adicionadas as seguintes CloudWatch métricas para monitorar a conexão com o broker MQTT.
	  • `IoTSiteWiseProcessor.IsConnectedToMqttBroker`
	  • `IoTSiteWiseProcessor.NumberOfSubscriptionsToMqttBroker`
	  • `IoTSiteWiseProcessor.NumberOfUniqueMqttTopicsReceived`
	  • `IoTSiteWiseProcessor.MqttMessageReceivedSuccessCount`
	  • `IoTSiteWiseProcessor.MqttReceivedSuccessBytes`
	  Para obter mais informações sobre essas métricas, consulte métricas do AWS IoT Greengrass Version 2 gateway.

Version (Versão)	Alterações
3.2.0	Melhorias na performance • Otimize os serviços de API para ocupar menos memória e exigir menos espaço em disco para instalação • Isso proporciona uma redução de 2 GB no uso inicial da memória (agora usa 7,5 GB de memória na inicialização, no entanto, 16 GB ainda é recomendado) e uma redução de 500 MB no tamanho do download (agora requer um download de 1,4 GB) para todo o componente. Novos atributos • `GetAssetPropertyValueAggregates` A API agora oferece suporte a janelas de agregação de 15 minutos na borda. • As portas 8081 e 8082 não precisam mais estar disponíveis para que esse componente seja executado corretamente. **Note** O endpoint local para APIs do plano de AWS IoT SiteWise dados, como`get-asset-property-value` , está sendo alterado de `http://localhost:8081` para. `http://localhost:11080/data` O endpoint local para APIs do plano de AWS IoT SiteWise controle, como`list-asset-models`, está sendo alterado de `http://localhost:11080` para. `http://localhost:11080/control` AWS sempre recomenda que você use os endpoints HTTPS do gateway SiteWise Edge. Esses endpoints não mudaram. Correções de erros e melhorias • A sincronização de agora AWS IoT SiteWise transformará os recursos em um estado válido se a sincronização anterior tiver sido interrompida. Isso corrigirá problemas com alguns recursos corrompidos após uma reinicialização forçada. • Corrige uma condição rara em que um recurso pode ser corrompido na borda se for modificado durante a sincronização. A sincronização agora

Version (Versão)	Alterações
	falhará se essa condição for detectada, e o recurso será repetido na próxima sincronização. • Corrige um problema que poderia ter permitido que o endpoint HTTP das APIs fosse chamado externamente. Somente HTTPS pode ser usado para chamar APIs fora do endereço de loopback local agora. • `ListAssets` A API agora mostra as hierarquias de ativos para ativos armazenados na borda. • Corrige um problema em que o Pacote de Processamento de Dados falhou ao reiniciar, atualizar ou fazer o downgrade no Windows. • Corrige um bug no Pacote de Processamento de Dados para o sistema operacional Windows que impedia que os clientes usassem credenciais para se conectar a um MQTT Broker.
3.1.3	Correções de erros e melhorias • Corrige o problema em que o Data Processing Pack relatou incorreta mente uma sincronização bem-sucedida quando alguns dos recursos realmente falharam. • Permita que vários ativos tenham o mesmo nome, desde que não tenham o mesmo pai.
3.1.1	Correções de erros e melhorias • Corrige o problema em que a solicitação SigV4 falha devido a uma incompatibilidade de fuso horário. • Corrige um problema em que as propriedades de transformação e métrica param de ser calculadas quando dependem de atributos após a reinicialização. • Ative o suporte à configuração personalizada da porta do Stream Manager. • Corrija um problema em que as propriedades sincronizadas com a borda podem parar de ser atualizadas.

Version (Versão)	Alterações
3.1.0	Correções de erros e melhorias • Corrige o problema em que a `ListAssetModels` API não consegue gerar o próximo token.
3.0.0	Novos atributos • Permite o suporte à ingestão de dados de um corretor MQTT.
2.2.1	Correções de erros e melhorias • Ajuste o processo de sincronização para tornar o armazenamento de dados do plano de controle mais consistente com a forma como a nuvem opera. Isso afeta um pouco a atualização. **Note** Os dados do plano de controle sincronizados na versão 2.2.1 ou superior não serão compatíveis com as versões anteriores. Para fazer o downgrade para versões anteriores, você precisará concluir uma nova instalação. Isso não afeta as atualizações. Os dados sincronizados nas versões anteriores funcionarão com a versão 2.2.1. • Modificações adicionais na cadeia de AWS credenciais para AWS IoT Greengrass V2 priorizar as credenciais.

Version (Versão)	Alterações
2.1.37	Correções de erros e melhorias • Desative o dependency-routing-service processo e transfira sua funcionalidade para o property-state-service processo para reduzir o uso de recursos dos processos de comunicação. • Aumente o limite máximo de resultados da `get-asset-property-value-history` API para 20.000 para corresponder ao limite usado por AWS IoT SiteWise. • Corrija um problema em que o próximo token não estava sendo fornecido nos resultados paginados da `get-asset-property-value-history` API quando nenhum limite máximo de resultados era especificado.
2.1.35	Correções de erros e melhorias • Modifica a cadeia de AWS credenciais para AWS IoT Greengrass prioriz ar as credenciais. • Corrige um problema com a detecção de contas ao implantar como parte de um grupo AWS IoT Thing.
2.1.34	Correções de erros e melhorias • Ajusta os cálculos de métrica/transformação para usar multithreading no Linux. O Windows continua executando cálculos de thread único para fins de compatibilidade. • Corrige um problema em que os cálculos métricos estariam ausentes em algumas janelas de computação.
2.1.3	Correções de erros e melhorias • Corrige um problema com o relatório do estado de erro para o console do Greengrass.
2.1.32	Correções de erros e melhorias • Adiciona suporte para grupos e nomes de usuário personalizados.

Version (Versão)	Alterações
2.1.31	Correções de erros e melhorias • Adiciona suporte para calcular a média ponderada pelo tempo e o desvio padrão ponderado pelo tempo para dados modelados. AWS IoT SiteWise
2.1.29	Correções de erros e melhorias • Adiciona suporte para filtrar ativos na funcionalidade de borda.
2.1.28	Correções de erros e melhorias • Otimiza a sincronização de recursos para permitir que um grande número de ativos seja sincronizado da borda Nuvem AWS até a borda.
2.1.24	Correções de erros e melhorias • Corrige um problema que fazia com que o painel desaparecesse ao sincronizar um recurso pela segunda vez.
2.1.23	Correções de erros e melhorias • Foi adicionado um tempo limite para o processo de `aws.iot.SiteWiseEdgeProcessor` instalação para evitar falhas na instalação se a conectividade com a Internet estiver lenta. • Sincronização otimizada de recursos para melhorar a eficiência da sincronização entre a nuvem e a borda.
2.1.21	**Warning** A atualização de 2.0.x para 2.1.x resultará na perda de dados locais. Novos atributos • Adiciona suporte para Windows Server 2019 ou superior. • Remove o docker para sistemas operacionais baseados em Linux.
2.0.16	Esta versão contém correções de erros e melhorias.

Version (Versão)	Alterações
2.0.15	Correções de erros e melhorias • Altera a porta que esse componente usa para operações da API de sincronização de recursos de 8085 para 8087. Como resultado, esse componente agora exige que a porta 8087 esteja disponível. Esse componente ainda exige que a porta 8085 esteja disponível. • Atualiza a AWS OpsHub autenticação para negar usuários não autorizados durante o login, em vez de quando um usuário tenta chamar operações de API.
2.0.14	Esta versão contém correções de erros e melhorias.
2.0.13	Correções de erros e melhorias • Corrige um problema para que, quando esse componente relata dados para CloudWatch as métricas da Amazon, agora indique corretamente quais dados não foram modelados.
2.0.9	Correções de erros e melhorias • Melhora a confiabilidade para criar e atualizar AWS IoT SiteWise recursos no dispositivo principal. • Adiciona operações de API locais adicionais que você pode usar para monitorar quais componentes estão instalados no dispositivo principal, a versão de cada componente e o status de cada componente. Você pode ver essas informações na guia Configurações do AWS IoT SiteWise aplicativo AWS OpsHub for no dispositivo principal. • Adiciona um status de integridade para os contêineres do Docker que esse componente executa. Você pode executar o `docker ps` comando para ver o status de saúde dos contêineres.
2.0.7	Correções de erros e melhorias • Corrige o suporte para visualização de AWS IoT SiteWise Monitor portais no dispositivo principal.

Version (Versão)	Alterações
2.0.6	Correções de erros e melhorias • Corrige as `latest()` funções AWS IoT SiteWise `statetime() earliest()` , e que esse componente computa no dispositivo principal.
2.0.5	Correções de erros e melhorias • Adiciona suporte para a AWS IoT SiteWise `pretrigger()` função nas transformações que esse componente computa no dispositivo principal. • Altera o caminho em que esse componente armazena a configuração do Lightweight Directory Access Protocol (LDAP) para autenticação.
2.0.2	Versão inicial.

Consulte também

- O que é AWS IoT SiteWise? no Guia do AWS IoT SiteWise usuário.

Componentes compatíveis com o editor

Os componentes compatíveis com o editor estão em uma versão prévia AWS IoT Greengrass e estão sujeitos a alterações. Esses componentes não são suportados peloAWS. Você deve entrar em contato com o editor para qualquer problema com cada um dos componentes.

Os componentes suportados pelo Greengrass Publisher são desenvolvidos, oferecidos e atendidos por fornecedores de componentes terceirizados. Os fornecedores terceirizados de componentes são do AWS Partner Device Catalog, AWS Heroes ou fornecedores da comunidade. Você pode comprar os componentes deste catálogo entrando em contato diretamente com o fornecedor terceirizado de componentes.

Os componentes suportados pelo Greengrass Publisher incluem o seguinte:

Tópicos

- AIShield.Edge
- EdgeLabs Sensor AI
- Investidor Greengrass S3

AIShield.Edge

Este componente foi desenvolvido e é suportado pelo AiShield, desenvolvido pela Bosch. Aumente sua segurança de IA com AIShield.edge. Esse componente foi projetado para implantar perfeitamente defesas personalizadas e informadas sobre ameaças em dispositivos de ponta, o que protege seus dispositivos contra ataques de IA.

Esse componente oferece os seguintes benefícios:

- Faça a transição perfeita da análise de vulnerabilidade com o AiShield AI Security para defesas de ponta reforçadas em AWS
- Implemente defesas personalizadas em vários dispositivos periféricos com facilidade
- Ampla proteção adaptada a diversas configurações de IA que oferece suporte a vários tipos de modelos e estruturas
- Mantenha-se atualizado com a integração perfeita com os fluxos de trabalho do Amazon SageMaker Greengrass
- Obtenha insights imediatos sobre possíveis ameaças, com dados retransmitidos diretamente para AWS IoT Core
- Um caminho coeso de segurança de IA para implantação de defesa na periferia do AiShield AI Security on the Marketplace AWS

Esse componente deve ser executado na seguinte plataforma:

- SO: Linux

Se você estiver interessado em comprar esse componente, entre em contato com a Bosch Software and Digital Solutions: `<AIShield.Contact@bosch.com>`.

EdgeLabs Sensor AI

Esse componente foi desenvolvido e é suportado pela IA EdgeLabs. O AI EdgeLabs Sensor é um aplicativo baseado em contêiner que contém recursos de detecção e prevenção de ameaças

baseados em IA. O AI Sensor é encapsulado em um componente do Greengrass e implantado como um contêiner independente no dispositivo principal junto com outros componentes do Greengrass.

Esse componente atual é um agente baseado em contêiner que verifica continuamente a comunicação de rede e procura padrões de ameaças no software executado no Edge Host ou no gateway de IoT. Esse componente usa o eBPF, a verificação comportamental da largura de banda dos processos e a configuração baseada em host. A principal funcionalidade desse componente é baseada nas funções NDR/IPS e EDR.

Esse componente oferece os seguintes benefícios:

- Detecção de ameaças baseada em IA contra ataques de rede e malware (EDR/NDR)
- Resposta automatizada a incidentes baseada em IA (IPS)
- Hospede inteligência local sobre ameaças com o mínimo de transferência de dados para fora
- Implantação leve com Docker e Greengrass

Esse componente deve ser executado em uma das seguintes plataformas:

- SO: Linux

Se você estiver interessado em comprar este componente, entre em contato com a AI EdgeLabs: `<contact@edgelabs.ai>`.

Investidor Greengrass S3

Esse componente foi desenvolvido e é apoiado por Nathan Glover. O componente Greengrass S3 Ingestor foi projetado para ser usado com o componente gerenciador de fluxo. Esse componente pega um fluxo delimitado por linhas de mensagens JSON do gerenciador de fluxos e as agrupa em um arquivo GZIP. Esse componente permite a ingestão eficiente de dados no Amazon S3 para processamento adicional ou armazenamento. Esse componente não suporta o envio de dados para o Nuvem AWS em tempo real.

Esse componente deve ser executado em uma das seguintes plataformas:

- SO: Linux
- SISTEMA OPERACIONAL: Windows

`<Se você estiver interessado em comprar este componente, entre em contato com Na`

Componentes da comunidade

O Catálogo de Software do Greengrass é um índice dos componentes do Greengrass desenvolvidos pela comunidade do Greengrass. A partir desse catálogo, você pode baixar, modificar e implantar componentes para criar seus aplicativos Greengrass. Você pode ver o catálogo no seguinte link: https://github.com/aws-greengrass/ aws-greengrass-software-catalog.

Cada componente tem um GitHub repositório público que você pode explorar. Veja o Catálogo de Software Greengrass em GitHub para encontrar a lista completa dos componentes da comunidade. Por exemplo, esse catálogo inclui os seguintes componentes:

- Amazon Kinesis Video Streams

  Esse componente ingere fluxos de áudio e vídeo de câmeras locais que usam o Real Time Streaming Protocol (RTSP). Em seguida, o componente carrega os streams de áudio e vídeo para o Amazon Kinesis Video Streams.
- Gateway IoT Bluetooth

  Esse componente usa a BluePybiblioteca que permite a comunicação com dispositivos Bluetooth Low Energy (LE) para criar interfaces de cliente Bluetooth LE.
- Rotador de certificados

  Esse componente fornece um meio de alternar o certificado do dispositivo AWS IoT Greengrass principal e a chave privada em toda a sua frota, em grande escala.
- Tunelamento seguro em contêineres

  Esse componente fornece um contêiner Docker para tunelamento seguro com todas as dependências e bibliotecas correspondentes em uma receita reutilizável que não depende de um sistema operacional host específico.
- Grafana

  Esse componente permite que você hospede um servidor Grafana em um dispositivo principal do Greengrass. Você pode usar os painéis do Grafana para visualizar e gerenciar dados no dispositivo principal.
- GStreamer para Amazon Lookout for Vision

  Esse componente fornece um plug-in do GStreamer para que você possa realizar a detecção de anomalias do Lookout for Vision em seus pipelines personalizados do GStreamer.

- Assistente doméstico

  Esse componente permite que o cliente use o Home Assistant para fornecer controle local de dispositivos domésticos inteligentes. Ele fornece integração com AWS serviços na borda e na nuvem para fornecer soluções de automação residencial que ampliam o Home Assistant.

- Painel InfluxDBGrafana

  Esse componente fornece uma experiência de um clique para configurar os componentes InfluxDB e Grafana. Ele conecta o InfluxDB ao Grafana e automatiza a configuração de um painel local do Grafana que renderiza a telemetria em tempo real. AWS IoT Greengrass

- InfluxDB

  Esse componente fornece um banco de dados de séries temporais do InfluxDB em um dispositivo principal do Greengrass. Você pode usar esse componente para processar dados de sensores de IoT, analisar dados em tempo real e monitorar operações na borda.

- Editora InfluxDB

  Esse componente retransmite a telemetria de integridade AWS IoT Greengrass do sistema do plug-in do emissor Nucleus para o InfluxDB. Esse componente também pode encaminhar telemetria personalizada para o InfluxDB.

- Estrutura pubsub de IoT

  Essa estrutura fornece uma arquitetura de aplicativo, código de modelo e exemplos implantáveis que ajudam a melhorar a qualidade do código para aplicativos pubsub de IoT distribuídos e orientados por eventos usando componentes personalizados v2. AWS IoT Greengrass Para ter mais informações, consulte Crie AWS IoT Greengrass componentes.

- Laboratórios Jupyter

  Esse componente é implantado JupyterLab em um dispositivo AWS IoT Greengrass principal. O ambiente Jupyter tem acesso aos recursos variáveis de processo e ambiente definidos por AWS IoT Greengrass, simplificando o processo de teste e desenvolvimento de componentes escritos em Python.

- Servidor web local

  Esse componente permite que você crie uma interface de usuário da web local em um dispositivo principal do Greengrass. Você pode criar uma interface de usuário da Web local que permita definir as configurações do dispositivo e do aplicativo ou monitorar o dispositivo, por exemplo.

- LoRaWaAdaptador de protocolo N

  Esse componente ingere dados de dispositivos sem fio locais que usam o protocolo LoRaWa N, que é um protocolo de rede de área ampla de baixa potência (LPWAN). O componente permite que você analise e atue nos dados localmente sem se comunicar com a nuvem.

- Modbus TCP

  Esse componente coleta dados de dispositivos locais usando o protocolo ModbusTCP e os publica em fluxos de dados selecionados.

- Node-red

  Esse componente instala o Node-RED em um dispositivo AWS IoT Greengrass principal usando o NPM. O componente depende do componente Node-RED Auth, que deve ser implantado e configurado explicitamente. Você pode usar a CLI do Node-RED para Greengrass para implantar fluxos do Node-RED em dispositivos. AWS IoT Greengrass

- Docker Node-RED

  Esse componente instala o Node-RED no dispositivo AWS IoT Greengrass principal usando o contêiner Docker oficial do Node-RED. O componente depende do componente Node-RED Auth, que deve ser implantado e configurado explicitamente. Você pode usar a CLI do Node-RED para Greengrass para implantar fluxos do Node-RED em dispositivos. AWS IoT Greengrass

- Autenticação Node-RED

  Esse componente configura um nome de usuário e uma senha para proteger a instância do Node-RED em execução em um AWS IoT Greengrass dispositivo principal.

- OpenThreadRoteador de fronteira

  Esse componente implanta o contêiner OpenThread Border Router Docker. O componente ajuda a compor um dispositivo Matter que inclui um roteador Thread border.

- Conector de dados de streaming OSI Pi

  Esse componente fornece streaming de ingestão de dados em tempo real do OSI Pi Data Archive para uma arquitetura de dados moderna em. AWS Ele se integra ao OSI Pi Asset Framework, que é gerenciado centralmente por meio de mensagens. AWS IoT PubSub

- Provedor de Parsec

Esse componente permite que AWS IoT Greengrass os dispositivos integrem soluções de segurança de hardware usando o projeto Parsec de código aberto da Cloud Native Computing Foundation (CNCF).

- Banco de dados PostgreSQL

  Esse componente fornece suporte para o banco de dados relacional PostgreSQL na borda. Os clientes podem usar esse componente para provisionar e gerenciar uma instância local do PostgreSQL dentro de um contêiner docker.

- Carregador de arquivos S3

  Esse componente monitora um diretório em busca de novos arquivos, os carrega no Amazon Simple Storage Service (Amazon S3) e os exclui após um upload bem-sucedido.

- Cliente Secrets Manager

  Esse componente fornece uma ferramenta CLI que pode ser usada por outros componentes que precisam recuperar segredos do componente Secrets Manager em um script de ciclo de vida da receita.

- Roteamento TES para contêiner

  Esse componente configura nftables ou iptables em um AWS IoT Greengrass dispositivo para que ele possa usar o Serviço de troca de tokens componente com contêineres.

- WebRTC

  Esse componente ingere fluxos de áudio e vídeo de câmeras RTSP conectadas ao dispositivo principal. AWS IoT Greengrass Em seguida, o componente transforma os streams de áudio e vídeo em peer-to-peer comunicação ou retransmissão por meio do Amazon Kinesis Video Streams.

Para solicitar um recurso ou relatar um bug, abra um GitHub problema no repositório desse componente. AWS não fornece suporte para componentes da comunidade. Para obter mais informações, consulte o CONTRIBUTING.mdarquivo no repositório de cada componente.

Vários componentes AWS fornecidos também são de código aberto. Para ter mais informações, consulte Software AWS IoT Greengrass principal de código aberto.

AWS IoT Greengrassferramentas de desenvolvimento

Use ferramentas de AWS IoT Greengrass desenvolvimento para criar, testar, criar, publicar e implantar componentes personalizados do Greengrass.

- Kit de desenvolvimento do Greengrass (CLI)

  Use a interface de linha de comando do kit de AWS IoT Greengrass desenvolvimento (GDK CLI) em seu ambiente de desenvolvimento local para criar componentes a partir de modelos e componentes da comunidade no catálogo de software do Greengrass. Você pode usar a CLI do GDK para criar o componente e publicá-lo AWS IoT Greengrass no serviço como um componente privado no seu. Conta da AWS
- Interface de linha de comando do Greengrass

  Use a interface de linha de comando do Greengrass (Greengrass CLI) nos dispositivos principais do Greengrass para implantar e depurar componentes do Greengrass. A CLI do Greengrass é um componente que você pode implantar em seus dispositivos principais para criar implantações locais, visualizar detalhes sobre os componentes instalados e explorar arquivos de log.
- Console de depuração local

  Use o console de depuração local nos dispositivos principais do Greengrass para implantar e depurar componentes do Greengrass usando uma interface web de painel local. O console de depuração local é um componente que você pode implantar em seus dispositivos principais para criar implantações locais e visualizar detalhes sobre os componentes instalados.

AWS IoT Greengrasstambém fornece os seguintes SDKs que você pode usar em componentes personalizados do Greengrass:

- OAWS IoT Device SDK, que contém a biblioteca de comunicação entre processos (IPC). Para ter mais informações, consulte Use o AWS IoT Device SDK para se comunicar com o núcleo do Greengrass, outros componentes e AWS IoT Core.
- O Stream Manager SDK, que você pode usar para transferir fluxos de dados para o. Nuvem AWS Para ter mais informações, consulte Gerencie fluxos de dados nos dispositivos principais do Greengrass.

Tópicos

- AWS IoT GreengrassInterface de linha de comando do kit de desenvolvimento

- Interface de linha de comando do Greengrass
- Use a estrutura AWS IoT Greengrass de teste

AWS IoT GreengrassInterface de linha de comando do kit de desenvolvimento

A interface de linha de comando do kit de AWS IoT Greengrass desenvolvimento (GDK CLI) fornece recursos que ajudam você a desenvolver componentes personalizados do Greengrass. Você pode usar a CLI do GDK para criar, criar e publicar componentes personalizados. Ao criar um repositório de componentes com a CLI do GDK, você pode começar com um modelo ou componente da comunidade do Catálogo de Software do Greengrass. Em seguida, você pode escolher um sistema de compilação que empacote arquivos como arquivos ZIP, use um script de compilação do Maven ou do Gradle ou execute um comando de compilação personalizado. Depois de criar um componente, você pode usar a CLI do GDK para publicá-lo AWS IoT Greengrass no serviço. Assim, você pode usar o console ou AWS IoT Greengrass a API para implantar o componente em seus dispositivos principais do Greengrass.

Ao desenvolver componentes do Greengrass sem a CLI do GDK, você deve atualizar os URIs da versão e do artefato no arquivo de receita do componente sempre que criar uma nova versão do componente. Quando você usa a CLI do GDK, ela pode atualizar automaticamente os URIs de versão e artefato sempre que você publica uma nova versão do componente.

A CLI do GDK é de código aberto e está disponível em. GitHub Você pode personalizar e estender a CLI do GDK para atender às suas necessidades de desenvolvimento de componentes. Convidamos você a abrir problemas e fazer pull requests no GitHub repositório. Você pode encontrar a fonte da CLI do GDK no seguinte link: https://github.com/aws-greengrass/. aws-greengrass-gdk-cli

Pré-requisitos

Para instalar e usar a CLI do Greengrass Development Kit, você precisa do seguinte:

- Uma Conta da AWS. Se você não tiver uma, consulte Configurar um Conta da AWS.
- Um computador de desenvolvimento semelhante ao Windows, macOS ou Unix com conexão à Internet.
- Para o GDK CLI versão 1.1.0 ou posterior, o Python 3.6 ou posterior instalado em seu computador de desenvolvimento.

Para o GDK CLI versão 1.0.0, o Python 3.8 ou posterior está instalado em seu computador de desenvolvimento.

- Git instalado em seu computador de desenvolvimento.
- AWS Command Line Interface(AWS CLI) instalado e configurado com credenciais em seu computador de desenvolvimento. Para obter mais informações, consulte Instalação, atualização e desinstalação do AWS CLI e Configuração do AWS CLI no Guia do AWS Command Line Interface Usuário.

> **Note**
>
> Se você usa um Raspberry Pi ou outro dispositivo ARM de 32 bits, instale AWS CLI a V1. AWS CLI A V2 não está disponível para dispositivos ARM de 32 bits. Para obter mais informações, consulte Instalando, atualizando e desinstalando a AWS CLI versão 1.

- Para usar a CLI do GDK para publicar componentes AWS IoT Greengrass no serviço, você deve ter as seguintes permissões:
  - `s3:CreateBucket`
  - `s3:GetBucketLocation`
  - `s3:PutObject`
  - `greengrass:CreateComponentVersion`
  - `greengrass:ListComponentVersions`
- Para usar a CLI do GDK para criar um componente cujos artefatos existem em um bucket do S3 e não no sistema de arquivos local, você deve ter as seguintes permissões:
  - `s3:ListBucket`

  Esse recurso está disponível para o GDK CLI v1.1.0 e versões posteriores.

Changelog

A tabela a seguir descreve as alterações em cada versão da CLI do GDK. Para obter mais informações, consulte a página de lançamentos da CLI do GDK em. GitHub

Version (Versão)	Alterações
1.6.2	Correções de erros e melhorias • Corrige um problema em que o gradlew.bat do Windows não funciona devido ao caminho relativo. • Pequenas melhorias no registro, nos testes e no empacotamento.
1.6.1	Correções de erros e melhorias • Adiciona uma correção de segurança para análise de argumentos da CLI. • Permite que o GDK obtenha o nome de versão mais recente do Greengrass Testing Framework (GTF) como a versão padrão do GTF. • Permite que o GDK recomende aos clientes que usam uma versão mais antiga do GTF que eles atualizem para a versão mais recente.
1.6.0	Novos atributos • Adiciona uma verificação de validação da receita em relação ao esquema da receita do Greengrass durante os comandos e. `component build` `component publish` Essa atualização ajuda os desenvolvedores a identificar problemas acionáveis em suas receitas de componentes no início do processo de criação de componentes. • Adiciona um conjunto de testes de confiança ao modelo que pode ser baixado pelo `test-e2e init` comando. Esse conjunto de testes de confiança inclui oito testes genéricos que podem ser usados e ampliados para atender às necessidades básicas de testes de componentes. Correções de erros e melhorias • Atualiza a versão padrão do Greengrass Testing Framework (GTF) usada pelo `test-e2e` comando para a versão 1.2.0.
1.5.0	Correções de erros e melhorias Atualiza os padrões reconhecidos pela opção de `excludes` construçã o quando `build_system` estiver `zip`. Esta versão agora reconhecerá padrões globais que correspondem aos nomes de caminho com base em

Version (Versão)	Alterações
	seus caracteres curinga. Isso permite a especificação personalizada de quais diretórios excluir.
1.4.0	Novos atributos • Adiciona um novo `config` comando que inicia um prompt interativo para modificar campos em um arquivo de configuração existente do GDK. • Modifica os `gdk component publish` comandos `gdk component build` e para verificar se o tamanho da receita está dentro dos requisitos do Greengrass (<=16000 bytes) antes de continuar. Correções de erros e melhorias • Adiciona registro adicional na saída do `gdk component build` comando quando um erro de sintaxe da receita está impedindo a conclusão da compilação para fins de reconhecimento. • Renomeia `otf-options` e `otf-version` para `gtf-options` e `gtf-version` respectivamente, devido à renomeação do Open Test Framework para Greengrass Testing Framework.
1.3.0	Novos atributos • Adiciona um novo `test-e2e` comando para oferecer suporte ao end-to-end teste de componentes usando o Open Test Framework. • Adiciona uma nova opção de configuração,`zip_name`, para oferecer suporte a nomes de arquivos zip configuráveis com o sistema de compilação zip. • Torna a `region` propriedade no arquivo de configuração do GDK opcional. Correções de erros e melhorias • Corrige um problema em que um novo diretório é criado mesmo quando o modelo ou repositório especificado não existe ao inicializar um projeto GDK com o argumento. `--name`

Version (Versão)	Alterações
1.2.3	Correções de erros e melhorias • Corrige um problema em que a criação do bucket falha devido ao tratamento incorreto de erros. • Corrige um problema em que as estruturas de lista na receita do componente são removidas.
1.2.2	Correções de erros e melhorias • As chaves da receita não diferenciam mais maiúsculas de minúsculas. • Adiciona uma verificação para determinar se um bucket existe em um Região da AWS e está acessível pelo usuário antes de criar um novo bucket. Requer que o usuário tenha a `GetBucketLocation` permissão. • Corrige um problema com a `excludes` palavra-chave no arquivo de configuração da CLI do GDK.
1.2.1	Correções de erros e melhorias • Aceita o Canadá (Central) (`ca-central-1`) Região da AWS na entrada de configuração da região no `gdk-config.json` arquivo. • Corrige problemas com o argumento da CLI do `--region` GDK para o `publish` comando.

Version (Versão)	Alterações
1.2.0	Novos atributos • Adiciona a `options` entrada à `build` configuração no arquivo de configuração da CLI do GDK. `excludes`Suporta `options` a exclusão de determinados arquivos do artefato zip ao usar o sistema de `zip` compilação. • Adiciona o sistema de `gradlew` compilação para usar o Gradle Wrapper para criar componentes. • Adiciona suporte para arquivos de compilação Kotlin DSL para a opção de compilação. `gradle` • Adiciona uma `options` entrada à `publish` configuração no arquivo de configuração da CLI do GDK. Suporta o `file_upload_args` under `options` para fornecer argumentos adicionais ao fazer o upload de arquivos para o Amazon S3. Correções de erros e melhorias • Corrige um problema em que as compilações do Gradle não eram limpas antes de executar um comando de compilação. • Corrige um problema em que a compilação não saía quando o comando de compilação falhava. • Melhora o formato de saída do `gdk component list` comando.

Version (Versão)	Alterações
1.1.0	Novos atributos • Adiciona suporte ao sistema de compilação Gradle. • Adiciona suporte para o sistema de compilação Maven em dispositivos Windows. • Adiciona o `--bucket` argumento ao comando de publicação do componente. Você pode usar esse argumento para especificar o bucket exato em que a CLI do GDK carrega artefatos do componente. • Adiciona o `--name` argumento ao comando init do componente. Você pode usar essa opção para especificar a pasta em que a CLI do GDK inicializa o componente. • Adiciona suporte para artefatos de componentes que existem em um bucket do S3, mas não na pasta local de criação de componentes. Você pode usar esse recurso para reduzir os custos de largura de banda para artefatos de componentes grandes, como modelos de aprendizado de máquina. Correções de erros e melhorias • Atualiza o comando de publicação do componente para verificar se o componente foi criado antes de publicá-lo. Se o componente não foi criado, esse comando agora cria o componente para você. • Corrige um problema em que o sistema de compilação zip falha na compilação em dispositivos Windows quando o nome do arquivo ZIP contém letras maiúsculas. • Melhora o formato da mensagem de log e altera o nível de log padrão para `INFO` em dispositivos que executam versões do Python anteriores à 3.8. • Altera o requisito mínimo da versão do Python para Python 3.6.
1.0.0	Versão inicial.

Instale ou atualize a interface de linha de comando do kit de AWS IoT Greengrass desenvolvimento

A interface de linha de comando do kit de AWS IoT Greengrass desenvolvimento (GDK CLI) é construída em Python, então você pode `pip` usá-la para instalá-la em seu computador de desenvolvimento.

> **Tip**
>
> Você também pode instalar a CLI do GDK em ambientes virtuais Python, como venv. Para obter mais informações, consulte Ambientes e pacotes virtuais na documentação do Python 3.

Para instalar ou atualizar a CLI do GDK

1. Execute o comando a seguir para instalar a versão mais recente da CLI do GDK a partir de GitHub seu repositório.

   ```
   python3 -m pip install -U git+https://github.com/aws-greengrass/aws-greengrass-gdk-cli.git@v1.6.2
   ```

   > **Note**
   >
   > Para instalar uma versão específica da CLI do GDK, *substitua* versionTag pela tag de versão a ser instalada. Você pode ver as tags de versão da CLI do GDK em GitHub seu repositório.
   >
   > ```
   > python3 -m pip install -U git+https://github.com/aws-greengrass/aws-greengrass-gdk-cli.git@versionTag
   > ```

2. Execute o comando a seguir para verificar se a CLI do GDK foi instalada com êxito.

   ```
   gdk --help
   ```

   Se o `gdk` comando não for encontrado, adicione sua pasta ao PATH.

- Em dispositivos Linux, adicione `/home/MyUser/.local/bin` ao PATH e *MyUser*substitua pelo nome do seu usuário.
- Em dispositivos Windows, adicione *PythonPath*`\\Scripts` ao PATH e *PythonPath*substitua pelo caminho para a pasta Python no seu dispositivo.

Agora você pode usar a CLI do GDK para criar, criar e publicar componentes do Greengrass. Para obter mais informações sobre como usar a CLI do GDK, consulte. AWS IoT GreengrassComandos da interface de linha de comando do kit de desenvolvimento

AWS IoT GreengrassComandos da interface de linha de comando do kit de desenvolvimento

A interface de linha de comando do kit de AWS IoT Greengrass desenvolvimento (GDK CLI) fornece uma interface de linha de comando que você pode usar para criar, criar e publicar componentes do Greengrass em seu computador de desenvolvimento. Os comandos da CLI do GDK usam o formato a seguir.

```
gdk <command> <subcommand> [arguments]
```

Quando você instala a CLI do GDK, o instalador `gdk` adiciona ao PATH para que você possa executar a CLI do GDK na linha de comando.

Você pode usar os seguintes argumentos com qualquer comando:

- Use `-h` ou `--help` para obter informações sobre um comando da CLI do GDK.
- Use `-v` ou `--version` para ver qual versão do GDK CLI está instalada.
- Use `-d` ou `--debug` para gerar registros detalhados que você pode usar para depurar a CLI do GDK.

Esta seção descreve os comandos da CLI do GDK e fornece exemplos para cada comando. A sinopse de cada comando mostra seus argumentos e seu uso. Os argumentos opcionais são mostrados entre colchetes.

Comandos disponíveis

- parte
- config

- teste-e2e

parte

Use o `component` comando na interface de linha de comando do kit de AWS IoT Greengrass desenvolvimento (GDK CLI) para criar, criar e publicar componentes personalizados do Greengrass.

Subcomandos

- init
- build
- publish
- list

init

Inicialize uma pasta de componentes do Greengrass a partir de um modelo de componente ou componente da comunidade.

A CLI do GDK recupera componentes da comunidade do Catálogo de Software do Greengrass e modelos de componentes do repositório de Modelos de AWS IoT Greengrass Componentes em. GitHub

> **Note**
>
> Se você usa o GDK CLI v1.0.0, deve executar esse comando em uma pasta vazia. A CLI do GDK baixa o modelo ou componente da comunidade para a pasta atual.
> Se você usa o GDK CLI v1.1.0 ou posterior, pode especificar `--name` o argumento para especificar a pasta em que o GDK CLI baixa o modelo ou o componente da comunidade. Se você usar esse argumento, especifique uma pasta que não existe. A CLI do GDK cria a pasta para você. Se você não especificar esse argumento, a CLI do GDK usará a pasta atual, que deve estar vazia.
> Se o componente usa o sistema de compilação zip, a CLI do GDK compacta determinados arquivos na pasta do componente em um arquivo zip com o mesmo nome da pasta do componente. Por exemplo, se o nome da pasta do componente for`HelloWorld`, a CLI do GDK criará um arquivo zip chamado. `HelloWorld.zip` Na receita do componente, o nome do artefato zip deve corresponder ao nome da pasta do componente. Se você usa o

GDK CLI versão 1.0.0 em um dispositivo Windows, a pasta do componente e os nomes dos arquivos zip devem conter somente letras minúsculas.
Se você inicializar um modelo ou componente da comunidade que usa o sistema de compilação zip em uma pasta com um nome diferente do modelo ou componente, deverá alterar o nome do artefato zip na receita do componente. Atualize as `Lifecycle` definições `Artifacts` e de forma que o nome do arquivo zip corresponda ao nome da pasta do componente. O exemplo a seguir destaca o nome do arquivo zip nas `Lifecycle` definições `Artifacts` e.

JSON

```
{
  ...
  "Manifests": [
    {
      "Platform": {
        "os": "all"
      },
      "Artifacts": [
        {
          "URI": "s3://BUCKET_NAME/COMPONENT_NAME/
COMPONENT_VERSION/HelloWorld.zip",
          "Unarchive": "ZIP"
        }
      ],
      "Lifecycle": {
        "run": "python3 -u {artifacts:decompressedPath}/HelloWorld/main.py
 {configuration:/Message}"
      }
    }
  ]
}
```

YAML

```
---
...
Manifests:
  - Platform:
      os: all
    Artifacts:
```

```
      - URI: "s3://BUCKET_NAME/COMPONENT_NAME/
COMPONENT_VERSION/HelloWorld.zip"
        Unarchive: ZIP
    Lifecycle:
      run: "python3 -u {artifacts:decompressedPath}/HelloWorld/main.py
 {configuration:/Message}"
```

Resumo

```
$ gdk component init
    [--language]
    [--template]
    [--repository]
    [--name]
```

Argumentos (inicializar a partir do modelo do componente)

- `-l`, `--language` — A linguagem de programação a ser usada para o modelo que você especificar.

  Você deve especificar um `--repository` ou `--language` `--template` e.
- `-t`, `--template` — O modelo de componente a ser usado em um projeto de componente local. Para visualizar os modelos disponíveis, use o comando list.

  Você deve especificar um `--repository` ou `--language` `--template` e.
- `-n`, `--name` — (Opcional) O nome da pasta local em que a CLI do GDK inicializa o componente. Especifique uma pasta que não existe. A CLI do GDK cria a pasta para você.

  Esse recurso está disponível para o GDK CLI v1.1.0 e versões posteriores.

Argumentos (inicializar a partir do componente da comunidade)

- `-r`, `--repository` — O componente da comunidade a ser verificado na pasta local. Para visualizar os componentes disponíveis da comunidade, use o comando list.

  Você deve especificar um `--repository` ou `--language` `--template` e.
- `-n`, `--name` — (Opcional) O nome da pasta local em que a CLI do GDK inicializa o componente. Especifique uma pasta que não existe. A CLI do GDK cria a pasta para você.

  Esse recurso está disponível para o GDK CLI v1.1.0 e versões posteriores.

Saída

O exemplo a seguir mostra a saída produzida quando você executa esse comando para inicializar uma pasta de componentes a partir do modelo Python Hello World.

```
$ gdk component init -l python -t HelloWorld
[2021-11-29 12:51:40] INFO - Initializing the project directory with a python
 component template - 'HelloWorld'.
[2021-11-29 12:51:40] INFO - Fetching the component template 'HelloWorld-python'
 from Greengrass Software Catalog.
```

O exemplo a seguir mostra a saída produzida quando você executa esse comando para inicializar uma pasta de componentes a partir de um componente da comunidade.

```
$ gdk component init -r aws-greengrass-labs-database-influxdb
[2022-01-24 15:44:33] INFO - Initializing the project directory with a component
 from repository catalog - 'aws-greengrass-labs-database-influxdb'.
[2022-01-24 15:44:33] INFO - Fetching the component repository 'aws-greengrass-labs-
database-influxdb' from Greengrass Software Catalog.
```

build

Crie a fonte de um componente em uma receita e artefatos que você possa publicar no AWS IoT Greengrass serviço. A CLI do GDK executa o sistema de compilação que você especifica no arquivo de configuração da CLI do GDK,. `gdk-config.json` Você deve executar esse comando na mesma pasta em que o `gdk-config.json` arquivo existe.

Quando você executa esse comando, a CLI do GDK cria uma receita e artefatos na pasta `greengrass-build` do componente. A CLI do GDK salva a receita na `greengrass-build/recipes` pasta e salva os artefatos na pasta. `greengrass-build/artifacts/`*componentName*`/`*componentVersion*

Se você usa o GDK CLI v1.1.0 ou posterior, a receita do componente pode especificar artefatos que existem em um bucket do S3, mas não na pasta de criação do componente local. Você pode usar esse recurso para reduzir o uso de largura de banda ao desenvolver componentes com grandes artefatos, como modelos de aprendizado de máquina.

Depois de criar um componente, você pode fazer o seguinte para testá-lo em um dispositivo principal do Greengrass:

- Se você desenvolve em um dispositivo diferente daquele em que executa o software AWS IoT Greengrass Core, você deve publicar o componente para implantá-lo em um dispositivo principal do Greengrass. Publique o componente no AWS IoT Greengrass serviço e implante-o no dispositivo principal do Greengrass. Para obter mais informações, consulte o comando publish Criar implantações e.
- Se você desenvolve no mesmo dispositivo em que executa o software AWS IoT Greengrass Core, pode publicar o componente no AWS IoT Greengrass serviço para implantação ou criar uma implantação local para instalar e executar o componente. Para criar uma implantação local, use a CLI do Greengrass. Para obter mais informações, consulte Teste AWS IoT Greengrass componentes com implantações locais e Interface de linha de comando do Greengrass. Ao criar a implantação local, especifique `greengrass-build/recipes` como pasta de receitas e `greengrass-build/artifacts` como pasta de artefatos.

Resumo

```
$ gdk component build
```

Arguments (Argumentos)

Nenhum

Saída

O exemplo a seguir mostra a saída produzida quando você executa esse comando.

```
$ gdk component build
[2021-11-29 13:18:49] INFO - Getting project configuration from gdk-config.json
[2021-11-29 13:18:49] INFO - Found component recipe file 'recipe.yaml' in the
 project directory.
[2021-11-29 13:18:49] INFO - Building the component 'com.example.PythonHelloWorld'
 with the given project configuration.
[2021-11-29 13:18:49] INFO - Using 'zip' build system to build the component.
[2021-11-29 13:18:49] WARNING - This component is identified as using 'zip' build
 system. If this is incorrect, please exit and specify custom build command in the
 'gdk-config.json'.
[2021-11-29 13:18:49] INFO - Zipping source code files of the component.
[2021-11-29 13:18:49] INFO - Copying over the build artifacts to the greengrass
 component artifacts build folder.
[2021-11-29 13:18:49] INFO - Updating artifact URIs in the recipe.
```

```
[2021-11-29 13:18:49] INFO - Creating component recipe in 'C:\Users\MyUser\Documents
\greengrass-components\python\HelloWorld\greengrass-build\recipes'.
```

publish

Publique esse componente no AWS IoT Greengrass serviço. Esse comando carrega artefatos de construção em um bucket do S3, atualiza o URI do artefato na receita e cria uma nova versão do componente a partir da receita. A CLI do GDK usa o bucket AWS e a região do S3 que você especifica no arquivo de configuração da CLI do GDK,. `gdk-config.json` Você deve executar esse comando na mesma pasta em que o `gdk-config.json` arquivo existe.

Se você usar o GDK CLI v1.1.0 ou posterior, poderá especificar `--bucket` o argumento para especificar o bucket do S3 em que o GDK CLI carrega os artefatos do componente. Se você não especificar esse argumento, a CLI do GDK será carregada no bucket do S3 cujo nome *`bucket`*`-`*`region`*`-`*`accountId`* é, *`onde`* bucket *`e`* region são os valores que você especifica e *`accountId gdk-config.json`* é seu ID. Conta da AWS A CLI do GDK cria o bucket se ele não existir.

Se você usa o GDK CLI v1.2.0 ou posterior, você pode substituir o Região da AWS especificado no arquivo de configuração do GDK CLI usando o parâmetro. `--region` Você também pode especificar opções adicionais usando o `--options` parâmetro. Para obter uma lista das opções disponíveis, consulteArquivo de configuração CLI do Greengrass Development Kit.

Quando você executa esse comando, a CLI do GDK publica o componente com a versão especificada na receita. Se você especificar`NEXT_PATCH`, a CLI do GDK usará a próxima versão do patch que ainda não existe. As versões semânticas usam um major. menor. sistema de numeração de patches. Para obter mais informações, consulte a especificação da versão semântica.

> **Note**
>
> Se você usa o GDK CLI v1.1.0 ou posterior, ao executar esse comando, o GDK CLI verifica se o componente foi criado. Se o componente não for criado, a CLI do GDK cria o componente antes de publicá-lo.

Resumo

```
$ gdk component publish
```

```
    [--bucket] [--region] [--options]
```

Arguments (Argumentos)

- `-b`, `--bucket` — (Opcional) Especifique o nome do bucket do S3 em que a CLI do GDK publica artefatos do componente.

  Se você não especificar esse argumento, a CLI do GDK será carregada no bucket do S3 cujo nome *`bucket`*`-`*`region`*`-`*`accountId`* é, *`onde`* bucket *`e`* region são os valores que você especifica e *`accountId gdk-config.json`* é seu ID. Conta da AWS A CLI do GDK cria o bucket se ele não existir.

  A CLI do GDK cria o bucket se ele não existir.

  Esse recurso está disponível para o GDK CLI v1.1.0 e versões posteriores.
- `-r`, `--region` — (Opcional) Especifique o nome do Região da AWS para quando o componente for criado. Esse argumento substitui o nome da região na configuração da CLI do GDK.

  Esse recurso está disponível para o GDK CLI v1.2.0 e versões posteriores.
- `-o`, `--options` (Opcional) Especifique uma lista de opções para publicar um componente. O argumento deve ser uma string JSON válida ou um caminho de arquivo para um arquivo JSON contendo as opções de publicação. Esse argumento substitui as opções na configuração da CLI do GDK.

  Esse recurso está disponível para o GDK CLI v1.2.0 e versões posteriores.

Saída

O exemplo a seguir mostra a saída produzida quando você executa esse comando.

```
$ gdk component publish
[2021-11-29 13:45:29] INFO - Getting project configuration from gdk-config.json
[2021-11-29 13:45:29] INFO - Found component recipe file 'recipe.yaml' in the
 project directory.
[2021-11-29 13:45:29] INFO - Found credentials in shared credentials file: ~/.aws/
credentials
[2021-11-29 13:45:30] INFO - Publishing the component 'com.example.PythonHelloWorld'
 with the given project configuration.
[2021-11-29 13:45:30] INFO - No private version of the component
 'com.example.PythonHelloWorld' exist in the account. Using '1.0.0' as the next
 version to create.
```

```
[2021-11-29 13:45:30] INFO - Uploading the component built artifacts to s3 bucket.
[2021-11-29 13:45:30] INFO - Uploading component artifacts to S3 bucket: {bucket}.
 If this is your first time using this bucket, add the 's3:GetObject' permission
 to each core device's token exchange role to allow it to download the component
 artifacts. For more information, see https://docs.aws.amazon.com/greengrass/v2/
developerguide/device-service-role.html.
[2021-11-29 13:45:30] INFO - Not creating an artifacts bucket as it already exists.
[2021-11-29 13:45:30] INFO - Updating the component recipe
 com.example.PythonHelloWorld-1.0.0.
[2021-11-29 13:45:30] INFO - Creating a new greengrass component
 com.example.PythonHelloWorld-1.0.0
[2021-11-29 13:45:30] INFO - Created private version '1.0.0' of the component in the
 account.'com.example.PythonHelloWorld'.
```

list

Recupere a lista de modelos de componentes e componentes da comunidade disponíveis.

A CLI do GDK recupera componentes da comunidade do Catálogo de Software do Greengrass e modelos de componentes do repositório de Modelos de AWS IoT Greengrass Componentes em. GitHub

Você pode passar a saída desse comando para o comando init para inicializar repositórios de componentes a partir de modelos e componentes da comunidade.

Resumo

```
$ gdk component list
    [--template]
    [--repository]
```

Arguments (Argumentos)

- `-t`, `--template` — (Opcional) Especifique esse argumento para listar os modelos de componentes disponíveis. Esse comando gera o nome e o idioma de cada modelo no formato*`name`*`-`*`language`*. Por exemplo, em`HelloWorld-python`, o nome do modelo é `HelloWorld` e o idioma é`python`.
- `-r`, `--repository` — (Opcional) Especifique esse argumento para listar os repositórios de componentes da comunidade disponíveis.

Saída

O exemplo a seguir mostra a saída produzida quando você executa esse comando.

```
$ gdk component list --template
[2021-11-29 12:29:04] INFO - Listing all the available component templates from
 Greengrass Software Catalog.
[2021-11-29 12:29:04] INFO - Found '2' component templates to display.
1. HelloWorld-python
2. HelloWorld-java
```

config

Use o `config` comando na interface de linha de comando do kit de AWS IoT Greengrass desenvolvimento (CLI do GDK) para modificar a configuração do GDK no arquivo de configuração,. `gdk-config.json`

Subcomandos

- update

update

Inicie um prompt interativo para modificar campos em um arquivo de configuração existente do GDK.

Resumo

```
$ gdk config update
    [--component]
```

Arguments (Argumentos)

- `-c`, `--component` — Para atualizar os campos relacionados ao componente no `gdk-config.json` arquivo. Esse argumento é obrigatório, pois é a única opção.

Resultado

O exemplo a seguir mostra a saída produzida quando você executa esse comando para configurar um componente.

```
$ gdk config update --component
```

```
Current value of the REQUIRED component_name is (default:
 com.example.PythonHelloWorld):
Current value of the REQUIRED author is (default: author):
Current value of the REQUIRED version is (default: NEXT_PATCH):
Do you want to change the build configurations? (y/n)
Do you want to change the publish configurations? (y/n)
[2023-09-26 10:19:48] INFO - Config file has been updated. Exiting...
```

teste-e2e

Use o `test-e2e` comando na interface de linha de comando do kit de AWS IoT Greengrass desenvolvimento (CLI do GDK) para inicializar, criar e end-to-end executar módulos de teste no projeto GDK.

Subcomandos

- init
- build
- run

init

Inicialize um projeto existente do GDK CLI com um módulo de teste que usa o Greengrass Testing Framework (GTF).

Por padrão, o GDK CLI recupera o modelo do módulo maven do repositório Component Templates AWS IoT Greengrass em. GitHub Esse módulo maven vem com uma dependência do arquivo `aws-greengrass-testing-standalone` JAR.

Esse comando cria um novo diretório chamado `gg-e2e-tests` dentro do projeto GDK. Se o diretório do módulo de teste já existir e não estiver vazio, o comando será encerrado sem fazer nada. Essa `gg-e2e-tests` pasta contém o recurso Cucumber e as definições de etapas estruturadas em um projeto maven.

Por padrão, esse comando tentará usar a versão mais recente do GTF.

Resumo

```
$ gdk test-e2e init
```

```
    [--gtf-version]
```

Arguments (Argumentos)

- `-ov`, `--gtf-version` — (Opcional) A versão do GTF a ser usada com o módulo de end-to-end teste no projeto GDK. Esse valor deve ser uma das versões GTF dos lançamentos. Esse argumento substitui o `gtf_version` na configuração da CLI do GDK.

Saída

O exemplo a seguir mostra a saída produzida quando você executa esse comando para inicializar o projeto GDK com o módulo de teste.

```
$ gdk test-e2e init
[2023-12-06 12:20:28] INFO - Using the GTF version provided in the GDK test config
 1.2.0
[2023-12-06 12:20:28] INFO - Downloading the E2E testing template from GitHub into
 gg-e2e-tests directory...
```

build

> **Note**
>
> Você deve criar o componente executando gdk component build antes de criar o módulo end-to-end de teste.

Crie o módulo end-to-end de teste. A CLI do GDK cria o módulo de teste usando o sistema de compilação que você especifica no arquivo de configuração da CLI do GDK,, sob a propriedade. `gdk-config.json` `test-e2e` Você deve executar esse comando na mesma pasta em que o `gdk-config.json` arquivo existe.

Por padrão, o GDK CLI usa o sistema de compilação maven para criar o módulo de teste. O Maven é necessário para executar o `gdk test-e2e build` comando.

Você deve criar o componente executando gdk-component-build antes de criar o módulo de teste, se os arquivos de recursos de teste tiverem variáveis como `GDK_COMPONENT_NAME` e `GDK_COMPONENT_RECIPE_FILE` para interpolar.

Quando você executa esse comando, a CLI do GDK interpola todas as variáveis da configuração do projeto GDK e cria o módulo para gerar `gg-e2e-tests` o arquivo JAR de teste final.

Resumo

```
$ gdk test-e2e build
```

Arguments (Argumentos)

Nenhum

Saída

O exemplo a seguir mostra a saída produzida quando você executa esse comando.

```
$ gdk test-e2e build
[2023-07-20 15:36:48] INFO - Updating feature file: file:///path/to//
HelloWorld/greengrass-build/gg-e2e-tests/src/main/resources/greengrass/features/
component.feature
[2023-07-20 15:36:48] INFO - Creating the E2E testing recipe file:///path/to/
HelloWorld/greengrass-build/recipes/e2e_test_recipe.yaml
[2023-07-20 15:36:48] INFO - Building the E2E testing module
[2023-07-20 15:36:48] INFO - Running the build command 'mvn package'
.........
```

run

Execute o módulo de teste com as opções de teste no arquivo de configuração do GDK.

> **Note**
>
> Você deve criar o módulo de teste executando gdk test-e2e build antes de executar os end-to-end testes.

Resumo

```
$ gdk test-e2e run
    [--gtf-options]
```

Arguments (Argumentos)

- `-oo`, `--gtf-options` — (Opcional) Especifique uma lista de opções para executar os end-to-end testes. O argumento deve ser uma string JSON válida ou um caminho de arquivo para um

arquivo JSON contendo as opções GTF. As opções fornecidas no arquivo de configuração são mescladas com as fornecidas nos argumentos do comando. Se uma opção estiver presente em ambos os lugares, a do argumento terá precedência sobre a do arquivo de configuração.

Se a `tags` opção não for especificada nesse comando, o GDK usará `Sample` para tags. Se não `ggc-archive` for especificado, o GDK baixa a versão mais recente do arquivo do núcleo do Greengrass.

Saída

O exemplo a seguir mostra a saída produzida quando você executa esse comando.

```
$ gdk test-e2e run
[2023-07-20 16:35:53] INFO - Downloading latest nucleus archive from url https://
d2s8p88vqu9w66.cloudfront.net/releases/greengrass-latest.zip
[2023-07-20 16:35:57] INFO - Running test jar with command java -jar /path/to/
greengrass-build/gg-e2e-tests/target/uat-features-1.0.0.jar –ggc-archive=/path/to/
aws-greengrass-gdk-cli/HelloWorld/greengrass-build/greengrass-nucleus-latest.zip –
tags=Sample

16:35:59.693 [] [] [] [INFO]
 com.aws.greengrass.testing.modules.GreengrassContextModule - Extracting /path/
to/workplace/aws-greengrass-gdk-cli/HelloWorld/greengrass-build/greengrass-
nucleus-latest.zip into /var/folders/7g/ltzcb_3s77nbtmkzfb6brwv40000gr/T/gg-
testing-7718418114158172636/greengrass
16:36:00.534 [gtf-1.1.0-SNAPSHOT] [] [] [INFO]
 com.aws.greengrass.testing.features.LoggerSteps - GTF Version is gtf-1.1.0-SNAPSHOT
.......
```

Arquivo de configuração CLI do Greengrass Development Kit

A interface de linha de comando do kit de AWS IoT Greengrass desenvolvimento (GDK CLI) lê um arquivo de configuração `gdk-config.json` chamado para criar e publicar componentes. Esse arquivo de configuração deve existir na raiz do repositório de componentes. Você pode usar o comando init da CLI do GDK para inicializar repositórios de componentes com esse arquivo de configuração.

Tópicos

- Formato de arquivo de configuração do GDK CLI
- Exemplos de arquivos de configuração do GDK CLI

Formato de arquivo de configuração do GDK CLI

Ao definir um arquivo de configuração da CLI do GDK para um componente, você especifica as seguintes informações no formato JSON.

`gdk_version`

A versão mínima da CLI do GDK compatível com esse componente. Esse valor deve ser uma das versões da CLI do GDK das versões.

`component`

A configuração desse componente.

`componentName`

`author`

O autor ou editor do componente.

`version`

A versão do componente. Especifique um dos seguintes:

- `NEXT_PATCH`— Quando você escolhe essa opção, a CLI do GDK define a versão quando você publica o componente. A CLI do GDK consulta AWS IoT Greengrass o serviço para identificar a versão mais recente publicada do componente. Em seguida, ele define a versão para a próxima versão de patch após essa versão. Se você não publicou o componente antes, a CLI do GDK usa a versão. `1.0.0`

  Se você escolher essa opção, não poderá usar a CLI do Greengrass para implantar e testar localmente o componente em seu computador de desenvolvimento local que executa o software Core. AWS IoT Greengrass Para habilitar implantações locais, você deve especificar uma versão semântica em vez disso.

- Uma versão semântica, como**`1.0.0`**. As versões semânticas usam um major. menor. sistema de numeração de patches. Para obter mais informações, consulte a especificação da versão semântica.

  Se você desenvolver componentes em um dispositivo principal do Greengrass em que deseja implantar e testar o componente, escolha essa opção. Você deve criar o componente com uma versão específica para criar implantações locais com a CLI do Greengrass.

`build`

A configuração a ser usada para criar a fonte desse componente em artefatos. Esse objeto contém as seguintes informações:

`build_system`

O sistema de compilação a ser usado. Escolha uma das seguintes opções:

- `zip`— Empacota a pasta do componente em um arquivo ZIP para definir como o único artefato do componente. Escolha essa opção para os seguintes tipos de componentes:
  - Componentes que usam linguagens de programação interpretadas, como JavaScript Python ou.
  - Componentes que empacotam arquivos que não sejam código, como modelos de aprendizado de máquina ou outros recursos.

  A CLI do GDK compacta a pasta do componente em um arquivo zip com o mesmo nome da pasta do componente. Por exemplo, se o nome da pasta do componente for`HelloWorld`, a CLI do GDK criará um arquivo zip chamado. `HelloWorld.zip`

  > **Note**
  >
  > Se você usa o GDK CLI versão 1.0.0 em um dispositivo Windows, a pasta do componente e os nomes dos arquivos zip devem conter somente letras minúsculas.

  Quando a CLI do GDK compacta a pasta do componente em um arquivo zip, ela ignora os seguintes arquivos:
  - O arquivo `gdk-config.json`
  - O arquivo da receita (`recipe.jsonourecipe.yaml`)
  - Crie pastas, como `greengrass-build`
- `maven`— Executa o `mvn clean package` comando para transformar a fonte do componente em artefatos. Escolha essa opção para componentes que usam o Maven, como componentes Java.

  Em dispositivos Windows, esse recurso está disponível para o GDK CLI v1.1.0 e versões posteriores.

- `gradle`— Executa o `gradle build` comando para transformar a fonte do componente em artefatos. Escolha essa opção para componentes que usam o Gradle. Esse recurso está disponível para o GDK CLI v1.1.0 e versões posteriores.

  O sistema de `gradle` compilação oferece suporte ao Kotlin DSL como arquivo de compilação. Esse recurso está disponível para o GDK CLI v1.2.0 e versões posteriores.

- `gradlew`— Executa o `gradlew` comando para transformar a fonte do componente em artefatos. Escolha essa opção para componentes que usam o Gradle Wrapper.

  Esse recurso está disponível para o GDK CLI v1.2.0 e versões posteriores.

- `custom`— Executa um comando personalizado para transformar a fonte do componente em uma receita e artefatos. Especifique o comando personalizado no `custom_build_command` parâmetro.

`custom_build_command`

(Opcional) O comando de compilação personalizado a ser executado em um sistema de compilação personalizado. Você deve especificar esse parâmetro se especificar `custom` para`build_system`.

> **Important**
>
> Esse comando deve criar uma receita e artefatos nas seguintes pastas dentro da pasta do componente. A CLI do GDK cria essas pastas para você quando você executa o comando de construção do componente.
>
> - Pasta de receitas: `greengrass-build/recipes`
> - Pasta de artefatos: `greengrass-build/artifacts/`*`componentName`*`/`*`componentVersion`*
>
>   Substitua *`componentName`* pelo nome do componente e substitua *`componentVersion pela versão`* do componente ou. `NEXT_PATCH`

Você pode especificar uma única string ou uma lista de strings, em que cada string é uma palavra no comando. Por exemplo, para executar um comando de compilação personalizado para um componente C++, você pode especificar **`cmake --build`**

`build --config Release` ou **`["cmake", "--build", "build", "--config", "Release"]`**.

Para ver um exemplo de um sistema de compilação personalizado, consulte GitHub o. aws.greengrass.labs.LocalWebServer community component

`options`

(Opcional) Opções de configuração adicionais usadas durante o processo de criação do componente.

Esse recurso está disponível para o GDK CLI v1.2.0 e versões posteriores.

`excludes`

Uma lista de padrões globais que definem quais arquivos excluir do diretório de componentes ao criar o arquivo zip. Válido somente quando o `build_system` é `zip`.

> **Note**
>
> Nas versões 1.4.0 e anteriores do GDK CLI, qualquer arquivo que corresponda a uma entrada na lista de exclusões é excluído de todos os subdiretórios do componente. Para obter o mesmo comportamento nas versões 1.5.0 e posteriores da CLI do GDK, acrescente às `**/` entradas existentes na lista de exclusões. Por exemplo, `*.txt` excluirá arquivos de texto apenas do diretório; `**/*.txt` excluirá arquivos de texto de todos os diretórios e subdiretórios.
> Nas versões 1.5.0 e posteriores da CLI do GDK, você pode ver um aviso durante a criação do componente, `excludes` quando definido no arquivo de configuração do GDK. Para desativar esse aviso, defina a variável de ambiente `GDK_EXCLUDES_WARN_IGNORE` como `true`.

A CLI do GDK sempre exclui os seguintes arquivos do arquivo zip:

- O arquivo `gdk-config.json`
- O arquivo da receita (`recipe.json` ou `recipe.yaml`)
- Crie pastas, como `greengrass-build`

Os arquivos a seguir são excluídos por padrão. No entanto, você pode controlar quais desses arquivos são excluídos com a `excludes` opção.

- Qualquer pasta que comece com o prefixo “test” () `test*`
- Todos os arquivos ocultos
- A pasta `node_modules`.

Se você especificar a `excludes` opção, a CLI do GDK excluirá somente os arquivos definidos com a opção. `excludes` Se você não especificar a `excludes` opção, a CLI do GDK excluirá os arquivos e pastas padrão mencionados anteriormente.

`zip_name`

O nome do arquivo zip a ser usado ao criar um artefato zip durante o processo de compilação. Válido somente quando o `build_system` é`zip`. Se `build_system` estiver vazio, o nome do componente será usado para o nome do arquivo zip.

`publish`

A configuração a ser usada para publicar esse componente no AWS IoT Greengrass serviço.

Se você usar o GDK CLI v1.1.0 ou posterior, poderá especificar `--bucket` o argumento para especificar o bucket do S3 em que o GDK CLI carrega os artefatos do componente. Se você não especificar esse argumento, a CLI do GDK será carregada no bucket do S3 cujo nome *`bucket`*`-`*`region`*`-`*`accountId`* é, *`onde`* bucket *`e`* region são os valores que você especifica e *`accountId`* *`gdk-config.json`* é seu ID. Conta da AWS A CLI do GDK cria o bucket se ele não existir.

Esse objeto contém as seguintes informações:

`bucket`

O nome do bucket do S3 a ser usado para hospedar artefatos de componentes.

`region`

O Região da AWS local onde a CLI do GDK publica esse componente.

Essa propriedade é opcional se você estiver usando o GDK CLI v1.3.0 ou posterior.

`options`

(Opcional) Opções de configuração adicionais usadas durante a criação da versão do componente.

Esse recurso está disponível para o GDK CLI v1.2.0 e versões posteriores.

`file_upload_args`

Uma estrutura JSON contendo argumentos enviados ao Amazon S3 durante o upload de arquivos para um bucket, como metadados e mecanismos de criptografia. Para obter uma lista dos argumentos permitidos, consulte a `S3Transfer`classe na documentação do Boto3. .

`test-e2e`

(Opcional) A configuração a ser usada durante o end-to-end teste do componente. Esse recurso está disponível para o GDK CLI v1.3.0 e versões posteriores.

`build`

`build_system`— O sistema de construção a ser usado. A opção padrão é`maven`. Escolha uma das seguintes opções:

- `maven`— Executa o `mvn package` comando para criar o módulo de teste. Escolha essa opção para criar o módulo de teste que usa o Maven.
- `gradle`— Executa o `gradle build` comando para criar o módulo de teste. Escolha essa opção para o módulo de teste que usa o Gradle.

`gtf_version`

(Opcional) A versão do Greengrass Testing Framework (GTF) a ser usada como dependência do módulo de end-to-end teste ao inicializar o projeto GDK com GTF. Esse valor deve ser uma das versões GTF dos lançamentos. O padrão é GTF versão 1.1.0.

`gtf_options`

(Opcional) Opções de configuração adicionais usadas durante o end-to-end teste do componente.

A lista a seguir inclui as opções que você pode usar com a versão 1.1.0 do GTF.

- `additional-plugins`— (Opcional) Plugins adicionais do Cucumber

- `aws-region`— Tem como alvo endpoints regionais específicos para AWS serviços. O padrão é o que o AWS SDK descobre.
- `credentials-path`— Caminho opcional AWS de credenciais do perfil. O padrão é credenciais descobertas no ambiente do host.
- `credentials-path-rotation`— Duração de rotação opcional para AWS credenciais. O padrão é 15 minutos ou. `PT15M`
- `csr-path`— O caminho para o CSR usando o qual o certificado do dispositivo será gerado.
- `device-mode`— O dispositivo alvo em teste. O padrão é dispositivo local.
- `env-stage`— Tem como alvo o ambiente de implantação do Greengrass. O padrão é produção.
- `existing-device-cert-arn`— O arn de um certificado existente que você deseja usar como certificado de dispositivo para o Greengrass.
- `feature-path`— Arquivo ou diretório contendo arquivos de recursos adicionais. O padrão é que nenhum arquivo de recurso adicional é usado.
- `gg-cli-version`— Substitui a versão da CLI do Greengrass. O padrão é o valor encontrado em. `ggc.version`
- `gg-component-bucket`— O nome de um bucket Amazon S3 existente que abriga componentes do Greengrass.
- `gg-component-overrides`— Uma lista de substituições de componentes do Greengrass.
- `gg-persist`— Uma lista de elementos de teste a serem persistidos após a execução do teste. O comportamento padrão é não persistir em nada. Os valores aceitos são: `aws.resourcesinstalled.software`, `generated.files` e.
- `gg-runtime`— Uma lista de valores para influenciar a forma como o teste interage com os recursos do teste. Esses valores substituem o `gg.persist` parâmetro. Se o padrão for vazio, ele presume que todos os recursos de teste são gerenciados pelo caso de teste, incluindo o tempo de execução do Greengrass instalado. Os valores aceitos são: `aws.resourcesinstalled.software`, `generated.files` e.
- `ggc-archive`— O caminho para o componente do núcleo arquivado do Greengrass.
- `ggc-install-root`— Diretório para instalar o componente do núcleo do Greengrass. O padrão é test.temp.path e test run folder.
- `ggc-log-level`— Defina o nível de log do núcleo do Greengrass para a execução do teste. O padrão é “INFO”.

- `ggc-tes-rolename`— A função do IAM que o AWS IoT Greengrass Core assumirá para acessar AWS os serviços. Se uma função com o nome fornecido não existir, será criada uma política de acesso padrão.
- `ggc-trusted-plugins`— A lista separada por vírgula dos caminhos (no host) dos plug-ins confiáveis que precisam ser adicionados ao Greengrass. Para fornecer o caminho no próprio DUT, prefixe o caminho com 'dut: '
- `ggc-user-name`— O valor user:group PosixUser para o núcleo Greengrass. O padrão é o nome de usuário atual que está conectado.
- `ggc-version`— Substitui a versão do componente do núcleo do Greengrass em execução. O padrão é o valor encontrado em ggc.archive.
- `log-level`— Nível de registro da execução do teste. O padrão é “INFO”.
- `parallel-config`— Conjunto de índice de lote e número de lotes como uma string JSON. O valor padrão do índice do lote é 0 e o número de lotes é 1.
- `proxy-url`— Configure todos os testes para rotear o tráfego por meio desse URL.
- `tags`— Execute apenas tags de recursos. Pode ser cruzado com '&'
- `test-id-prefix`— Um prefixo comum aplicado a todos os recursos específicos do teste, incluindo nomes e tags de AWS recursos. O padrão é um prefixo “gg”.
- `test-log-path`— Diretório que conterá os resultados de toda a execução do teste. O padrão é “TestResults”.
- `test-results-json`— Sinalize para determinar se um relatório JSON do Cucumber resultante foi gerado e gravado no disco. O valor padrão é verdadeiro.
- `test-results-log`— Sinalize para determinar se a saída do console foi gerada gravada no disco. O padrão é falso.
- `test-results-xml`— Sinalize para determinar se um relatório XML JUnit resultante é gerado e gravado em disco. O valor padrão é verdadeiro.
- `test-temp-path`— Diretório para gerar artefatos de teste locais. O padrão é um diretório temporário aleatório prefixado com gg-testing.
- `timeout-multiplier`— Multiplicador fornecido para todos os tempos limite de teste. O padrão é 1.0.

Exemplos de arquivos de configuração do GDK CLI

Você pode consultar os seguintes exemplos de arquivos de configuração da CLI do GDK para ajudá-lo a configurar ambientes de componentes do Greengrass.

Hello World (Python)

O arquivo de configuração da CLI do GDK a seguir é compatível com um componente Hello World que executa um script Python. Esse arquivo de configuração usa o sistema de `zip` compilação para empacotar o script Python do componente em um arquivo ZIP que a CLI do GDK carrega como um artefato.

```
{
  "component": {
    "com.example.PythonHelloWorld": {
      "author": "Amazon",
      "version": "NEXT_PATCH",
      "build": {
        "build_system" : "zip",
        "options": {
           "excludes": [".*"]
        }
      },
      "publish": {
        "bucket": "greengrass-component-artifacts",
        "region": "us-west-2",
        "options": {
           "file_upload_args": {
              "Metadata": {
                 "some-key": "some-value"
              }
           }
        }
      }
    },
  "test-e2e":{
    "build":{
        "build_system": "maven"
    },
    "gtf_version": "1.1.0",
    "gtf_options": {
         "tags": "Sample"
     }
  },
  "gdk_version": "1.6.1"
  }
}
```

Olá Mundo (Java)

O arquivo de configuração da CLI do GDK a seguir é compatível com um componente Hello World que executa um aplicativo Java. Esse arquivo de configuração usa o sistema de `maven` compilação para empacotar o código-fonte Java do componente em um arquivo JAR que a CLI do GDK carrega como um artefato.

```
{
  "component": {
    "com.example.JavaHelloWorld": {
      "author": "Amazon",
      "version": "NEXT_PATCH",
      "build": {
        "build_system" : "maven"
      },
      "publish": {
        "bucket": "greengrass-component-artifacts",
        "region": "us-west-2",
        "options": {
           "file_upload_args": {
              "Metadata": {
                 "some-key": "some-value"
              }
           }
        }
      }
  },
  "test-e2e":{
    "build":{
        "build_system": "maven"
    },
    "gtf_version": "1.1.0",
    "gtf_options": {
         "tags": "Sample"
     }
  },
  "gdk_version": "1.6.1"
  }
}
```

Componentes da comunidade

Vários componentes da comunidade no Catálogo de Software do Greengrass usam a CLI do GDK. Você pode explorar os arquivos de configuração da CLI do GDK nos repositórios desses componentes.

Para visualizar os arquivos de configuração da CLI do GDK dos componentes da comunidade

1. Execute o comando a seguir para listar os componentes da comunidade que usam a CLI do GDK.

   ```
   gdk component list --repository
   ```

   A resposta lista o nome do GitHub repositório para cada componente da comunidade que usa a CLI do GDK. Cada repositório existe na `awslabs` organização.

   ```
   [2022-02-22 17:27:31] INFO - Listing all the available component repositories from
    Greengrass Software Catalog.
   [2022-02-22 17:27:31] INFO - Found '6' component repositories to display.
   1. aws-greengrass-labs-database-influxdb
   2. aws-greengrass-labs-telemetry-influxdbpublisher
   3. aws-greengrass-labs-dashboard-grafana
   4. aws-greengrass-labs-dashboard-influxdb-grafana
   5. aws-greengrass-labs-local-web-server
   6. aws-greengrass-labs-lookoutvision-gstreamer
   ```

2. Abra o GitHub repositório de um componente da comunidade no seguinte URL. *`community-component-name`*Substitua pelo nome de um componente da comunidade da etapa anterior.

   ```
   https://github.com/awslabs/community-component-name
   ```

Interface de linha de comando do Greengrass

A interface de linha de comando (CLI) do Greengrass permite que você interaja com o AWS IoT Greengrass Core em seu dispositivo para desenvolver componentes localmente e depurar problemas. Por exemplo, você pode usar a CLI do Greengrass para criar uma implantação local e reiniciar um componente no dispositivo principal.

Implante o componente Greengrass CLI (`aws.greengrass.Cli`) para instalar o Greengrass CLI em seu dispositivo principal.

> **Important**
>
> Recomendamos que você use esse componente somente em ambientes de desenvolvimento, não em ambientes de produção. Esse componente fornece acesso a informações e operações que você normalmente não precisará em um ambiente de produção. Siga o princípio do menor privilégio implantando esse componente somente nos dispositivos principais onde você precisar.

Tópicos

Instale a CLI do Greengrass

Você pode instalar a CLI do Greengrass de uma das seguintes formas:

- Use o `--deploy-dev-tools` argumento ao configurar o software AWS IoT Greengrass Core pela primeira vez em seu dispositivo. Você também deve especificar `--provision true` para aplicar esse argumento.
- Implante o componente CLI do Greengrass (`aws.greengrass.Cli`) em seu dispositivo.

Esta seção descreve as etapas para implantar o componente CLI do Greengrass. Para obter informações sobre a instalação da CLI do Greengrass durante a configuração inicial, consulte. Tutorial: Conceitos básicos do AWS IoT Greengrass V2

Pré-requisitos

Para implantar o componente CLI do Greengrass, você deve atender aos seguintes requisitos:

- AWS IoT Greengrass Software principal instalado e configurado em seu dispositivo principal. Para ter mais informações, consulte Tutorial: Conceitos básicos do AWS IoT Greengrass V2.

- Para usar o AWS CLI para implantar a CLI do Greengrass, você deve ter instalado e configurado o. AWS CLI Para obter mais informações, consulte Configuração da AWS CLI no Guia do usuário da AWS Command Line Interface .
- Você deve estar autorizado a usar a CLI do Greengrass para interagir com o software principal. AWS IoT Greengrass Faça o seguinte para usar a CLI do Greengrass:
  - Use o usuário do sistema que executa o software AWS IoT Greengrass Core.
  - Use um usuário com permissões de root ou administrativas. Nos dispositivos principais do Linux, você pode usar `sudo` para obter permissões de root.
  - Use um usuário do sistema que esteja em um grupo que você especifica nos parâmetros de `AuthorizedWindowsGroups` configuração `AuthorizedPosixGroups` ou ao implantar o componente. Para obter mais informações, consulte Configuração do componente CLI do Greengrass.

Implemente o componente CLI do Greengrass

Conclua as etapas a seguir para implantar o componente CLI do Greengrass em seu dispositivo principal:

Para implantar o componente CLI do Greengrass (console)

1. Faça login no console do AWS IoT Greengrass.
2. No menu de navegação, escolha Componentes.
3. Na página Componentes, na guia Componentes públicos, escolha `aws.greengrass.Cli`.
4. Na página aws.greengrass.Cli, escolha Implantar.
5. Em Adicionar à implantação, escolha Criar nova implantação.
6. Na página Especificar destino, em Alvos de implantação, na lista Nome do alvo, escolha o grupo Greengrass no qual você deseja implantar e escolha Avançar.
7. Na página Selecionar componentes, verifique se o aws.greengrass.Clicomponente está selecionado e escolha Avançar.
8. Na página Configurar componentes, mantenha as configurações padrão e escolha Avançar.
9. Na página Definir configuração avançada, mantenha as configurações padrão e escolha Avançar.
10. Na página de revisão, clique em Implantar

Para implantar o componente CLI do Greengrass ()AWS CLI

1. No seu dispositivo, crie um `deployment.json` arquivo para definir a configuração de implantação do componente CLI do Greengrass. Esse arquivo deve ter a seguinte aparência:

```
{
  "targetArn":"targetArn",
  "components": {
    "aws.greengrass.Cli": {
      "componentVersion": "2.12.6",
      "configurationUpdate": {
        "merge": "{\"AuthorizedPosixGroups\":\"<group1>,<group2>,...,<groupN>\",
\"AuthorizedWindowsGroups\":\"<group1>,<group2>,...,<groupN>\"}"
      }
    }
  }
}
```

   - No campo `target`, substitua *`targetArn`* pelo nome do recurso da Amazon (ARN) da coisa ou do grupo de coisas do destino da implantação, no seguinte formato:
     - Coisa: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`thingName`*
     - Grupo de coisas: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thinggroup/`*`thingGroupName`*
   - No objeto `aws.greengrass.Cli` componente, especifique os valores da seguinte forma:

     `version`

     A versão do componente Greengrass CLI.

     `configurationUpdate.AuthorizedPosixGroups`

     (Opcional) Uma string que contém uma lista separada por vírgulas dos grupos do sistema. Você autoriza esses grupos de sistema a usar a CLI do Greengrass para interagir com AWS IoT Greengrass o software principal. Você pode especificar nomes de grupos ou IDs de grupos. Por exemplo, `group1,1002,group3` autoriza três grupos do sistema (`group1`1002, `egroup3`) a usar a CLI do Greengrass.

     Se você não especificar nenhum grupo para autorizar, poderá usar a CLI do Greengrass como `sudo` usuário raiz () ou como usuário do sistema que AWS IoT Greengrass executa o software Core.

`configurationUpdate.AuthorizedWindowsGroups`

(Opcional) Uma string que contém uma lista separada por vírgulas dos grupos do sistema. Você autoriza esses grupos de sistema a usar a CLI do Greengrass para interagir com AWS IoT Greengrass o software principal. Você pode especificar nomes de grupos ou IDs de grupos. Por exemplo, `group1,1002,group3` autoriza três grupos do sistema (`group11002`, `egroup3`) a usar a CLI do Greengrass.

Se você não especificar nenhum grupo para autorizar, poderá usar a CLI do Greengrass como administrador ou como usuário do sistema que AWS IoT Greengrass executa o software Core.

2. Execute o comando a seguir para implantar o componente CLI do Greengrass no dispositivo:

```
$ aws greengrassv2 create-deployment --cli-input-json file://path/
to/deployment.json
```

Durante a instalação, o componente adiciona um link simbólico à `greengrass-cli` */greengrass/v2*`/bin` pasta do seu dispositivo e você executa a CLI do Greengrass a partir desse caminho. Para executar a CLI do Greengrass sem seu caminho absoluto, adicione sua */greengrass/v2*`/bin` pasta à variável PATH. Para verificar a instalação da CLI do Greengrass, execute o seguinte comando:

Linux or Unix

```
/greengrass/v2/bin/greengrass-cli help
```

Windows

```
C:\greengrass\v2\bin\greengrass-cli help
```

A seguinte saída deverá ser mostrada:

```
Usage: greengrass-cli [-hV] [--ggcRootPath=<ggcRootPath>] [COMMAND]
Greengrass command line interface

      --ggcRootPath=<ggcRootPath>
                  The AWS IoT Greengrass V2 root directory.
```

```
  -h, --help      Show this help message and exit.
  -V, --version   Print version information and exit.
Commands:
  help                Show help information for a command.
  component           Retrieve component information and stop or restart
                        components.
  deployment          Create local deployments and retrieve deployment status.
  logs                Analyze Greengrass logs.
  get-debug-password  Generate a password for use with the HTTP debug view
                        component.
```

Se `greengrass-cli` não for encontrado, a implantação pode ter falhado ao instalar a CLI do Greengrass. Para ter mais informações, consulte Solução de problemas AWS IoT Greengrass V2.

Comandos da CLI do Greengrass

A CLI do Greengrass fornece uma interface de linha de comando para interagir localmente com seu dispositivo principal. AWS IoT Greengrass Os comandos da CLI do Greengrass usam o seguinte formato.

```
$ greengrass-cli <command> <subcommand> [arguments]
```

Por padrão, o arquivo `greengrass-cli` executável na */greengrass/v2*`/bin/` pasta interage com a versão do software AWS IoT Greengrass Core em execução na */greengrass/v2* pasta. Se você chamar um executável que não esteja colocado nesse local ou se quiser interagir com o software AWS IoT Greengrass Core em um local diferente, deverá usar um dos métodos a seguir para especificar explicitamente o caminho raiz do software AWS IoT Greengrass Core com o qual você deseja interagir:

- Defina a variável de ambiente `GGC_ROOT_PATH` como */greengrass/v2*.
- Adicione o `--ggcRootPath` */greengrass/v2* argumento ao seu comando conforme mostrado no exemplo a seguir.

  ```
  greengrass-cli --ggcRootPath /greengrass/v2 <command> <subcommand> [arguments]
  ```

Você pode usar os seguintes argumentos com qualquer comando:

- Use `--help` para obter informações sobre um comando específico da CLI do Greengrass.
- Use `--version` para obter informações sobre a versão da CLI do Greengrass.

Esta seção descreve os comandos da CLI do Greengrass e fornece exemplos desses comandos. A sinopse de cada comando mostra seus argumentos e seu uso. Os argumentos opcionais são mostrados entre colchetes.

Comandos disponíveis

- parte
- implantação
- logs
- get-debug-password

parte

Use o `component` comando para interagir com componentes locais em seu dispositivo principal.

Subcomandos

- detalhes
- list
- reiniciar
- stop

detalhes

Recupere a versão, o status e a configuração de um componente.

Resumo

```
greengrass-cli component details --name <component-name>
```

Arguments (Argumentos)

`--name`,`-n`. O nome do componente.

Resultado

O exemplo a seguir mostra a saída produzida quando você executa esse comando.

```
$ sudo greengrass-cli component details --name MyComponent
```

```
Component Name: MyComponent
Version: 1.0.0
State: RUNNING
Configuration: null
```

list

Recupere o nome, a versão, o status e a configuração de cada componente instalado no dispositivo.

Resumo

```
greengrass-cli component list
```

Arguments (Argumentos)

Nenhum

Resultado

O exemplo a seguir mostra a saída produzida quando você executa esse comando.

```
$ sudo greengrass-cli component list

Components currently running in Greengrass:
Component Name: FleetStatusService
Version: 0.0.0
State: RUNNING
Configuration: {"periodicUpdateIntervalSec":86400.0}
Component Name: UpdateSystemPolicyService
Version: 0.0.0
State: RUNNING
Configuration: null
Component Name: aws.greengrass.Nucleus
Version: 2.0.0
State: FINISHED
Configuration: {"awsRegion":"region","runWithDefault":
{"posixUser":"ggc_user:ggc_group"},"telemetry":{}}
Component Name: DeploymentService
Version: 0.0.0
State: RUNNING
Configuration: null
Component Name: TelemetryAgent
Version: 0.0.0
```

```
State: RUNNING
Configuration: null
Component Name: aws.greengrass.Cli
Version: 2.0.0
State: RUNNING
Configuration: {"AuthorizedPosixGroups":"ggc_user"}
```

reiniciar

Reinicie os componentes.

Resumo

```
greengrass-cli component restart --names <component-name>,...
```

Arguments (Argumentos)

`--names,-n`. O nome do componente. É necessário pelo menos um nome de componente. Você pode especificar nomes de componentes adicionais, separando cada nome com uma vírgula.

Resultado

Nenhum

stop

Pare de executar componentes.

Resumo

```
greengrass-cli component stop --names <component-name>,...
```

Arguments (Argumentos)

`--names,-n`. O nome do componente. É necessário pelo menos um nome de componente. Você pode especificar nomes de componentes adicionais, se necessário, separando cada nome com uma vírgula.

Resultado

Nenhum

implantação

Use o `deployment` comando para interagir com componentes locais em seu dispositivo principal.

Para monitorar o progresso de uma implantação local, use o `status` subcomando. Você não pode monitorar o progresso de uma implantação local usando o console.

Subcomandos

- criar
- cancelar
- list
- status

criar

Crie ou atualize uma implantação local usando receitas de componentes, artefatos e argumentos de tempo de execução especificados.

Resumo

```
greengrass-cli deployment create
    --recipeDir path/to/component/recipe
    [--artifactDir path/to/artifact/folder ]
    [--update-config {component-configuration}]
    [--groupId <thing-group>]
    [--merge "<component-name>=<component-version>"]...
    [--runWith "<component-name>:posixUser=<user-name>[:<group-name>]"]...
    [--systemLimits "{component-system-resource-limits}]"]...
    [--remove <component-name>,...]
    [--failure-handling-policy <policy name[ROLLBACK, DO_NOTHING]>]
```

Arguments (Argumentos)

- `--recipeDir`,`-r`. O caminho completo para a pasta que contém os arquivos de receita do componente.
- `--artifactDir`,`-a`. O caminho completo para a pasta que contém os arquivos de artefatos que você deseja incluir na sua implantação. A pasta de artefatos deve conter a seguinte estrutura de diretórios:

```
/path/to/artifact/folder/<component-name>/<component-version>/<artifacts>
```

- `--update-config,-c`. Os argumentos de configuração para a implantação, fornecidos como uma string JSON ou um arquivo JSON. A string JSON deve estar no seguinte formato:

  ```
  { \
    "componentName": { \
      "MERGE": {"config-key": "config-value"}, \
      "RESET": ["path/to/reset/"] \
    } \
  }
  ```

  `MERGE`e `RESET` diferenciam maiúsculas de minúsculas e devem estar em maiúsculas.

- `--groupId,-g`. O grupo-alvo da implantação.
- `--merge,-m`. O nome e a versão do componente de destino que você deseja adicionar ou atualizar. Você deve fornecer as informações do componente no formato`<component>`=`<version>`. Use um argumento separado para cada componente adicional a ser especificado. Se necessário, use o `--runWith` argumento para fornecer as `windowsUser` informações `posixUserposixGroup`, e para executar o componente.
- `--runWith`. As `windowsUser` informações `posixUserposixGroup`, e para executar um componente genérico ou Lambda. Você deve fornecer essas informações no formato`<component>`:`{posixUser|windowsUser}`=`<user>`[:`<=posixGroup>`]. Por exemplo, você pode especificar **`HelloWorld:posixUser=ggc_user:ggc_group`** ou**`HelloWorld:windowsUser=ggc_user`**. Use um argumento separado para cada opção adicional a ser especificada.

  Para ter mais informações, consulte Configurar o usuário que executa os componentes.

- `--systemLimits`. Os limites de recursos do sistema devem ser aplicados aos processos dos componentes Lambda genéricos e não conteinerizados no dispositivo principal. Você pode configurar a quantidade máxima de uso de CPU e RAM que os processos de cada componente podem usar. Especifique um objeto JSON serializado ou um caminho de arquivo para um arquivo JSON. O objeto JSON deve ter o seguinte formato.

  ```
  {  \
    "componentName": { \
      "cpus": cpuTimeLimit, \
      "memory": memoryLimitInKb \
  ```

```
  } \
}
```

Você pode configurar os seguintes limites de recursos do sistema para cada componente:

- `cpus`— A quantidade máxima de tempo de CPU que os processos desse componente podem usar no dispositivo principal. O tempo total da CPU de um dispositivo essencial é equivalente ao número de núcleos da CPU do dispositivo. Por exemplo, em um dispositivo principal com 4 núcleos de CPU, você pode definir esse valor 2 para limitar os processos desse componente a 50% de uso de cada núcleo da CPU. Em um dispositivo com 1 núcleo de CPU, você pode definir esse valor `0.25` para limitar os processos desse componente a 25% de uso da CPU. Se você definir esse valor como um número maior que o número de núcleos de CPU, o software AWS IoT Greengrass Core não limitará o uso da CPU do componente.
- `memory`— A quantidade máxima de RAM (em kilobytes) que os processos desse componente podem usar no dispositivo principal.

Para ter mais informações, consulte Configurar limites de recursos do sistema para componentes.

Esse recurso está disponível para a versão 2.4.0 e posterior do componente Greengrass nucleus e da CLI do Greengrass em dispositivos principais do Linux. AWS IoT Greengrass atualmente não oferece suporte a esse recurso nos dispositivos principais do Windows.

- `--remove`. O nome do componente de destino que você deseja remover de uma implantação local. Para remover um componente que foi mesclado de uma implantação na nuvem, você deve fornecer o ID do grupo de itens de destino no seguinte formato:

  Greengrass nucleus v2.4.0 and later

  ```
  --remove <component-name> --groupId <group-name>
  ```

  Earlier than v2.4.0

  ```
  --remove <component-name> --groupId thinggroup/<group-name>
  ```

- `--failure-handling-policy`. Define a ação tomada quando uma implantação falha. Há duas ações que você pode especificar:
  - `ROLLBACK` –
  - `DO_NOTHING` –

Esse recurso está disponível para v2.11.0 e versões posteriores do. Núcleo Greengrass

Saída

O exemplo a seguir mostra a saída produzida quando você executa esse comando.

```
$ sudo greengrass-cli deployment create \
    --merge MyApp1=1.0.0 \
    --merge MyApp2=1.0.0 --runWith MyApp2:posixUser=ggc_user \
    --remove MyApp3 \
    --recipeDir recipes/ \
    --artifactDir artifacts/

Local deployment has been submitted! Deployment Id: 44d89f46-1a29-4044-
ad89-5151213dfcbc
```

cancelar

Cancela a implantação especificada.

Resumo

```
greengrass-cli deployment cancel
    -i <deployment-id>
```

Argumentos

`-i`. O identificador exclusivo da implantação a ser cancelada. O ID de implantação é retornado na saída do `create` comando.

Saída

- Nenhum

list

Recupere o status das últimas 10 implantações locais.

Resumo

```
greengrass-cli deployment list
```

Arguments (Argumentos)

Nenhum

Saída

O exemplo a seguir mostra a saída produzida quando você executa esse comando. Dependendo do status da sua implantação, a saída mostra um dos seguintes valores de status:`IN_PROGRESS`,`SUCCEEDED`, ou`FAILED`.

```
$ sudo greengrass-cli deployment list

44d89f46-1a29-4044-ad89-5151213dfcbc: SUCCEEDED
Created on: 6/27/23 11:05 AM
```

status

Recupere o status de uma implantação específica.

Resumo

```
greengrass-cli deployment status -i <deployment-id>
```

Arguments (Argumentos)

`-i`. O ID da implantação.

Saída

O exemplo a seguir mostra a saída produzida quando você executa esse comando. Dependendo do status da sua implantação, a saída mostra um dos seguintes valores de status:`IN_PROGRESS`,`SUCCEEDED`, ou`FAILED`.

```
$ sudo greengrass-cli deployment status -i 44d89f46-1a29-4044-ad89-5151213dfcbc

44d89f46-1a29-4044-ad89-5151213dfcbc: FAILED
Created on: 6/27/23 11:05 AM
Detailed Status: <Detailed deployment status>
Deployment Error Stack: List of error codes
Deployment Error Types: List of error types
Failure Cause: Cause
```

logs

Use o `logs` comando para analisar os registros do Greengrass em seu dispositivo principal.

Subcomandos

- get
- palavras-chave da lista
- list-log-files

get

Colete, filtre e visualize os arquivos de log do Greengrass. Esse comando suporta somente arquivos de log formatados em JSON. Você pode especificar o formato de registro na configuração do núcleo.

Resumo

```
greengrass-cli logs get
    [--log-dir path/to/a/log/folder]
    [--log-file path/to/a/log/file]
    [--follow true | false ]
    [--filter <filter> ]
    [--time-window <start-time>,<end-time> ]
    [--verbose ]
    [--no-color ]
    [--before <value> ]
    [--after <value> ]
    [--syslog ]
    [--max-long-queue-size <value> ]
```

Arguments (Argumentos)

- `--log-dir`,`-ld`. O caminho para o diretório para verificar os arquivos de log, como**/*greengrass/v2*/logs**. Não use com`--syslog`. Use um argumento separado para cada diretório adicional para especificar. Você deve usar pelo menos um dos `--log-dir` ou`--log-file`. Você também pode usar os dois argumentos em um único comando.
- `--log-file`,`-lf`. Os caminhos para os diretórios de log que você deseja usar. Use um argumento separado para cada diretório adicional para especificar. Você deve usar pelo menos um dos `--log-dir` ou`--log-file`. Você também pode usar os dois argumentos em um único comando.

- `--follow`,`-fol`. Mostre as atualizações de registro à medida que elas ocorrem. A CLI do Greengrass continua sendo executada e lendo os registros especificados. Se você especificar uma janela de tempo, a CLI do Greengrass interromperá o monitoramento dos registros após o término de todas as janelas de tempo.
- `--filter`,`-f`. A palavra-chave, as expressões regulares ou o par de valores-chave a serem usados como filtro. Forneça esse valor como uma string, uma expressão regular ou como um par de valores-chave. Use um argumento separado para cada filtro adicional para especificar.

  Quando avaliados, vários filtros especificados em um único argumento são separados por operadores OR, e os filtros especificados em argumentos adicionais são combinados com operadores AND. Por exemplo, se seu comando incluir`--filter "installed" --filter "name=alpha,name=beta"`, a CLI do Greengrass filtrará e exibirá mensagens de log que contêm a palavra-chave `installed` e uma `name` chave com os valores ou. `alpha` `beta`
- `--time-window`,`-t`. A janela de tempo para a qual mostrar as informações do registro. Você pode usar carimbos de data/hora exatos e compensações relativas. Você deve fornecer essas informações no formato*`<begin-time>`*`,`*`<end-time>`*. Se você não especificar a hora de início nem a hora de término, o valor dessa opção será padronizado para a data e a hora atuais do sistema. Use um argumento separado para cada janela de tempo adicional para especificar.

  O Greengrass CLI é compatível com os seguintes formatos de timestamps:

  - `yyyy-MM-DD`, por exemplo,`2020-06-30`. O horário padrão é 00:00:00 quando você usa esse formato.

    `yyyyMMDD`, por exemplo,`20200630`. O horário padrão é 00:00:00 quando você usa esse formato.

    `HH:mm:ss`, por exemplo,`15:30:45`. A data é padronizada para a data atual do sistema quando você usa esse formato.

    `HH:mm:ssSSS`, por exemplo,`15:30:45`. A data assume como padrão a data atual do sistema quando você usa esse formato.

    `YYYY-MM-DD'T'HH:mm:ss'Z'`, por exemplo,`2020-06-30T15:30:45Z`.

    `YYYY-MM-DD'T'HH:mm:ss`, por exemplo,`2020-06-30T15:30:45`.

    `yyyy-MM-dd'T'HH:mm:ss.SSS`, por exemplo,`2020-06-30T15:30:45.250`.

As compensações relativas especificam uma diferença de período em relação à hora atual do sistema. O Greengrass CLI suporta o seguinte formato para compensações relativas:. `+|-[<value>h|hr|hours][value]m|min|minutes][value]s|sec|seconds`

Por exemplo, o argumento a seguir para especificar uma janela de tempo entre 1 hora e 2 horas, 15 minutos antes da hora atual`--time-window -2h15min,-1hr`.

- `--verbose`. Mostra todos os campos das mensagens de registro. Não use com`--syslog`.
- `--no-color`,`-nc`. Remova o código de cores. O código de cores padrão para mensagens de registro usa texto vermelho em negrito. Suporta apenas terminais do tipo Unix porque usa sequências de escape ANSI.
- `--before`,`-b`. O número de linhas a serem mostradas antes de uma entrada de registro correspondente. O padrão é 0.
- `--after`,`-a`. O número de linhas a serem mostradas após uma entrada de registro correspondente. O padrão é 0.
- `--syslog`. Processe todos os arquivos de log usando o protocolo syslog definido pelo RFC3164. Não use com `--log-dir --verbose` e. O protocolo syslog usa o seguinte formato:. `"<$Priority>$Timestamp $Host $Logger ($Class): $Message"` Se você não especificar um arquivo de log, a CLI do Greengrass lerá as mensagens de log dos seguintes locais:`/var/log/messages`,, `/var/log/syslog` ou o. `/var/log/system.log`

  AWS IoT Greengrassatualmente não oferece suporte a esse recurso nos dispositivos principais do Windows.
- `--max-log-queue-size`,`-m`. O número máximo de entradas de registro a serem alocadas na memória. Use essa opção para otimizar o uso da memória. O padrão é de 100.

Saída

O exemplo a seguir mostra a saída produzida quando você executa esse comando.

```
$ sudo greengrass-cli logs get --verbose \
    --log-file /greengrass/v2/logs/greengrass.log \
    --filter deployment,serviceName=DeploymentService \
    --filter level=INFO \
    --time-window 2020-12-08T01:11:17,2020-12-08T01:11:22

2020-12-08T01:11:17.615Z [INFO] (pool-2-thread-14)
 com.aws.greengrass.deployment.DeploymentService: Current deployment finished.
```

```
 {DeploymentId=44d89f46-1a29-4044-ad89-5151213dfcbc, serviceName=DeploymentService,
 currentState=RUNNING}
2020-12-08T01:11:17.675Z [INFO] (pool-2-thread-14)
 com.aws.greengrass.deployment.IotJobsHelper: Updating status of persisted
 deployment. {Status=SUCCEEDED, StatusDetails={detailed-deployment-
status=SUCCESSFUL}, ThingName=MyThing, JobId=22d89f46-1a29-4044-ad89-5151213dfcbc
```

palavras-chave da lista

Mostre as palavras-chave sugeridas que você pode usar para filtrar arquivos de log.

Resumo

```
greengrass-cli logs list-keywords [arguments]
```

Arguments (Argumentos)

Nenhum

Saída

Os exemplos a seguir mostram a saída produzida quando você executa esse comando.

```
$ sudo greengrass-cli logs list-keywords

Here is a list of suggested keywords for Greengrass log:
level=$str
thread=$str
loggerName=$str
eventType=$str
serviceName=$str
error=$str
```

```
$ sudo greengrass-cli logs list-keywords --syslog

Here is a list of suggested keywords for syslog:
priority=$int
host=$str
logger=$str
class=$str
```

list-log-files

Mostra arquivos de log localizados em um diretório especificado.

Resumo

```
greengrass-cli logs list-log-files [arguments]
```

Arguments (Argumentos)

`--log-dir`,`-ld`. O caminho para o diretório para verificar os arquivos de log.

Saída

O exemplo a seguir mostra a saída produzida quando você executa esse comando.

```
$ sudo greengrass-cli logs list-log-files -ld /greengrass/v2/logs/

/greengrass/v2/logs/aws.greengrass.Nucleus.log
/greengrass/v2/logs/main.log
/greengrass/v2/logs/greengrass.log
Total 3 files found.
```

get-debug-password

Usar a`get-debug-password`para imprimir uma senha gerada aleatoriamente para ocomponente do console de depuração local(`aws.greengrass.LocalDebugConsole`). A senha expira 8 horas depois de ser gerada.

Resumo

```
greengrass-cli get-debug-password
```

Arguments (Argumentos)

Nenhum

Resultado

O exemplo a seguir mostra a saída produzida quando você executa esse comando.

```
$ sudo greengrass-cli get-debug-password
```

```
Username: debug
Password: bEDp3MOHdj8ou2w5de_sCBI2XAaguy3a8XxREXAMPLE
Password expires at: 2021-04-01T17:01:43.921999931-07:00
The local debug console is configured to use TLS security. The certificate is self-
signed so you will need to bypass your web browser's security warnings to open the
 console.
Before you bypass the security warning, verify that the certificate fingerprint
 matches the following fingerprints.
SHA-256: 15 0B 2C E2 54 8B 22 DE 08 46 54 8A B1 2B 25 DE FB 02 7D 01 4E 4A 56 67 96
 DA A6 CC B1 D2 C4 1B
SHA-1: BC 3E 16 04 D3 80 70 DA E0 47 25 F9 90 FA D6 02 80 3E B5 C1
```

Use a estrutura AWS IoT Greengrass de teste

O Greengrass Testing Framework (GTF) é uma coleção de componentes básicos que oferece suporte à end-to-end automação do ponto de vista do cliente. O GTF usa Cucumber como driver de recursos. AWS IoT Greengrass usa os mesmos componentes básicos para qualificar as alterações de software em vários dispositivos. Para obter mais informações, consulte Greengrass Testing Framework no Github.

O GTF é implementado usando o Cucumber, uma ferramenta usada para executar testes automatizados, para incentivar o desenvolvimento orientado pelo comportamento (BDD) dos componentes. No Cucumber, os recursos desse sistema são descritos em um tipo especial de arquivo chamado. `feature` Cada recurso é descrito em um formato legível por humanos chamado cenários, que são especificações que podem ser convertidas em testes automatizados. Cada cenário é descrito como uma série de etapas que definem as interações e os resultados desse sistema em teste usando uma linguagem específica de domínio chamada Gherkin. Uma etapa Gherkin é vinculada ao código de programação usando um método chamado definição de etapa, que conecta a especificação ao fluxo de teste. As definições de etapas no GTF são implementadas com Java.

Tópicos

- Como funciona
- Changelog
- Opções de configuração do Greengrass Testing Framework
- Tutorial: Execute end-to-end testes usando o Greengrass Testing Framework e o Greengrass Development Kit
- Tutorial: Use um teste de confiança da suíte de testes de confiança

Como funciona

AWS IoT Greengrass distribui o GTF como um JAR autônomo que consiste em vários módulos Java. Para usar o GTF para end-to-end testar componentes, você deve implementar os testes em um projeto Java. Adicionar o JAR padrão de teste como uma dependência em seu projeto Java permite que você use a funcionalidade existente do GTF e a estenda escrevendo seus próprios casos de teste personalizados. Para executar os casos de teste personalizados, você pode criar seu projeto Java e executar o JAR de destino com as opções de configuração descritas emOpções de configuração do Greengrass Testing Framework.

JAR autônomo GTF

O Greengrass usa o Cloudfront como um repositório Maven para hospedar diferentes versões do JAR autônomo do GTF. Para obter uma lista completa das versões do GTF, consulte os lançamentos do GTF.

O JAR autônomo do GTF inclui os seguintes módulos. Não se limita apenas a esses módulos. Você pode escolher cada uma dessas dependências separadamente em seu projeto ou incluí-las todas de uma vez com o arquivo JAR autônomo de teste.

- `aws-greengrass-testing-resources`: este módulo fornece abstração para gerenciar o ciclo de vida de um AWS recurso durante o curso de um teste. Você pode usar isso para definir seus AWS recursos personalizados usando `ResourceSpec` abstração para que o GTF possa cuidar da criação e remoção desses recursos para você.
- `aws-greengrass-testing-platform`: este módulo fornece abstração em nível de plataforma para o dispositivo em teste durante o ciclo de vida do teste. Ele contém APIs usadas para interagir com o sistema operacional independente da plataforma e pode ser usado para simular os comandos em execução no shell do dispositivo.
- `aws-greengrass-testing-components`: Este módulo consiste em amostras de componentes que são usados para testar os principais recursos do Greengrass, como implantações, IPC e outros recursos.
- `aws-greengrass-testing-features`: Este módulo consiste em etapas comuns reutilizáveis e suas definições, que são usadas para testes no ambiente Greengrass.

Tópicos

- Tutorial: Execute end-to-end testes usando o Greengrass Testing Framework e o Greengrass Development Kit
- Tutorial: Use um teste de confiança da suíte de testes de confiança

Changelog

A tabela a seguir descreve as mudanças em cada versão do GTF. Para obter mais informações, consulte a página de lançamentos do GTF em. GitHub

Version (Versão)	Alterações
1.2.0	Novos atributos • Adiciona etapas relacionadas à rede para configurar o MQTT e a conectividade de rede com a Internet durante os testes. • Adiciona etapas métricas do sistema para monitorar o uso da RAM e da CPU do dispositivo. Correções de erros e melhorias • A etapa de implantação local do Greengrass CLI tenta novamente até ser bem-sucedida. • Os testes interrompem graciosamente o núcleo do Greengrass em vez de matá-lo. • Adiciona melhorias em que o GTF pesquisa o endpoint de AWS IoT credenciais até que as credenciais possam ser recuperadas para o alias da coisa e da função. • Corrige artefatos e diretórios de receitas ausentes. Essa versão também corrige as versões de componentes ausentes. • Corrige um problema em que o GTF falha durante a limpeza da imagem do docker se a imagem do docker não existir. • Adiciona a palavra-chave CURRENT como versão do componente.
1.1.0	Novos atributos • Adiciona a capacidade de instalar um componente personalizado com configuração. Isso requer uma receita para o componente personali zado.

Version (Versão)	Alterações
	• Adiciona a capacidade de atualizar uma implantação local com uma configuração personalizada. Correções de erros e melhorias • Corrige o problema de inconsistência da versão GTF do contexto de log.
1.0.0	Versão inicial.

Opções de configuração do Greengrass Testing Framework

Opções de configuração do GTF

O Greengrass Testing Framework (GTF) permite que você configure determinados parâmetros durante o lançamento do end-to-end processo de teste para orquestrar o fluxo de teste. Você pode especificar essas opções de configuração como argumentos de CLI para o JAR autônomo do GTF.

A versão 1.1.0 e posterior do GTF fornece as seguintes opções de configuração.

- `additional-plugins`— (Opcional) Plugins adicionais do Cucumber
- `aws-region`— Tem como alvo endpoints regionais específicos paraAWSserviços. O padrão é o que oAWSO SDK descobre.
- `credentials-path`— OpcionalAWScaminho das credenciais do perfil. O padrão é credenciais descobertas no ambiente do host.
- `credentials-path-rotation`— Duração de rotação opcional paraAWScredenciais. O padrão é 15 minutos ou`PT15M`.
- `csr-path`— O caminho para o CSR usando o qual o certificado do dispositivo será gerado.
- `device-mode`— O dispositivo alvo em teste. O padrão é dispositivo local.
- `env-stage`— Tem como alvo o ambiente de implantação do Greengrass. O padrão é produção.
- `existing-device-cert-arn`— O arn de um certificado existente que você deseja usar como certificado de dispositivo para o Greengrass.
- `feature-path`— Arquivo ou diretório contendo arquivos de recursos adicionais. O padrão é que nenhum arquivo de recurso adicional é usado.
- `gg-cli-version`— Substitui a versão da CLI do Greengrass. O padrão é o valor encontrado em`ggc.version`.

- `gg-component-bucket`— O nome de um bucket Amazon S3 existente que abriga componentes do Greengrass.
- `gg-component-overrides`— Uma lista de substituições de componentes do Greengrass.
- `gg-persist`— Uma lista de elementos de teste a serem persistidos após a execução do teste. O comportamento padrão é não persistir em nada. Os valores aceitos são:`aws.resources`,`installed.software`, e`generated.files`.
- `gg-runtime`— Uma lista de valores para influenciar a forma como o teste interage com os recursos do teste. Esses valores substituem o`gg.persist`parâmetro. Se o padrão estiver vazio, ele presume que todos os recursos de teste são gerenciados pelo caso de teste, incluindo o tempo de execução instalado do Greengrass. Os valores aceitos são:`aws.resources`,`installed.software`, e`generated.files`.
- `ggc-archive`— O caminho para o componente do núcleo arquivado do Greengrass.
- `ggc-install-root`— Diretório para instalar o componente do núcleo do Greengrass. O padrão é test.temp.path e test run folder.
- `ggc-log-level`— Defina o nível de log do núcleo do Greengrass para a execução do teste. O padrão é “INFO”.
- `ggc-tes-rolename`— A função do IAM queAWS IoT GreengrassO Core assumirá o acessoAWSserviços. Se uma função com o nome fornecido não existir, será criada uma política de acesso padrão.
- `ggc-trusted-plugins`— A lista separada por vírgula dos caminhos (no host) dos plug-ins confiáveis que precisam ser adicionados ao Greengrass. Para fornecer o caminho no próprio DUT, prefixe o caminho com 'dut: '
- `ggc-user-name`— O valor user:group PosixUser para o núcleo Greengrass. O padrão é o nome de usuário atual que está conectado.
- `ggc-version`— Substitui a versão do componente central do Greengrass em execução. O padrão é o valor encontrado em ggc.archive.
- `log-level`— Nível de registro da execução do teste. O padrão é “INFO”.
- `parallel-config`— Conjunto de índice de lote e número de lotes como uma string JSON. O valor padrão do índice do lote é 0 e o número de lotes é 1.
- `proxy-url`— Configure todos os testes para rotear o tráfego por meio desse URL.
- `tags`— Execute apenas tags de recursos. Pode ser cruzado com '&'
- `test-id-prefix`— Um prefixo comum aplicado a todos os recursos específicos do teste, incluindoAWSnomes e tags de recursos. O padrão é um prefixo “gg”.

- `test-log-path`— Diretório que conterá os resultados de toda a execução do teste. O padrão é “TestResults”.
- `test-results-json`— Sinalize para determinar se um relatório JSON do Cucumber resultante foi gerado e gravado no disco. O valor padrão é verdadeiro.
- `test-results-log`— Sinalize para determinar se a saída do console foi gerada gravada no disco. O padrão é falso.
- `test-results-xml`— Sinalize para determinar se um relatório XML JUnit resultante é gerado e gravado em disco. O valor padrão é verdadeiro.
- `test-temp-path`— Diretório para gerar artefatos de teste locais. O padrão é um diretório temporário aleatório prefixado com gg-testing.
- `timeout-multiplier`— Multiplicador fornecido para todos os tempos limite de teste. O padrão é 1.0.

Tutorial: Execute end-to-end testes usando o Greengrass Testing Framework e o Greengrass Development Kit

AWS IoT GreengrassO Testing Framework (GTF) e o Greengrass Development Kit (GDK) oferecem aos desenvolvedores maneiras de executar testes. end-to-end Você pode concluir este tutorial para inicializar um projeto do GDK com um componente, inicializar um projeto do GDK com um módulo de end-to-end teste e criar um caso de teste personalizado. Depois de criar seu caso de teste personalizado, você poderá executar o teste.

Neste tutorial, você faz o seguinte:

1. Inicialize um projeto do GDK com um componente.
2. Inicialize um projeto GDK com um módulo de end-to-end teste.
3. Crie um caso de teste personalizado.
4. Adicione uma tag ao novo caso de teste.
5. Crie o JAR de teste.
6. Execute o teste do .

Tópicos

Pré-requisitos

Para concluir este tutorial, você precisará do seguinte:

- GDK versão 1.3.0 ou posterior
- Java
- Maven
- Git

Etapa 1: inicializar um projeto do GDK com um componente

- Inicialize uma pasta vazia com um projeto do GDK. Faça o download do `HelloWorld` componente implementado em Python executando o comando a seguir.

  ```
  gdk component init -t HelloWorld -l python -n HelloWorld
  ```

  Esse comando cria um novo diretório nomeado `HelloWorld` no diretório atual.

Etapa 2: inicializar um projeto do GDK com um end-to-end módulo de teste

- O GDK permite que você baixe o modelo do módulo de teste que consiste em uma implementação de recursos e etapas. Execute o comando a seguir para abrir o `HelloWorld` diretório e inicializar o projeto GDK existente usando um módulo de teste.

  ```
  cd HelloWorld
  gdk test-e2e init
  ```

Esse comando cria um novo diretório chamado `gg-e2e-tests` dentro do `HelloWorld` diretório. Esse diretório de teste é um projeto Maven que depende do JAR independente de testes do Greengrass.

Etapa 3: criar um caso de teste personalizado

Escrever um caso de teste personalizado consiste basicamente em duas etapas: criar um arquivo de recurso com um cenário de teste e implementar as definições das etapas. Para ver um exemplo de criação de um caso de teste personalizado, consulteExemplo: criar um caso de teste personalizado. Use as etapas a seguir para criar seu caso de teste personalizado:

1. Crie um arquivo de recurso com um cenário de teste

   Um recurso geralmente descreve uma funcionalidade específica do software que está sendo testado. No Cucumber, cada recurso é especificado como um arquivo de recurso individual com um título, uma descrição detalhada e um ou mais exemplos de casos específicos chamados cenários. Cada cenário consiste em um título, uma descrição detalhada e uma série de etapas que definem as interações e os resultados esperados. Os cenários são escritos em um formato estruturado usando as palavras-chave “determinado”, “quando” e “então”.
2. Implementar definições de etapas

   Uma definição de etapa vincula a etapa Gherkin em linguagem simples ao código programático. Quando o Cucumber identifica uma etapa do Gherkin em um cenário, ele procura uma definição de etapa correspondente para ser executada.

Etapa 4: adicionar uma tag ao novo caso de teste

- Você pode atribuir tags aos recursos e cenários para organizar o processo de teste. Você pode usar tags para categorizar os subconjuntos de cenários e também selecionar ganchos condicionalmente para execução. Os recursos e cenários podem ter várias tags separadas por um espaço.

  Neste exemplo, estamos usando o `HelloWorld` componente.

  No arquivo de recurso, adicione uma nova tag chamada `@HelloWorld` ao lado da `@Sample` tag.

  ```
  @Sample @HelloWorld
  ```

```
Scenario: As a developer, I can create a component and deploy it on my device
....
```

Etapa 5: criar o JAR de teste

1. Crie o componente. Você deve criar o componente antes de criar o módulo de teste.

```
gdk component build
```

2. Crie o módulo de teste usando o comando a seguir. Esse comando criará o JAR de teste na `greengrass-build` pasta.

```
gdk test-e2e build
```

Etapa 6: executar o teste

Quando você executa um caso de teste personalizado, o GTF automatiza o ciclo de vida do teste junto com o gerenciamento de recursos que foram criados durante o teste. Primeiro, ele provisiona um dispositivo em teste (DUT) como uma AWS IoT coisa e instala o software principal do Greengrass nele. Em seguida, ele criará um novo componente chamado `HelloWorld` usando a receita especificada nesse caminho. O `HelloWorld` componente é então implantado no dispositivo principal por meio de uma implantação do Greengrass Thing. Em seguida, será verificado se a implantação foi bem-sucedida. O status da implantação será alterado para `COMPLETED` dentro de 3 minutos se a implantação for bem-sucedida.

1. Acesse o `gdk-config.json` arquivo no diretório do projeto para direcionar os testes com a `HelloWorld` tag. Atualize a `test-e2e` chave usando o comando a seguir.

```
"test-e2e":{
  "gtf_options" : {
       "tags":"HelloWorld"
   }
}
```

2. Antes de executar os testes, você deve fornecer AWS credenciais para o dispositivo host. O GTF usa essas credenciais para gerenciar os AWS recursos durante o processo de teste. Certifique-se de que a função fornecida tenha permissões para automatizar as operações necessárias incluídas no teste.

Execute os comandos a seguir para fornecer as AWS credenciais.

- Linux or Unix

```
export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
```

Windows Command Prompt (CMD)

```
set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
```

PowerShell

```
$env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
$env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
```

3. Execute o teste usando o comando a seguir.

```
gdk test-e2e run
```

Esse comando baixa a versão mais recente do núcleo do Greengrass na `greengrass-build` pasta e executa testes usando-a. Esse comando também visa somente os cenários com a `HelloWorld` tag e gera um relatório para esses cenários. Você verá que os AWS recursos criados durante esse teste serão descartados no final do teste.

Exemplo: criar um caso de teste personalizado

Example

O módulo de teste baixado no projeto GDK consiste em um recurso de amostra e um arquivo de implementação de etapas.

No exemplo a seguir, criamos um arquivo de recurso para testar o recurso de implantação do software Greengrass. Testamos parcialmente a funcionalidade desse recurso com um cenário que executa a implantação de um componente por meio do GreengrassNuvem AWS. Essa é uma série de etapas que nos ajudam a entender as interações e os resultados esperados desse caso de uso.

1. Criar um arquivo de recurso

   Navegue até a `gg-e2e-tests/src/main/resources/greengrass/features` pasta no diretório atual. Você pode encontrar a amostra `component.feature` que se parece com o exemplo a seguir.

   Nesse arquivo de recurso, você pode testar o recurso de implantação do software Greengrass. Você pode testar parcialmente a funcionalidade desse recurso com um cenário que executa a implantação de um componente por meio da nuvem Greengrass. O cenário é uma série de etapas que ajudam a entender as interações e os resultados esperados desse caso de uso.

   ```
   Feature: Testing features of Greengrassv2 component

   Background:
       Given my device is registered as a Thing
       And my device is running Greengrass

   @Sample
   Scenario: As a developer, I can create a component and deploy it on my device
       When I create a Greengrass deployment with components
           HelloWorld | /path/to/recipe/file
       And I deploy the Greengrass deployment configuration
       Then the Greengrass deployment is COMPLETED on the device after 180 seconds
       And I call my custom step
   ```

   O GTF contém as definições de etapas de todas as etapas a seguir, exceto a etapa chamada:`And I call my custom step`.

2. Implementar definições de etapas

   O JAR autônomo do GTF contém as definições de etapas de todas as etapas, exceto de uma etapa:. `And I call my custom step` Você pode implementar essa etapa no módulo de teste.

   Navegue até o código-fonte do arquivo de teste. Você pode vincular sua etapa personalizada usando uma definição de etapa usando o comando a seguir.

   ```
   @And("I call my custom step")
   public void customStep() {
       System.out.println("My custom step was called ");
   ```

```
}
```

Tutorial: Use um teste de confiança da suíte de testes de confiança

AWS IoT GreengrassO Testing Framework (GTF) e o Greengrass Development Kit (GDK) oferecem aos desenvolvedores maneiras de executar testes. end-to-end Você pode concluir este tutorial para inicializar um projeto do GDK com um componente, inicializar um projeto do GDK com um módulo de end-to-end teste e usar um teste de confiança do conjunto de testes de confiança. Depois de criar seu caso de teste personalizado, você poderá executar o teste.

Um teste de confiança é um teste genérico fornecido pelo Greengrass que valida os comportamentos dos componentes fundamentais. Esses testes podem ser modificados ou estendidos para atender às necessidades mais específicas dos componentes.

Para este tutorial, usaremos um HelloWorld componente. Se você estiver usando outro componente, substitua o HelloWorld componente pelo seu componente.

Neste tutorial, você faz o seguinte:

1. Inicialize um projeto do GDK com um componente.
2. Inicialize um projeto GDK com um módulo de end-to-end teste.
3. Use um teste da suíte de testes de confiança.
4. Adicione uma tag ao novo caso de teste.
5. Crie o JAR de teste.
6. Execute o teste do .

Tópicos

Pré-requisitos

Para concluir este tutorial, você precisará do seguinte:

- GDK versão 1.6.0 ou posterior
- Java
- Maven
- Git

Etapa 1: inicializar um projeto do GDK com um componente

- Inicialize uma pasta vazia com um projeto do GDK. Faça o download do `HelloWorld` componente implementado em Python executando o comando a seguir.

  ```
  gdk component init -t HelloWorld -l python -n HelloWorld
  ```

  Esse comando cria um novo diretório nomeado `HelloWorld` no diretório atual.

Etapa 2: inicializar um projeto do GDK com um end-to-end módulo de teste

- O GDK permite que você baixe o modelo do módulo de teste que consiste em uma implementação de recursos e etapas. Execute o comando a seguir para abrir o `HelloWorld` diretório e inicializar o projeto GDK existente usando um módulo de teste.

  ```
  cd HelloWorld
  gdk test-e2e init
  ```

  Esse comando cria um novo diretório chamado `gg-e2e-tests` dentro do `HelloWorld` diretório. Esse diretório de teste é um projeto Maven que depende do JAR independente de testes do Greengrass.

Etapa 3: use um teste da suíte de testes de confiança

Escrever um caso de teste de confiança consiste em usar o arquivo de recurso fornecido e, se necessário, modificar os cenários. Para obter um exemplo de uso de um teste de confiança, consulteExemplo: criar um caso de teste personalizado. Use as etapas a seguir para usar um teste de confiança:

- Use o arquivo de recurso fornecido.

  Navegue até a `gg-e2e-tests/src/main/resources/greengrass/features` pasta no diretório atual. Abra o `confidenceTest.feature` arquivo de amostra para usar o teste de confiança.

Etapa 4: adicionar uma tag ao novo caso de teste

- Você pode atribuir tags aos recursos e cenários para organizar o processo de teste. Você pode usar tags para categorizar os subconjuntos de cenários e também selecionar ganchos condicionalmente para execução. Os recursos e cenários podem ter várias tags separadas por um espaço.

  Neste exemplo, estamos usando o `HelloWorld` componente.

  Cada cenário é marcado com`@ConfidenceTest`. Altere ou adicione tags se quiser executar somente um subconjunto da suíte de testes. Cada cenário de teste é descrito na parte superior de cada teste de confiança. O cenário é uma série de etapas que ajudam a entender as interações e os resultados esperados de cada caso de teste. Você pode estender esses testes adicionando suas próprias etapas ou modificando as existentes.

  ```
  @ConfidenceTest
  Scenario: As a Developer, I can deploy GDK_COMPONENT_NAME to my device and see it
   is working as expected
  ....
  ```

Etapa 5: criar o JAR de teste

1. Crie o componente. Você deve criar o componente antes de criar o módulo de teste.

   ```
   gdk component build
   ```

2. Crie o módulo de teste usando o comando a seguir. Esse comando criará o JAR de teste na `greengrass-build` pasta.

   ```
   gdk test-e2e build
   ```

Etapa 6: executar o teste

Quando você executa um teste de confiança, o GTF automatiza o ciclo de vida do teste, além de gerenciar os recursos que foram criados durante o teste. Primeiro, ele provisiona um dispositivo em teste (DUT) como uma AWS IoT coisa e instala o software principal do Greengrass nele. Em seguida, ele criará um novo componente chamado `HelloWorld` usando a receita especificada nesse caminho. O `HelloWorld` componente é então implantado no dispositivo principal por meio de uma implantação do Greengrass Thing. Em seguida, será verificado se a implantação foi bem-sucedida. O status da implantação será alterado para `COMPLETED` dentro de 3 minutos se a implantação for bem-sucedida.

1. Acesse o `gdk-config.json` arquivo no diretório do projeto para direcionar os testes com a `ConfidenceTest` tag ou qualquer tag que você especificou na Etapa 4. Atualize a `test-e2e` chave usando o comando a seguir.

   ```
   "test-e2e":{
     "gtf_options" : {
          "tags":"ConfidenceTest"
      }
   }
   ```

2. Antes de executar os testes, você deve fornecer AWS credenciais para o dispositivo host. O GTF usa essas credenciais para gerenciar os AWS recursos durante o processo de teste. Certifique-se de que a função fornecida tenha permissões para automatizar as operações necessárias incluídas no teste.

   Execute os comandos a seguir para fornecer as AWS credenciais.

   - Linux or Unix

     ```
     export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
     export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
     ```

     Windows Command Prompt (CMD)

     ```
     set AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
     set AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
     ```

PowerShell

```
$env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
$env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
```

3. Execute o teste usando o comando a seguir.

```
gdk test-e2e run
```

Esse comando baixa a versão mais recente do núcleo do Greengrass na `greengrass-build` pasta e executa testes usando-a. Esse comando também visa somente os cenários com a `ConfidenceTest` tag e gera um relatório para esses cenários. Você verá que os AWS recursos criados durante esse teste serão descartados no final do teste.

Exemplo: use um teste de confiança

Example

O módulo de teste baixado no projeto GDK consiste em um arquivo de recurso fornecido.

No exemplo a seguir, usamos um arquivo de recurso para testar o recurso de implantação do software Greengrass. Testamos parcialmente a funcionalidade desse recurso com um cenário que executa a implantação de um componente por meio do GreengrassNuvem AWS. Essa é uma série de etapas que nos ajudam a entender as interações e os resultados esperados desse caso de uso.

- Use o arquivo de recurso fornecido.

  Navegue até a `gg-e2e-tests/src/main/resources/greengrass/features` pasta no diretório atual. Você pode encontrar a amostra `confidenceTest.feature` que se parece com o exemplo a seguir.

```
Feature: Confidence Test Suite

Background:
    Given my device is registered as a Thing
    And my device is running Greengrass

@ConfidenceTest
Scenario: As a Developer, I can deploy GDK_COMPONENT_NAME to my device and see it
 is working as expected
```

```
    When I create a Greengrass deployment with components
      | GDK_COMPONENT_NAME | GDK_COMPONENT_RECIPE_FILE |
      | aws.greengrass.Cli | LATEST                    |
    And I deploy the Greengrass deployment configuration
    Then the Greengrass deployment is COMPLETED on the device after 180 seconds
    # Update component state accordingly. Possible states: {RUNNING, FINISHED,
 BROKEN, STOPPING}
    And I verify the GDK_COMPONENT_NAME component is RUNNING using the greengrass-
cli
```

Cada cenário de teste é descrito na parte superior de cada teste de confiança. O cenário é uma série de etapas que ajudam a entender as interações e os resultados esperados de cada caso de teste. Você pode estender esses testes adicionando suas próprias etapas ou modificando as existentes. Cada um dos cenários inclui comentários que ajudam você a fazer esses ajustes.

Desenvolva AWS IoT Greengrass componentes

Você pode desenvolver e testar componentes em seu dispositivo principal do Greengrass. Como resultado, você pode criar e iterar seu AWS IoT Greengrass software sem interagir com o. Nuvem AWS Ao finalizar uma versão do seu componente, você pode carregá-la AWS IoT Greengrass na nuvem, para que você e sua equipe possam implantar o componente em outros dispositivos da sua frota. Para obter mais informações sobre como implantar componentes, consulteImplemente AWS IoT Greengrass componentes em dispositivos.

Cada componente é composto por uma receita e artefatos.

- Receitas

  Cada componente contém um arquivo de receita, que define seus metadados. A receita também especifica os parâmetros de configuração do componente, as dependências do componente, o ciclo de vida e a compatibilidade da plataforma. O ciclo de vida do componente define os comandos que instalam, executam e desligam o componente. Para ter mais informações, consulte AWS IoT Greengrass referência da receita do componente.

  Você pode definir receitas no formato JSON ou YAML.

- Artefatos

  Os componentes podem ter qualquer número de artefatos, que são binários de componentes. Os artefatos podem incluir scripts, código compilado, recursos estáticos e quaisquer outros

arquivos que um componente consuma. Os componentes também podem consumir artefatos das dependências dos componentes.

AWS IoT Greengrassfornece componentes pré-criados que você pode usar em seus aplicativos e implantar em seus dispositivos. Por exemplo, você pode usar o componente stream manager para fazer upload de dados para vários AWS serviços ou pode usar o componente de CloudWatch métricas para publicar métricas personalizadas na Amazon CloudWatch. Para ter mais informações, consulte AWS-componentes fornecidos.

AWS IoT Greengrassorganiza um índice dos componentes do Greengrass, chamado Catálogo de Software do Greengrass. Este catálogo rastreia os componentes do Greengrass que são desenvolvidos pela comunidade do Greengrass. A partir desse catálogo, você pode baixar, modificar e implantar componentes para criar seus aplicativos Greengrass. Para ter mais informações, consulte Componentes da comunidade.

O software AWS IoT Greengrass Core executa componentes como usuário e grupo do sistema, como `ggc_user` e`ggc_group`, que você configura no dispositivo principal. Isso significa que os componentes têm as permissões desse usuário do sistema. Se você usar um usuário do sistema sem um diretório inicial, os componentes não poderão usar comandos de execução ou códigos que usem um diretório inicial. Isso significa que você não pode usar o `pip install some-library --user` comando para instalar pacotes Python, por exemplo. Se você seguiu o tutorial de introdução para configurar seu dispositivo principal, o usuário do sistema não tem um diretório inicial. Para obter mais informações sobre como configurar o usuário e o grupo que executam componentes, consulteConfigurar o usuário que executa os componentes.

> **Note**
>
> AWS IoT Greengrassusa versões semânticas para componentes. As versões semânticas seguem um sistema de numeração principal.secundária.patch. Por exemplo, a versão `1.0.0` representa a primeira versão principal de um componente. Para obter mais informações, consulte a especificação da versão semântica.

Tópicos

- Ciclo de vida do componente
- Tipos de componentes
- Crie AWS IoT Greengrass componentes

- Teste AWS IoT Greengrass componentes com implantações locais
- Publish components to deploy to your core devices
- Interaja com AWS os serviços
- Execute um contêiner Docker
- AWS IoT Greengrass referência da receita do componente
- Referência de variáveis de ambiente de componentes

Ciclo de vida do componente

O ciclo de vida do componente define os estágios que o software AWS IoT Greengrass principal usa para instalar e executar componentes. Cada estágio define um script e outras informações que especificam como o componente se comporta. Por exemplo, quando você instala um componente, o software AWS IoT Greengrass Core executa o script de `Install` ciclo de vida desse componente. Os componentes nos dispositivos principais têm os seguintes estados de ciclo de vida:

- `NEW`— A receita e os artefatos do componente são carregados no dispositivo principal, mas o componente não está instalado. Depois que um componente entra nesse estado, ele executa seu script de instalação.
- `INSTALLED`— O componente é instalado no dispositivo principal. O componente entra nesse estado depois de executar o script de instalação.
- `STARTING`— O componente está iniciando no dispositivo principal. O componente entra nesse estado quando executa seu script de inicialização. Se a inicialização for bem-sucedida, o componente entrará no `RUNNING` estado.
- `RUNNING`— O componente está sendo executado no dispositivo principal. O componente entra nesse estado quando executa o script de execução ou quando tem processos ativos em segundo plano a partir do script de inicialização.
- `FINISHED`— O componente foi executado com sucesso e concluiu sua execução.
- `STOPPING`— O componente está parando. O componente entra nesse estado quando executa seu script de desligamento.
- `ERRORED`— O componente encontrou um erro. Quando o componente entra nesse estado, ele executa seu script de recuperação. Em seguida, o componente é reiniciado para tentar retornar ao uso normal. Se o componente entrar no `ERRORED` estado três vezes sem uma execução bem-sucedida, o componente se tornará`BROKEN`.

- `BROKEN`— O componente encontrou erros várias vezes e não conseguiu se recuperar. Você deve implantar o componente novamente para corrigi-lo.

Tipos de componentes

O tipo de componente especifica como o software AWS IoT Greengrass Core executa o componente. Os componentes podem ter os seguintes tipos:

- Núcleo () `aws.greengrass.nucleus`

  O núcleo do Greengrass é o componente que fornece a funcionalidade mínima do AWS IoT Greengrass software Core. Para ter mais informações, consulte Núcleo Greengrass.

- Plug-in (`aws.greengrass.plugin`)

  O núcleo do Greengrass executa um componente de plug-in na mesma Java Virtual Machine (JVM) do núcleo. O núcleo reinicia quando você altera a versão de um componente de plug-in em um dispositivo principal. Para instalar e executar componentes do plug-in, você deve configurar o núcleo do Greengrass para ser executado como um serviço do sistema. Para ter mais informações, consulte Configurar o núcleo do Greengrass como um serviço do sistema.

  Vários componentes fornecidos pelo AWS são componentes de plug-ins, o que permite que eles interajam diretamente com o núcleo do Greengrass. Os componentes do plug-in usam o mesmo arquivo de log do Greengrass nucleus. Para ter mais informações, consulte Monitore AWS IoT Greengrass os registros.

- Genérico (`aws.greengrass.generic`)

  O núcleo do Greengrass executa os scripts de ciclo de vida de um componente genérico, se o componente definir um ciclo de vida.

  Esse tipo é o tipo padrão para componentes personalizados.

- Lambda (1) `aws.greengrass.lambda`

  O núcleo do Greengrass executa um componente de função Lambda usando o componente Lambda launcher.

  Quando você cria um componente a partir de uma função Lambda, o componente tem esse tipo. Para ter mais informações, consulte Executar AWS Lambda funções.

> **Note**
>
> Não recomendamos que você especifique o tipo de componente em uma receita. AWS IoT Greengrassdefine o tipo para você ao criar um componente.

Crie AWS IoT Greengrass componentes

Você pode desenvolver AWS IoT Greengrass componentes personalizados em um computador de desenvolvimento local ou em um dispositivo principal do Greengrass. AWS IoT Greengrass fornece a interface de linha de comando do kit de AWS IoT Greengrass desenvolvimento (GDK CLI) para ajudá-lo a criar, criar e publicar componentes a partir de modelos de componentes predefinidos e componentes da comunidade. Você também pode executar comandos de shell integrados para criar, criar e publicar componentes. Escolha entre as seguintes opções para criar componentes personalizados do Greengrass:

- Use o Greengrass Development Kit CLI

  Use a CLI do GDK para desenvolver componentes em um computador de desenvolvimento local. A CLI do GDK cria e empacota o código-fonte do componente em uma receita e artefatos que você pode publicar como um componente privado no serviço. AWS IoT Greengrass Você pode configurar a CLI do GDK para atualizar automaticamente a versão do componente e os URIs do artefato ao publicar o componente, para que você não precise atualizar a receita toda vez. Para desenvolver um componente usando a CLI do GDK, você pode começar com um modelo ou componente da comunidade do Catálogo de Software do Greengrass. Para ter mais informações, consulte AWS IoT GreengrassInterface de linha de comando do kit de desenvolvimento.
- Execute comandos de shell integrados

  Você pode executar comandos de shell integrados para desenvolver componentes em um computador de desenvolvimento local ou em um dispositivo principal do Greengrass. Você usa comandos shell para copiar ou criar o código-fonte do componente em artefatos. Cada vez que você cria uma nova versão de um componente, você deve criar ou atualizar a receita com a nova versão do componente. Ao publicar o componente no AWS IoT Greengrass serviço, você deve atualizar o URI para cada artefato do componente na receita.

Tópicos

- Criar um componente (GDK CLI)

- Crie um componente (comandos shell)

Criar um componente (GDK CLI)

Siga as instruções nesta seção para criar e criar um componente usando a CLI do GDK.

Para desenvolver um componente do Greengrass (GDK CLI)

1. Caso ainda não tenha feito isso, instale a CLI do GDK em seu computador de desenvolvimento. Para ter mais informações, consulte Instale ou atualize a interface de linha de comando do kit de AWS IoT Greengrass desenvolvimento.
2. Vá para a pasta em que você deseja criar pastas de componentes.

   Linux or Unix

   ```
   mkdir ~/greengrassv2
   cd ~/greengrassv2
   ```

   Windows Command Prompt (CMD)

   ```
   mkdir %USERPROFILE%\greengrassv2
   cd %USERPROFILE%\greengrassv2
   ```

   PowerShell

   ```
   mkdir ~/greengrassv2
   cd ~/greengrassv2
   ```

3. Escolha um modelo de componente ou componente da comunidade para baixar. A CLI do GDK baixa o modelo ou o componente da comunidade, para que você possa começar com um exemplo funcional. Use o comando component list para recuperar a lista de modelos ou componentes da comunidade disponíveis.

   - Para listar modelos de componentes, execute o comando a seguir. Cada linha na resposta inclui o nome do modelo e a linguagem de programação.

     ```
     gdk component list --template
     ```

   - Para listar os componentes da comunidade, execute o comando a seguir.

```
gdk component list --repository
```

4. Crie e altere para uma pasta de componentes na qual a CLI do GDK baixa o modelo ou o componente da comunidade. *HelloWorld*Substitua pelo nome do componente ou por outro nome que ajude a identificar essa pasta de componentes.

   Linux or Unix

   ```
   mkdir HelloWorld
   cd HelloWorld
   ```

   Windows Command Prompt (CMD)

   ```
   mkdir HelloWorld
   cd HelloWorld
   ```

   PowerShell

   ```
   mkdir HelloWorld
   cd HelloWorld
   ```

5. Baixe o modelo ou componente da comunidade para a pasta atual. Use o comando component init.

   - Para criar uma pasta de componentes a partir de um modelo, execute o comando a seguir. *HelloWorld*Substitua pelo nome do modelo e substitua *python* pelo nome da linguagem de programação.

     ```
     gdk component init --template HelloWorld --language python
     ```

   - Para criar uma pasta de componentes a partir de um componente da comunidade, execute o comando a seguir. *ComponentName*Substitua pelo nome do componente da comunidade.

     ```
     gdk component init --repository ComponentName
     ```

   > **Note**
   >
   > Se você usa o GDK CLI v1.0.0, deve executar esse comando em uma pasta vazia. A CLI do GDK baixa o modelo ou componente da comunidade para a pasta atual.

Se você usa o GDK CLI v1.1.0 ou posterior, pode especificar `--name` o argumento para especificar a pasta em que o GDK CLI baixa o modelo ou o componente da comunidade. Se você usar esse argumento, especifique uma pasta que não existe. A CLI do GDK cria a pasta para você. Se você não especificar esse argumento, a CLI do GDK usará a pasta atual, que deve estar vazia.

6. A CLI do GDK lê o arquivo de configuração da CLI do GDK, `gdk-config.json` chamado, para criar e publicar componentes. Esse arquivo de configuração existe na raiz da pasta do componente. A etapa anterior cria esse arquivo para você. Nesta etapa, você atualiza `gdk-config.json` com informações sobre seu componente. Faça o seguinte:

   a. Abra `gdk-config.json` em um editor de texto.

   b. (Opcional) Altere o nome do componente. O nome do componente é a chave no `component` objeto.

   c. Altere o autor do componente.

   d. (Opcional) Altere a versão do componente. Especifique um dos seguintes:

      - `NEXT_PATCH`— Quando você escolhe essa opção, a CLI do GDK define a versão quando você publica o componente. A CLI do GDK consulta AWS IoT Greengrass o serviço para identificar a versão mais recente publicada do componente. Em seguida, ele define a versão para a próxima versão de patch após essa versão. Se você não publicou o componente antes, a CLI do GDK usa a versão. `1.0.0`

        Se você escolher essa opção, não poderá usar a CLI do Greengrass para implantar e testar localmente o componente em seu computador de desenvolvimento local que executa o software Core. AWS IoT Greengrass Para habilitar implantações locais, você deve especificar uma versão semântica em vez disso.

      - Uma versão semântica, como**`1.0.0`**. As versões semânticas usam um major. menor. sistema de numeração de patches. Para obter mais informações, consulte a especificação da versão semântica.

        Se você desenvolver componentes em um dispositivo principal do Greengrass em que deseja implantar e testar o componente, escolha essa opção. Você deve criar o componente com uma versão específica para criar implantações locais com a CLI do Greengrass.

e. (Opcional) Altere a configuração de compilação do componente. A configuração de compilação define como a CLI do GDK transforma a fonte do componente em artefatos. Escolha entre as seguintes opções para`build_system`:

- `zip`— Empacota a pasta do componente em um arquivo ZIP para definir como o único artefato do componente. Escolha essa opção para os seguintes tipos de componentes:
  - Componentes que usam linguagens de programação interpretadas, como JavaScript Python ou.
  - Componentes que empacotam arquivos que não sejam código, como modelos de aprendizado de máquina ou outros recursos.

  A CLI do GDK compacta a pasta do componente em um arquivo zip com o mesmo nome da pasta do componente. Por exemplo, se o nome da pasta do componente for`HelloWorld`, a CLI do GDK criará um arquivo zip chamado. `HelloWorld.zip`

  > **Note**
  >
  > Se você usa o GDK CLI versão 1.0.0 em um dispositivo Windows, a pasta do componente e os nomes dos arquivos zip devem conter somente letras minúsculas.

  Quando a CLI do GDK compacta a pasta do componente em um arquivo zip, ela ignora os seguintes arquivos:

  - O arquivo `gdk-config.json`
  - O arquivo da receita (`recipe.jsonourecipe.yaml`)
  - Crie pastas, como `greengrass-build`
- `maven`— Executa o `mvn clean package` comando para transformar a fonte do componente em artefatos. Escolha essa opção para componentes que usam o Maven, como componentes Java.

  Em dispositivos Windows, esse recurso está disponível para o GDK CLI v1.1.0 e versões posteriores.
- `gradle`— Executa o `gradle build` comando para transformar a fonte do componente em artefatos. Escolha essa opção para componentes que usam o Gradle. Esse recurso está disponível para o GDK CLI v1.1.0 e versões posteriores.

O sistema de `gradle` compilação oferece suporte ao Kotlin DSL como arquivo de compilação. Esse recurso está disponível para o GDK CLI v1.2.0 e versões posteriores.

- `gradlew`— Executa o `gradlew` comando para transformar a fonte do componente em artefatos. Escolha essa opção para componentes que usam o Gradle Wrapper.

  Esse recurso está disponível para o GDK CLI v1.2.0 e versões posteriores.

- `custom`— Executa um comando personalizado para transformar a fonte do componente em uma receita e artefatos. Especifique o comando personalizado no `custom_build_command` parâmetro.

f. Se você especificar `custom` para`build_system`, adicione o `custom_build_command` ao `build` objeto. Em`custom_build_command`, especifique uma única string ou lista de strings, em que cada string é uma palavra no comando. Por exemplo, para executar um comando de compilação personalizado para um componente C++, você pode especificar**`["cmake", "--build", "build", "--config", "Release"]`**.

g. Se você usar o GDK CLI v1.1.0 ou posterior, poderá especificar `--bucket` o argumento para especificar o bucket do S3 em que o GDK CLI carrega os artefatos do componente. Se você não especificar esse argumento, a CLI do GDK será carregada no bucket do S3 cujo nome *`bucket`*`-`*`region`*`-`*`accountId`* é, *`onde`* bucket *`e`* region são os valores que você especifica e *`accountId`* *`gdk-config.json`* é seu ID. Conta da AWS A CLI do GDK cria o bucket se ele não existir.

   Altere a configuração de publicação do componente. Faça o seguinte:

   i. Especifique o nome do bucket do S3 a ser usado para hospedar artefatos do componente.

   ii. Especifique Região da AWS onde a CLI do GDK publica o componente.

Quando você concluir essa etapa, o `gdk-config.json` arquivo poderá ter uma aparência semelhante ao exemplo a seguir.

```
{
  "component": {
    "com.example.PythonHelloWorld": {
      "author": "Amazon",
      "version": "NEXT_PATCH",
      "build": {
        "build_system" : "zip"
```

```
      },
      "publish": {
        "bucket": "greengrass-component-artifacts",
        "region": "us-west-2"
      }
    }
  },
  "gdk_version": "1.0.0"
}
```

7. Atualize o arquivo de receita do componente, chamado `recipe.yaml` ou`recipe.json`. Faça o seguinte:

   a. Se você baixou um modelo ou componente da comunidade que usa o sistema de `zip` compilação, verifique se o nome do artefato zip corresponde ao nome da pasta do componente. A CLI do GDK compacta a pasta do componente em um arquivo zip com o mesmo nome da pasta do componente. A receita contém o nome do artefato zip na lista de artefatos do componente e nos scripts de ciclo de vida que usam arquivos no artefato zip. Atualize as `Lifecycle` definições `Artifacts` e de forma que o nome do arquivo zip corresponda ao nome da pasta do componente. Os exemplos de receitas parciais a seguir destacam o nome do arquivo zip nas `Lifecycle` definições `Artifacts` e.

      JSON

      ```
      {
        ...
        "Manifests": [
          {
            "Platform": {
              "os": "all"
            },
            "Artifacts": [
              {
                "URI": "s3://{COMPONENT_NAME}/{COMPONENT_VERSION}/HelloWorld.zip",
                "Unarchive": "ZIP"
              }
            ],
            "Lifecycle": {
              "run": "python3 -u {artifacts:decompressedPath}/HelloWorld/main.py
       {configuration:/Message}"
            }
          }
      ```

```
  ]
}
```

YAML

```
---
...
Manifests:
  - Platform:
      os: all
    Artifacts:
      - URI: "s3://BUCKET_NAME/COMPONENT_NAME/
COMPONENT_VERSION/HelloWorld.zip"
        Unarchive: ZIP
    Lifecycle:
      run: "python3 -u {artifacts:decompressedPath}/HelloWorld/main.py
 {configuration:/Message}"
```

b. (Opcional) Atualize a descrição do componente, a configuração padrão, os artefatos, os scripts de ciclo de vida e o suporte à plataforma. Para ter mais informações, consulte AWS IoT Greengrass referência da receita do componente.

Quando você terminar essa etapa, o arquivo da receita poderá ter uma aparência semelhante aos exemplos a seguir.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "{COMPONENT_NAME}",
  "ComponentVersion": "{COMPONENT_VERSION}",
  "ComponentDescription": "This is a simple Hello World component written in
 Python.",
  "ComponentPublisher": "{COMPONENT_AUTHOR}",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "Message": "World"
    }
  },
  "Manifests": [
    {
      "Platform": {
```

```
        "os": "all"
      },
      "Artifacts": [
        {
          "URI": "s3://{COMPONENT_NAME}/{COMPONENT_VERSION}/HelloWorld.zip",
          "Unarchive": "ZIP"
        }
      ],
      "Lifecycle": {
        "run": "python3 -u {artifacts:decompressedPath}/HelloWorld/main.py
 {configuration:/Message}"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: "2020-01-25"
ComponentName: "{COMPONENT_NAME}"
ComponentVersion: "{COMPONENT_VERSION}"
ComponentDescription: "This is a simple Hello World component written in
 Python."
ComponentPublisher: "{COMPONENT_AUTHOR}"
ComponentConfiguration:
  DefaultConfiguration:
    Message: "World"
Manifests:
  - Platform:
      os: all
    Artifacts:
      - URI: "s3://BUCKET_NAME/COMPONENT_NAME/COMPONENT_VERSION/HelloWorld.zip"
        Unarchive: ZIP
    Lifecycle:
      run: "python3 -u {artifacts:decompressedPath}/HelloWorld/main.py
 {configuration:/Message}"
```

8. Desenvolva e construa o componente Greengrass. O comando de construção do componente produz uma receita e artefatos na `greengrass-build` pasta do componente. Execute o seguinte comando .

```
gdk component build
```

Quando estiver pronto para testar seu componente, use a CLI do GDK para publicá-lo no serviço. AWS IoT Greengrass Em seguida, você pode implantar o componente nos dispositivos principais do Greengrass. Para ter mais informações, consulte Publish components to deploy to your core devices.

Crie um componente (comandos shell)

Siga as instruções nesta seção para criar pastas de receitas e artefatos que contêm código-fonte e artefatos para vários componentes.

Para desenvolver um componente do Greengrass (comandos shell)

1. Crie uma pasta para seus componentes com subpastas para receitas e artefatos. Execute os comandos a seguir em seu dispositivo principal do Greengrass para criar essas pastas e mudar para a pasta do componente. Substitua *~/greengrassv2* ou *%USERPROFILE%\greengrassv2* pelo caminho para a pasta a ser usada no desenvolvimento local.

   Linux or Unix

   ```
   mkdir -p ~/greengrassv2/{recipes,artifacts}
   cd ~/greengrassv2
   ```

   Windows Command Prompt (CMD)

   ```
   mkdir %USERPROFILE%\greengrassv2\\recipes, %USERPROFILE%\greengrassv2\\artifacts
   cd %USERPROFILE%\greengrassv2
   ```

   PowerShell

   ```
   mkdir ~/greengrassv2/recipes, ~/greengrassv2/artifacts
   cd ~/greengrassv2
   ```

2. Use um editor de texto para criar um arquivo de receita que defina os metadados, os parâmetros, as dependências, o ciclo de vida e a capacidade da plataforma do seu componente. Inclua a versão do componente no nome do arquivo da receita para que você possa identificar qual receita reflete qual versão do componente. Você pode escolher o formato YAML ou JSON para sua receita.

Por exemplo, em um sistema baseado em Linux, você pode executar o seguinte comando para usar o GNU nano para criar o arquivo.

JSON

```
nano recipes/com.example.HelloWorld-1.0.0.json
```

YAML

```
nano recipes/com.example.HelloWorld-1.0.0.yaml
```

> **Note**
>
> AWS IoT Greengrass usa versões semânticas para componentes. As versões semânticas seguem um sistema de numeração principal.secundária.patch. Por exemplo, a versão `1.0.0` representa a primeira versão principal de um componente. Para obter mais informações, consulte a especificação da versão semântica.

3. Defina a receita do seu componente. Para ter mais informações, consulte AWS IoT Greengrass referência da receita do componente.

   Sua receita pode ser semelhante à seguinte receita de exemplo do Hello World.

   JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.HelloWorld",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "My first AWS IoT Greengrass component.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "Message": "world"
    }
  },
  "Manifests": [
    {
      "Platform": {
```

```
        "os": "linux"
      },
      "Lifecycle": {
        "run": "python3 -u {artifacts:path}/hello_world.py {configuration:/
Message}"
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "run": "py -3 -u {artifacts:path}/hello_world.py {configuration:/
Message}"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.HelloWorld
ComponentVersion: '1.0.0'
ComponentDescription: My first AWS IoT Greengrass component.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    Message: world
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      run: |
        python3 -u {artifacts:path}/hello_world.py "{configuration:/Message}"
  - Platform:
      os: windows
    Lifecycle:
      run: |
        py -3 -u {artifacts:path}/hello_world.py "{configuration:/Message}"
```

Essa receita executa um script Hello World Python, que pode ser semelhante ao script de exemplo a seguir.

```
import sys

message = "Hello, %s!" % sys.argv[1]

# Print the message to stdout, which Greengrass saves in a log file.
print(message)
```

4. Crie uma pasta para a versão do componente ser desenvolvida. Recomendamos que você use uma pasta separada para os artefatos de cada versão do componente para que você possa identificar quais artefatos são de cada versão do componente. Execute o seguinte comando .

   Linux or Unix

   ```
   mkdir -p artifacts/com.example.HelloWorld/1.0.0
   ```

   Windows Command Prompt (CMD)

   ```
   mkdir artifacts/com.example.HelloWorld/1.0.0
   ```

   PowerShell

   ```
   mkdir artifacts/com.example.HelloWorld/1.0.0
   ```

   Important

   Você deve usar o seguinte formato para o caminho da pasta de artefatos. Inclua o nome e a versão do componente que você especificar na receita.

   ```
   artifacts/componentName/componentVersion/
   ```

5. Crie os artefatos para seu componente na pasta que você criou na etapa anterior. Os artefatos podem incluir software, imagens e quaisquer outros binários que seu componente usa.

   Quando seu componente estiver pronto, teste seu componente.

Teste AWS IoT Greengrass componentes com implantações locais

Se você desenvolver um componente do Greengrass em um dispositivo principal, poderá criar uma implantação local para instalá-lo e testá-lo. Siga as etapas desta seção para criar uma implantação local.

Se você desenvolver o componente em um computador diferente, como um computador de desenvolvimento local, não poderá criar uma implantação local. Em vez disso, publique o componente no AWS IoT Greengrass serviço para que você possa implantá-lo nos dispositivos principais do Greengrass para testá-lo. Para obter mais informações, consulte Implemente AWS IoT Greengrass componentes em dispositivos e Publish components to deploy to your core devices.

Para testar um componente em um dispositivo principal do Greengrass

1. O dispositivo principal registra eventos, como atualizações de componentes. Você pode visualizar esse arquivo de log para descobrir e solucionar erros em seu componente, como uma receita inválida. Esse arquivo de log também exibe mensagens que seu componente imprime na saída padrão (stdout). Recomendamos que você abra uma sessão de terminal adicional em seu dispositivo principal para observar novas mensagens de registro em tempo real. Abra uma nova sessão de terminal, por exemplo, por meio de SSH, e execute o comando a seguir para visualizar os registros. */greengrass/v2*Substitua pelo caminho para a pasta AWS IoT Greengrass raiz.

   Linux or Unix

   ```
   sudo tail -f /greengrass/v2/logs/greengrass.log
   ```

   PowerShell

   ```
   gc C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
   ```

   Você também pode visualizar o arquivo de log do seu componente.

   Linux or Unix

   ```
   sudo tail -f /greengrass/v2/logs/com.example.HelloWorld.log
   ```

PowerShell

```
gc C:\greengrass\v2\logs\com.example.HelloWorld.log -Tail 10 -Wait
```

2. Na sessão original do terminal, execute o comando a seguir para atualizar o dispositivo principal com seu componente. */greengrass/v2*Substitua pelo caminho para a pasta AWS IoT Greengrass raiz e substitua *~/greengrassv2* pelo caminho para sua pasta de desenvolvimento local.

   Linux or Unix

   ```
   sudo /greengrass/v2/bin/greengrass-cli deployment create \
     --recipeDir ~/greengrassv2/recipes \
     --artifactDir ~/greengrassv2/artifacts \
     --merge "com.example.HelloWorld=1.0.0"
   ```

   Windows Command Prompt (CMD)

   ```
   C:\greengrass\v2\bin\greengrass-cli deployment create ^
     --recipeDir %USERPROFILE%\greengrassv2\recipes ^
     --artifactDir %USERPROFILE%\greengrassv2\artifacts ^
     --merge "com.example.HelloWorld=1.0.0"
   ```

   PowerShell

   ```
   C:\greengrass\v2\bin\greengrass-cli deployment create `
     --recipeDir ~/greengrassv2/recipes `
     --artifactDir ~/greengrassv2/artifacts `
     --merge "com.example.HelloWorld=1.0.0"
   ```

   > **Note**
   >
   > Você também pode usar o `greengrass-cli deployment create` comando para definir o valor dos parâmetros de configuração do seu componente. Para ter mais informações, consulte criar.

3. Use o `greengrass-cli deployment status` comando para monitorar o progresso da implantação do seu componente.

Unix or Linux

```
sudo /greengrass/v2/bin/greengrass-cli deployment status \
  -i deployment-id
```

Windows Command Prompt (CMD)

```
C:\greengrass\v2\bin\greengrass-cli deployment status ^
  -i deployment-id
```

PowerShell

```
C:\greengrass\v2\bin\greengrass-cli deployment status `
  -i deployment-id
```

4. Teste seu componente enquanto ele é executado no dispositivo principal do Greengrass. Ao concluir essa versão do seu componente, você poderá carregá-la no AWS IoT Greengrass serviço. Em seguida, você pode implantar o componente em outros dispositivos principais. Para ter mais informações, consulte Publish components to deploy to your core devices.

Publish components to deploy to your core devices

Depois de criar ou concluir uma versão de um componente, você pode publicá-la no AWS IoT Greengrass serviço. Em seguida, você pode implantá-lo nos dispositivos principais do Greengrass.

Se você usa a CLI do Greengrass Development Kit (GDK CLI) para desenvolver e criar um componente, você pode usar a CLI do GDK para publicar o componente no. Nuvem AWS Caso contrário, use os comandos shell integrados e o AWS CLI para publicar o componente.

Você também pode usar AWS CloudFormation para criar componentes e outros AWS recursos a partir de modelos. Para obter mais informações, consulte O que é AWS CloudFormation? e AWS::GreengrassV2::ComponentVersionno Guia do AWS CloudFormation usuário.

Tópicos

- Publicar um componente (GDK CLI)
- Publicar um componente (comandos do shell)

Publicar um componente (GDK CLI)

Siga as instruções nesta seção para publicar um componente usando a CLI do GDK. A CLI do GDK carrega artefatos de construção em um bucket do S3, atualiza os URIs do artefato na receita e cria o componente a partir da receita. Você especifica o bucket e a região do S3 a serem usados no arquivo de configuração da CLI do GDK.

Se você usar o GDK CLI v1.1.0 ou posterior, poderá especificar `--bucket` o argumento para especificar o bucket do S3 em que o GDK CLI carrega os artefatos do componente. Se você não especificar esse argumento, a CLI do GDK será carregada no bucket do S3 cujo nome *`bucket`*`-`*`region`*`-`*`accountId`* é, *`onde`* bucket *`e`* region são os valores que você especifica e *`accountId gdk-config.json`* é seu ID. Conta da AWS A CLI do GDK cria o bucket se ele não existir.

> **Important**
>
> As funções principais do dispositivo não permitem acesso aos buckets do S3 por padrão. Se for a primeira vez que você usa esse bucket do S3, você deve adicionar permissões à função para permitir que os dispositivos principais recuperem artefatos de componentes desse bucket do S3. Para ter mais informações, consulte Permitir acesso aos buckets do S3 para artefatos de componentes.

Para publicar um componente do Greengrass (GDK CLI)

1. Abra a pasta do componente em um prompt de comando ou terminal.
2. Se você ainda não o fez, crie o componente Greengrass. O comando de construção do componente produz uma receita e artefatos na `greengrass-build` pasta do componente. Execute o seguinte comando .

   ```
   gdk component build
   ```

3. Publique o componente no Nuvem AWS. O comando de publicação do componente carrega os artefatos do componente para o Amazon S3 e atualiza a receita do componente com o URI de cada artefato. Em seguida, ele cria o componente no AWS IoT Greengrass serviço.

> **Note**
>
> AWS IoT Greengrass calcula o resumo de cada artefato quando você cria o componente. Isso significa que você não pode modificar os arquivos de artefatos em seu bucket do S3 depois de criar um componente. Se você fizer isso, as implantações que incluam esse componente falharão, porque o resumo do arquivo não corresponde. Se você modificar um arquivo de artefato, deverá criar uma nova versão do componente.

Se você especificar `NEXT_PATCH` a versão do componente no arquivo de configuração da CLI do GDK, a CLI do GDK usará a próxima versão do patch que ainda não existe no serviço. AWS IoT Greengrass

Execute o seguinte comando .

```
gdk component publish
```

A saída informa a versão do componente que a CLI do GDK criou.

Depois de publicar o componente, você pode implantar o componente nos dispositivos principais. Para ter mais informações, consulte Implemente AWS IoT Greengrass componentes em dispositivos.

Publicar um componente (comandos do shell)

Use o procedimento a seguir para publicar um componente usando comandos shell e o AWS Command Line Interface (AWS CLI). Ao publicar um componente, você faz o seguinte:

1. Publique artefatos de componentes em um bucket do S3.
2. Adicione o URI do Amazon S3 de cada artefato à receita do componente.
3. Crie uma versão do componente a AWS IoT Greengrass partir da receita do componente.

> **Note**
>
> Cada versão do componente que você carrega deve ser exclusiva. Certifique-se de fazer o upload da versão correta do componente, pois você não poderá editá-la depois de carregá-la.

Você pode seguir estas etapas para publicar um componente do seu computador de desenvolvimento ou do seu dispositivo principal do Greengrass.

Para publicar um componente (comandos shell)

1. Se o componente usar uma versão que existe no AWS IoT Greengrass serviço, você deverá alterar a versão do componente. Abra a receita em um editor de texto, incremente a versão e salve o arquivo. Escolha uma nova versão que reflita as alterações feitas no componente.

   > **Note**
   >
   > AWS IoT Greengrass usa versões semânticas para componentes. As versões semânticas seguem um sistema de numeração principal.secundária.patch. Por exemplo, a versão `1.0.0` representa a primeira versão principal de um componente. Para obter mais informações, consulte a especificação da versão semântica.

2. Se seu componente tiver artefatos, faça o seguinte:

   a. Publique os artefatos do componente em um bucket do S3 no seu. Conta da AWS

      > **Tip**
      >
      > Recomendamos que você inclua o nome e a versão do componente no caminho para o artefato no bucket do S3. Esse esquema de nomenclatura pode ajudá-lo a manter os artefatos que as versões anteriores do componente usam, para que você possa continuar oferecendo suporte às versões anteriores do componente.

      Execute o comando a seguir para publicar um arquivo de artefato em um bucket do S3. *Substitua DOC-EXAMPLE-BUCKET pelo nome do bucket e substitua*

artifacts/com.example. HelloWorld/1.0.0/artifact.py com o caminho para o arquivo do artefato.

```
aws s3 cp artifacts/com.example.HelloWorld/1.0.0/artifact.py s3://DOC-EXAMPLE-BUCKET/artifacts/com.example.HelloWorld/1.0.0/artifact.py
```

> **Important**
>
> As funções principais do dispositivo não permitem acesso aos buckets do S3 por padrão. Se for a primeira vez que você usa esse bucket do S3, você deve adicionar permissões à função para permitir que os dispositivos principais recuperem artefatos de componentes desse bucket do S3. Para ter mais informações, consulte Permitir acesso aos buckets do S3 para artefatos de componentes.

b. Adicione uma lista nomeada `Artifacts` à receita do componente se ela não estiver presente. A `Artifacts` lista aparece em cada manifesto, que define os requisitos do componente em cada plataforma que ele suporta (ou os requisitos padrão do componente para todas as plataformas).

c. Adicione cada artefato à lista de artefatos ou atualize o URI dos artefatos existentes. O URI do Amazon S3 é composto pelo nome do bucket e pelo caminho para o objeto de artefato no bucket. Os URIs do Amazon S3 de seus artefatos devem ser semelhantes ao exemplo a seguir.

```
s3://DOC-EXAMPLE-BUCKET/artifacts/com.example.HelloWorld/1.0.0/artifact.py
```

Depois de concluir essas etapas, sua receita deve ter uma `Artifacts` lista semelhante à seguinte.

JSON

```
{
  ...
  "Manifests": [
    {
      "Lifecycle": {
        ...
      },
      "Artifacts": [
```

```
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/MyGreengrassComponent/1.0.0/
artifact.py",
          "Unarchive": "NONE"
        }
      ]
    }
  ]
}
```

> **Note**
>
> Você pode adicionar a `"Unarchive": "ZIP"` opção de um artefato ZIP para configurar o software AWS IoT Greengrass Core para descompactar o artefato quando o componente for implantado.

YAML

```
...
Manifests:
  - Lifecycle:
      ...
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/MyGreengrassComponent/1.0.0/
artifact.py
        Unarchive: NONE
```

> **Note**
>
> Você pode usar a `Unarchive: ZIP` opção de configurar o software AWS IoT Greengrass Core para descompactar um artefato ZIP quando o componente for implantado. Para obter mais informações sobre como usar artefatos ZIP em um componente, consulte a variável de receita artifacts:decompressedPath.

Para obter mais informações sobre receitas, consulte AWS IoT Greengrass referência da receita do componente.

3. Use o AWS IoT Greengrass console para criar um componente a partir do arquivo de receita.

Execute o comando a seguir para criar o componente a partir de um arquivo de receita. Esse comando cria o componente e o publica como um AWS IoT Greengrass componente privado no seu Conta da AWS. Substitua *path/to/recipeFile pelo caminho para o arquivo* de receita.

```
aws greengrassv2 create-component-version --inline-recipe fileb://path/to/
recipeFile
```

Copie o `arn` da resposta para verificar o estado do componente na próxima etapa.

> **Note**
>
> AWS IoT Greengrass calcula o resumo de cada artefato quando você cria o componente. Isso significa que você não pode modificar os arquivos de artefatos em seu bucket do S3 depois de criar um componente. Se você fizer isso, as implantações que incluam esse componente falharão, porque o resumo do arquivo não corresponde. Se você modificar um arquivo de artefato, deverá criar uma nova versão do componente.

4. Cada componente no AWS IoT Greengrass serviço tem um estado. Execute o comando a seguir para confirmar o estado da versão do componente que você publica neste procedimento. Substitua *com.example. HelloWorld*e *1.0.0* com a versão do componente a ser consultada. `arn`Substitua o pelo ARN da etapa anterior.

   ```
   aws greengrassv2 describe-component --arn "arn:aws:greengrass:region:account-
   id:components:com.example.HelloWorld:versions:1.0.0"
   ```

   A operação retorna uma resposta que contém os metadados do componente. Os metadados contêm um `status` objeto que contém o estado do componente e quaisquer erros, se aplicável.

   Quando o estado do componente é`DEPLOYABLE`, você pode implantar o componente nos dispositivos. Para ter mais informações, consulte Implemente AWS IoT Greengrass componentes em dispositivos.

Interaja com AWS os serviços

Os principais dispositivos do Greengrass usam certificados X.509 para se conectar AWS IoT Core usando protocolos de autenticação mútua TLS. Esses certificados permitem que os dispositivos

interajam AWS IoT sem AWS credenciais, que normalmente incluem uma ID de chave de acesso e uma chave de acesso secreta. Outros AWS serviços exigem AWS credenciais em vez de certificados X.509 para chamar operações de API nos endpoints do serviço. AWS IoT Coretem um provedor de credenciais que permite que os dispositivos usem seu certificado X.509 para autenticar solicitações. AWS O provedor de AWS IoT credenciais autentica dispositivos usando um certificado X.509 e emite AWS credenciais na forma de um token de segurança temporário com privilégios limitados. Os dispositivos podem usar esse token para assinar e autenticar qualquer AWS solicitação. Isso elimina a necessidade de armazenar AWS credenciais nos dispositivos principais do Greengrass. Para obter mais informações, consulte Autorização de chamadas diretas para AWS serviços no Guia do AWS IoT Core desenvolvedor.

Para obter credenciais do AWS IoT Greengrass, os dispositivos principais usam um alias de função que aponta para uma AWS IoT função do IAM. Essa função do IAM é chamada de função de troca de tokens. Você cria o alias da função e a função de troca de tokens ao instalar o software AWS IoT Greengrass Core. Para especificar o alias de função que um dispositivo principal usa, configure o `iotRoleAlias` parâmetro doNúcleo Greengrass.

O provedor de AWS IoT credenciais assume a função de troca de tokens em seu nome para fornecer AWS credenciais aos dispositivos principais. Você pode anexar políticas apropriadas do IAM a essa função para permitir que seus dispositivos principais acessem seus AWS recursos, como artefatos de componentes em buckets do S3. Para obter mais informações sobre como configurar a função de troca de tokens, consulteAutorize os dispositivos principais a interagir com os serviços AWS.

Os principais dispositivos do Greengrass armazenam AWS as credenciais na memória e, por padrão, as credenciais expiram após uma hora. Se o software AWS IoT Greengrass principal for reiniciado, ele deverá buscar as credenciais novamente. Você pode usar a UpdateRoleAliasoperação para configurar a duração em que as credenciais são válidas.

AWS IoT Greengrassfornece um componente público, o componente do serviço de troca de tokens, que você pode definir como uma dependência em seu componente personalizado para interagir com AWS os serviços. O serviço de troca de tokens fornece ao seu componente uma variável de ambiente,`AWS_CONTAINER_CREDENTIALS_FULL_URI`, que define o URI para um servidor local que fornece AWS credenciais. Quando você cria um cliente AWS SDK, o cliente verifica essa variável de ambiente e se conecta ao servidor local para recuperar AWS as credenciais e as usa para assinar solicitações de API. Isso permite que você use AWS SDKs e outras ferramentas para chamar AWS serviços em seus componentes. Para ter mais informações, consulte Serviço de troca de tokens.

Important

Support para adquirir AWS credenciais dessa forma foi adicionado aos AWS SDKs em 13 de julho de 2016. Seu componente deve usar uma versão do AWS SDK criada nessa data ou após essa data. Para obter mais informações, consulte Como usar um AWS SDK compatível no Amazon Elastic Container Service Developer Guide.

Para adquirir AWS credenciais em seu componente personalizado, defina `aws.greengrass.TokenExchangeService` como uma dependência na receita do componente. O exemplo de receita a seguir define um componente que instala o boto3 e executa um script Python que usa AWS credenciais do serviço de troca de tokens para listar buckets do Amazon S3.

Note

Para executar esse componente de exemplo, seu dispositivo deve ter a `s3:ListAllMyBuckets` permissão. Para ter mais informações, consulte Autorize os dispositivos principais a interagir com os serviços AWS.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.ListS3Buckets",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that uses the token exchange service to list
 S3 buckets.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.TokenExchangeService": {
      "VersionRequirement": "^2.0.0",
      "DependencyType": "HARD"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
```

```
      "Lifecycle": {
        "install": "pip3 install --user boto3",
        "run": "python3 -u {artifacts:path}/list_s3_buckets.py"
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "install": "pip3 install --user boto3",
        "run": "py -3 -u {artifacts:path}/list_s3_buckets.py"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.ListS3Buckets
ComponentVersion: '1.0.0'
ComponentDescription: A component that uses the token exchange service to list S3
 buckets.
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.TokenExchangeService:
    VersionRequirement: '^2.0.0'
    DependencyType: HARD
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install:
        pip3 install --user boto3
      run: |-
        python3 -u {artifacts:path}/list_s3_buckets.py
  - Platform:
      os: windows
    Lifecycle:
      install:
        pip3 install --user boto3
```

```
      run: |-
        py -3 -u {artifacts:path}/list_s3_buckets.py
```

Esse componente de exemplo executa o seguinte script Python, `list_s3_buckets.py` que lista os buckets do Amazon S3.

```
import boto3
import os

try:
    print("Creating boto3 S3 client...")
    s3 = boto3.client('s3')
    print("Successfully created boto3 S3 client")
except Exception as e:
    print("Failed to create boto3 s3 client. Error: " + str(e))
    exit(1)

try:
    print("Listing S3 buckets...")
    response = s3.list_buckets()
    for bucket in response['Buckets']:
        print(f'\t{bucket["Name"]}')
    print("Successfully listed S3 buckets")
except Exception as e:
    print("Failed to list S3 buckets. Error: " + str(e))
    exit(1)
```

Execute um contêiner Docker

Você pode configurar AWS IoT Greengrass componentes para executar um contêiner Docker a partir de imagens armazenadas nos seguintes locais:

- Repositórios de imagens públicos e privados no Amazon Elastic Container Registry (Amazon ECR)
- Repositório público do Docker Hub
- Registro confiável do Public Docker
- Bucket do S3

Em seu componente personalizado, inclua o URI da imagem do Docker como um artefato para recuperar a imagem e executá-la no dispositivo principal. Para imagens do Amazon ECR e do

Docker Hub, você pode usar o componente Docker Application Manager para baixar as imagens e gerenciar credenciais para repositórios privados do Amazon ECR.

Tópicos

Requisitos

Para executar um contêiner Docker em um componente, você precisa do seguinte:

- Um dispositivo principal do Greengrass. Se você não tiver uma, consulte Tutorial: Conceitos básicos do AWS IoT Greengrass V2.
- Docker Engine 1.9.1 ou posterior instalado no dispositivo principal do Greengrass. A versão 20.10 é a versão mais recente verificada para funcionar com o software AWS IoT Greengrass Core. Você deve instalar o Docker diretamente no dispositivo principal antes de implantar componentes que executam contêineres do Docker.

> **Tip**
>
> Você também pode configurar o dispositivo principal para instalar o Docker Engine quando o componente for instalado. Por exemplo, o script de instalação a seguir instala o Docker Engine antes de carregar a imagem do Docker. Esse script de instalação funciona em distribuições Linux baseadas no Debian, como o Ubuntu. Se você configurar o componente para instalar o Docker Engine com esse comando, talvez seja necessário `RequiresPrivilege` configurá-lo `true` no script do ciclo de vida para executar a instalação e os comandos. `docker` Para ter mais informações, consulte AWS IoT Greengrass referência da receita do componente.

```
apt-get install docker-ce docker-ce-cli containerd.io && docker load -i
 {artifacts:path}/hello-world.tar
```

- O usuário do sistema que executa um componente de contêiner do Docker deve ter permissões de raiz ou administrador, ou você deve configurar o Docker para executá-lo como usuário não raiz ou não administrador.
  - Em dispositivos Linux, você pode adicionar um usuário ao `docker` grupo para chamar `docker` comandos sem `sudo`.
  - Em dispositivos Windows, você pode adicionar um usuário ao `docker-users` grupo para chamar `docker` comandos sem privilégios de administrador.

  Linux or Unix

  Para adicionar `ggc_user` ou o usuário não root que você usa para executar componentes de contêiner do Docker ao `docker` grupo, execute o comando a seguir.

  ```
  sudo usermod -aG docker ggc_user
  ```

  Para obter mais informações, consulte Gerenciar o Docker como usuário não root.

  Windows Command Prompt (CMD)

  Para adicionar `ggc_user` ou o usuário que você usa para executar componentes de contêiner do Docker ao `docker-users` grupo, execute o comando a seguir como administrador.

  ```
  net localgroup docker-users ggc_user /add
  ```

  Windows PowerShell

  Para adicionar `ggc_user` ou o usuário que você usa para executar componentes de contêiner do Docker ao `docker-users` grupo, execute o comando a seguir como administrador.

  ```
  Add-LocalGroupMember -Group docker-users -Member ggc_user
  ```

- Arquivos acessados pelo componente de contêiner Docker montado como um volume no contêiner Docker.
- Se você configurar o software AWS IoT Greengrass Core para usar um proxy de rede, deverá configurar o Docker para usar o mesmo servidor proxy.

Além desses requisitos, você também deve atender aos seguintes requisitos se eles se aplicarem ao seu ambiente:

- Para usar o Docker Compose para criar e iniciar seus contêineres do Docker, instale o Docker Compose em seu dispositivo principal do Greengrass e carregue seu arquivo Docker Compose em um bucket do S3. Você deve armazenar seu arquivo Compose em um bucket do S3 no mesmo componente Conta da AWS e no Região da AWS mesmo. Para ver um exemplo que usa o `docker-compose up` comando em um componente personalizado, consulteExecute um contêiner Docker a partir de uma imagem pública no Amazon ECR ou no Docker Hub.
- Se você estiver executando AWS IoT Greengrass por trás de um proxy de rede, configure o daemon do Docker para usar um servidor proxy.
- Se suas imagens do Docker estiverem armazenadas no Amazon ECR ou no Docker Hub, inclua o componente gerenciador de componentes do Docker como uma dependência em seu componente de contêiner do Docker. Você deve iniciar o daemon do Docker no dispositivo principal antes de implantar seu componente.

  Além disso, inclua os URIs da imagem como artefatos do componente. Os URIs de imagem devem estar no formato `docker:`*`registry`*`/`*`image`*`[`*`:tag|@digest`*`]` mostrado nos exemplos a seguir:

  - Imagem privada do Amazon ECR: `docker:`*`account-id`*`.dkr.ecr.`*`region`*`.amazonaws.com/`*`repository`*`/`*`image`*`[`*`:tag|@digest`*`]`
  - Imagem pública do Amazon ECR: `docker:public.ecr.aws/`*`repository`*`/`*`image`*`[`*`:tag|@digest`*`]`
  - Imagem pública do Docker Hub: `docker:`*`name`*`[`*`:tag|@digest`*`]`

  Para obter mais informações sobre como executar contêineres do Docker a partir de imagens armazenadas em repositórios públicos, consulte. Execute um contêiner Docker a partir de uma imagem pública no Amazon ECR ou no Docker Hub
- Se suas imagens do Docker estiverem armazenadas em um repositório privado do Amazon ECR, você deverá incluir o componente do serviço de troca de tokens como uma dependência no componente de contêiner do Docker. Além disso, a função de dispositivo do Greengrass deve permitir as `ecr:GetDownloadUrlForLayer` ações`ecr:GetAuthorizationToken`,`ecr:BatchGetImage`, e, conforme mostrado no exemplo de política do IAM a seguir.

```
{
  "Version": "2012-10-17",
  "Statement": [
```

```
    {
      "Action": [
        "ecr:GetAuthorizationToken",
        "ecr:BatchGetImage",
        "ecr:GetDownloadUrlForLayer"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

Para obter informações sobre a execução de contêineres Docker a partir de imagens armazenadas em um repositório privado do Amazon ECR, consulte. Execute um contêiner Docker a partir de uma imagem privada no Amazon ECR

- Para usar imagens do Docker armazenadas em um repositório privado do Amazon ECR, o repositório privado deve estar no mesmo que o dispositivo principal Região da AWS .
- Se suas imagens do Docker ou arquivos do Compose estiverem armazenados em um bucket do S3, a função de dispositivo do Greengrass deve permitir que os `s3:GetObject` dispositivos principais baixem as imagens como artefatos de componentes, conforme mostrado no exemplo de política do IAM a seguir.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

Para obter informações sobre a execução de contêineres Docker a partir de imagens armazenadas no Amazon S3, consulte. Execute um contêiner Docker a partir de uma imagem no Amazon S3

- Para usar comunicação entre processos (IPC), AWS credenciais ou gerenciador de fluxo em seu componente de contêiner do Docker, você deve especificar opções adicionais ao executar o contêiner do Docker. Para mais informações, consulte:
  - Use a comunicação entre processos nos componentes do contêiner Docker
  - Use AWS credenciais em componentes de contêiner do Docker (Linux)
  - Use o gerenciador de fluxo em componentes de contêiner Docker (Linux)

Execute um contêiner Docker a partir de uma imagem pública no Amazon ECR ou no Docker Hub

Esta seção descreve como você pode criar um componente personalizado que usa o Docker Compose para executar um contêiner Docker a partir de imagens do Docker armazenadas no Amazon ECR e no Docker Hub.

Para executar um contêiner Docker usando o Docker Compose

1. Crie e faça upload de um arquivo Docker Compose para um bucket do Amazon S3. Certifique-se de que a função de dispositivo do Greengrass permita que o `s3:GetObject` dispositivo acesse o arquivo Compose. O exemplo de arquivo Compose mostrado no exemplo a seguir inclui a imagem do Amazon CloudWatch Agent do Amazon ECR e a imagem do MySQL do Docker Hub.

   ```
   version: "3"
   services:
     cloudwatchagent:
       image: "public.ecr.aws/cloudwatch-agent/cloudwatch-agent:latest"
     mysql:
       image: "mysql:8.0"
   ```

2. Crie um componente personalizado em seu dispositivo AWS IoT Greengrass principal. A receita de exemplo mostrada no exemplo a seguir tem as seguintes propriedades:

   - O componente do gerenciador de aplicativos Docker como uma dependência. Esse componente permite AWS IoT Greengrass baixar imagens dos repositórios públicos do Amazon ECR e do Docker Hub.

- Um artefato de componente que especifica uma imagem do Docker em um repositório público do Amazon ECR.
- Um artefato de componente que especifica uma imagem do Docker em um repositório público do Docker Hub.
- Um artefato de componente que especifica o arquivo Docker Compose que inclui contêineres para as imagens do Docker que você deseja executar.
- Um script de execução do ciclo de vida que usa docker-compose up para criar e iniciar um contêiner a partir das imagens especificadas.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.MyDockerComposeComponent",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that uses Docker Compose to run images
 from public Amazon ECR and Docker Hub.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.DockerApplicationManager": {
      "VersionRequirement": "~2.0.0"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "all"
      },
      "Lifecycle": {
        "run": "docker-compose -f {artifacts:path}/docker-compose.yaml up"
      },
      "Artifacts": [
        {
          "URI": "docker:public.ecr.aws/cloudwatch-agent/cloudwatch-
agent:latest"
        },
        {
          "URI": "docker:mysql:8.0"
        },
        {
```

```
          "URI": "s3://DOC-EXAMPLE-BUCKET/folder/docker-compose.yaml"
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.MyDockerComposeComponent
ComponentVersion: '1.0.0'
ComponentDescription: 'A component that uses Docker Compose to run images from
 public Amazon ECR and Docker Hub.'
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.DockerApplicationManager:
    VersionRequirement: ~2.0.0
Manifests:
  - Platform:
      os: all
    Lifecycle:
        run: docker-compose -f {artifacts:path}/docker-compose.yaml up
    Artifacts:
      - URI: "docker:public.ecr.aws/cloudwatch-agent/cloudwatch-agent:latest"
      - URI: "docker:mysql:8.0"
      - URI: "s3://DOC-EXAMPLE-BUCKET/folder/docker-compose.yaml"
```

> **Note**
>
> Para usar comunicação entre processos (IPC), AWS credenciais ou gerenciador de fluxo em seu componente de contêiner do Docker, você deve especificar opções adicionais ao executar o contêiner do Docker. Para mais informações, consulte:
>
> - Use a comunicação entre processos nos componentes do contêiner Docker
> - Use AWS credenciais em componentes de contêiner do Docker (Linux)
> - Use o gerenciador de fluxo em componentes de contêiner Docker (Linux)

3. Teste o componente para verificar se ele funciona conforme o esperado.

> **Important**
>
> Você deve instalar e iniciar o daemon do Docker antes de implantar o componente.

Depois de implantar o componente localmente, você pode executar o comando docker container ls para verificar se o contêiner está sendo executado.

```
docker container ls
```

4. Quando o componente estiver pronto, faça o upload do componente AWS IoT Greengrass para implantá-lo em outros dispositivos principais. Para ter mais informações, consulte Publish components to deploy to your core devices.

Execute um contêiner Docker a partir de uma imagem privada no Amazon ECR

Esta seção descreve como você pode criar um componente personalizado que executa um contêiner do Docker a partir de uma imagem do Docker armazenada em um repositório privado no Amazon ECR.

Para executar um contêiner Docker

1. Crie um componente personalizado em seu dispositivo AWS IoT Greengrass principal. Use o exemplo de receita a seguir, que tem as seguintes propriedades:

   - O componente do gerenciador de aplicativos Docker como uma dependência. Esse componente permite AWS IoT Greengrass gerenciar credenciais para baixar imagens de repositórios privados.
   - O componente do serviço de troca de tokens como uma dependência. Esse componente permite recuperar AWS credenciais AWS IoT Greengrass para interagir com o Amazon ECR.
   - Um artefato de componente que especifica uma imagem do Docker em um repositório privado do Amazon ECR.
   - Um script de execução do ciclo de vida que usa docker run para criar e iniciar um contêiner a partir da imagem.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.MyPrivateDockerComponent",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that runs a Docker container from a
 private Amazon ECR image.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.DockerApplicationManager": {
      "VersionRequirement": "~2.0.0"
    },
    "aws.greengrass.TokenExchangeService": {
      "VersionRequirement": "~2.0.0"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "all"
      },
      "Lifecycle": {
        "run": "docker run account-
id.dkr.ecr.region.amazonaws.com/repository[:tag|@digest]"
      },
      "Artifacts": [
        {
          "URI": "docker:account-
id.dkr.ecr.region.amazonaws.com/repository[:tag|@digest]"
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.MyPrivateDockerComponent
```

```
ComponentVersion: '1.0.0'
ComponentDescription: 'A component that runs a Docker container from a private
 Amazon ECR image.'
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.DockerApplicationManager:
    VersionRequirement: ~2.0.0
  aws.greengrass.TokenExchangeService:
    VersionRequirement: ~2.0.0
Manifests:
  - Platform:
      os: all
    Lifecycle:
        run: docker run account-id.dkr.ecr.region.amazonaws.com/repository[:tag|
@digest]
    Artifacts:
      - URI: "docker:account-id.dkr.ecr.region.amazonaws.com/repository[:tag|
@digest]"
```

> **Note**
>
> Para usar comunicação entre processos (IPC), AWS credenciais ou gerenciador de fluxo em seu componente de contêiner do Docker, você deve especificar opções adicionais ao executar o contêiner do Docker. Para mais informações, consulte:
>
> - Use a comunicação entre processos nos componentes do contêiner Docker
> - Use AWS credenciais em componentes de contêiner do Docker (Linux)
> - Use o gerenciador de fluxo em componentes de contêiner Docker (Linux)

2. Teste o componente para verificar se ele funciona conforme o esperado.

> **Important**
>
> Você deve instalar e iniciar o daemon do Docker antes de implantar o componente.

Depois de implantar o componente localmente, você pode executar o comando docker container ls para verificar se o contêiner está sendo executado.

```
docker container ls
```

3. Faça o upload do componente AWS IoT Greengrass para implantá-lo em outros dispositivos principais. Para ter mais informações, consulte Publish components to deploy to your core devices.

Execute um contêiner Docker a partir de uma imagem no Amazon S3

Esta seção descreve como você pode executar um contêiner do Docker em um componente a partir de uma imagem do Docker armazenada no Amazon S3.

Para executar um contêiner Docker em um componente a partir de uma imagem no Amazon S3

1. Execute o comando docker save para criar um backup de um contêiner Docker. Você fornece esse backup como um artefato de componente para executar o contêiner. AWS IoT Greengrass Substitua *`hello-world`* pelo nome da imagem e substitua *`hello-world.tar`* pelo nome do arquivo a ser criado.

   ```
   docker save hello-world > artifacts/com.example.MyDockerComponent/1.0.0/hello-world.tar
   ```

2. Crie um componente personalizado em seu dispositivo AWS IoT Greengrass principal. Use o exemplo de receita a seguir, que tem as seguintes propriedades:

   - Um script de instalação do ciclo de vida que usa docker load para carregar uma imagem do Docker de um arquivo.
   - Um script de execução do ciclo de vida que usa docker run para criar e iniciar um contêiner a partir da imagem. A `--rm` opção limpa o contêiner quando ele sai.

   JSON

   ```
   {
     "RecipeFormatVersion": "2020-01-25",
     "ComponentName": "com.example.MyS3DockerComponent",
     "ComponentVersion": "1.0.0",
     "ComponentDescription": "A component that runs a Docker container from an
    image in an S3 bucket.",
     "ComponentPublisher": "Amazon",
   ```

```
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": {
          "Script": "docker load -i {artifacts:path}/hello-world.tar"
        },
        "run": {
          "Script": "docker run --rm hello-world"
        }
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.MyS3DockerComponent
ComponentVersion: '1.0.0'
ComponentDescription: 'A component that runs a Docker container from an image in
 an S3 bucket.'
ComponentPublisher: Amazon
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install:
        Script: docker load -i {artifacts:path}/hello-world.tar
      run:
        Script: docker run --rm hello-world
```

> **Note**
>
> Para usar comunicação entre processos (IPC), AWS credenciais ou gerenciador de fluxo em seu componente de contêiner do Docker, você deve especificar opções adicionais ao executar o contêiner do Docker. Para mais informações, consulte:

- Use a comunicação entre processos nos componentes do contêiner Docker
- Use AWS credenciais em componentes de contêiner do Docker (Linux)
- Use o gerenciador de fluxo em componentes de contêiner Docker (Linux)

3. Teste o componente para verificar se ele funciona conforme o esperado.

   Depois de implantar o componente localmente, você pode executar o comando docker container ls para verificar se o contêiner está sendo executado.

   ```
   docker container ls
   ```

4. Quando o componente estiver pronto, faça o upload do arquivo de imagens do Docker em um bucket do S3 e adicione seu URI à receita do componente. Em seguida, você pode carregar o componente AWS IoT Greengrass para implantá-lo em outros dispositivos principais. Para ter mais informações, consulte Publish components to deploy to your core devices.

   Quando terminar, a receita do componente deve ter a aparência do exemplo a seguir.

   JSON

   ```
   {
     "RecipeFormatVersion": "2020-01-25",
     "ComponentName": "com.example.MyS3DockerComponent",
     "ComponentVersion": "1.0.0",
     "ComponentDescription": "A component that runs a Docker container from an
    image in an S3 bucket.",
     "ComponentPublisher": "Amazon",
     "Manifests": [
       {
         "Platform": {
           "os": "linux"
         },
         "Lifecycle": {
           "install": {
             "Script": "docker load -i {artifacts:path}/hello-world.tar"
           },
           "run": {
             "Script": "docker run --rm hello-world"
           }
         },
   ```

```
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.MyDockerComponent/1.0.0/hello-world.tar"
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.MyS3DockerComponent
ComponentVersion: '1.0.0'
ComponentDescription: 'A component that runs a Docker container from an image in
 an S3 bucket.'
ComponentPublisher: Amazon
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install:
        Script: docker load -i {artifacts:path}/hello-world.tar
      run:
        Script: docker run --rm hello-world
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.MyDockerComponent/1.0.0/hello-world.tar
```

Use a comunicação entre processos nos componentes do contêiner Docker

Você pode usar a biblioteca de comunicação entre processos (IPC) do Greengrass no AWS IoT Device SDK para se comunicar com o núcleo do Greengrass, outros componentes do Greengrass e. AWS IoT Core Para ter mais informações, consulte Use o AWS IoT Device SDK para se comunicar com o núcleo do Greengrass, outros componentes e AWS IoT Core.

Para usar o IPC em um componente de contêiner do Docker, você deve executar o contêiner do Docker com os seguintes parâmetros:

- Monte o soquete IPC no contêiner. O núcleo do Greengrass fornece o caminho do arquivo do soquete IPC na variável de ambiente. `AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT`
- Defina as variáveis de `AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT` ambiente `SVCUID` e de acordo com os valores que o núcleo do Greengrass fornece aos componentes. Seu componente usa essas variáveis de ambiente para autenticar conexões com o núcleo do Greengrass.

Example Exemplo de receita: publicar uma mensagem MQTT em AWS IoT Core (Python)

A receita a seguir define um exemplo de componente de contêiner do Docker que publica uma mensagem MQTT no. AWS IoT Core Essa fórmula tem as seguintes propriedades:

- Uma política de autorização (`accessControl`) que permite que o componente publique mensagens MQTT AWS IoT Core em todos os tópicos. Para obter mais informações, consulte Autorize componentes a realizar operações de IPC e autorização do AWS IoT Core MQTT IPC.
- Um artefato de componente que especifica uma imagem do Docker como um arquivo TAR no Amazon S3.
- Um script de instalação do ciclo de vida que carrega a imagem do Docker do arquivo TAR.
- Um script de execução do ciclo de vida que executa um contêiner Docker a partir da imagem. O comando de execução do Docker tem os seguintes argumentos:
  - O `-v` argumento monta o soquete Greengrass IPC no contêiner.
  - Os dois primeiros `-e` argumentos definem as variáveis de ambiente necessárias no contêiner do Docker.
  - Os `-e` argumentos adicionais definem as variáveis de ambiente usadas neste exemplo.
  - O `--rm` argumento limpa o contêiner quando ele sai.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.python.docker.PublishToIoTCore",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Uses interprocess communication to publish an MQTT
 message to IoT Core.",
  "ComponentPublisher": "Amazon",
```

```
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "topic": "test/topic/java",
      "message": "Hello, World!",
      "qos": "1",
      "accessControl": {
        "aws.greengrass.ipc.mqttproxy": {
          "com.example.python.docker.PublishToIoTCore:pubsub:1": {
            "policyDescription": "Allows access to publish to IoT Core on all
 topics.",
            "operations": [
              "aws.greengrass#PublishToIoTCore"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "all"
      },
      "Lifecycle": {
        "install": "docker load -i {artifacts:path}/publish-to-iot-core.tar",
        "run": "docker run -v $AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT:
$AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT -e SVCUID -e
 AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT -e MQTT_TOPIC=
\"{configuration:/topic}\" -e MQTT_MESSAGE=\"{configuration:/message}\" -e MQTT_QOS=
\"{configuration:/qos}\" --rm publish-to-iot-core"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.python.docker.PublishToIoTCore/1.0.0/publish-to-iot-core.tar"
        }
      ]
    }
  ]
}
```

YAML

```
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.python.docker.PublishToIoTCore
ComponentVersion: 1.0.0
ComponentDescription: Uses interprocess communication to publish an MQTT message to
 IoT Core.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    topic: 'test/topic/java'
    message: 'Hello, World!'
    qos: '1'
    accessControl:
      aws.greengrass.ipc.mqttproxy:
        'com.example.python.docker.PublishToIoTCore:pubsub:1':
          policyDescription: Allows access to publish to IoT Core on all topics.
          operations:
            - 'aws.greengrass#PublishToIoTCore'
          resources:
            - '*'
Manifests:
  - Platform:
      os: all
    Lifecycle:
      install: 'docker load -i {artifacts:path}/publish-to-iot-core.tar'
      run: |
        docker run \
          -v $AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT:
$AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT \
          -e SVCUID \
          -e AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT \
          -e MQTT_TOPIC="{configuration:/topic}" \
          -e MQTT_MESSAGE="{configuration:/message}" \
          -e MQTT_QOS="{configuration:/qos}" \
          --rm publish-to-iot-core
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.python.docker.PublishToIoTCore/1.0.0/publish-to-iot-core.tar
```

Use AWS credenciais em componentes de contêiner do Docker (Linux)

Você pode usar o componente de serviço de troca de tokens para interagir com AWS os serviços nos componentes do Greengrass. Esse componente fornece AWS credenciais da função de troca de tokens do dispositivo principal usando um servidor de contêiner local. Para ter mais informações, consulte Interaja com AWS os serviços.

> **Note**
>
> O exemplo nesta seção funciona somente nos dispositivos principais do Linux.

Para usar AWS as credenciais do serviço de troca de tokens em um componente de contêiner do Docker, você deve executar o contêiner do Docker com os seguintes parâmetros:

- Forneça acesso à rede host usando o `--network=host` argumento. Essa opção permite que o contêiner Docker se conecte ao serviço local de troca de tokens para recuperar AWS as credenciais. Esse argumento funciona somente no Docker para Linux.

  > **Warning**
  >
  > Essa opção dá ao contêiner acesso a todas as interfaces de rede local no host, portanto, essa opção é menos segura do que se você executasse contêineres do Docker sem esse acesso à rede do host. Considere isso ao desenvolver e executar componentes de contêiner do Docker que usam essa opção. Para obter mais informações, consulte Rede: host na documentação do Docker.

- Defina as variáveis de `AWS_CONTAINER_AUTHORIZATION_TOKEN` ambiente `AWS_CONTAINER_CREDENTIALS_FULL_URI` e de acordo com os valores que o núcleo do Greengrass fornece aos componentes. AWS Os SDKs usam essas variáveis de ambiente para recuperar credenciais AWS .

Example Exemplo de receita: listar buckets do S3 em um componente de contêiner do Docker (Python)

A receita a seguir define um exemplo de componente de contêiner Docker que lista os buckets S3 em seu. Conta da AWS Essa fórmula tem as seguintes propriedades:

- O componente do serviço de troca de tokens como uma dependência. Essa dependência permite que o componente recupere AWS credenciais para interagir com outros serviços. AWS
- Um artefato de componente que especifica uma imagem do Docker como um arquivo tar no Amazon S3.
- Um script de instalação do ciclo de vida que carrega a imagem do Docker do arquivo TAR.
- Um script de execução do ciclo de vida que executa um contêiner Docker a partir da imagem. O comando de execução do Docker tem os seguintes argumentos:
  - O `--network=host` argumento fornece ao contêiner acesso à rede host, para que o contêiner possa se conectar ao serviço de troca de tokens.
  - O `-e` argumento define as variáveis de ambiente necessárias no contêiner do Docker.
  - O `--rm` argumento limpa o contêiner quando ele sai.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.python.docker.ListS3Buckets",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Uses the token exchange service to lists your S3
 buckets.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.TokenExchangeService": {
      "VersionRequirement": "^2.0.0",
      "DependencyType": "HARD"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "docker load -i {artifacts:path}/list-s3-buckets.tar",
        "run": "docker run --network=host -e AWS_CONTAINER_AUTHORIZATION_TOKEN -e
 AWS_CONTAINER_CREDENTIALS_FULL_URI --rm list-s3-buckets"
      },
      "Artifacts": [
        {
```

```
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.python.docker.ListS3Buckets/1.0.0/list-s3-buckets.tar"
        }
      ]
    }
  ]
}
```

YAML

```
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.python.docker.ListS3Buckets
ComponentVersion: 1.0.0
ComponentDescription: Uses the token exchange service to lists your S3 buckets.
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.TokenExchangeService:
    VersionRequirement: ^2.0.0
    DependencyType: HARD
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: 'docker load -i {artifacts:path}/list-s3-buckets.tar'
      run: |
        docker run \
          --network=host \
          -e AWS_CONTAINER_AUTHORIZATION_TOKEN \
          -e AWS_CONTAINER_CREDENTIALS_FULL_URI \
          --rm list-s3-buckets
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.python.docker.ListS3Buckets/1.0.0/list-s3-buckets.tar
```

Use o gerenciador de fluxo em componentes de contêiner Docker (Linux)

Você pode usar o componente gerenciador de fluxo para gerenciar fluxos de dados nos componentes do Greengrass. Esse componente permite processar fluxos de dados e transferir dados de IoT de alto volume para o. Nuvem AWS AWS IoT Greengrass fornece um SDK do gerenciador de stream que você usa para interagir com o componente do stream manager. Para ter mais informações, consulte Gerencie fluxos de dados nos dispositivos principais do Greengrass.

> **Note**
>
> O exemplo nesta seção funciona somente nos dispositivos principais do Linux.

Para usar o SDK do gerenciador de stream em um componente de contêiner do Docker, você deve executar o contêiner do Docker com os seguintes parâmetros:

- Forneça acesso à rede host usando o `--network=host` argumento. Essa opção permite que o contêiner Docker interaja com o componente gerenciador de fluxo por meio de uma conexão TLS local. Esse argumento funciona apenas no Docker para Linux

  > **Warning**
  >
  > Essa opção dá ao contêiner acesso a todas as interfaces de rede local no host, portanto, essa opção é menos segura do que se você executasse contêineres do Docker sem esse acesso à rede do host. Considere isso ao desenvolver e executar componentes de contêiner do Docker que usam essa opção. Para obter mais informações, consulte Rede: host na documentação do Docker.

- Se você configurar o componente do gerenciador de fluxo para exigir autenticação, que é o comportamento padrão, defina a variável de `AWS_CONTAINER_CREDENTIALS_FULL_URI` ambiente com o valor que o núcleo do Greengrass fornece aos componentes. Para obter mais informações, consulte Configuração do gerenciador de stream.
- Se você configurar o componente do gerenciador de fluxo para usar uma porta não padrão, use a comunicação entre processos (IPC) para obter a porta da configuração do componente do gerenciador de fluxo. Você deve executar o contêiner Docker com opções adicionais para usar o IPC. Para mais informações, consulte:
  - Conecte-se ao gerenciador de streaming no código do aplicativo
  - Use a comunicação entre processos nos componentes do contêiner Docker

Example Exemplo de receita: transmitir um arquivo para um bucket do S3 em um componente de contêiner do Docker (Python)

A receita a seguir define um exemplo de componente de contêiner do Docker que cria um arquivo e o transmite para um bucket do S3. Essa fórmula tem as seguintes propriedades:

- O componente do gerenciador de fluxo como uma dependência. Essa dependência permite que o componente use o SDK do gerenciador de fluxo para interagir com o componente do gerenciador de fluxo.
- Um artefato de componente que especifica uma imagem do Docker como um arquivo TAR no Amazon S3.
- Um script de instalação do ciclo de vida que carrega a imagem do Docker do arquivo TAR.
- Um script de execução do ciclo de vida que executa um contêiner Docker a partir da imagem. O comando de execução do Docker tem os seguintes argumentos:
  - O `--network=host` argumento fornece ao contêiner acesso à rede host, para que o contêiner possa se conectar ao componente do gerenciador de fluxo.
  - O primeiro `-e` argumento define a variável de `AWS_CONTAINER_AUTHORIZATION_TOKEN` ambiente necessária no contêiner do Docker.
  - Os `-e` argumentos adicionais definem as variáveis de ambiente usadas neste exemplo.
  - O `-v` argumento monta a pasta de trabalho do componente no contêiner. Este exemplo cria um arquivo na pasta de trabalho para fazer o upload desse arquivo para o Amazon S3 usando o gerenciador de stream.
  - O `--rm` argumento limpa o contêiner quando ele sai.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.python.docker.StreamFileToS3",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Creates a text file and uses stream manager to stream the
 file to S3.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.StreamManager": {
      "VersionRequirement": "^2.0.0",
      "DependencyType": "HARD"
    }
  },
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "bucketName": ""
    }
  },
```

```
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "docker load -i {artifacts:path}/stream-file-to-s3.tar",
        "run": "docker run --network=host -e AWS_CONTAINER_AUTHORIZATION_TOKEN
 -e BUCKET_NAME=\"{configuration:/bucketName}\" -e WORK_PATH=\"{work:path}\" -v
 {work:path}:{work:path} --rm stream-file-to-s3"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.python.docker.StreamFileToS3/1.0.0/stream-file-to-s3.tar"
        }
      ]
    }
  ]
}
```

YAML

```
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.python.docker.StreamFileToS3
ComponentVersion: 1.0.0
ComponentDescription: Creates a text file and uses stream manager to stream the file
 to S3.
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.StreamManager:
    VersionRequirement: ^2.0.0
    DependencyType: HARD
ComponentConfiguration:
  DefaultConfiguration:
    bucketName: ''
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: 'docker load -i {artifacts:path}/stream-file-to-s3.tar'
      run: |
        docker run \
```

```
            --network=host \
            -e AWS_CONTAINER_AUTHORIZATION_TOKEN \
            -e BUCKET_NAME="{configuration:/bucketName}" \
            -e WORK_PATH="{work:path}" \
            -v {work:path}:{work:path} \
            --rm stream-file-to-s3
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.python.docker.StreamFileToS3/1.0.0/stream-file-to-s3.tar
```

AWS IoT Greengrass referência da receita do componente

A receita do componente é um arquivo que define os detalhes, dependências, artefatos e ciclos de vida de um componente. O ciclo de vida do componente especifica os comandos a serem executados para instalar, executar e desligar o componente, por exemplo. O AWS IoT Greengrass núcleo usa os ciclos de vida que você define na receita para instalar e executar componentes. O AWS IoT Greengrass serviço usa a receita para identificar as dependências e os artefatos a serem implantados em seus dispositivos principais quando você implanta o componente.

Na receita, você pode definir dependências e ciclos de vida exclusivos para cada plataforma que um componente suporta. Você pode usar esse recurso para implantar um componente em dispositivos com várias plataformas que tenham requisitos diferentes. Você também pode usar isso para evitar a instalação AWS IoT Greengrass de um componente em dispositivos que não o suportam.

Cada receita contém uma lista de manifestos. Cada manifesto especifica um conjunto de requisitos da plataforma, o ciclo de vida e os artefatos a serem usados nos dispositivos principais cuja plataforma atende a esses requisitos. O dispositivo principal usa o primeiro manifesto com os requisitos de plataforma que o dispositivo atende. Especifique um manifesto sem nenhum requisito de plataforma que corresponda a qualquer dispositivo principal.

Você também pode especificar um ciclo de vida global que não esteja em um manifesto. No ciclo de vida global, você pode usar chaves de seleção que identificam subseções do ciclo de vida. Em seguida, você pode especificar essas chaves de seleção em um manifesto para usar essas seções do ciclo de vida global, além do ciclo de vida do manifesto. O dispositivo principal usa as chaves de seleção do manifesto somente se o manifesto não definir um ciclo de vida. Você pode usar a `all` seleção em um manifesto para combinar seções do ciclo de vida global sem chaves de seleção.

Depois que o software AWS IoT Greengrass Core seleciona um manifesto que corresponda ao dispositivo principal, ele faz o seguinte para identificar as etapas do ciclo de vida a serem usadas:

- Se o manifesto selecionado definir um ciclo de vida, o dispositivo principal usará esse ciclo de vida.
- Se o manifesto selecionado não definir um ciclo de vida, o dispositivo principal usa o ciclo de vida global. O dispositivo principal faz o seguinte para identificar quais seções do ciclo de vida global usar:
  - Se o manifesto definir chaves de seleção, o dispositivo principal usará as seções do ciclo de vida global que contêm as chaves de seleção do manifesto.
  - Se o manifesto não definir chaves de seleção, o dispositivo principal usa as seções do ciclo de vida global que não têm chaves de seleção. Esse comportamento é equivalente a um manifesto que define a `all` seleção.

> **Important**
>
> Um dispositivo principal deve atender aos requisitos de plataforma de pelo menos um manifesto para instalar o componente. Se nenhum manifesto corresponder ao dispositivo principal, o software AWS IoT Greengrass Core não instalará o componente e a implantação falhará.

Você pode definir receitas no formato JSON ou YAML. A seção de exemplos de receitas inclui receitas em cada formato.

Tópicos

Validação da receita

O Greengrass valida uma receita de componente JSON ou YAML ao criar uma versão do componente. Essa validação de receita verifica se há erros comuns na receita do componente JSON ou YAML para evitar possíveis problemas de implantação. A validação verifica a receita em busca de erros comuns (por exemplo, vírgulas, chaves e campos ausentes) e para garantir que a receita esteja bem formada.

Se você receber uma mensagem de erro de validação da receita, verifique se há vírgulas, chaves ou campos ausentes em sua receita. Verifique se não está faltando nenhum campo examinando o formato da receita.

Formato da receita

Ao definir uma receita para um componente, você especifica as seguintes informações no documento da receita. A mesma estrutura se aplica às receitas nos formatos YAML e JSON.

`RecipeFormatVersion`

A versão modelo da receita. Escolha a seguinte opção:

- `2020-01-25`

`ComponentName`

O nome do componente que essa receita define. O nome do componente deve ser exclusivo Conta da AWS em cada região.

> **Dicas**
>
> - Use o formato de nome de domínio inverso para evitar colisões de nomes dentro da sua empresa. Por exemplo, se sua empresa possui `example.com` e você trabalha em um projeto de energia solar, você pode nomear seu componente Hello World`com.example.solar.HelloWorld`. Isso ajuda a evitar colisões de nomes de componentes em sua empresa.
> - Evite o `aws.greengrass` prefixo nos nomes dos componentes. AWS IoT Greengrass usa esse prefixo para os componentes públicos que ele fornece. Se você escolher o mesmo nome de um componente público, seu componente substituirá esse componente. Em seguida, AWS IoT Greengrass fornece seu componente em vez do componente público ao implantar componentes com dependência desse componente público. Esse recurso permite que você substitua o comportamento de componentes públicos, mas também pode interromper outros componentes se você não pretende substituir um componente público.

`ComponentVersion`

A versão do componente. O valor máximo para os valores principais, secundários e de patch é 999999.

> **Note**
>
> AWS IoT Greengrass usa versões semânticas para componentes. As versões semânticas seguem um sistema de numeração principal.secundária.patch. Por exemplo, a versão `1.0.0` representa a primeira versão principal de um componente. Para obter mais informações, consulte a especificação da versão semântica.

`ComponentDescription`

(Opcional) A descrição do componente.

`ComponentPublisher`

O editor ou autor do componente.

`ComponentConfiguration`

(Opcional) Um objeto que define a configuração ou os parâmetros do componente. Você define a configuração padrão e, ao implantar o componente, pode especificar o objeto de configuração a ser fornecido ao componente. A configuração do componente oferece suporte a parâmetros e estruturas aninhados. Esse objeto contém as seguintes informações:

`DefaultConfiguration`

Um objeto que define a configuração padrão do componente. Você define a estrutura desse objeto.

> **Note**
>
> AWS IoT Greengrass usa JSON para valores de configuração. O JSON especifica um tipo de número, mas não diferencia entre números inteiros e flutuantes. Como resultado, os valores de configuração podem ser convertidos em floats in AWS IoT Greengrass. Para garantir que seu componente use o tipo de dados correto, recomendamos que você defina valores de configuração numérica como cadeias de caracteres. Em seguida, faça com que seu componente os analise como números inteiros ou flutuantes. Isso garante que seus valores de configuração tenham o mesmo tipo na configuração e no seu dispositivo principal.

`ComponentDependencies`

(Opcional) Um dicionário de objetos em que cada um define uma dependência de componente para o componente. A chave para cada objeto identifica o nome da dependência do componente. AWS IoT Greengrass instala dependências do componente quando o componente é instalado. AWS IoT Greengrass espera que as dependências comecem antes de iniciar o componente. Cada objeto contém as seguintes informações:

`VersionRequirement`

A restrição de versão semântica no estilo npm que define as versões de componentes compatíveis para essa dependência. Você pode especificar uma versão ou um intervalo de versões. Para obter mais informações, consulte a calculadora da versão semântica npm.

`DependencyType`

(Opcional) O tipo dessa dependência. Escolha entre as opções a seguir.

- `SOFT` – O componente não é reiniciado se a dependência muda de estado.
- `HARD` – O componente é reiniciado se a dependência muda de estado.

Padronizado como `HARD`.

`ComponentType`

(Opcional) O tipo de componente.

> **Note**
>
> Não recomendamos que você especifique o tipo de componente em uma receita. AWS IoT Greengrass define o tipo para você ao criar um componente.

O tipo pode ser um dos seguintes tipos:

- `aws.greengrass.generic`— O componente executa comandos ou fornece artefatos.
- `aws.greengrass.lambda`— O componente executa uma função Lambda usando o componente Lambda launcher. O `ComponentSource` parâmetro especifica o ARN da função Lambda que esse componente executa.

  Não recomendamos que você use essa opção, pois ela é definida AWS IoT Greengrass quando você cria um componente a partir de uma função Lambda. Para ter mais informações, consulte Executar AWS Lambda funções.

- `aws.greengrass.plugin`— O componente é executado na mesma Java Virtual Machine (JVM) do núcleo Greengrass. Se você implantar ou reiniciar um componente de plug-in, o núcleo do Greengrass será reiniciado.

  Os componentes do plug-in usam o mesmo arquivo de log do Greengrass nucleus. Para ter mais informações, consulte Monitore AWS IoT Greengrass os registros.

  Não recomendamos que você use essa opção em receitas de componentes, porque ela se destina a componentes AWS fornecidos escritos em Java que interagem diretamente com o núcleo do Greengrass. Para obter mais informações sobre quais componentes públicos são plug-ins, consulteAWS-componentes fornecidos.
- `aws.greengrass.nucleus`— O componente do núcleo. Para ter mais informações, consulte Núcleo Greengrass.

  Não recomendamos que você use essa opção em receitas de componentes. Ele é destinado ao componente do núcleo Greengrass, que fornece a funcionalidade mínima do software Core. AWS IoT Greengrass

O padrão é `aws.greengrass.generic` quando você cria um componente a partir de uma receita ou `aws.greengrass.lambda` quando você cria um componente a partir de uma função Lambda.

Para ter mais informações, consulte Tipos de componentes.

`ComponentSource`

(Opcional) O ARN da função Lambda que um componente executa.

Não recomendamos que você especifique a origem do componente em uma receita. AWS IoT Greengrass define esse parâmetro para você quando você cria um componente a partir de uma função Lambda. Para ter mais informações, consulte Executar AWS Lambda funções.

`Manifests`

Uma lista de objetos em que cada um define o ciclo de vida, os parâmetros e os requisitos do componente para uma plataforma. Se um dispositivo principal atender aos requisitos de plataforma de vários manifestos, AWS IoT Greengrass use o primeiro manifesto correspondente ao dispositivo principal. Para garantir que os dispositivos principais usem o manifesto correto, defina primeiro os manifestos com requisitos de plataforma mais rígidos. Um manifesto que se aplica a todas as plataformas deve ser o último manifesto na lista.

> **Important**
>
> Um dispositivo principal deve atender aos requisitos de plataforma de pelo menos um manifesto para instalar o componente. Se nenhum manifesto corresponder ao dispositivo principal, o software AWS IoT Greengrass Core não instalará o componente e a implantação falhará.

Cada objeto contém as seguintes informações:

`Name`

(Opcional) Um nome amigável para a plataforma que esse manifesto define.

Se você omitir esse parâmetro, AWS IoT Greengrass cria um nome da plataforma `os` e. `architecture`

`Platform`

(Opcional) Um objeto que define a plataforma à qual esse manifesto se aplica. Omita esse parâmetro para definir um manifesto que se aplica a todas as plataformas.

Esse objeto especifica pares de valores-chave sobre a plataforma na qual um dispositivo principal é executado. Quando você implanta esse componente, o software AWS IoT Greengrass Core compara esses pares de valores-chave com os atributos da plataforma no dispositivo principal. O software AWS IoT Greengrass principal sempre define `os` `earchitecture`, e pode definir atributos adicionais. Você pode especificar atributos de plataforma personalizados para um dispositivo principal ao implantar o componente nucleus do Greengrass. Para obter mais informações, consulte o parâmetro de substituição da plataforma do componente do núcleo do Greengrass.

Para cada par de valores-chave, você pode especificar um dos seguintes valores:

- Um valor exato, como `linux` ou`windows`. Os valores exatos devem começar com uma letra ou um número.
- `*`, que corresponde a qualquer valor. Isso também corresponde quando um valor não está presente.
- Uma expressão regular no estilo Java, como. `/windows|linux/` A expressão regular deve começar e terminar com um caractere de barra (`/`). Por exemplo, a expressão regular `/.+/` corresponde a qualquer valor que não esteja em branco.

Esse objeto contém as seguintes informações:

`os`

(Opcional) O nome do sistema operacional da plataforma compatível com esse manifesto. As plataformas comuns incluem os seguintes valores:

- `linux`
- `windows`
- `darwin` (macOS)

`architecture`

(Opcional) A arquitetura do processador para a plataforma que esse manifesto suporta. As arquiteturas comuns incluem os seguintes valores:

- `amd64`
- `arm`
- `aarch64`
- `x86`

`architecture.detail`

(Opcional) Os detalhes da arquitetura do processador para a plataforma que esse manifesto suporta. Os detalhes comuns da arquitetura incluem os seguintes valores:

- `arm61`
- `arm71`
- `arm81`

`key`

(Opcional) Um atributo de plataforma que você define para esse manifesto. Substitua a *`chave`* pelo nome do atributo da plataforma. O software AWS IoT Greengrass Core combina esse atributo da plataforma com os pares de valores-chave que você especifica na configuração do componente nuclear do Greengrass. Para obter mais informações, consulte o parâmetro de substituição da plataforma do componente do núcleo do Greengrass.

> **Tip**
>
> Use o formato de nome de domínio inverso para evitar colisões de nomes dentro da sua empresa. Por exemplo, se sua empresa possui `example.com` e você

trabalha em um projeto de rádio, você pode nomear um atributo de plataforma personalizado`com.example.radio.RadioModule`. Isso ajuda a evitar colisões de nomes de atributos de plataforma em sua empresa.

Por exemplo, você pode definir um atributo de plataforma,`com.example.radio.RadioModule`, para especificar um manifesto diferente com base em qual módulo de rádio está disponível em um dispositivo principal. Cada manifesto pode incluir artefatos diferentes que se aplicam a diferentes configurações de hardware, para que você implante o conjunto mínimo de software no dispositivo principal.

`Lifecycle`

Um objeto ou string que define como instalar e executar o componente na plataforma definida por esse manifesto. Você também pode definir um ciclo de vida global que se aplique a todas as plataformas. O dispositivo principal usa o ciclo de vida global somente se o manifesto a ser usado não especificar um ciclo de vida.

> **Note**
>
> Você define esse ciclo de vida em um manifesto. As etapas do ciclo de vida que você especifica aqui se aplicam somente à plataforma definida por esse manifesto. Você também pode definir um ciclo de vida global que se aplique a todas as plataformas.

Esse objeto ou string contém as seguintes informações:

`Setenv`

(Opcional) Um dicionário de variáveis de ambiente para fornecer a todos os scripts de ciclo de vida. Você pode substituir essas variáveis de ambiente `Setenv` em cada script de ciclo de vida.

`install`

(Opcional) Um objeto ou string que define o script a ser executado quando o componente é instalado. O software AWS IoT Greengrass principal também executa essa etapa do ciclo de vida toda vez que o software é lançado.

Se o `install` script sair com um código de sucesso, o componente entrará no `INSTALLED` estado.

Esse objeto ou string contém as seguintes informações:

`Script`

O script a ser executado.

`RequiresPrivilege`

(Opcional) Você pode executar o script com privilégios de root. Se você definir essa opção como`true`, o software AWS IoT Greengrass Core executará esse script de ciclo de vida como root em vez de como usuário do sistema que você configura para executar esse componente. Padronizado como `false`.

`Skipif`

(Opcional) A verificação para determinar se o script deve ou não ser executado. Você pode definir para verificar se um executável está no caminho ou se existe um arquivo. Se a saída for verdadeira, o software AWS IoT Greengrass Core pulará a etapa. Escolha uma das seguintes verificações:

- `onpath` *`runnable`*— Verifique se um executável está no caminho do sistema. Por exemplo, use **`onpath python3`** para pular essa etapa do ciclo de vida se o Python 3 estiver disponível.
- `exists` *`file`*— Verifique se existe um arquivo. Por exemplo, use **`exists /tmp/my-configuration.db`** para pular essa etapa do ciclo de vida, se estiver `/tmp/my-configuration.db` presente.

`Timeout`

(Opcional) O tempo máximo em segundos que o script pode ser executado antes que o software AWS IoT Greengrass principal encerre o processo.

Padrão: 120 segundos

`Setenv`

(Opcional) O dicionário de variáveis de ambiente a serem fornecidas ao script. Essas variáveis de ambiente substituem as variáveis que você fornece. `Lifecycle.Setenv`

`run`

(Opcional) Um objeto ou string que define o script a ser executado quando o componente é iniciado.

O componente entra no `RUNNING` estado em que essa etapa do ciclo de vida é executada. Se o `run` script sair com um código de sucesso, o componente entrará no `STOPPING` estado. Se um `shutdown` script for especificado, ele será executado; caso contrário, o componente entrará no `FINISHED` estado.

Os componentes que dependem desse componente são iniciados quando essa etapa do ciclo de vida é executada. Para executar um processo em segundo plano, como um serviço usado por componentes dependentes, use a etapa do `startup` ciclo de vida.

Quando você implanta componentes com um `run` ciclo de vida, o dispositivo principal pode relatar a implantação como concluída assim que esse script de ciclo de vida é executado. Como resultado, a implantação pode ser concluída e bem-sucedida mesmo se o script do `run` ciclo de vida falhar logo após a execução. Se você quiser que o status de implantação dependa do resultado do script de inicialização do componente, use a etapa do `startup` ciclo de vida em vez disso.

> **Note**
>
> Você pode definir somente um `run` ciclo `startup` de vida.

Esse objeto ou string contém as seguintes informações:

`Script`

O script a ser executado.

`RequiresPrivilege`

(Opcional) Você pode executar o script com privilégios de root. Se você definir essa opção como`true`, o software AWS IoT Greengrass Core executará esse script de ciclo de vida como root em vez de como usuário do sistema que você configura para executar esse componente. Padronizado como `false`.

`Skipif`

(Opcional) A verificação para determinar se o script deve ou não ser executado. Você pode definir para verificar se um executável está no caminho ou se existe um arquivo. Se a saída for verdadeira, o software AWS IoT Greengrass Core pulará a etapa. Escolha uma das seguintes verificações:

- `onpath` *`runnable`*— Verifique se um executável está no caminho do sistema. Por exemplo, use **`onpath python3`** para pular essa etapa do ciclo de vida se o Python 3 estiver disponível.
- `exists` *`file`*— Verifique se existe um arquivo. Por exemplo, use **`exists /tmp/my-configuration.db`** para pular essa etapa do ciclo de vida, se estiver `/tmp/my-configuration.db` presente.

`Timeout`

(Opcional) O tempo máximo em segundos que o script pode ser executado antes que o software AWS IoT Greengrass principal encerre o processo.

Essa etapa do ciclo de vida não atinge o tempo limite por padrão. Se você omitir esse tempo limite, o `run` script será executado até ser encerrado.

`Setenv`

(Opcional) O dicionário de variáveis de ambiente a serem fornecidas ao script. Essas variáveis de ambiente substituem as variáveis que você fornece. `Lifecycle.Setenv`

`startup`

(Opcional) Um objeto ou string que define o processo em segundo plano a ser executado quando o componente é iniciado.

Use `startup` para executar um comando que deve sair com êxito ou atualizar o status do componente para `RUNNING` antes que os componentes dependentes possam ser iniciados. Use a operação UpdateStateIPC para definir o status do componente como `RUNNING` ou `ERRORED` quando o componente inicia um script que não sai. Por exemplo, você pode definir uma `startup` etapa que inicie o processo do MySQL com. `/etc/init.d/mysqld start`

O componente entra no `STARTING` estado em que essa etapa do ciclo de vida é executada. Se o `startup` script sair com um código de sucesso, o componente entrará no `RUNNING` estado. Em seguida, os componentes dependentes podem ser iniciados.

Quando você implanta componentes com um `startup` ciclo de vida, o dispositivo principal pode relatar a implantação como concluída após a saída desse script de ciclo de vida ou relatar seu estado. Em outras palavras, o status da implantação é `IN_PROGRESS` até que os scripts de inicialização de todos os componentes saiam ou relatem um estado.

> **Note**
>
> Você pode definir somente um `run` ciclo `startup` de vida.

Esse objeto ou string contém as seguintes informações:

`Script`

O script a ser executado.

`RequiresPrivilege`

(Opcional) Você pode executar o script com privilégios de root. Se você definir essa opção como`true`, o software AWS IoT Greengrass Core executará esse script de ciclo de vida como root em vez de como usuário do sistema que você configura para executar esse componente. Padronizado como `false`.

`Skipif`

(Opcional) A verificação para determinar se o script deve ou não ser executado. Você pode definir para verificar se um executável está no caminho ou se existe um arquivo. Se a saída for verdadeira, o software AWS IoT Greengrass Core pulará a etapa. Escolha uma das seguintes verificações:

- `onpath` *`runnable`*— Verifique se um executável está no caminho do sistema. Por exemplo, use **`onpath python3`** para pular essa etapa do ciclo de vida se o Python 3 estiver disponível.
- `exists` *`file`*— Verifique se existe um arquivo. Por exemplo, use **`exists /tmp/my-configuration.db`** para pular essa etapa do ciclo de vida, se estiver `/tmp/my-configuration.db` presente.

`Timeout`

(Opcional) O tempo máximo em segundos que o script pode ser executado antes que o software AWS IoT Greengrass principal encerre o processo.

Padrão: 120 segundos

`Setenv`

(Opcional) O dicionário de variáveis de ambiente a serem fornecidas ao script. Essas variáveis de ambiente substituem as variáveis que você fornece. `Lifecycle.Setenv`

`shutdown`

(Opcional) Um objeto ou string que define o script a ser executado quando o componente é encerrado. Use o ciclo de vida de desligamento para executar o código que você deseja executar quando o componente estiver no estado. `STOPPING` O ciclo de vida de desligamento pode ser usado para interromper um processo iniciado pelos `startup` scripts ou. `run`

Se você iniciar um processo em segundo plano no`startup`, use a `shutdown` etapa para interromper esse processo quando o componente for encerrado. Por exemplo, você pode definir uma `shutdown` etapa que interrompa o processo do MySQL com. `/etc/init.d/mysqld stop`

O `shutdown` script é executado depois que o componente entra no `STOPPING` estado. Se o script for concluído com êxito, o componente entrará no `FINISHED` estado.

Esse objeto ou string contém as seguintes informações:

`Script`

O script a ser executado.

`RequiresPrivilege`

(Opcional) Você pode executar o script com privilégios de root. Se você definir essa opção como`true`, o software AWS IoT Greengrass Core executará esse script de ciclo de vida como root em vez de como usuário do sistema que você configura para executar esse componente. Padronizado como `false`.

`Skipif`

(Opcional) A verificação para determinar se o script deve ou não ser executado. Você pode definir para verificar se um executável está no caminho ou se existe um arquivo. Se a saída for verdadeira, o software AWS IoT Greengrass Core pulará a etapa. Escolha uma das seguintes verificações:

- `onpath` *`runnable`*— Verifique se um executável está no caminho do sistema. Por exemplo, use **`onpath python3`** para pular essa etapa do ciclo de vida se o Python 3 estiver disponível.
- `exists` *`file`*— Verifique se existe um arquivo. Por exemplo, use **`exists /tmp/my-configuration.db`** para pular essa etapa do ciclo de vida, se estiver `/tmp/my-configuration.db` presente.

`Timeout`

(Opcional) O tempo máximo em segundos que o script pode ser executado antes que o software AWS IoT Greengrass principal encerre o processo.

Padrão: 15 segundos.

`Setenv`

(Opcional) O dicionário de variáveis de ambiente a serem fornecidas ao script. Essas variáveis de ambiente substituem as variáveis que você fornece. `Lifecycle.Setenv`

`recover`

(Opcional) Um objeto ou string que define o script a ser executado quando o componente encontra um erro.

Essa etapa é executada quando um componente entra no `ERRORED` estado. Se o componente se tornar `ERRORED` três vezes sem se recuperar com sucesso, o componente mudará para o `BROKEN` estado. Para corrigir um `BROKEN` componente, você deve implantá-lo novamente.

Esse objeto ou string contém as seguintes informações:

`Script`

O script a ser executado.

`RequiresPrivilege`

(Opcional) Você pode executar o script com privilégios de root. Se você definir essa opção como`true`, o software AWS IoT Greengrass Core executará esse script de ciclo de vida como root em vez de como usuário do sistema que você configura para executar esse componente. Padronizado como `false`.

`Skipif`

(Opcional) A verificação para determinar se o script deve ou não ser executado. Você pode definir para verificar se um executável está no caminho ou se existe um arquivo. Se a saída for verdadeira, o software AWS IoT Greengrass Core pulará a etapa. Escolha uma das seguintes verificações:

- `onpath` *`runnable`*— Verifique se um executável está no caminho do sistema. Por exemplo, use **`onpath python3`** para pular essa etapa do ciclo de vida se o Python 3 estiver disponível.

- `exists` *`file`*— Verifique se existe um arquivo. Por exemplo, use **`exists /tmp/my-configuration.db`** para pular essa etapa do ciclo de vida, se estiver `/tmp/my-configuration.db` presente.

`Timeout`

(Opcional) O tempo máximo em segundos que o script pode ser executado antes que o software AWS IoT Greengrass principal encerre o processo.

Padrão: 60 segundos.

`Setenv`

(Opcional) O dicionário de variáveis de ambiente a serem fornecidas ao script. Essas variáveis de ambiente substituem as variáveis que você fornece. `Lifecycle.Setenv`

`bootstrap`

(Opcional) Um objeto ou string que define um script que exige que o software AWS IoT Greengrass principal ou o dispositivo principal seja reiniciado. Isso permite desenvolver um componente que executa uma reinicialização após instalar atualizações do sistema operacional ou atualizações de tempo de execução, por exemplo.

> **Note**
>
> Para instalar atualizações ou dependências que não exijam a reinicialização do software ou dispositivo AWS IoT Greengrass Core, use o ciclo de vida da instalação.

Essa etapa do ciclo de vida é executada antes da etapa do ciclo de vida da instalação nos seguintes casos, quando o software AWS IoT Greengrass principal implanta o componente:

- O componente é implantado no dispositivo principal pela primeira vez.
- A versão do componente muda.
- O script de bootstrap muda como resultado de uma atualização da configuração do componente.

Depois que o software AWS IoT Greengrass principal concluir a etapa de inicialização de todos os componentes que têm uma etapa de inicialização em uma implantação, o software é reiniciado.

> **Important**
>
> Você deve configurar o software AWS IoT Greengrass Core como um serviço do sistema para reiniciar o software AWS IoT Greengrass Core ou o dispositivo principal. Se você não configurar o software AWS IoT Greengrass Core como um serviço do sistema, o software não será reiniciado. Para ter mais informações, consulte Configurar o núcleo do Greengrass como um serviço do sistema.

Esse objeto ou string contém as seguintes informações:

`BootstrapOnRollback`

> **Note**
>
> Quando esse recurso estiver ativado, `BootstrapOnRollback` será executado somente para componentes que concluíram ou tentaram executar as etapas do ciclo de vida do bootstrap como parte de uma implantação de destino com falha. Esse recurso está disponível para as versões 2.12.0 e posteriores do Greengrass nucleus.

(Opcional) Você pode executar as etapas do ciclo de vida do bootstrap como parte de uma implantação de reversão. Se você definir essa opção como`true`, as etapas do ciclo de vida do bootstrap definidas em uma implantação de reversão serão executadas. Quando uma implantação falha, a versão anterior do ciclo de vida de bootstrap do componente será executada novamente durante uma implantação de reversão.

Padronizado como `false`.

`Script`

O script a ser executado. O código de saída desse script define a instrução de reinicialização. Use os seguintes códigos de saída:

- `0`— Não reinicie o software AWS IoT Greengrass principal nem o dispositivo principal. O software AWS IoT Greengrass Core ainda reinicia após a inicialização de todos os componentes.
- `100`— Solicitação para reiniciar o software AWS IoT Greengrass principal.

- `101`— Solicitação para reiniciar o dispositivo principal.

Os códigos de saída 100 a 199 são reservados para comportamentos especiais. Outros códigos de saída representam erros de script.

`RequiresPrivilege`

(Opcional) Você pode executar o script com privilégios de root. Se você definir essa opção como`true`, o software AWS IoT Greengrass Core executará esse script de ciclo de vida como root em vez de como usuário do sistema que você configura para executar esse componente. Padronizado como `false`.

`Timeout`

(Opcional) O tempo máximo em segundos que o script pode ser executado antes que o software AWS IoT Greengrass principal encerre o processo.

Padrão: 120 segundos

`Setenv`

(Opcional) O dicionário de variáveis de ambiente a serem fornecidas ao script. Essas variáveis de ambiente substituem as variáveis que você fornece. `Lifecycle.Setenv`

`Selections`

(Opcional) Uma lista de chaves de seleção que especificam seções do ciclo de vida global a serem executadas para esse manifesto. No ciclo de vida global, você pode definir etapas do ciclo de vida com chaves de seleção em qualquer nível para selecionar subseções do ciclo de vida. Em seguida, o dispositivo principal usa as seções que correspondem às teclas de seleção nesse manifesto. Para obter mais informações, consulte os exemplos de ciclo de vida global.

> **Important**
>
> O dispositivo principal usa as seleções do ciclo de vida global somente se esse manifesto não definir um ciclo de vida.

Você pode especificar a chave `all` de seleção para executar seções do ciclo de vida global que não têm chaves de seleção.

`Artifacts`

(Opcional) Uma lista de objetos em que cada um define um artefato binário para o componente na plataforma que esse manifesto define. Por exemplo, você pode definir código ou imagens como artefatos.

Quando o componente é implantado, o software AWS IoT Greengrass Core baixa o artefato em uma pasta no dispositivo principal. Você também pode definir artefatos como arquivos de arquivamento que o software extrai depois de baixá-los.

Você pode usar variáveis de receita para obter os caminhos para as pastas em que os artefatos são instalados no dispositivo principal.

- Arquivos normais — Use a variável de receita artifacts:path para obter o caminho para a pasta que contém os artefatos. Por exemplo, especifique `{artifacts:path}/my_script.py` em uma receita para obter o caminho para um artefato que tenha o URI`s3://DOC-EXAMPLE-BUCKET/path/to/my_script.py`.
- Arquivos extraídos — Use a variável de receita artifacts:DecompressedPath para obter o caminho para a pasta que contém os artefatos do arquivo extraído. O software AWS IoT Greengrass Core extrai cada arquivo em uma pasta com o mesmo nome do arquivo. Por exemplo, especifique `{artifacts:decompressedPath}/my_archive/my_script.py` em uma receita para obter o caminho `my_script.py` no artefato de arquivamento que tem o URI`s3://DOC-EXAMPLE-BUCKET/path/to/my_archive.zip`.

> **Note**
>
> Ao desenvolver um componente com um artefato de arquivamento em um dispositivo central local, talvez você não tenha um URI para esse artefato. Para testar seu componente com uma `Unarchive` opção que extrai o artefato, especifique um URI em que o nome do arquivo corresponda ao nome do arquivo do artefato arquivado. Você pode especificar o URI no qual você espera carregar o artefato de arquivamento ou pode especificar um novo URI de espaço reservado. Por exemplo, para extrair o `my_archive.zip` artefato durante uma implantação local, você pode especificar`s3://DOC-EXAMPLE-BUCKET/my_archive.zip`.

Cada objeto contém as seguintes informações:

`URI`

O URI de um artefato em um bucket do S3. O software AWS IoT Greengrass Core busca o artefato desse URI quando o componente é instalado, a menos que o artefato já exista no dispositivo. Cada artefato deve ter um nome de arquivo exclusivo em cada manifesto.

`Unarchive`

(Opcional) O tipo de arquivo a ser descompactado. Escolha uma das seguintes opções:

- `NONE`— O arquivo não é um arquivo para descompactar. O software AWS IoT Greengrass Core instala o artefato em uma pasta no dispositivo principal. Você pode usar a variável de receita artifacts:path para obter o caminho para essa pasta.
- `ZIP`— O arquivo é um arquivo ZIP. O software AWS IoT Greengrass Core extrai o arquivo em uma pasta com o mesmo nome do arquivo. Você pode usar a variável de receita artifacts:decompressedPath para obter o caminho para a pasta que contém essa pasta.

Padronizado como `NONE`.

`Permission`

(Opcional) Um objeto que define as permissões de acesso a serem definidas para esse arquivo de artefato. Você pode definir a permissão de leitura e a permissão de execução.

> **Note**
>
> Você não pode definir a permissão de gravação, porque o software AWS IoT Greengrass Core não permite que os componentes editem arquivos de artefatos na pasta de artefatos. Para editar um arquivo de artefato em um componente, copie-o para outro local ou publique e implante um novo arquivo de artefato.

Se você definir um artefato como um arquivo a ser descompactado, o software AWS IoT Greengrass Core definirá essas permissões de acesso nos arquivos que ele descompacta do arquivamento. O software AWS IoT Greengrass Core define as permissões de acesso da pasta `ALL` para `Read Execute` e. Isso permite que os componentes visualizem os arquivos descompactados na pasta. Para definir permissões em arquivos individuais do arquivamento, você pode definir as permissões no script do ciclo de vida da instalação.

Esse objeto contém as seguintes informações:

`Read`

(Opcional) A permissão de leitura a ser definida para esse arquivo de artefato. Para permitir que outros componentes acessem esse artefato, como componentes que dependem desse componente, especifique`ALL`. Escolha uma das seguintes opções:

- `NONE`— O arquivo não está legível.
- `OWNER`— O arquivo pode ser lido pelo usuário do sistema que você configura para executar esse componente.
- `ALL`— O arquivo pode ser lido por todos os usuários.

Padronizado como `OWNER`.

`Execute`

(Opcional) A permissão de execução a ser definida para esse arquivo de artefato. A `Execute` permissão implica a `Read` permissão. Por exemplo, se você especificar `ALL` para`Execute`, todos os usuários poderão ler e executar esse arquivo de artefato.

Escolha uma das seguintes opções:

- `NONE`— O arquivo não pode ser executado.
- `OWNER`— O arquivo pode ser executado pelo usuário do sistema que você configura para executar o componente.
- `ALL`— O arquivo pode ser executado por todos os usuários.

Padronizado como `NONE`.

`Digest`

(Somente leitura) O hash criptográfico do resumo do artefato. Quando você cria um componente, AWS IoT Greengrass usa um algoritmo de hash para calcular um hash do arquivo de artefato. Então, quando você implanta o componente, o núcleo do Greengrass calcula o hash do artefato baixado e compara o hash com esse resumo para verificar o artefato antes da instalação. Se o hash não corresponder ao resumo, a implantação falhará.

Se você definir esse parâmetro, AWS IoT Greengrass substituirá o valor definido ao criar o componente.

`Algorithm`

(Somente leitura) O algoritmo de hash AWS IoT Greengrass usado para calcular o hash de resumo do artefato.

Se você definir esse parâmetro, AWS IoT Greengrass substituirá o valor definido ao criar o componente.

`Lifecycle`

Um objeto que define como instalar e executar o componente. O dispositivo principal usa o ciclo de vida global somente se o manifesto a ser usado não especificar um ciclo de vida.

> **Note**
>
> Você define esse ciclo de vida fora de um manifesto. Você também pode definir um ciclo de vida do manifesto que se aplica às plataformas que correspondem a esse manifesto.

No ciclo de vida global, você pode especificar ciclos de vida que são executados para determinadas chaves de seleção que você especifica em cada manifesto. As chaves de seleção são cadeias de caracteres que identificam seções do ciclo de vida global a serem executadas para cada manifesto.

A tecla de `all` seleção é o padrão em qualquer seção sem uma chave de seleção. Isso significa que você pode especificar a chave `all` de seleção em um manifesto para executar as seções do ciclo de vida global sem chaves de seleção. Você não precisa especificar a chave de `all` seleção no ciclo de vida global.

Se um manifesto não definir um ciclo de vida ou chaves de seleção, o dispositivo principal usará a seleção como padrão. `all` Isso significa que, nesse caso, o dispositivo principal usa as seções do ciclo de vida global que não usam teclas de seleção.

Esse objeto contém as mesmas informações do ciclo de vida do manifesto, mas você pode especificar chaves de seleção em qualquer nível para selecionar subseções do ciclo de vida.

> **Tip**
>
> Recomendamos que você use somente letras minúsculas para cada chave de seleção para evitar conflitos entre as chaves de seleção e as chaves do ciclo de vida. As chaves do ciclo de vida começam com letra maiúscula.

Example Exemplo de ciclo de vida global com chaves de seleção de alto nível

```
Lifecycle:
  key1:
    install:
      Skipif: either onpath executable or exists file
      Script: command1
  key2:
    install:
      Script: command2
  all:
    install:
      Script: command3
```

Example Exemplo de ciclo de vida global com chaves de seleção de nível inferior

```
Lifecycle:
  install:
    Script:
      key1: command1
      key2: command2
      all: command3
```

Example Exemplo de ciclo de vida global com vários níveis de chaves de seleção

```
Lifecycle:
  key1:
    install:
      Skipif: either onpath executable or exists file
      Script: command1
  key2:
    install:
      Script: command2
  all:
```

```
    install:
      Script:
        key3: command3
        key4: command4
        all: command5
```

Variáveis da receita

As variáveis da receita expõem informações do componente e do núcleo atuais para você usar em suas receitas. Por exemplo, você pode usar uma variável de receita para passar parâmetros de configuração do componente para um aplicativo executado em um script de ciclo de vida.

Você pode usar variáveis de receita nas seguintes seções das receitas de componentes:

- Definições do ciclo de vida.
- Definições de configuração de componentes, se você usar o Greengrass nucleus v2.6.0 ou posterior e definir a opção de configuração como. interpolateComponentConfiguration`true` Você também pode usar variáveis de receitas ao implantar atualizações de configuração de componentes.

As variáveis da receita usam `{recipe_variable}` sintaxe. Os colchetes indicam uma variável de receita.

AWS IoT Greengrass suporta as seguintes variáveis de receita:

`component_dependency_name``:configuration:`*`json_pointer`*

O valor de um parâmetro de configuração para o componente que essa receita define ou para um componente do qual esse componente depende.

Você pode usar essa variável para fornecer um parâmetro para um script executado no ciclo de vida do componente.

> **Note**
>
> AWS IoT Greengrass suporta essa variável de receita somente nas definições do ciclo de vida do componente.

Essa variável de receita tem as seguintes entradas:

- `component_dependency_name`— (Opcional) O nome da dependência do componente a ser consultada. Omita esse segmento para consultar o componente definido por essa receita. Você pode especificar somente dependências diretas.
- `json_pointer`— O ponteiro JSON para o valor da configuração a ser avaliado. Os ponteiros JSON começam com uma barra. `/` Para identificar um valor em uma configuração de componente aninhado, use barras (`/`) para separar as chaves de cada nível na configuração. Você pode usar um número como chave para especificar um índice em uma lista. Para obter mais informações, consulte a especificação do ponteiro JSON.

  AWS IoT Greengrass O Core usa ponteiros JSON para receitas no formato YAML.

O ponteiro JSON pode fazer referência aos seguintes tipos de nós:

- Um nó de valor. AWS IoT Greengrass O Core substitui a variável da receita pela representação em cadeia do valor. Valores nulos são convertidos em `null` uma string.
- Um nó de objeto. AWS IoT Greengrass O Core substitui a variável de receita pela representação serializada da string JSON desse objeto.
- Sem nó. AWS IoT Greengrass O núcleo não substitui a variável da receita.

Por exemplo, a variável de `{configuration:/Message}` receita recupera o valor da `Message` chave na configuração do componente. A variável de `{com.example.MyComponentDependency:configuration:/server/port}` receita recupera o valor de `port` no objeto de `server` configuração de uma dependência de componente.

`component_dependency_name``:artifacts:path`

O caminho raiz dos artefatos para o componente que essa receita define ou para um componente do qual esse componente depende.

Quando um componente é instalado, AWS IoT Greengrass copia os artefatos do componente para a pasta que essa variável expõe. Você pode usar essa variável para identificar a localização de um script a ser executado no ciclo de vida do componente, por exemplo.

A pasta nesse caminho é somente para leitura. Para modificar arquivos de artefatos, copie os arquivos para outro local, como o diretório de trabalho atual (`$PWDou.`). Em seguida, modifique os arquivos lá.

Para ler ou executar um artefato a partir de uma dependência de componente, essa `Execute` permissão `Read` ou artefato deve ser. `ALL` Para obter mais informações, consulte as permissões de artefato que você define na receita do componente.

Essa variável de receita tem as seguintes entradas:

- `component_dependency_name`— (Opcional) O nome da dependência do componente a ser consultada. Omita esse segmento para consultar o componente definido por essa receita. Você pode especificar somente dependências diretas.

`component_dependency_name``:artifacts:decompressedPath`

O caminho raiz dos artefatos de arquivamento descompactado para o componente que essa receita define ou para um componente do qual esse componente depende.

Quando um componente é instalado, AWS IoT Greengrass descompacta os artefatos de arquivamento do componente na pasta que essa variável expõe. Você pode usar essa variável para identificar a localização de um script a ser executado no ciclo de vida do componente, por exemplo.

Cada artefato é descompactado em uma pasta dentro do caminho descompactado, onde a pasta tem o mesmo nome do artefato menos sua extensão. Por exemplo, um artefato ZIP chamado `models.zip` descompacta na pasta. `{artifacts:decompressedPath}/models`

A pasta nesse caminho é somente para leitura. Para modificar arquivos de artefatos, copie os arquivos para outro local, como o diretório de trabalho atual (`$PWDou .`). Em seguida, modifique os arquivos lá.

Para ler ou executar um artefato a partir de uma dependência de componente, essa `Execute` permissão `Read` ou artefato deve ser. `ALL` Para obter mais informações, consulte as permissões de artefato que você define na receita do componente.

Essa variável de receita tem as seguintes entradas:

- `component_dependency_name`— (Opcional) O nome da dependência do componente a ser consultada. Omita esse segmento para consultar o componente definido por essa receita. Você pode especificar somente dependências diretas.

`component_dependency_name``:work:path`

Esse recurso está disponível para a versão 2.0.4 e posterior do componente de núcleo do Greengrass.

O caminho de trabalho para o componente que essa receita define ou para um componente do qual esse componente depende. O valor dessa variável de receita é equivalente à saída da variável de `$PWD` ambiente e do comando pwd quando executado a partir do contexto do componente.

Você pode usar essa variável de receita para compartilhar arquivos entre um componente e uma dependência.

A pasta nesse caminho pode ser lida e gravada pelo componente definido por essa receita e por outros componentes que são executados pelo mesmo usuário e grupo.

Essa variável de receita tem as seguintes entradas:

- `component_dependency_name`— (Opcional) O nome da dependência do componente a ser consultada. Omita esse segmento para consultar o componente definido por essa receita. Você pode especificar somente dependências diretas.

`kernel:rootPath`

O caminho raiz AWS IoT Greengrass principal.

`iot:thingName`

Esse recurso está disponível para a versão 2.3.0 e posterior do componente de núcleo do Greengrass.

O nome da AWS IoT coisa do dispositivo principal.

Exemplos de receitas

Você pode consultar os exemplos de receitas a seguir para ajudá-lo a criar receitas para seus componentes.

AWS IoT Greengrass organiza um índice dos componentes do Greengrass, chamado Catálogo de Software do Greengrass. Este catálogo rastreia os componentes do Greengrass que são desenvolvidos pela comunidade do Greengrass. A partir desse catálogo, você pode baixar, modificar e implantar componentes para criar seus aplicativos Greengrass. Para ter mais informações, consulte Componentes da comunidade.

Tópicos

- Receita de componentes Hello World

- Exemplo de componente de tempo de execução em Python
- Receita de componente que especifica vários campos

Receita de componentes Hello World

A receita a seguir descreve um componente Hello World que executa um script Python. Esse componente suporta todas as plataformas e aceita um `Message` parâmetro que AWS IoT Greengrass passa como argumento para o script Python. Esta é a receita do componente Hello World no tutorial de introdução.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.HelloWorld",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "My first AWS IoT Greengrass component.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "Message": "world"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "run": "python3 -u {artifacts:path}/hello_world.py {configuration:/Message}"
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "run": "py -3 -u {artifacts:path}/hello_world.py {configuration:/Message}"
      }
    }
  ]
```

```
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.HelloWorld
ComponentVersion: '1.0.0'
ComponentDescription: My first AWS IoT Greengrass component.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    Message: world
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      run: |
        python3 -u {artifacts:path}/hello_world.py "{configuration:/Message}"
  - Platform:
      os: windows
    Lifecycle:
      run: |
        py -3 -u {artifacts:path}/hello_world.py "{configuration:/Message}"
```

Exemplo de componente de tempo de execução em Python

A receita a seguir descreve um componente que instala o Python. Esse componente é compatível com dispositivos Linux de 64 bits.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PythonRuntime",
  "ComponentDescription": "Installs Python 3.7",
  "ComponentPublisher": "Amazon",
  "ComponentVersion": "3.7.0",
  "Manifests": [
    {
      "Platform": {
        "os": "linux",
```

```
        "architecture": "amd64"
      },
      "Lifecycle": {
        "install": "apt-get update\napt-get install python3.7"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PythonRuntime
ComponentDescription: Installs Python 3.7
ComponentPublisher: Amazon
ComponentVersion: '3.7.0'
Manifests:
  - Platform:
      os: linux
      architecture: amd64
    Lifecycle:
      install: |
        apt-get update
        apt-get install python3.7
```

Receita de componente que especifica vários campos

A receita do componente a seguir usa vários campos de receita.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.FooService",
  "ComponentDescription": "Complete recipe for AWS IoT Greengrass components",
  "ComponentPublisher": "Amazon",
  "ComponentVersion": "1.0.0",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "TestParam": "TestValue"
    }
```

```
  },
  "ComponentDependencies": {
    "BarService": {
      "VersionRequirement": "^1.1.0",
      "DependencyType": "SOFT"
    },
    "BazService": {
      "VersionRequirement": "^2.0.0"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux",
        "architecture": "amd64"
      },
      "Lifecycle": {
        "install": {
          "Skipif": "onpath git",
          "Script": "sudo apt-get install git"
        },
        "Setenv": {
          "environment_variable1": "variable_value1",
          "environment_variable2": "variable_value2"
        }
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/hello_world.zip",
          "Unarchive": "ZIP"
        },
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/hello_world_linux.py"
        }
      ]
    },
    {
      "Lifecycle": {
        "install": {
          "Skipif": "onpath git",
          "Script": "sudo apt-get install git",
          "RequiresPrivilege": "true"
        }
      },
```

```
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/hello_world.py"
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.FooService
ComponentDescription: Complete recipe for AWS IoT Greengrass components
ComponentPublisher: Amazon
ComponentVersion: 1.0.0
ComponentConfiguration:
  DefaultConfiguration:
    TestParam: TestValue
ComponentDependencies:
  BarService:
    VersionRequirement: ^1.1.0
    DependencyType: SOFT
  BazService:
    VersionRequirement: ^2.0.0
Manifests:
  - Platform:
      os: linux
      architecture: amd64
    Lifecycle:
      install:
        Skipif: onpath git
        Script: sudo apt-get install git
      Setenv:
        environment_variable1: variable_value1
        environment_variable2: variable_value2
    Artifacts:
      - URI: 's3://DOC-EXAMPLE-BUCKET/hello_world.zip'
        Unarchive: ZIP
      - URI: 's3://DOC-EXAMPLE-BUCKET/hello_world_linux.py'
  - Lifecycle:
      install:
```

```
        Skipif: onpath git
        Script: sudo apt-get install git
        RequiresPrivilege: 'true'
    Artifacts:
      - URI: 's3://DOC-EXAMPLE-BUCKET/hello_world.py'
```

Referência de variáveis de ambiente de componentes

O softwareAWS IoT Greengrass Core define variáveis de ambiente ao executar scripts de ciclo de vida para componentes. Você pode obter essas variáveis de ambiente em seus componentes para obter o nome da coisa e a versão do núcleo do Greengrass.Região da AWS O software também define as variáveis de ambiente que seu componente precisa para usar o SDK de comunicação entre processos e interagir comAWS os serviços.

Você também pode definir variáveis de ambiente personalizadas para os scripts de ciclo de vida do seu componente. Para obter mais informações, consulte Setenv.

O softwareAWS IoT Greengrass Core define as seguintes variáveis de ambiente:

`AWS_IOT_THING_NAME`

O nome daAWS IoT coisa que representa esse dispositivo central do Greengrass.

`AWS_REGION`

Região da AWSOnde esse dispositivo central do Greengrass opera.

OsAWS SDKs usam essa variável de ambiente para identificar a região padrão a ser usada. Essa variável é equivalente`AWS_DEFAULT_REGION` a.

`AWS_DEFAULT_REGION`

Região da AWSOnde esse dispositivo central do Greengrass opera.

OAWS CLI usa essa variável de ambiente para identificar a região padrão a ser usada. Essa variável é equivalente`AWS_REGION` a.

`GGC_VERSION`

A versão do componente do núcleo Greengrass que é executado neste dispositivo central do Greengrass.

`GG_ROOT_CA_PATH`

Esse recurso está disponível para a v2.5.5 e versões posteriores do componente de núcleo Greengrass.

O caminho para o certificado da autoridade de certificação raiz (CA) que o núcleo Greengrass usa.

`AWS_GG_NUCLEUS_DOMAIN_SOCKET_FILEPATH_FOR_COMPONENT`

O caminho para o soquete IPC que os componentes usam para se comunicar com o softwareAWS IoT Greengrass Core. Para obter mais informações, consulte Use o AWS IoT Device SDK para se comunicar com o núcleo do Greengrass, outros componentes e AWS IoT Core.

`SVCUID`

O token secreto que os componentes usam para se conectar ao soquete IPC e se comunicar com o softwareAWS IoT Greengrass Core. Para obter mais informações, consulte Use o AWS IoT Device SDK para se comunicar com o núcleo do Greengrass, outros componentes e AWS IoT Core.

`AWS_CONTAINER_AUTHORIZATION_TOKEN`

O token secreto que os componentes usam para recuperar credenciais do componente do serviço de troca de tokens.

`AWS_CONTAINER_CREDENTIALS_FULL_URI`

O URI que os componentes solicitam para recuperar credenciais do componente do serviço de troca de tokens.

Implemente AWS IoT Greengrass componentes em dispositivos

Você pode usar AWS IoT Greengrass para implantar componentes em dispositivos ou grupos de dispositivos. Você usa implantações para definir os componentes e as configurações que são enviados aos dispositivos. AWS IoT Greengrassé implantado em alvos, AWS IoT coisas ou grupos de coisas que representam os principais dispositivos do Greengrass. AWS IoT Greengrassusa AWS IoT Coretrabalhos para implantar em seus dispositivos principais. Você pode configurar como o trabalho é implementado em seus dispositivos.

Implantações de dispositivos principais

Cada dispositivo principal executa os componentes das implantações desse dispositivo. Uma nova implantação no mesmo destino substitui a implantação anterior no destino. Ao criar uma implantação, você define os componentes e as configurações a serem aplicados ao software existente do dispositivo principal.

Ao revisar uma implantação para um destino, você substitui os componentes da revisão anterior pelos componentes da nova revisão. Por exemplo, você implanta os Gerente secreto componentes Gerenciador de registros e no grupo de coisas`TestGroup`. Em seguida, você cria outra implantação para `TestGroup` que especifique somente o componente do gerenciador secreto. Como resultado, os dispositivos principais desse grupo não executam mais o gerenciador de registros.

Resolução da dependência da plataforma

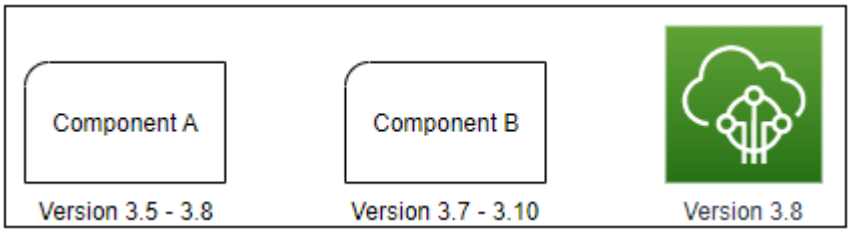
Quando um dispositivo principal recebe uma implantação, ele verifica se os componentes são compatíveis com o dispositivo principal. Por exemplo, se você Firehose implantar o em um destino do Windows, a implantação falhará.

Resolução de dependência de componentes

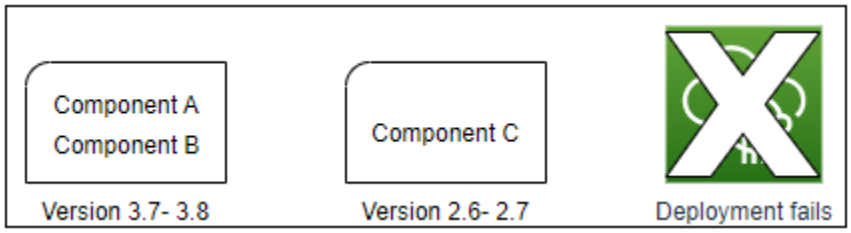
O dispositivo principal também verifica se as dependências de cada componente são compatíveis com as restrições de versão para implantações de outros componentes nesse grupo de itens. Quando as restrições de versão de um componente se sobrepõem, o Greengrass usa a versão mais alta aplicável do componente. Por exemplo: .

- Você implanta o componente A em`TestGroup`. O componente A depende das `com.example.PythonRuntime` versões 3.5 a 3.10 do componente.
- Em seguida, você implanta o componente B em`TestGroup`. O componente B depende das `com.example.PythonRuntime` versões 3.7 a 3.8 do componente.

Como resultado, os dispositivos principais `TestGroup` determinam que podem implantar a versão 3.8 do `com.example.PythonRuntime` componente porque essa versão é a versão mais alta aplicável em que as restrições de versão se sobrepõem.



Em seguida, você implanta o componente C em`TestGroup`. O componente C depende das `com.example.PythonRuntime` versões 2.6 a 2.7 do componente. Essa implantação falha porque não há nenhuma versão do componente que atenda às restrições 2.6 - 2.7 e 3.7 - 3.8.



Removendo um dispositivo de um grupo de coisas

Quando você remove um dispositivo principal de um grupo de coisas, o comportamento de implantação do componente depende da versão do núcleo do Greengrass que o dispositivo principal executa.

2.5.1 and later

Quando você remove um dispositivo principal de um grupo de coisas, o comportamento depende se a AWS IoT política concede a `greengrass:ListThingGroupsForCoreDevice` permissão. Para obter mais informações sobre essa permissão e AWS IoT políticas para dispositivos principais, consulteAutorização e autenticação do dispositivo para o AWS IoT Greengrass.

- Se a AWS IoT política conceder essa permissão

  Quando você remove um dispositivo principal de um grupo de coisas, AWS IoT Greengrass remove os componentes do grupo de coisas na próxima vez que uma implantação for feita no dispositivo. Se um componente no dispositivo for incluído na próxima implantação, esse componente não será removido do dispositivo.

- Se a AWS IoT política não conceder essa permissão

  Quando você remove um dispositivo principal de um grupo de coisas, AWS IoT Greengrass não exclui os componentes desse grupo de coisas do dispositivo.

Para remover um componente de um dispositivo, use o comando deployment create da CLI do Greengrass. Especifique o componente a ser removido com o `--remove` argumento e especifique o grupo de coisas com o `--groupId` argumento.

2.5.0

Quando você remove um dispositivo principal de um grupo de coisas, AWS IoT Greengrass remove os componentes do grupo de coisas na próxima vez que uma implantação for feita no dispositivo. Se um componente no dispositivo for incluído na próxima implantação, esse componente não será removido do dispositivo.

Esse comportamento exige que a AWS IoT política do dispositivo principal conceda a `greengrass:ListThingGroupsForCoreDevice` permissão. Se um dispositivo principal não tiver essa permissão, o dispositivo principal não aplicará implantações. Para ter mais informações, consulte Autorização e autenticação do dispositivo para o AWS IoT Greengrass.

2.0.x - 2.4.x

Quando você remove um dispositivo principal de um grupo de coisas, AWS IoT Greengrass não exclui os componentes desse grupo de coisas do dispositivo.

Para remover um componente de um dispositivo, use o comando deployment create da CLI do Greengrass. Especifique o componente a ser removido com o `--remove` argumento e especifique o grupo de coisas com o `--groupId` argumento.

Implantações

As implantações são contínuas. Quando você cria uma implantação, AWS IoT Greengrass implementa a implantação nos dispositivos de destino que estão on-line. Se um dispositivo de destino não estiver on-line, ele receberá a implantação na próxima vez em que se conectarAWS IoT Greengrass. Quando você adiciona um dispositivo principal a um grupo de itens de destino, AWS IoT Greengrass envia ao dispositivo a implantação mais recente desse grupo de itens.

Antes de um dispositivo principal implantar um componente, por padrão, ele notifica cada componente no dispositivo. Os componentes do Greengrass podem responder à notificação para adiar a implantação. Talvez você queira adiar a implantação se o dispositivo tiver um nível de bateria baixo ou estiver executando um processo que não pode ser interrompido. Para ter mais informações, consulte Tutorial: Desenvolva um componente do Greengrass que adia as atualizações

de componentes. Ao criar uma implantação, você pode configurá-la para implantação sem notificar os componentes.

Cada item ou grupo de itens de destino pode ter uma implantação por vez. Isso significa que quando você cria uma implantação para um destino, AWS IoT Greengrass não implanta mais a revisão anterior da implantação desse alvo.

Opções de implantação

As implantações oferecem várias opções que permitem controlar quais dispositivos recebem uma atualização e como a atualização é implantada. Ao criar uma implantação, você pode configurar as seguintes opções:

- AWS IoT Greengrasscomponentes

  Defina os componentes a serem instalados e executados nos dispositivos de destino. AWS IoT Greengrasscomponentes são módulos de software que você implanta e executa nos dispositivos principais do Greengrass. Os dispositivos recebem componentes somente se o componente suportar a plataforma do dispositivo. Isso permite que você implante em grupos de dispositivos, mesmo que os dispositivos de destino sejam executados em várias plataformas. Se um componente não for compatível com a plataforma do dispositivo, o componente não será implantado no dispositivo.

  Você pode implantar componentes AWS personalizados e componentes fornecidos em seus dispositivos. Quando você implanta um componente, AWS IoT Greengrass identifica todas as dependências do componente e as implanta também. Para obter mais informações, consulte AWS-componentes fornecidos e Desenvolva AWS IoT Greengrass componentes.

  Você define a versão e a atualização de configuração a serem implantadas em cada componente. A atualização de configuração especifica como modificar a configuração existente do componente no dispositivo principal ou a configuração padrão do componente se o componente não existir no dispositivo principal. Você pode especificar quais valores de configuração serão redefinidos para os valores padrão e os novos valores de configuração a serem mesclados no dispositivo principal. Quando um dispositivo principal recebe implantações para destinos diferentes e cada implantação especifica versões de componentes compatíveis, o dispositivo principal aplica as atualizações de configuração em ordem com base no registro de data e hora de quando você cria a implantação. Para ter mais informações, consulte Atualizar configurações de componentes.

> **Important**
>
> Quando você implanta um componente, AWS IoT Greengrass instala as versões mais recentes suportadas de todas as dependências desse componente. Por esse motivo, novas versões AWS de patch dos componentes públicos fornecidos podem ser implantadas automaticamente em seus dispositivos principais se você adicionar novos dispositivos a um grupo de coisas ou atualizar a implantação que visa esses dispositivos. Algumas atualizações automáticas, como a atualização do nucleus, podem fazer com que seus dispositivos reiniciem inesperadamente.
> Para evitar atualizações não intencionais para um componente que está sendo executado em seu dispositivo, recomendamos que você inclua diretamente sua versão preferida desse componente ao criar uma implantação. Para obter mais informações sobre o comportamento de atualização AWS IoT Greengrass do software Core, consulteAtualize o software AWS IoT Greengrass principal (OTA).

- Políticas de implantação

  Defina quando é seguro implantar uma configuração e o que fazer se a implantação falhar. Você pode especificar se deve ou não esperar que os componentes relatem que podem ser atualizados. Você também pode especificar se deseja ou não reverter os dispositivos para a configuração anterior se eles aplicarem uma implantação que falhe.

- Parar a configuração

  Defina quando e como interromper uma implantação. A implantação é interrompida e falha se os critérios definidos por você forem atendidos. Por exemplo, você pode configurar uma implantação para ser interrompida se uma porcentagem de dispositivos não conseguir aplicá-la após um número mínimo de dispositivos recebê-la.

- Configuração de distribuição

  Defina a taxa na qual uma implantação é implementada nos dispositivos de destino. Você pode configurar um aumento exponencial da taxa com limites mínimos e máximos de taxa.

- Configuração de tempo limite

  Defina o tempo máximo que cada dispositivo tem para aplicar uma implantação. Se um dispositivo exceder a duração especificada, o dispositivo não conseguirá aplicar a implantação.

Important

Componentes personalizados podem definir artefatos em buckets do S3. Quando o software AWS IoT Greengrass principal implanta um componente, ele baixa os artefatos do componente do. Nuvem AWS As funções principais do dispositivo não permitem acesso aos buckets do S3 por padrão. Para implantar componentes personalizados que definem artefatos em um bucket do S3, a função principal do dispositivo deve conceder permissões para baixar artefatos desse bucket. Para ter mais informações, consulte Permitir acesso aos buckets do S3 para artefatos de componentes.

Tópicos

Criar implantações

Você pode criar uma implantação que tenha como alvo uma coisa ou um grupo de coisas.

Ao criar uma implantação, você configura os componentes de software a serem implantados e como o trabalho de implantação é implementado nos dispositivos de destino. Você pode definir a implantação no arquivo JSON que você fornece aoAWS CLI.

O destino de implantação determina os dispositivos nos quais você deseja executar seus componentes. Para implantar em um dispositivo principal, especifique uma coisa. Para implantar em vários dispositivos principais, especifique um grupo de coisas que inclua esses dispositivos. Para obter mais informações sobre como configurar grupos de coisas, consulte Grupos de coisas estáticos e Grupos de coisas dinâmicas no Guia do AWS IoT desenvolvedor.

Siga as etapas desta seção para criar uma implantação em um destino. Para obter mais informações sobre como atualizar os componentes de software em um destino que tenha uma implantação, consulteRevise as implantações.

> **Warning**
>
> A CreateDeploymentoperação pode desinstalar componentes dos dispositivos principais. Se um componente estiver presente na implantação anterior e não na nova implantação, o dispositivo principal desinstalará esse componente. Para evitar a desinstalação de componentes, primeiro use a ListDeploymentsoperação para verificar se o destino da implantação já tem uma implantação existente. Em seguida, use a GetDeploymentoperação para começar a partir da implantação existente ao criar uma nova implantação.

Para criar uma implantação (AWS CLI)

1. Crie um arquivo chamado e`deployment.json`, em seguida, copie o seguinte objeto JSON para o arquivo. Substitua *`targetArn`* pelo ARN da coisa ou grupo de coisas AWS IoT a ser direcionado para a implantação. Os ARNs de grupos de coisas e coisas têm o seguinte formato:

   - Coisa: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`thingName`*
   - Grupo de coisas: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thinggroup/`*`thingGroupName`*

   ```
   {
     "targetArn": "targetArn"
   }
   ```

2. Verifique se o destino de implantação tem uma implantação existente que você deseja revisar. Faça o seguinte:

   a. Execute o comando a seguir para listar as implantações para o destino de implantação. Substitua *`targetArn`* pelo ARN da coisa ou grupo de coisas de destino. AWS IoT

      ```
      aws greengrassv2 list-deployments --target-arn targetArn
      ```

      A resposta contém uma lista com a implantação mais recente do destino. Se a resposta estiver vazia, o destino não tem uma implantação existente e você pode pular paraStep 3. Caso contrário, copie o `deploymentId` da resposta para usar na próxima etapa.

> **Note**
>
> Você também pode revisar uma implantação diferente da revisão mais recente do destino. Especifique o `--history-filter ALL` argumento para listar todas as implantações do destino. Em seguida, copie a ID da implantação que você deseja revisar.

b. Execute o comando a seguir para obter os detalhes da implantação. Esses detalhes incluem metadados, componentes e configuração do trabalho. Substitua *`deploymentID`* pelo ID da etapa anterior.

```
aws greengrassv2 get-deployment --deployment-id deploymentId
```

A resposta contém os detalhes da implantação.

c. Copie qualquer um dos seguintes pares de valores-chave da resposta do comando anterior para. `deployment.json` Você pode alterar esses valores para a nova implantação.

- `deploymentName`— O nome da implantação.
- `components`— Os componentes da implantação. Para desinstalar um componente, remova-o desse objeto.
- `deploymentPolicies`— As políticas de implantação.
- `iotJobConfiguration`— A configuração do trabalho da implantação.
- `tags`— As tags da implantação.

3. (Opcional) Defina um nome para a implantação. Substitua *`DeploymentName`* pelo nome da implantação.

```
{
  "targetArn": "targetArn",
  "deploymentName": "deploymentName"
}
```

4. Adicione cada componente para implantar os dispositivos de destino. Para fazer isso, adicione pares de valores-chave ao `components` objeto, onde a chave é o nome do componente e o valor é um objeto que contém os detalhes desse componente. Especifique os seguintes detalhes para cada componente que você adicionar:

- `version`— A versão do componente a ser implantada.
- `configurationUpdate`— A atualização de configuração a ser implantada. A atualização é uma operação de patch que modifica a configuração existente do componente em cada dispositivo de destino ou a configuração padrão do componente se ela não existir no dispositivo de destino. Você pode especificar as seguintes atualizações de configuração:
  - Redefinir atualizações (`reset`) — (Opcional) Uma lista de ponteiros JSON que definem os valores de configuração a serem redefinidos para seus valores padrão no dispositivo de destino. O software AWS IoT Greengrass Core aplica as atualizações de redefinição antes da aplicação das atualizações de mesclagem. Para ter mais informações, consulte Redefinir atualizações.
  - Merge updates (`merge`) — (Opcional) Um documento JSON que define os valores de configuração a serem mesclados no dispositivo de destino. Você deve serializar o documento JSON como uma string. Para ter mais informações, consulte Mesclar atualizações.
- `runWith`— (Opcional) As opções de processo do sistema que o software AWS IoT Greengrass Core usa para executar os processos desse componente no dispositivo principal. Se você omitir um parâmetro no `runWith` objeto, o software AWS IoT Greengrass Core usará os valores padrão que você configura no componente do núcleo do Greengrass.

  Você pode especificar qualquer uma das seguintes opções:
  - `posixUser`— O usuário do sistema POSIX e, opcionalmente, o grupo a ser usado para executar esse componente nos dispositivos principais do Linux. O usuário e o grupo, se especificados, devem existir em cada dispositivo principal do Linux. Especifique o usuário e o grupo separando-os por dois pontos (`:`), no seguinte formato: `user:group`. O grupo é opcional. Se você não especificar um grupo, o software AWS IoT Greengrass Core usará o grupo primário para o usuário. Para ter mais informações, consulte Configurar o usuário que executa os componentes.
  - `windowsUser`— O usuário do Windows a ser usado para executar esse componente nos dispositivos principais do Windows. O usuário deve existir em cada dispositivo principal do Windows e seu nome e senha devem ser armazenados na instância do Gerenciador de Credenciais da LocalSystem conta. Para ter mais informações, consulte Configurar o usuário que executa os componentes.

    Esse recurso está disponível para a versão 2.5.0 e posterior do componente de núcleo do Greengrass.

- `systemResourceLimits`— Os limites de recursos do sistema a serem aplicados aos processos desse componente. Você pode aplicar limites de recursos do sistema a componentes Lambda genéricos e não conteinerizados. Para ter mais informações, consulte Configurar limites de recursos do sistema para componentes.

  Você pode especificar qualquer uma das seguintes opções:

  - `cpus`— A quantidade máxima de tempo de CPU que os processos desse componente podem usar no dispositivo principal. O tempo total da CPU de um dispositivo essencial é equivalente ao número de núcleos da CPU do dispositivo. Por exemplo, em um dispositivo principal com 4 núcleos de CPU, você pode definir esse valor 2 para limitar os processos desse componente a 50% de uso de cada núcleo da CPU. Em um dispositivo com 1 núcleo de CPU, você pode definir esse valor `0.25` para limitar os processos desse componente a 25% de uso da CPU. Se você definir esse valor como um número maior que o número de núcleos de CPU, o software AWS IoT Greengrass Core não limitará o uso da CPU do componente.
  - `memory`— A quantidade máxima de RAM (em kilobytes) que os processos desse componente podem usar no dispositivo principal.

  Esse recurso está disponível para a versão 2.4.0 e posterior do componente de núcleo do Greengrass. AWS IoT Greengrassatualmente não oferece suporte a esse recurso nos dispositivos principais do Windows.

Example Exemplo de atualização de configuração básica

O `components` objeto de exemplo a seguir especifica a implantação de um componente,`com.example.PythonRuntime`, que espera um parâmetro de configuração chamado`pythonVersion`.

```
{
  "targetArn": "targetArn",
  "deploymentName": "deploymentName",
  "components": {
    "com.example.PythonRuntime": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "merge": "{\"pythonVersion\":\"3.7\"}"
```

```
      }
    }
  }
}
```

Example Exemplo de atualização de configuração com atualizações de redefinição e mesclagem

Considere um exemplo de componente de painel industrial`com.example.IndustrialDashboard`,, que tem a seguinte configuração padrão.

```
{
  "name": null,
  "mode": "REQUEST",
  "network": {
    "useHttps": true,
    "port": {
      "http": 80,
      "https": 443
    },
  },
  "tags": []
}
```

A atualização de configuração a seguir especifica as seguintes instruções:

1. Redefina a configuração HTTPS para seu valor padrão (`true`).
2. Redefina a lista de etiquetas industriais para uma lista vazia.
3. Combine uma lista de etiquetas industriais que identificam fluxos de dados de temperatura e pressão para duas caldeiras.

```
{
  "reset": [
    "/network/useHttps",
    "/tags"
  ],
  "merge": {
    "tags": [
      "/boiler/1/temperature",
      "/boiler/1/pressure",
      "/boiler/2/temperature",
```

```
      "/boiler/2/pressure"
    ]
  }
}
```

O `components` objeto de exemplo a seguir especifica a implantação desse componente de painel industrial e a atualização de configuração.

```
{
  "targetArn": "targetArn",
  "deploymentName": "deploymentName",
  "components": {
    "com.example.IndustrialDashboard": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "reset": [
          "/network/useHttps",
          "/tags"
        ],
        "merge": "{\"tags\":[\"/boiler/1/temperature\",\"/boiler/1/pressure\",\"/
boiler/2/temperature\",\"/boiler/2/pressure\"]}"
      }
    }
  }
}
```

5. (Opcional) Defina políticas de implantação para a implantação. Você pode configurar quando os dispositivos principais podem aplicar uma implantação com segurança ou o que fazer se um dispositivo principal não conseguir aplicar a implantação. Para fazer isso, adicione um `deploymentPolicies` objeto e`deployment.json`, em seguida, faça o seguinte:

   1. (Opcional) Especifique a política de atualização do componente (`componentUpdatePolicy`). Essa política define se a implantação permite ou não que os componentes adiem uma atualização até que estejam prontos para serem atualizados. Por exemplo, os componentes podem precisar limpar recursos ou concluir ações críticas antes de serem reiniciados para aplicar uma atualização. Essa política também define a quantidade de tempo que os componentes têm para responder a uma notificação de atualização.

      Essa política é um objeto com os seguintes parâmetros:

      - `action`— (Opcional) Se deve ou não notificar os componentes e esperar que eles relatem quando estiverem prontos para a atualização. Escolha uma das seguintes opções:

- `NOTIFY_COMPONENTS`: a implantação notifica cada componente antes que ele seja interrompido e atualiza esse componente. Os componentes podem usar a operação SubscribeToComponentUpdates IPC para receber essas notificações.
- `SKIP_NOTIFY_COMPONENTS`: a implantação não notifica os componentes ou aguarda que eles possam ser atualizados com segurança.

Padronizado como `NOTIFY_COMPONENTS`.

- `timeoutInSeconds`A quantidade de tempo em segundos que cada componente tem para responder a uma notificação de atualização com a operação DeferComponentUpdate IPC. Se o componente não responder dentro desse período de tempo, a implantação prosseguirá no dispositivo principal.

  O padrão é 60 segundos.

2. (Opcional) Especifique a política de validação da configuração (`configurationValidationPolicy`). Essa política define quanto tempo cada componente tem para validar uma atualização de configuração de uma implantação. Os componentes podem usar a operação SubscribeToValidateConfigurationUpdates IPC para assinar notificações para suas próprias atualizações de configuração. Em seguida, os componentes podem usar a operação SendConfigurationValidityReport IPC para informar ao software AWS IoT Greengrass Core se a atualização de configuração é válida. Se a atualização de configuração não for válida, a implantação falhará.

   Essa política é um objeto com o seguinte parâmetro:

   - `timeoutInSeconds`(Opcional) A quantidade de tempo em segundos que cada componente tem para validar uma atualização de configuração. Se o componente não responder dentro desse período de tempo, a implantação prosseguirá no dispositivo principal.

     O padrão é 30 segundos.

3. (Opcional) Especifique a política de tratamento de falhas (`failureHandlingPolicy`). Essa política é uma string que define se os dispositivos devem ser revertidos ou não se a implantação falhar. Escolha uma das seguintes opções:

   - `ROLLBACK`— Se a implantação falhar em um dispositivo principal, o software AWS IoT Greengrass Core reverterá esse dispositivo principal para a configuração anterior.

- `DO_NOTHING`— Se a implantação falhar em um dispositivo principal, o software AWS IoT Greengrass Core manterá a nova configuração. Isso pode resultar em componentes quebrados se a nova configuração não for válida.

Padronizado como `ROLLBACK`.

Sua implantação em `deployment.json` pode ser semelhante ao exemplo a seguir:

```
{
  "targetArn": "targetArn",
  "deploymentName": "deploymentName",
  "components": {
    "com.example.IndustrialDashboard": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "reset": [
          "/network/useHttps",
          "/tags"
        ],
        "merge": "{\"tags\":[\"/boiler/1/temperature\",\"/boiler/1/pressure\",\"/
boiler/2/temperature\",\"/boiler/2/pressure\"]}"
      }
    }
  },
  "deploymentPolicies": {
    "componentUpdatePolicy": {
      "action": "NOTIFY_COMPONENTS",
      "timeoutInSeconds": 30
    },
    "configurationValidationPolicy": {
      "timeoutInSeconds": 60
    },
    "failureHandlingPolicy": "ROLLBACK"
  }
}
```

6. (Opcional) Defina como a implantação é interrompida, implementada ou expira. AWS IoT Greengrassusa AWS IoT Core trabalhos para enviar implantações aos dispositivos principais, portanto, essas opções são idênticas às opções de configuração dos AWS IoT Core trabalhos. Para obter mais informações, consulte Job Rollout and abort configuration no Developer Guide. AWS IoT

Para definir as opções de trabalho, adicione um `iotJobConfiguration` objeto `deployment.json` a. Em seguida, defina as opções a serem configuradas.

Sua implantação em `deployment.json` pode ser semelhante ao exemplo a seguir:

```
{
  "targetArn": "targetArn",
  "deploymentName": "deploymentName",
  "components": {
    "com.example.IndustrialDashboard": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "reset": [
          "/network/useHttps",
          "/tags"
        ],
        "merge": "{\"tags\":[\"/boiler/1/temperature\",\"/boiler/1/pressure\",\"/
boiler/2/temperature\",\"/boiler/2/pressure\"]}"
      }
    }
  },
  "deploymentPolicies": {
    "componentUpdatePolicy": {
      "action": "NOTIFY_COMPONENTS",
      "timeoutInSeconds": 30
    },
    "configurationValidationPolicy": {
      "timeoutInSeconds": 60
    },
    "failureHandlingPolicy": "ROLLBACK"
  },
  "iotJobConfiguration": {
    "abortConfig": {
      "criteriaList": [
        {
          "action": "CANCEL",
          "failureType": "ALL",
          "minNumberOfExecutedThings": 100,
          "thresholdPercentage": 5
        }
      ]
    },
```

```
    "jobExecutionsRolloutConfig": {
      "exponentialRate": {
        "baseRatePerMinute": 5,
        "incrementFactor": 2,
        "rateIncreaseCriteria": {
          "numberOfNotifiedThings": 10,
          "numberOfSucceededThings": 5
        }
      },
      "maximumPerMinute": 50
    },
    "timeoutConfig":  {
      "inProgressTimeoutInMinutes": 5
    }
  }
}
```

7. (Opcional) Adicione tags (`tags`) para a implantação. Para ter mais informações, consulte Marcar com tag os recursos do AWS IoT Greengrass Version 2.
8. Execute o comando a seguir para criar a implantação a partir de`deployment.json`.

   ```
   aws greengrassv2 create-deployment --cli-input-json file://deployment.json
   ```

   A resposta inclui uma `deploymentId` que identifica essa implantação. Você pode usar o ID de implantação para verificar o status da implantação. Para ter mais informações, consulte Verificar status da implantação.

Atualizar configurações de componentes

As configurações de componentes são objetos JSON que definem os parâmetros de cada componente. A receita de cada componente define sua configuração padrão, que você modifica ao implantar componentes nos dispositivos principais.

Ao criar uma implantação, você pode especificar a atualização de configuração a ser aplicada a cada componente. As atualizações de configuração são operações de patch, o que significa que a atualização modifica a configuração do componente que existe no dispositivo principal. Se o dispositivo principal não tiver o componente, a atualização da configuração modifica e aplica a configuração padrão para essa implantação.

A atualização de configuração define atualizações de redefinição e atualizações de mesclagem. As atualizações de redefinição definem quais valores de configuração devem ser redefinidos para seus padrões ou removidos. As atualizações de mesclagem definem os novos valores de configuração a serem definidos para o componente. Quando você implanta uma atualização de configuração, o software AWS IoT Greengrass Core executa a atualização de redefinição antes da atualização de mesclagem.

Os componentes podem validar as atualizações de configuração que você implanta. O componente se inscreve para receber uma notificação quando uma implantação altera sua configuração e pode rejeitar uma configuração que não suporta. Para ter mais informações, consulte Interaja com a configuração do componente.

Tópicos

- Redefinir atualizações
- Mesclar atualizações
- Exemplos

Redefinir atualizações

As atualizações de redefinição definem quais valores de configuração devem ser redefinidos para seus valores padrão no dispositivo principal. Se um valor de configuração não tiver um valor padrão, a atualização de redefinição removerá esse valor da configuração do componente. Isso pode ajudá-lo a corrigir um componente que falha como resultado de uma configuração inválida.

Use uma lista de ponteiros JSON para definir quais valores de configuração devem ser redefinidos. Os ponteiros JSON começam com uma barra. / Para identificar um valor em uma configuração de componente aninhado, use barras (/) para separar as chaves de cada nível na configuração. Para obter mais informações, consulte a especificação do ponteiro JSON.

> **Note**
>
> Você pode redefinir somente uma lista inteira para seus valores padrão. Você não pode usar as atualizações de redefinição para redefinir um elemento individual em uma lista.

Para redefinir toda a configuração de um componente para seus valores padrão, especifique uma única string vazia como atualização de redefinição.

```
"reset": [""]
```

Mesclar atualizações

As atualizações de mesclagem definem os valores de configuração a serem inseridos na configuração do componente no núcleo. A atualização de mesclagem é um objeto JSON que o software AWS IoT Greengrass Core mescla depois de redefinir os valores nos caminhos especificados na atualização de redefinição. Ao usar os AWS SDKs AWS CLI ou, você deve serializar esse objeto JSON como uma string.

Você pode mesclar um par de valores-chave que não existe na configuração padrão do componente. Você também pode mesclar um par de valores-chave que tenha um tipo diferente do valor com a mesma chave. O novo valor substitui o valor antigo. Isso significa que você pode alterar a estrutura do objeto de configuração.

Você pode mesclar valores nulos e sequências de caracteres, listas e objetos vazios.

> **Note**
>
> Você não pode usar atualizações de mesclagem com a finalidade de inserir ou acrescentar um elemento a uma lista. Você pode substituir uma lista inteira ou definir um objeto em que cada elemento tenha uma chave exclusiva.
> AWS IoT Greengrass usa JSON para valores de configuração. O JSON especifica um tipo de número, mas não diferencia entre números inteiros e flutuantes. Como resultado, os valores de configuração podem ser convertidos em floats in AWS IoT Greengrass. Para garantir que seu componente use o tipo de dados correto, recomendamos que você defina valores de configuração numérica como cadeias de caracteres. Em seguida, faça com que seu componente os analise como números inteiros ou flutuantes. Isso garante que seus valores de configuração tenham o mesmo tipo na configuração e no seu dispositivo principal.

Use variáveis de receita em atualizações de mesclagem

Esse recurso está disponível para a versão 2.6.0 e posterior do componente núcleo do Greengrass.

Se você definir a opção de ComponentConfiguration configuração de interpolação do núcleo Greengrass como`true`, poderá usar variáveis de receita, além da variável de receita, em atualizações de *`component_dependency_name`*`:configuration:`*`json_pointer`* mesclagem.

Por exemplo, você pode usar a variável de `{iot:thingName}` receita em uma atualização de mesclagem para incluir o nome do item do AWS IoT dispositivo principal em um valor de configuração do componente, como uma política de autorização de comunicação entre processos (IPC).

Exemplos

O exemplo a seguir demonstra as atualizações de configuração para um componente do painel que tem a seguinte configuração padrão. Este componente de exemplo exibe informações sobre equipamentos industriais.

```
{
  "name": null,
  "mode": "REQUEST",
  "network": {
    "useHttps": true,
    "port": {
      "http": 80,
      "https": 443
    },
  },
  "tags": []
}
```

Receita de componente de painel industrial

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.IndustrialDashboard",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Displays information about industrial equipment.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "name": null,
      "mode": "REQUEST",
      "network": {
        "useHttps": true,
        "port": {
          "http": 80,
```

```
          "https": 443
        },
      },
      "tags": []
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "run": "python3 -u {artifacts:path}/industrial_dashboard.py"
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "run": "py -3 -u {artifacts:path}/industrial_dashboard.py"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.IndustrialDashboard
ComponentVersion: '1.0.0'
ComponentDescription: Displays information about industrial equipment.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    name: null
    mode: REQUEST
    network:
      useHttps: true
      port:
        http: 80
        https: 443
```

```
    tags: []
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      run: |
        python3 -u {artifacts:path}/industrial_dashboard.py
  - Platform:
      os: windows
    Lifecycle:
      run: |
        py -3 -u {artifacts:path}/industrial_dashboard.py
```

Example Exemplo 1: atualização de mesclagem

Você cria uma implantação que aplica a seguinte atualização de configuração, que especifica uma atualização de mesclagem, mas não uma atualização de redefinição. Essa atualização de configuração instrui o componente a exibir o painel na porta HTTP 8080 com dados de duas caldeiras.

Console

Configuração para mesclar

```
{
  "name": "Factory 2A",
  "network": {
    "useHttps": false,
    "port": {
      "http": 8080
    }
  },
  "tags": [
    "/boiler/1/temperature",
    "/boiler/1/pressure",
    "/boiler/2/temperature",
    "/boiler/2/pressure"
  ]
}
```

AWS CLI

O comando a seguir cria uma implantação em um dispositivo principal.

```
aws greengrassv2 create-deployment --cli-input-json file://dashboard-deployment.json
```

O `dashboard-deployment.json` arquivo contém o seguinte documento JSON.

```
{
  "targetArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
  "deploymentName": "Deployment for MyGreengrassCore",
  "components": {
    "com.example.IndustrialDashboard": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "merge": "{\"name\":\"Factory 2A\",\"network\":{\"useHttps\":false,\"port
\":{\"http\":8080}},\"tags\":[\"/boiler/1/temperature\",\"/boiler/1/pressure\",\"/
boiler/2/temperature\",\"/boiler/2/pressure\"]}"
      }
    }
  }
}
```

Greengrass CLI

O comando CLI do Greengrass a seguir cria uma implantação local em um dispositivo principal.

```
sudo greengrass-cli deployment create \
  --recipeDir recipes \
  --artifactDir artifacts \
  --merge "com.example.IndustrialDashboard=1.0.0" \
  --update-config dashboard-configuration.json
```

O `dashboard-configuration.json` arquivo contém o seguinte documento JSON.

```
{
  "com.example.IndustrialDashboard": {
    "MERGE": {
      "name": "Factory 2A",
      "network": {
        "useHttps": false,
```

```
        "port": {
          "http": 8080
        }
      },
      "tags": [
        "/boiler/1/temperature",
        "/boiler/1/pressure",
        "/boiler/2/temperature",
        "/boiler/2/pressure"
      ]
    }
  }
}
```

Após essa atualização, o componente do painel tem a seguinte configuração.

```
{
  "name": "Factory 2A",
  "mode": "REQUEST",
  "network": {
    "useHttps": false,
    "port": {
      "http": 8080,
      "https": 443
    }
  },
  "tags": [
    "/boiler/1/temperature",
    "/boiler/1/pressure",
    "/boiler/2/temperature",
    "/boiler/2/pressure"
  ]
}
```

Example Exemplo 2: redefinir e mesclar atualizações

Em seguida, você cria uma implantação que aplica a seguinte atualização de configuração, que especifica uma atualização de redefinição e uma atualização de mesclagem. Essas atualizações especificam a exibição do painel na porta HTTPS padrão com dados de diferentes caldeiras. Essas atualizações modificam a configuração resultante das atualizações de configuração no exemplo anterior.

Console

Redefinir caminhos

```
[
  "/network/useHttps",
  "/tags"
]
```

Configuração para mesclar

```
{
  "tags": [
    "/boiler/3/temperature",
    "/boiler/3/pressure",
    "/boiler/4/temperature",
    "/boiler/4/pressure"
  ]
}
```

AWS CLI

O comando a seguir cria uma implantação em um dispositivo principal.

```
aws greengrassv2 create-deployment --cli-input-json file://dashboard-
deployment2.json
```

O `dashboard-deployment2.json` arquivo contém o seguinte documento JSON.

```
{
  "targetArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
  "deploymentName": "Deployment for MyGreengrassCore",
  "components": {
    "com.example.IndustrialDashboard": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "reset": [
          "/network/useHttps",
          "/tags"
        ],
```

```
        "merge": "{\"tags\":[\"/boiler/3/temperature\",\"/boiler/3/pressure\",\"/
boiler/4/temperature\",\"/boiler/4/pressure\"]}"
      }
    }
  }
}
```

Greengrass CLI

O comando CLI do Greengrass a seguir cria uma implantação local em um dispositivo principal.

```
sudo greengrass-cli deployment create \
  --recipeDir recipes \
  --artifactDir artifacts \
  --merge "com.example.IndustrialDashboard=1.0.0" \
  --update-config dashboard-configuration2.json
```

O `dashboard-configuration2.json` arquivo contém o seguinte documento JSON.

```
{
  "com.example.IndustrialDashboard": {
    "RESET": [
      "/network/useHttps",
      "/tags"
    ],
    "MERGE": {
      "tags": [
        "/boiler/3/temperature",
        "/boiler/3/pressure",
        "/boiler/4/temperature",
        "/boiler/4/pressure"
      ]
    }
  }
}
```

Após essa atualização, o componente do painel tem a seguinte configuração.

```
{
  "name": "Factory 2A",
  "mode": "REQUEST",
```

```
  "network": {
    "useHttps": true,
    "port": {
      "http": 8080,
      "https": 443
    }
  },
  "tags": [
    "/boiler/3/temperature",
    "/boiler/3/pressure",
    "/boiler/4/temperature",
    "/boiler/4/pressure",
  ]
}
```

Crie subimplantações

> **Note**
>
> O recurso de subimplantação está disponível no Greengrass nucleus versão 2.9.0 e posterior. Não é possível implantar uma configuração em uma subimplantação com versões anteriores de componentes do Greengrass nucleus.

Uma subimplantação é uma implantação que tem como alvo um subconjunto menor de dispositivos em uma implantação principal. Você pode usar subimplantações para implantar uma configuração em um subconjunto menor de dispositivos. Você também pode criar subimplantações para tentar novamente uma implantação principal malsucedida quando um ou mais dispositivos nessa implantação principal falharem. Com esse recurso, você pode selecionar dispositivos que falharam nessa implantação principal e criar uma subimplantação para testar as configurações até que a subimplantação seja bem-sucedida. Depois que a subimplantação for bem-sucedida, você poderá reimplantar essa configuração na implantação principal.

Siga as etapas desta seção para criar uma subimplantação e verificar seu status. Para obter mais informações sobre como criar implantações, consulte Criar implantações.

Para criar uma subimplantação () AWS CLI

1. Execute o comando a seguir para recuperar as implantações mais recentes de um grupo de coisas. Substitua o ARN no comando pelo ARN do grupo de coisas a ser consultado. --

`history-filter`Defina como **`LATEST_ONLY`** para ver a implantação mais recente desse grupo de coisas.

```
aws greengrassv2 list-deployments --target-arn arn:aws:iot:region:account-
id:thinggroup/thingGroupName --history-filter LATEST_ONLY
```

2. Copie o `deploymentId` da resposta ao list-deployments comando para usar na próxima etapa.
3. Execute o comando a seguir para recuperar o status de uma implantação. *`deploymentId`*Substitua pelo ID da implantação a ser consultada.

```
aws greengrassv2 get-deployment --deployment-id deploymentId
```

4. Copie o `iotJobId` da resposta ao get-deployment comando a ser usado na etapa a seguir.
5. Execute o comando a seguir para recuperar a lista de execuções de trabalhos para o trabalho especificado. Substitua *`JobID`* pelo `iotJobId` da etapa anterior. *`Substitua o status pelo status que você deseja filtrar.`* Você pode filtrar os resultados com os seguintes status:

   - `QUEUED`
   - `IN_PROGRESS`
   - `SUCCEEDED`
   - `FAILED`
   - `TIMED_OUT`
   - `REJECTED`
   - `REMOVED`
   - `CANCELED`

```
aws iot list-job-executions-for-job --job-id jobID --status status
```

6. Crie um novo grupo de AWS IoT coisas ou use um grupo de coisas existente para sua subimplantação. Em seguida, adicione AWS IoT algo a esse grupo de coisas. Você usa grupos de coisas para gerenciar frotas de dispositivos principais do Greengrass. Ao implantar componentes de software em seus dispositivos, você pode segmentar dispositivos individuais ou grupos de dispositivos. Você pode adicionar um dispositivo a um grupo de coisas com uma implantação ativa do Greengrass. Depois de adicionado, você pode implantar os componentes de software desse grupo de coisas nesse dispositivo.

Para criar um novo grupo de coisas e adicionar seus dispositivos a ele, faça o seguinte:

a. Crie um grupo de AWS IoT coisas. *MyGreengrassCoreGroup*Substitua pelo nome do novo grupo de coisas. Você não pode usar dois pontos (:) no nome de um grupo de coisas.

> **Note**
>
> Se um grupo de coisas para uma subimplantação for usado com um`parentTargetArn`, ele não poderá ser reutilizado com uma frota principal diferente. Se um grupo de coisas já tiver sido usado para criar uma subimplantação para outra frota, a API retornará um erro.

```
aws iot create-thing-group --thing-group-name MyGreengrassCoreGroup
```

Se a solicitação for bem-sucedida, a resposta será semelhante ao exemplo a seguir:

```
{
  "thingGroupName": "MyGreengrassCoreGroup",
  "thingGroupArn": "arn:aws:iot:us-
west-2:123456789012:thinggroup/MyGreengrassCoreGroup",
  "thingGroupId": "4df721e1-ff9f-4f97-92dd-02db4e3f03aa"
}
```

b. Adicione um núcleo provisionado do Greengrass ao seu grupo de coisas. Execute o comando a seguir com esses parâmetros:

- *MyGreengrassCore*Substitua pelo nome do seu núcleo provisionado do Greengrass.
- *MyGreengrassCoreGroup*Substitua pelo nome do seu grupo de coisas.

```
aws iot add-thing-to-thing-group --thing-name MyGreengrassCore --thing-group-
name MyGreengrassCoreGroup
```

O comando não terá nenhuma saída se a solicitação for bem-sucedida.

7. Crie um arquivo chamado `edeployment.json`, em seguida, copie o seguinte objeto JSON para o arquivo. Substitua *targetArn* pelo ARN do grupo de coisas a ser direcionado para AWS

IoT a subimplantação. Um alvo de subimplantação só pode ser um grupo de coisas. Os ARNs do Thing Group têm o seguinte formato:

- Grupo Thing — `arn:aws:iot:region:account-id:thinggroup/thingGroupName`

```
{
  "targetArn": "targetArn"
}
```

8. Execute o comando a seguir novamente para obter os detalhes da implantação original. Esses detalhes incluem metadados, componentes e configuração do trabalho. Substitua *deploymentID* pela ID de. Step 1 Você pode usar essa configuração de implantação para configurar sua subimplantação e fazer alterações conforme necessário.

```
aws greengrassv2 get-deployment --deployment-id deploymentId
```

A resposta contém os detalhes da implantação. Copie qualquer um dos seguintes pares de valores-chave da resposta do get-deployment comando para. `deployment.json` Você pode alterar esses valores para a subimplantação. Para obter mais informações sobre os detalhes desse comando, consulte GetDeployment.

- `components`— Os componentes da implantação. Para desinstalar um componente, remova-o desse objeto.
- `deploymentName`— O nome da implantação.
- `deploymentPolicies`— As políticas de implantação.
- `iotJobConfiguration`— A configuração do trabalho da implantação.
- `parentTargetArn`— O alvo da implantação principal.
- `tags`— As tags da implantação.

9. Execute o comando a seguir para criar a subimplantação a partir de`deployment.json`. Substitua *subdeploymentName por um nome* para a subimplantação.

```
aws greengrassv2 create-deployment --deployment-name subdeploymentName --cli-input-
json file://deployment.json
```

A resposta inclui uma `deploymentId` que identifica essa subimplantação. Você pode usar o ID de implantação para verificar o status da implantação. Para obter mais informações, consulte Verificar o status da implantação.

10. Se a subimplantação for bem-sucedida, você poderá usar sua configuração para revisar a implantação principal. Copie o `deployment.json` que você usou na etapa anterior. Substitua o `targetArn` no arquivo JSON pelo ARN da implantação principal e execute o comando a seguir para criar a implantação principal usando essa nova configuração.

    > **Note**
    >
    > Se você criar uma nova revisão de implantação da frota principal, ela substituirá todas as revisões e subimplantações dessa implantação principal. Para obter mais informações, consulte Revisar implantações.

    ```
    aws greengrassv2 create-deployment --cli-input-json file://deployment.json
    ```

    A resposta inclui uma `deploymentId` que identifica essa implantação. Você pode usar o ID de implantação para verificar o status da implantação. Para ter mais informações, consulte Verificar status da implantação.

Revise as implantações

Cada item ou grupo de itens de destino pode ter uma implantação ativa por vez. Quando você cria uma implantação para um destino que já tem uma implantação, os componentes de software na nova implantação substituem os da implantação anterior. Se a nova implantação não definir um componente definido pela implantação anterior, o software AWS IoT Greengrass principal removerá esse componente dos dispositivos principais de destino. Você pode revisar uma implantação existente para não remover os componentes executados nos dispositivos principais de uma implantação anterior em um destino.

Para revisar uma implantação, você cria uma implantação que começa com os mesmos componentes e configurações existentes em uma implantação anterior. Você usa a CreateDeploymentoperação, que é a mesma operação usada para criar implantações.

Para revisar uma implantação () AWS CLI

1. Execute o comando a seguir para listar as implantações para o destino de implantação. Substitua *targetArn* pelo ARN da coisa ou grupo de coisas de destino. AWS IoT

   ```
   aws greengrassv2 list-deployments --target-arn targetArn
   ```

   A resposta contém uma lista com a implantação mais recente do destino. Copie o `deploymentId` da resposta para usar na próxima etapa.

   > **Note**
   >
   > Você também pode revisar uma implantação diferente da revisão mais recente do destino. Especifique o `--history-filter ALL` argumento para listar todas as implantações do destino. Em seguida, copie a ID da implantação que você deseja revisar.

2. Execute o comando a seguir para obter os detalhes da implantação. Esses detalhes incluem metadados, componentes e configuração do trabalho. Substitua *deploymentID* pelo ID da etapa anterior.

   ```
   aws greengrassv2 get-deployment --deployment-id deploymentId
   ```

   A resposta contém os detalhes da implantação.

3. Crie um arquivo chamado `deployment.json` e copie a resposta do comando anterior no arquivo.

4. Remova os seguintes pares de chave-valor do objeto JSON em `deployment.json`:

   - `deploymentId`
   - `revisionId`
   - `iotJobId`
   - `iotJobArn`
   - `creationTimestamp`
   - `isLatestForTarget`
   - `deploymentStatus`

A CreateDeploymentoperação espera uma carga útil com a seguinte estrutura.

```
{
  "targetArn": "String",
  "components": Map of components,
  "deploymentPolicies": DeploymentPolicies,
  "iotJobConfiguration": DeploymentIoTJobConfiguration,
  "tags": Map of tags
}
```

5. Em `deployment.json` proceda de uma das seguintes maneiras:
   - Altere o nome da implantação (`deploymentName`).
   - Altere os componentes da implantação (`components`).
   - Altere as políticas da implantação (`deploymentPolicies`).
   - Altere a configuração do trabalho da implantação (`iotJobConfiguration`).
   - Altere as tags da implantação (`tags`).

   Para obter mais informações sobre como definir esses detalhes de implantação, consulteCriar implantações.
6. Execute o comando a seguir para criar a implantação a partir de`deployment.json`.

   ```
   aws greengrassv2 create-deployment --cli-input-json file://deployment.json
   ```

   A resposta inclui uma `deploymentId` que identifica essa implantação. Você pode usar o ID de implantação para verificar o status da implantação. Para ter mais informações, consulte Verificar status da implantação.

Cancelar implantações

Você pode cancelar uma implantação ativa para impedir que seus componentes de software sejam instalados nos dispositivosAWS IoT Greengrass principais. Se você cancelar uma implantação direcionada a um grupo de coisas, os dispositivos principais que você adicionar ao grupo não receberão essa implantação contínua. Se um dispositivo principal já executa a implantação, você não alterará os componentes desse dispositivo ao cancelar a implantação. Você deve criar uma nova implantação ou revisar a implantação para modificar os componentes que são executados nos dispositivos principais que receberam a implantação cancelada.

Para cancelar uma implantação (AWS CLI)

1. Execute o comando a seguir para encontrar o ID da revisão de implantação mais recente para um destino. A revisão mais recente é a única implantação que pode estar ativa para um destino, porque as implantações anteriores são canceladas quando você cria uma nova revisão. Substitua da *targetArn* pelo ARN ou grupo deAWS IoT itens da de destino.

   ```
   aws greengrassv2 list-deployments --target-arn targetArn
   ```

   A resposta contém uma lista com a implantação mais recente para o alvo. Copie o item`deploymentId` da resposta para usar na próxima etapa.

2. Execute o seguinte comando para cancelar a implantação. *Substitua deploymentId* pelo ID da etapa anterior.

   ```
   aws greengrassv2 cancel-deployment --deployment-id deploymentId
   ```

   Se a operação ocorrer com êxito, o status da implantação será alterado para`CANCELED`.

Verificar status da implantação

Você pode verificar o status de uma implantação na qual foi criadaAWS IoT Greengrass. Você também pode verificar o status dosAWS IoT trabalhos que implementam a implantação em cada dispositivo principal. Enquanto uma implantação está ativa, o status doAWS IoT trabalho é`IN_PROGRESS`. Depois de criar uma nova revisão de uma implantação, o status daAWS IoT tarefa da revisão anterior é alterado para`CANCELLED`.

Tópicos

- Verificar status da implantação
- Verifique o status de implantação do dispositivo

Verificar status da implantação

Você pode verificar o status de uma implantação que você identifica por seu destino ou seu ID.

Para verificar o status da implantação por destino (AWS CLI)

- Execute o comando a seguir para recuperar o status da implantação mais recente de um de destino. Substitua *targetArn* pelo nome de recurso da Amazon (ARN) daAWS IoT coisa ou do grupo de coisas de destino da implantação.

  ```
  aws greengrassv2 list-deployments --target-arn targetArn
  ```

  A resposta contém uma lista com a implantação mais recente para o alvo. Esse objeto de implantação inclui o status da implantação.

Para verificar o status da implantação por ID (AWS CLI)

- Execute o comando a seguir para recuperar o status de uma implantação. Substitua *deploymentId* pelo ID da implantação a ser consultada.

  ```
  aws greengrassv2 get-deployment --deployment-id deploymentId
  ```

  A resposta contém o status da implantação.

Verifique o status de implantação do dispositivo

Você pode verificar o status de um trabalho de implantação que se aplica a um dispositivo de núcleo individual. Você também pode verificar o status de uma tarefa de implantação para a implantação de um grupo de coisas.

Para verificar o status do trabalho de implantação de um dispositivo principal (AWS CLI)

- Execute o comando a seguir para recuperar o status de todas as tarefas de implantação de um dispositivo principal. *coreDeviceName*Substitua pelo nome do dispositivo principal a ser consultado.

  ```
  aws greengrassv2 list-effective-deployments --core-device-thing-name coreDeviceName
  ```

  A resposta contém a lista de tarefas de implantação do dispositivo principal. Você pode identificar o trabalho para uma implantação pelo trabalho`deploymentId` out`argetArn`. Cada tarefa de implantação contém o status da tarefa no dispositivo principal.

Para verificar o status de implantação de um grupo de coisas (AWS CLI)

1. Execute o comando a seguir para recuperar o ID de uma implantação existente. Substitua *targetArn* pelo ARN do grupo de itens de destino.

   ```
   aws greengrassv2 list-deployments --target-arn targetArn
   ```

   A resposta contém uma lista com a implantação mais recente para o alvo. Copie o`deploymentId` da resposta para usar na próxima etapa.

   > **Note**
   >
   > Você também pode listar uma implantação diferente da implantação mais recente para o destino. Especifique o`--history-filter ALL` argumento para listar todas as implantações para o destino. Em seguida, copie o ID da implantação da qual você deseja verificar o status.

2. Execute o comando a seguir para obter os detalhes da implantação. *Substitua TargetArc* pelo ID da etapa anterior.

   ```
   aws greengrassv2 get-deployment --deployment-id deploymentId
   ```

   A resposta contém informações sobre a implantação. Copie o`iotJobId` da resposta para usar na etapa a seguir.

3. Execute o comando a seguir para descrever a execução do trabalho de um dispositivo principal para a implantação. Substitua *iotJobId*e *coreDeviceThingnomeie* pelo ID do trabalho da etapa anterior e pelo dispositivo principal do qual você deseja verificar o status.

   ```
   aws iot describe-job-execution --job-id iotJobId --thing-name coreDeviceThingName
   ```

   A resposta contém o status da execução do trabalho de implantação do dispositivo principal e detalhes sobre o status. O`detailsMap` contém as seguintes informações:

   - `detailed-deployment-status`— Estado do resultado da implantação, que pode ser um dos seguintes valores:
     - `SUCCESSFUL`— A implantação foi bem-sucedida.

- `FAILED_NO_STATE_CHANGE`— A implantação falhou enquanto o dispositivo principal se preparava para aplicar a implantação.
- `FAILED_ROLLBACK_NOT_REQUESTED`— A implantação falhou e a implantação não especificou a reversão para uma configuração de trabalho anterior, portanto, o dispositivo principal pode não estar funcionando corretamente.
- `FAILED_ROLLBACK_COMPLETE`— A implantação falhou e o dispositivo principal foi revertido com êxito para uma configuração de trabalho anterior.
- `FAILED_UNABLE_TO_ROLLBACK`— A implantação falhou e o dispositivo principal não conseguiu reverter para uma configuração de trabalho anterior, portanto, o dispositivo principal pode não estar funcionando corretamente.

  Se a implantação falhar, verifique o`deployment-failure-cause` valor e os arquivos de log do dispositivo principal para identificar o problema. Para obter mais informações sobre como acessar os arquivos de log do dispositivo principal, consulteMonitore AWS IoT Greengrass os registros.

- `deployment-failure-cause`— Uma mensagem de erro que fornece detalhes adicionais sobre por que a execução do trabalho falhou.

A resposta é semelhante ao seguinte exemplo.


```
{
  "execution": {
    "jobId": "2cc2698a-5175-48bb-adf2-1dd345606ebd",
    "status": "FAILED",
    "statusDetails": {
      "detailsMap": {
        "deployment-failure-cause": "No local or cloud component version satisfies
 the requirements. Check whether the version constraints conflict and that
 the component exists in your Conta da AWS with a version that matches the
 version constraints. If the version constraints conflict, revise deployments
 to resolve the conflict. Component com.example.HelloWorld version constraints:
 LOCAL_DEPLOYMENT requires =1.0.0, thinggroup/MyGreengrassCoreGroup requires
 =1.0.1.",
        "detailed-deployment-status": "FAILED_NO_STATE_CHANGE"
      }
    },
    "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
    "queuedAt": "2022-02-15T14:45:53.098000-08:00",
    "startedAt": "2022-02-15T14:46:05.670000-08:00",
```

```
    "lastUpdatedAt": "2022-02-15T14:46:20.892000-08:00",
    "executionNumber": 1,
    "versionNumber": 3
  }
}
```

Registrar em log e monitorar no AWS IoT Greengrass

O monitoramento é uma parte importante da manutenção da confiabilidade, da disponibilidade e do desempenho do AWS IoT Greengrass e de soluções da AWS. Você deve coletar dados de monitoramento de todas as partes de sua solução da AWS para facilitar a depuração de uma falha multipontos, caso ela ocorra. Antes de começar a monitorar o AWS IoT Greengrass, crie um plano de monitoramento que inclua as respostas para as seguintes perguntas:

- Quais são seus objetivos de monitoramento?
- Quais recursos você vai monitorar?
- Com que frequência você vai monitorar esses recursos?
- Quais ferramentas de monitoramento você usará?
- Quem realizará o monitoramento das tarefas?
- Quem deve ser notificado quando algo der errado?

Tópicos

Ferramentas de monitoramento

A AWS fornece ferramentas que você pode usar para monitorar o AWS IoT Greengrass. Você pode configurar algumas dessas ferramentas para que façam o monitoramento para você. Algumas das ferramentas exigem intervenção manual. Recomendamos que as tarefas de monitoramento sejam automatizadas ao máximo possível.

É possível usar as seguintes ferramentas de monitoramento automatizadas para supervisionar o AWS IoT Greengrass e relatar os seguintes problemas:

- Amazon CloudWatch Logs — Monitore, armazene e acesse seus arquivos de log de AWS CloudTrail ou de outras fontes. Para obter mais informações, consulte Monitoramento de arquivos de log no Guia CloudWatch do usuário da Amazon.
- AWS CloudTrailMonitoramento de registros — compartilhe arquivos de log entre contas, monitore arquivos de CloudTrail log em tempo real enviando-os para o CloudWatch Logs, grave aplicativos de processamento de log em Java e valide se seus arquivos de log não foram alterados após a entrega. CloudTrail Para obter mais informações, consulte Como trabalhar com arquivos de CloudTrail log no Guia AWS CloudTrail do usuário.
- Telemetria da integridade do sistema Greengrass: inscreva-se para receber dados de telemetria enviados a partir do núcleo do Greengrass. Para ter mais informações, consulte the section called “Colete dados de telemetria de integridade do sistema”.
- Notificações de saúde do dispositivo Crie eventos usando EventBridge a Amazon para receber atualizações de status sobre implantações e componentes. Para ter mais informações, consulte Receba notificações de status de integridade de componentes e implantações.
- Serviço de status da frota — Use as operações da API de status da frota para verificar o status dos dispositivos principais e seus componentes do Greengrass. Você também pode visualizar as informações sobre o status da frota no AWS IoT Greengrass console. Para ter mais informações, consulte Verifique o status do dispositivo principal do Greengrass.

Monitore AWS IoT Greengrass os registros

O AWS IoT Greengrass consiste no serviço em nuvem e no software do AWS IoT Greengrass Core. O software AWS IoT Greengrass principal pode gravar registros no Amazon CloudWatch Logs e no sistema de arquivos local do dispositivo principal. Os componentes do Greengrass que são executados no dispositivo principal também podem gravar registros no Logs e no sistema de arquivos local. CloudWatch Você pode usar os logs para monitorar eventos e solucionar problemas. Todas as entradas de log do AWS IoT Greengrass incluem um time stamp, nível de log e informações sobre o evento.

Por padrão, o software AWS IoT Greengrass Core grava registros somente no sistema de arquivos local. Você pode visualizar os registros do sistema de arquivos em tempo real, para poder depurar os componentes do Greengrass que você desenvolve e implanta. Você também pode configurar um dispositivo principal para gravar registros no CloudWatch Logs, para que você possa solucionar problemas do dispositivo principal sem acessar o sistema de arquivos local. Para ter mais informações, consulte Habilitar o registro em CloudWatch registros.

Tópicos

Acesse os registros do sistema de arquivos

O software AWS IoT Greengrass Core armazena os registros na *`/greengrass/v2`*`/logs` pasta em um dispositivo principal, onde *`/greengrass/v2`* está o caminho para a pasta AWS IoT Greengrass raiz. A pasta de registros tem a seguinte estrutura.

```
/greengrass/v2
### logs
    ### greengrass.log
    ### greengrass_2021_09_14_15_0.log
    ### ComponentName.log
    ### ComponentName_2021_09_14_15_0.log
    ### main.log
```

- `greengrass.log`— O arquivo de log do software AWS IoT Greengrass principal. Use esse arquivo de log para visualizar informações em tempo real sobre componentes e implantações. Esse arquivo de log inclui registros do núcleo Greengrass, que é o núcleo do software AWS IoT Greengrass Core, e componentes de plug-in, como gerenciador de registros e gerenciador secreto.
- *`ComponentName`*`.log`— Arquivos de log do componente Greengrass. Use arquivos de log de componentes para visualizar informações em tempo real sobre um componente do Greengrass que é executado no dispositivo principal. Os componentes genéricos e os componentes do Lambda gravam a saída padrão (stdout) e o erro padrão (stderr) nesses arquivos de log.
- `main.log`— O arquivo de log do `main` serviço que manipula os ciclos de vida dos componentes. Esse arquivo de log estará sempre vazio.

Para obter mais informações sobre as diferenças entre componentes de plug-in, genéricos e Lambda, consulte. Tipos de componentes

As seguintes considerações se aplicam ao usar logs de sistema de arquivos:

- Permissões de usuário root

  É necessário ter permissões raiz para ler logs do AWS IoT Greengrass no sistema de arquivos.

- Rotação do arquivo de log

  O software AWS IoT Greengrass Core gira os arquivos de log a cada hora ou quando eles excedem o limite de tamanho do arquivo. Os arquivos de log rotacionados contêm um carimbo de data/hora no nome do arquivo. Por exemplo, um arquivo de log AWS IoT Greengrass do software Core rotacionado pode ser nomeado`greengrass_2021_09_14_15_0.log`. O limite de tamanho de arquivo padrão é 1.024 KB (1 MB). Você pode configurar o limite de tamanho do arquivo no componente do núcleo do Greengrass.

- Exclusão do arquivo de log

  O software AWS IoT Greengrass Core limpa os arquivos de log anteriores quando o tamanho dos arquivos de log do software AWS IoT Greengrass Core ou dos arquivos de log dos componentes do Greengrass, incluindo arquivos de log rotacionados, excede o limite de espaço em disco. O limite de espaço em disco padrão para o log do software AWS IoT Greengrass Core e cada registro de componente é de 10.240 KB (10 MB). Você pode configurar o limite de espaço em disco do log do software AWS IoT Greengrass Core no componente nucleus do Greengrass ou no componente do gerenciador de registros. Você pode configurar o limite de espaço em disco de registro de cada componente no componente do gerenciador de registros.

Para visualizar o arquivo de log do software AWS IoT Greengrass principal

- Execute o comando a seguir para visualizar o arquivo de log em tempo real. */greengrass/v2*Substitua pelo caminho para a pasta AWS IoT Greengrass raiz.

  Linux or Unix

  ```
  sudo tail -f /greengrass/v2/logs/greengrass.log
  ```

  Windows Command Prompt (CMD)

  ```
  type C:\greengrass\v2\logs\com.example.HelloWorld.log
  ```

O `type` comando grava o conteúdo do arquivo no terminal. Execute esse comando várias vezes para observar as alterações no arquivo.

PowerShell

```
gc C:\greengrass\v2\logs\greengrass.log -Tail 10 -Wait
```

Para visualizar o arquivo de log de um componente

- Execute o comando a seguir para visualizar o arquivo de log em tempo real. Substitua */greengrass/v2* ou *C:\greengrass\v2* pelo caminho para a pasta AWS IoT Greengrass raiz e substitua *com.example. HelloWorld*com o nome do componente.

  Linux or Unix

  ```
  sudo tail -f /greengrass/v2/logs/com.example.HelloWorld.log
  ```

  PowerShell

  ```
  gc C:\greengrass\v2\logs\com.example.HelloWorld.log -Tail 10 -Wait
  ```

Você também pode usar o `logs` comando da CLI do Greengrass para analisar os registros do Greengrass em um dispositivo principal. Para usar o `logs` comando, você deve configurar o núcleo do Greengrass para gerar arquivos de log no formato JSON. Para obter mais informações, consulte logs e Interface de linha de comando do Greengrass.

CloudWatch Registros de acesso

Você pode implantar o componente do gerenciador de registros para configurar o dispositivo principal para gravar CloudWatch nos registros. Para ter mais informações, consulte Habilitar o registro em CloudWatch registros. Em seguida, você pode visualizar os registros na página Logs do CloudWatch console da Amazon ou usando a API CloudWatch Logs.

Nome do grupo de registros

```
/aws/greengrass/componentType/region/componentName
```

O nome do grupo de registros usa as seguintes variáveis:

- `componentType`— O tipo do componente, que pode ser um dos seguintes:
  - `GreengrassSystemComponent`— Esse grupo de registros inclui registros para o núcleo e os componentes do plug-in, que são executados na mesma JVM que o núcleo do Greengrass. O componente faz parte do núcleo do Greengrass.
  - `UserComponent`— Esse grupo de registros inclui registros de componentes genéricos, componentes Lambda e outros aplicativos no dispositivo. O componente não faz parte do núcleo do Greengrass.

  Para ter mais informações, consulte Tipos de componentes.
- `region`— A AWS região que o dispositivo principal usa.
- `componentName`— O nome do componente. Para registros do sistema, esse valor é`System`.

Nome do fluxo de log

```
/date/thing/thingName
```

O nome do fluxo de log usa as seguintes variáveis:

- `date`— A data do registro, como`2020/12/15`. O componente do gerenciador de registros usa o `yyyy/MM/dd` formato.
- `thingName`— O nome do dispositivo principal.

> **Note**
>
> Se o nome de uma coisa contiver dois pontos (`:`), o gerenciador de registros substituirá os dois pontos por um sinal de adição (). +

As considerações a seguir se aplicam quando você usa o componente gerenciador de CloudWatch registros para gravar em registros:

- Atrasos no registro

> **Note**
>
> Recomendamos que você atualize para a versão 2.3.0 do gerenciador de registros, que reduz os atrasos nos arquivos de log rotacionados e ativos. Ao atualizar para o log

manager 2.3.0, recomendamos que você também atualize para o Greengrass nucleus 2.9.1.

A versão 2.2.8 (e anterior) do componente gerenciador de registros processa e carrega registros somente de arquivos de log rotacionados. Por padrão, o software AWS IoT Greengrass Core gira os arquivos de log a cada hora ou depois de atingirem 1.024 KB. Como resultado, o componente do gerenciador de registros carrega registros somente depois que o software AWS IoT Greengrass Core ou um componente do Greengrass grava mais de 1.024 KB de registros. Você pode configurar um limite menor de tamanho de arquivo de log para fazer com que os arquivos de log girem com mais frequência. Isso faz com que o componente do gerenciador de registros faça upload de registros para o CloudWatch Logs com mais frequência.

A versão 2.3.0 (e posterior) do componente gerenciador de registros processa e carrega todos os registros. Quando você grava um novo registro, a versão 2.3.0 (e posterior) do gerenciador de registros processa e carrega diretamente esse arquivo de log ativo em vez de esperar que ele seja rotacionado. Isso significa que você pode visualizar o novo registro em 5 minutos ou menos.

O componente gerenciador de registros carrega novos registros periodicamente. Por padrão, o componente do gerenciador de registros carrega novos registros a cada 5 minutos. Você pode configurar um intervalo de upload menor, para que o componente do gerenciador de registros faça o upload dos CloudWatch registros para o Logs com mais frequência configurando o. `periodicUploadIntervalSec` Para obter mais informações sobre como configurar esse intervalo periódico, consulte Configuração.

Os registros podem ser carregados quase em tempo real a partir do mesmo sistema de arquivos do Greengrass. Se você precisar observar os registros em tempo real, considere usar os registros do sistema de arquivos.

> **Note**
>
> Se você estiver usando sistemas de arquivos diferentes para gravar registros, o gerenciador de registros retornará ao comportamento nas versões 2.2.8 e anteriores do componente do gerenciador de registros. Para obter informações sobre como acessar registros do sistema de arquivos, consulte Acessar registros do sistema de arquivos.

- Inclinação do relógio

O componente do gerenciador de registros usa o processo de assinatura padrão do Signature versão 4 para criar solicitações de API para o CloudWatch Logs. Se a hora do sistema em um dispositivo principal estiver fora de sincronia por mais de 15 minutos, o CloudWatch Logs rejeitará as solicitações. Para obter mais informações, consulte Processo de assinatura do Signature versão 4 na Referência geral da AWS.

Acesse os registros de serviços do sistema

Se você configurar o software AWS IoT Greengrass Core como um serviço do sistema, poderá visualizar os registros do serviço do sistema para solucionar problemas, como a falha na inicialização do software.

Para visualizar os registros de serviços do sistema (CLI)

1. Execute o comando a seguir para visualizar os registros de serviços do sistema de software AWS IoT Greengrass principal.

   Linux or Unix (systemd)

   ```
   sudo journalctl -u greengrass.service
   ```

   Windows Command Prompt (CMD)

   ```
   type C:\greengrass\v2\logs\greengrass.wrapper.log
   ```

   PowerShell

   ```
   gc C:\greengrass\v2\logs\greengrass.wrapper.log
   ```

2. Em dispositivos Windows, o software AWS IoT Greengrass Core cria um arquivo de log separado para erros de serviço do sistema. Execute o comando a seguir para visualizar os registros de erros do serviço do sistema.

   Windows Command Prompt (CMD)

   ```
   type C:\greengrass\v2\logs\greengrass.err.log
   ```

PowerShell

```
gc C:\greengrass\v2\logs\greengrass.err.log
```

Em dispositivos Windows, você também pode usar o aplicativo Visualizador de Eventos para visualizar os registros de serviços do sistema.

Para visualizar os registros de serviços do Windows (Visualizador de eventos)

1. Abra o aplicativo Visualizador de Eventos.
2. Selecione Registros do Windows para expandi-lo.
3. Escolha Aplicativo para visualizar os registros do serviço do aplicativo.
4. Encontre e abra registros de eventos cuja fonte é greengrass.

Habilitar o registro em CloudWatch registros

Você pode implantar o componente do gerenciador de registros para configurar um dispositivo principal para gravar registros no CloudWatch Logs. Você pode ativar CloudWatch os registros para os registros do software AWS IoT Greengrass principal e pode habilitar CloudWatch os registros para componentes específicos do Greengrass.

> **Note**
>
> A função de troca de tokens do dispositivo principal do Greengrass deve permitir que o dispositivo principal grave nos CloudWatch registros, conforme mostrado no exemplo de política do IAM a seguir. Se você instalou o software AWS IoT Greengrass Core com provisionamento automático de recursos, seu dispositivo principal tem essas permissões.
>
> ```
> {
>   "Version": "2012-10-17",
>   "Statement": [
>     {
>       "Action": [
>         "logs:CreateLogGroup",
>         "logs:CreateLogStream",
>         "logs:PutLogEvents",
>         "logs:DescribeLogStreams"
> ```

```
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:logs:*:*:*"
    }
  ]
}
```

Para configurar um dispositivo principal para gravar registros AWS IoT Greengrass do software principal no CloudWatch Logs, crie uma implantação que especifique uma atualização de configuração definida `uploadToCloudWatch` `true` para o `aws.greengrass.LogManager` componente. AWS IoT Greengrass Os principais registros do software incluem registros do núcleo do Greengrass e dos componentes do plug-in.

```
{
  "logsUploaderConfiguration": {
    "systemLogsConfiguration": {
      "uploadToCloudWatch": "true"
    }
  }
}
```

Para configurar um dispositivo principal para gravar os registros de um componente do Greengrass em Logs, crie uma implantação que especifique uma atualização de configuração que adicione o componente à lista de configurações de CloudWatch registro de componentes. Quando você adiciona um componente a essa lista, o componente do gerenciador de registros grava seus registros em CloudWatch Logs. Os registros de componentes incluem registros de componentes genéricos e componentes Lambda.

```
{
  "logsUploaderConfiguration": {
    "componentLogsConfigurationMap": {
      "com.example.HelloWorld": {

      }
    }
  }
}
```

Ao implantar o componente do gerenciador de registros, você também pode configurar os limites de espaço em disco e se o dispositivo principal excluirá os arquivos de log depois de gravá-los no CloudWatch Logs. Para ter mais informações, consulte Configurar o registro em log para o AWS IoT Greengrass.

Configurar o registro em log para o AWS IoT Greengrass

Você pode configurar as seguintes opções para personalizar o registro nos dispositivos principais do Greengrass. Para configurar essas opções, crie uma implantação que especifique uma atualização de configuração para os componentes do Greengrass nucleus ou do gerenciador de registros.

- Gravando registros em CloudWatch registros

  Para solucionar problemas remotamente nos dispositivos principais, você pode configurar os dispositivos principais para gravar registros de software e componentes AWS IoT Greengrass principais nos registros. CloudWatch Para fazer isso, implante e configure o componente do gerenciador de registros. Para ter mais informações, consulte Habilitar o registro em CloudWatch registros.

- Excluindo arquivos de log enviados

  Para reduzir o uso do espaço em disco, você pode configurar os dispositivos principais para excluir arquivos de log depois de gravá-los em CloudWatch Logs. Para obter mais informações, consulte o `deleteLogFileAfterCloudUpload` parâmetro do componente gerenciador de registros, que você pode especificar para os registros AWS IoT Greengrassdo software principal e os registros de componentes.

- Registrar limites de espaço em disco

  Para limitar o uso do espaço em disco, você pode configurar o espaço máximo em disco para cada log, incluindo seus arquivos de log rotacionados, em um dispositivo principal. Por exemplo, você pode configurar o espaço máximo combinado em disco para `greengrass.log` `greengrass.log` arquivos rotacionados. Para obter mais informações, consulte o parâmetro do componente Greengrass nucleus e o `logging.totalLogsSizeKB` parâmetro do componente gerenciador de registros, que você pode especificar para os registros do `diskSpaceLimit`software AWS IoT Greengrass principal e os registros de componentes.

- Limite de tamanho do arquivo de log

  Você pode configurar o tamanho máximo de arquivo para cada arquivo de log. Depois que um arquivo de log excede esse limite de tamanho de arquivo, o software AWS IoT Greengrass

Core cria um novo arquivo de log. A versão 2.28 (e anterior) do componente gerenciador de registros grava somente arquivos de log rotacionados em CloudWatch Logs, então você pode especificar um limite menor de tamanho de arquivo para gravar registros em Logs com mais CloudWatch frequência. A versão 2.3.0 (e posterior) do componente gerenciador de registros processa e carrega todos os registros em vez de esperar que eles sejam alternados. Para obter mais informações, consulte o parâmetro de limite de tamanho do arquivo de log do componente Greengrass nucleus (). `logging.fileSizeKB`

- Níveis mínimos de registro

  Você pode configurar o nível mínimo de registro que o componente nucleus do Greengrass grava nos registros do sistema de arquivos. Por exemplo, você pode especificar registros de `DEBUG` nível para ajudar na solução de problemas ou pode especificar registros de `ERROR` nível para reduzir a quantidade de registros que um dispositivo principal cria. Para obter mais informações, consulte o parâmetro de nível de log do componente Greengrass nucleus (). `logging.level`

  Você também pode configurar o nível mínimo de registro que o componente gerenciador de registros grava CloudWatch nos registros. Por exemplo, você pode especificar um nível de registro mais alto para reduzir os custos de registro. Para obter mais informações, consulte o `minimumLogLevel` parâmetro do componente gerenciador de registros, que você pode especificar para os registros AWS IoT Greengrassdo software principal e os registros de componentes.

- Intervalo para verificar os registros para gravar CloudWatch nos registros

  Para aumentar ou diminuir a frequência com que o componente do gerenciador de registros grava CloudWatch registros em registros, você pode configurar o intervalo em que ele verifica a gravação de novos arquivos de log. Por exemplo, você pode especificar um intervalo menor para visualizar CloudWatch registros em Registros mais cedo do que faria com o intervalo padrão de 5 minutos. Você pode especificar um intervalo maior para reduzir custos, porque o componente do gerenciador de registros agrupa os arquivos de log em menos solicitações. Para obter mais informações, consulte o parâmetro de intervalo de upload do componente gerenciador de registros (`periodicUploadIntervalSec`).

- Formato de registro

  Você pode escolher se o software AWS IoT Greengrass Core grava registros no formato texto ou JSON. Escolha o formato de texto se você ler registros ou escolha o formato JSON se usar um aplicativo para ler ou analisar registros. Para obter mais informações, consulte o parâmetro de formato de log do componente Greengrass nucleus (). `logging.format`

- Pasta de registros do sistema de arquivos local

  Você pode alterar a pasta de registros `/greengrass/v2/logs` para outra pasta no dispositivo principal. Para obter mais informações, consulte o parâmetro do diretório de saída do componente Greengrass nucleus (). `logging.outputDirectory`

Logs do AWS CloudTrail

AWS IoT Greengrassintegra-se comAWS CloudTrail, um serviço que fornece um registro das ações realizadas por um usuário, função ou AWS service (Serviço da AWS) emAWS IoT Greengrass. Para ter mais informações, consulte Registre chamadas de AWS IoT Greengrass V2 API com AWS CloudTrail.

Registre chamadas de AWS IoT Greengrass V2 API com AWS CloudTrail

AWS IoT Greengrass V2 é integrado com AWS CloudTrail, um serviço que fornece um registro das ações realizadas por um usuário, função ou AWS serviço em AWS IoT Greengrass Version 2. CloudTrail captura todas as chamadas de API AWS IoT Greengrass como eventos. As chamadas capturadas incluem chamadas do AWS IoT Greengrass console e chamadas de código para as operações da AWS IoT Greengrass API.

Se você criar uma trilha, poderá habilitar a entrega contínua de CloudTrail eventos em um bucket do S3, incluindo eventos para AWS IoT Greengrass. Se você não configurar uma trilha, ainda poderá ver os eventos mais recentes no CloudTrail console no Histórico de eventos. Usando as informações coletadas por CloudTrail, você pode determinar a solicitação que foi feita AWS IoT Greengrass, o endereço IP do qual a solicitação foi feita, quem fez a solicitação, quando ela foi feita e detalhes adicionais.

Para obter mais informações sobre CloudTrail, consulte o Guia AWS CloudTrail do usuário.

AWS IoT Greengrass V2 informações em CloudTrail

CloudTrail é ativado no seu Conta da AWS quando você cria a conta. Quando a atividade ocorre em AWS IoT Greengrass, essa atividade é registrada em um CloudTrail evento junto com outros eventos AWS de serviço no histórico de eventos. Você pode exibir, pesquisar e baixar eventos recentes em sua Conta da AWS. Para obter mais informações, consulte Visualização de eventos com histórico de CloudTrail eventos.

Para um registro contínuo dos eventos em sua Conta da AWS, incluindo eventos para AWS IoT Greengrass, crie uma trilha. Uma trilha permite CloudTrail entregar arquivos de log para um bucket do S3. Por padrão, quando você cria uma trilha no console, a trilha se aplica a todos os Região da AWS s. A trilha registra eventos de todas as regiões na AWS partição e entrega os arquivos de log ao bucket do S3 que você especificar. Além disso, você pode configurar outros AWS serviços para analisar e agir com base nos dados de eventos coletados nos CloudTrail registros. Para obter mais informações, consulte:

- Visão geral da criação de uma trilha
- CloudTrail serviços e integrações suportados
- Configurando notificações do Amazon SNS para CloudTrail
- Recebendo arquivos de CloudTrail log de várias regiões e Recebendo arquivos de CloudTrail log de várias contas

Todas AWS IoT Greengrass V2 as ações são registradas CloudTrail e documentadas na Referência da AWS IoT Greengrass V2 API. Por exemplo, chamadas para o `CreateComponentVersion` `CreateDeployment` e `CancelDeployment` as ações geram entradas nos arquivos de CloudTrail log.

Cada entrada de log ou evento contém informações sobre quem gerou a solicitação. As informações de identidade ajudam a determinar:

- Se a solicitação foi feita com credenciais de usuário root ou AWS Identity and Access Management (IAM).
- Se a solicitação foi feita com credenciais de segurança temporárias de um perfil ou de um usuário federado.
- Se a solicitação foi feita por outro AWS serviço.

Para obter mais informações, consulte Elemento userIdentity do CloudTrail .

AWS IoT Greengrass eventos de dados em CloudTrail

Os eventos de dados fornecem informações sobre as operações de recursos realizadas em ou em um recurso (por exemplo, obter uma versão do componente ou a configuração de uma implantação). Elas também são conhecidas como operações de plano de dados. Eventos de dados geralmente são atividades de alto volume. Por padrão, CloudTrail não registra eventos de dados. O histórico de CloudTrail eventos não registra eventos de dados.

Há cobranças adicionais para eventos de dados. Para obter mais informações sobre CloudTrail preços, consulte AWS CloudTrail Preços.

Você pode registrar eventos de dados para os tipos de AWS IoT Greengrass recursos usando o CloudTrail console ou AWS CLI as operações CloudTrail da API. A tabela nesta seção mostra os tipos de recursos disponíveis para AWS IoT Greengrass.

- Para registrar eventos de dados usando o CloudTrail console, crie um armazenamento de dados de trilhas ou eventos para registrar eventos de dados ou atualize um armazenamento de dados de trilhas ou eventos existente para registrar eventos de dados.

  1. Escolha Eventos de dados para registrar eventos de dados.
  2. Na lista Tipo de evento de dados, escolha o tipo de recurso para o qual você deseja registrar eventos de dados.
  3. Escolha o modelo do seletor de registros que você deseja usar. Você pode registrar todos os eventos de dados do tipo de recurso, registrar todos os `readOnly` eventos, registrar todos os `writeOnly` eventos ou criar um modelo de seletor de registros personalizado para filtrar os `resources.ARN` campos `readOnlyeventName`, e.

- Para registrar eventos de dados usando o AWS CLI, configure o `--advanced-event-selectors` parâmetro para definir o `eventCategory` campo igual `Data` e o `resources.type` campo igual ao valor do tipo de recurso (consulte a tabela). Você pode adicionar condições para filtrar os valores dos `resources.ARN` campos `readOnlyeventName`, e.
  - Para configurar uma trilha para registrar eventos de dados, execute o put-event-selectorscomando. Para obter mais informações, consulte Registro de eventos de dados para trilhas com AWS CLI o.
  - Para configurar um armazenamento de dados de eventos para registrar eventos de dados, execute o create-event-data-storecomando para criar um novo armazenamento de dados de eventos para registrar eventos de dados ou execute o update-event-data-storecomando para atualizar um armazenamento de dados de eventos existente. Para obter mais informações, consulte Registro de eventos de dados para armazenamentos de dados de eventos com AWS CLI o.

A tabela a seguir lista os tipos de AWS IoT Greengrass recursos. A coluna Tipo de evento de dados (console) mostra o valor a ser escolhido na lista Tipo de evento de dados no CloudTrail console. A coluna de valor resources.type mostra o **`resources.type`** valor, que você especificaria ao configurar seletores de eventos avançados usando as APIs ou. AWS CLI CloudTrail A CloudTrail

coluna Data APIs logged to mostra as chamadas de API registradas CloudTrail para o tipo de recurso.

Tipo de evento de dados (console)	valor resources.type	APIs de dados registradas em CloudTrail
Certificado de IoT	`AWS::IoT::Certificate`	• VerifyClientDeviceIdentity • VerifyClientDeviceIoTCertificateAssociation
Versão do componente IoT Greengrass	`AWS::GreengrassV2::ComponentVersion`	• ResolveComponentCandidates
Implantação do IoT Greengrass	`AWS::GreengrassV2::Deployment`	• GetDeploymentConfiguration
Coisa de IoT	`AWS::IoT::Thing`	• ListThingGroupsForCoreDevices • PutCertificateAuthorities • VerifyClientDeviceIoTCertificateAssociation

> **Note**
>
> O Greengrass não registra eventos de acesso negado.

É possível configurar seletores de eventos avançados para filtrar os campos `eventName`, `readOnly` e `resources.ARN` para registrar somente os eventos que são importantes para você.

Adicione um filtro `eventName` para incluir ou excluir APIs de dados específicas.

Consulte mais informações sobre esses campos em AdvancedFieldSelector.

Os exemplos a seguir mostram como configurar seletores avançados usando o. AWS CLI*`TrailName`*Substitua uma *`região`* por suas próprias informações.

Example — Registre eventos de dados para coisas de IoT

```
aws cloudtrail put-event-selectors --trail-name TrailName --region region \
--advanced-event-selectors \
'[
    {
            "Name": "Log all thing data events",
            "FieldSelectors": [
                { "Field": "eventCategory", "Equals": ["Data"] },
                { "Field": "resources.type", "Equals": ["AWS::IoT::Thing"] }
            ]
        }
]'
```

Example — Filtrar em uma API específica de IoT

```
aws cloudtrail put-event-selectors --trail-name TrailName --region region \
--advanced-event-selectors \
'[
    {
            "Name": "Log IoT Greengrass PutCertificateAuthorities API calls",
            "FieldSelectors": [
                { "Field": "eventCategory", "Equals": ["Data"] },
                { "Field": "resources.type", "Equals": ["AWS::IoT::Thing"] },
                { "Field": "eventName", "Equals": ["PutCertificateAuthorities"] }
            ]
    }
]'
```

Example — Registre todos os eventos de dados do Greengrass

```
aws cloudtrail put-event-selectors --trail-name TrailName --region region \
--advanced-event-selectors \
'[
    {
        "Name": "Log all certificate data events",
        "FieldSelectors": [
            {
                "Field": "eventCategory",
                "Equals": [
                    "Data"
                ]
            },
```

```
            {
                “Field”: “resources.type”,
                “Equals”: [
                    “AWS::IoT::Certificate”
                ]
            }
        ]
    },
    {
        “Name”: “Log all component version data events”,
        “FieldSelectors”: [
            {
                “Field”: “eventCategory”,
                “Equals”: [
                    “Data”
                ]
            },
            {
                “Field”: “resources.type”,
                “Equals”: [
                    “AWS::GreengrassV2::ComponentVersion”
                ]
            }
        ]
    },
    {
        “Name”: “Log all deployment version”,
        “FieldSelectors”: [
            {
                “Field”: “eventCategory”,
                “Equals”: [
                    “Data”
                ]
            },
            {
                “Field”: “resources.type”,
                “Equals”: [
                    “AWS::GreengrassV2::Deployment”
                ]
            }
        ]
    },
    {
        “Name”: “Log all thing data events”,
```

```
        “FieldSelectors”: [
            {
                “Field”: “eventCategory”,
                “Equals”: [
                    “Data”
                ]
            },
            {
                “Field”: “resources.type”,
                “Equals”: [
                    “AWS::IoT::Thing”
                ]
            }
        ]
    }
]’
```

AWS IoT Greengrass eventos de gerenciamento em CloudTrail

Os eventos de gerenciamento fornecem informações sobre as operações de gerenciamento que são realizadas nos recursos AWS da sua conta. Elas também são conhecidas como operações de plano de controle. Por padrão, CloudTrail registra eventos de gerenciamento.

AWS IoT Greengrass registra todas as operações do plano de AWS IoT Greengrass controle como eventos de gerenciamento. Para ver uma lista das operações do plano de AWS IoT Greengrass controle AWS IoT Greengrass registradas CloudTrail, consulte a referência da AWS IoT Greengrass API, versão 2.

Entendendo as entradas do arquivo de AWS IoT Greengrass V2 log

Uma trilha é uma configuração que permite a entrega de eventos como arquivos de log para um bucket do S3 que você especificar. CloudTrail os arquivos de log contêm uma ou mais entradas de log. Um evento representa uma solicitação única de qualquer fonte. Ele inclui informações sobre a ação solicitada, a data e a hora da ação, os parâmetros da solicitação e assim por diante. CloudTrail os arquivos de log não são um rastreamento de pilha ordenado das chamadas públicas de API, portanto, eles não aparecem em nenhuma ordem específica.

O exemplo a seguir mostra uma entrada de CloudTrail registro que demonstra a `CreateDeployment` ação.

```
{
```

```
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/Administrator",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "Administrator"
    },
    "eventTime": "2021-01-06T02:38:05Z",
    "eventSource": "greengrass.amazonaws.com",
    "eventName": "CreateDeployment",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "203.0.113.0",
    "userAgent": "aws-cli/2.1.9 Python/3.7.9 Windows/10 exe/AMD64 prompt/off command/
greengrassv2.create-deployment",
    "requestParameters": {
        "deploymentPolicies": {
            "failureHandlingPolicy": "DO_NOTHING",
            "componentUpdatePolicy": {
                "timeoutInSeconds": 60,
                "action": "NOTIFY_COMPONENTS"
            },
            "configurationValidationPolicy": {
                "timeoutInSeconds": 60
            }
        },
        "deploymentName": "Deployment for MyGreengrassCoreGroup",
        "components": {
            "aws.greengrass.Cli": {
                "componentVersion": "2.0.3"
            }
        },
        "iotJobConfiguration": {},
        "targetArn": "arn:aws:iot:us-west-2:123456789012:thinggroup/
MyGreengrassCoreGroup"
    },
    "responseElements": {
        "iotJobArn": "arn:aws:iot:us-west-2:123456789012:job/fdfeba1d-ac6d-44ef-
ab28-54f684ea578d",
        "iotJobId": "fdfeba1d-ac6d-44ef-ab28-54f684ea578d",
        "deploymentId": "4196dddc-0a21-4c54-a985-66a525f6946e"
    },
    "requestID": "311b9529-4aad-42ac-8408-c06c6fec79a9",
```

```
    "eventID": "c0f3aa2c-af22-48c1-8161-bad4a2ab1841",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "eventCategory": "Management",
    "recipientAccountId": "123456789012"
}
```

Colete dados de telemetria de integridade do sistema a partir dos dispositivos principais AWS IoT Greengrass

Os dados de telemetria da integridade do sistema são dados de diagnóstico que podem ajudá-lo a monitorar o desempenho de operações críticas em seus dispositivos de núcleo do Greengrass. Você pode criar projetos e aplicativos para recuperar, analisar, transformar e relatar dados de telemetria de seus dispositivos periféricos. Especialistas em domínio, como engenheiros de processo, podem usar esses aplicativos para obter informações sobre a integridade da frota.

Você pode usar os seguintes métodos para coletar dados de telemetria de seus dispositivos principais do Greengrass:

- Componente emissor de telemetria do núcleo — O componente emissor de telemetria do núcleo (`aws.greengrass.telemetry.NucleusEmitter`) em um dispositivo principal do Greengrass publica dados de telemetria no tópico por padrão. `$local/greengrass/telemetry` Você pode usar os dados publicados neste tópico para agir localmente em seu dispositivo principal, mesmo quando seu dispositivo tem conectividade limitada com a nuvem. Opcionalmente, você também pode configurar o componente para publicar dados de telemetria em um tópico do AWS IoT Core MQTT de sua escolha.

  Você deve implantar o componente emissor de núcleo em um dispositivo principal para publicar dados de telemetria. Não há custos associados à publicação de dados de telemetria no tópico local. No entanto, o uso de um tópico do MQTT para publicar dados no Nuvem AWS está sujeito a AWS IoT Corepreços.

  AWS IoT Greengrassfornece vários componentes da comunidade para ajudá-lo a analisar e visualizar dados de telemetria localmente em seu dispositivo principal usando o InfluxDB e o Grafana. Esses componentes usam dados de telemetria do componente emissor do núcleo. Para obter mais informações, consulte o README do componente editor do InfluxDB.

- Agente de telemetria — O agente de telemetria nos principais dispositivos do Greengrass coleta dados de telemetria locais e os publica na Amazon sem exigir nenhuma interação com o cliente. EventBridge Os dispositivos principais publicam dados de telemetria com EventBridge base no melhor esforço. Por exemplo, os dispositivos de núcleo podem falhar em fornecer dados de telemetria quando estão off-line.

  O recurso de agente de telemetria está ativado por padrão para todos os dispositivos principais do Greengrass. Você começa a receber dados automaticamente assim que configura um dispositivo principal do Greengrass. Além dos custos do link de dados, a transferência de dados do dispositivo principal para o AWS IoT Core é gratuita. Isso ocorre porque o atendente publica em um tópico reservado AWS. No entanto, dependendo do seu caso de uso, você pode incorrer em custos ao receber ou processar os dados.

  > **Note**
  >
  > EventBridge A Amazon é um serviço de ônibus de eventos que você pode usar para conectar seus aplicativos a dados de várias fontes, como os principais dispositivos do Greengrass. Para obter mais informações, consulte O que é a Amazon EventBridge? no Guia do EventBridge usuário da Amazon.

Para garantir que o software AWS IoT Greengrass principal funcione adequadamente, AWS IoT Greengrass use os dados para fins de desenvolvimento e melhoria da qualidade. Esse recurso também ajuda a informar recursos de ponta novos e aprimorados. AWS IoT Greengrassretém os dados de telemetria por até sete dias.

Esta seção descreve como configurar e usar o agente de telemetria. Para obter informações sobre como configurar o componente emissor de telemetria do núcleo, consulte. Emissor de telemetria Nucleus

Tópicos

- Métricas de telemetria
- Definir as configurações do agente de telemetria
- Assine os dados de telemetria em EventBridge

Métricas de telemetria

A tabela a seguir descreve as métricas publicadas pelo agente de telemetria.

Nome	Descrição	
Sistema		
`SystemMemUsage`	A quantidade de memória atualmente em uso por todos os aplicativos no dispositivo de núcleo do Greengrass, incluindo o sistema operacional.	
`CpuUsage`	A quantidade de CPU atualmente em uso por todos os aplicativos no dispositivo de núcleo do Greengrass, incluindo o sistema operacional.	
`TotalNumberOfFDs`	O número de descritores de arquivo armazenados pelo sistema operacional do dispositivo de núcleo do Greengrass. Um descritor de arquivo identifica exclusivamente um arquivo aberto.	
Núcleo Greengrass		
`NumberOfComponentsRunning`	O número de componentes que estão sendo executados no dispositivo principal do Greengrass.	
`NumberOfComponentsErrored`	O número de componentes que estão em estado de erro	

Nome	Descrição
	no dispositivo principal do Greengrass.
`NumberOfComponentsInstalled`	O número de componentes que estão instalados no dispositivo principal do Greengrass.
`NumberOfComponentsStarting`	O número de componentes que estão começando no dispositivo principal do Greengrass.
`NumberOfComponentsNew`	O número de componentes que são novos no dispositivo principal do Greengrass.
`NumberOfComponentsStopping`	O número de componentes que estão parando no dispositivo principal do Greengrass.
`NumberOfComponentsFinished`	O número de componentes finalizados no dispositivo principal do Greengrass.
`NumberOfComponentsBroken`	O número de componentes que estão quebrados no dispositivo principal do Greengrass.
`NumberOfComponentsStateless`	O número de componentes sem estado no dispositivo principal do Greengrass.

Nome	Descrição	
Autenticação do dispositivo cliente — Esse recurso requer a versão 2.4.0 ou posterior do componente de autenticação do dispositivo cliente.		
`VerifyClientDeviceIdentity.Success`	O número de vezes que a identidade do dispositivo cliente foi verificada com êxito.	
`VerifyClientDeviceIdentity.Failure`	O número de vezes em que a identidade do dispositivo cliente falhou.	
`AuthorizeClientDeviceActions.Success`	O número de vezes que o dispositivo cliente está autorizado a concluir as ações solicitadas.	
`AuthorizeClientDeviceActions.Failure`	O número de vezes que o dispositivo cliente não está autorizado a concluir as ações solicitadas.	
`GetClientDeviceAuthToken.Success`	O número de vezes que o dispositivo cliente foi autenticado com sucesso.	
`GetClientDeviceAuthToken.Failure`	O número de vezes que o dispositivo cliente não pode ser autenticado.	
`SubscribeToCertificateUpdates.Success`	O número de assinaturas bem-sucedidas de atualizações de certificados.	

Nome	Descrição
`SubscribeToCertificateUpdates.Failure`	O número de tentativas malsucedidas de assinar as atualizações do certificado.
`ServiceError`	O número de erros internos não tratados na autenticação do dispositivo cliente.
Gerenciador de fluxo — Esse recurso requer a versão 2.7.0 ou posterior do componente do núcleo do Greengrass.	
`BytesAppended`	O número de bytes de dados anexados ao gerenciador de fluxo.
`BytesUploadedToIoTAnalytics`	O número de bytes de dados que o gerenciador de fluxos exporta para canais em AWS IoT Analytics.
`BytesUploadedToKinesis`	O número de bytes de dados que o gerenciador de fluxos exporta para streams no Amazon Kinesis Data Streams.
`BytesUploadedToIoTSiteWise`	O número de bytes de dados que o gerenciador de fluxos exporta para as propriedades do ativo AWS IoT SiteWise.

Nome	Descrição	
`BytesUploadedToS3`	O número de bytes de dados que o gerenciador de fluxos exporta para objetos no Amazon S3.	

Definir as configurações do agente de telemetria

O agente de telemetria usa as seguintes configurações padrão:

- O atendente de telemetria agrega dados de telemetria a cada hora.
- O atendente de telemetria publica uma mensagem de telemetria a cada 24 horas.

O agente de telemetria publica dados usando o protocolo MQTT com um nível de qualidade de serviço (QoS) de 0, o que significa que ele não confirma a entrega nem tenta publicar novamente. As mensagens de telemetria compartilham uma conexão MQTT com outras mensagens para assinatura destinadas a AWS IoT Core.

Além dos custos do link de dados, a transferência de dados do núcleo para o AWS IoT Core é gratuita. Isso ocorre porque o atendente publica em um tópico reservado AWS. No entanto, dependendo do seu caso de uso, você pode incorrer em custos ao receber ou processar os dados.

Você pode ativar ou desativar o recurso de agente de telemetria para cada dispositivo principal do Greengrass. Você também pode configurar os intervalos nos quais o dispositivo principal agrega e publica dados. Para configurar a telemetria, personalize o parâmetro de configuração da telemetria ao implantar o componente nucleus do Greengrass.

Assine os dados de telemetria em EventBridge

Você pode criar regras na Amazon EventBridge que definam como processar dados de telemetria publicados pelo agente de telemetria no dispositivo principal do Greengrass. Quando EventBridge recebe os dados, ele invoca as ações-alvo definidas em suas regras. Por exemplo, crie regras de eventos que enviem notificações, armazenem informações, tomem medidas corretivas ou invoquem outros eventos.

Eventos de telemetria

Os eventos de telemetria usam o formato a seguir.


```
{
  "version": "0",
  "id": "a09d303e-2f6e-3d3c-a693-8e33f4fe3955",
  "detail-type": "Greengrass Telemetry Data",
  "source": "aws.greengrass",
  "account": "123456789012",
  "time": "2020-11-30T20:45:53Z",
  "region": "us-east-1",
  "resources": [],
  "detail": {
    "ThingName": "MyGreengrassCore",
    "Schema": "2020-07-30",
    "ADP": [
      {
        "TS": 1602186483234,
        "NS": "SystemMetrics",
        "M": [
          {
            "N": "TotalNumberOfFDs",
            "Sum": 6447.0,
            "U": "Count"
          },
          {
            "N": "CpuUsage",
            "Sum": 15.458333333333332,
            "U": "Percent"
          },
          {
            "N": "SystemMemUsage",
            "Sum": 10201.0,
            "U": "Megabytes"
          }
        ]
      },
      {
        "TS": 1602186483234,
        "NS": "GreengrassComponents",
        "M": [
          {
            "N": "NumberOfComponentsStopping",
```

```
            "Sum": 0.0,
            "U": "Count"
          },
          {
            "N": "NumberOfComponentsStarting",
            "Sum": 0.0,
            "U": "Count"
          },
          {
            "N": "NumberOfComponentsBroken",
            "Sum": 0.0,
            "U": "Count"
          },
          {
            "N": "NumberOfComponentsFinished",
            "Sum": 1.0,
            "U": "Count"
          },
          {
            "N": "NumberOfComponentsInstalled",
            "Sum": 0.0,
            "U": "Count"
          },
          {
            "N": "NumberOfComponentsRunning",
            "Sum": 7.0,
            "U": "Count"
          },
          {
            "N": "NumberOfComponentsNew",
            "Sum": 0.0,
            "U": "Count"
          },
          {
            "N": "NumberOfComponentsErrored",
            "Sum": 0.0,
            "U": "Count"
          },
          {
            "N": "NumberOfComponentsStateless",
            "Sum": 0.0,
            "U": "Count"
          }
        ]
```

```
    },
    {
      "TS": 1602186483234,
      "NS": "aws.greengrass.ClientDeviceAuth",
      "M": [
        {
          "N": "VerifyClientDeviceIdentity.Success",
          "Sum": 3.0,
          "U": "Count"
        },
        {
          "N": "VerifyClientDeviceIdentity.Failure",
          "Sum": 1.0,
          "U": "Count"
        },
        {
          "N": "AuthorizeClientDeviceActions.Success",
          "Sum": 20.0,
          "U": "Count"
        },
        {
          "N": "AuthorizeClientDeviceActions.Failure",
          "Sum": 5.0,
          "U": "Count"
        },
        {
          "N": "GetClientDeviceAuthToken.Success",
          "Sum": 5.0,
          "U": "Count"
        },
        {
          "N": "GetClientDeviceAuthToken.Failure",
          "Sum": 2.0,
          "U": "Count"
        },
        {
          "N": "SubscribeToCertificateUpdates.Success",
          "Sum": 10.0,
          "U": "Count"
        },
        {
          "N": "SubscribeToCertificateUpdates.Failure",
          "Sum": 1.0,
          "U": "Count"
```

```
          },
          {
            "N": "ServiceError",
            "Sum": 3.0,
            "U": "Count"
          }
        ]
      },
      {
        "TS": 1602186483234,
        "NS": "aws.greengrass.StreamManager",
        "M": [
          {
            "N": "BytesAppended",
            "Sum": 157745524.0,
            "U": "Bytes"
          },
          {
            "N": "BytesUploadedToIoTAnalytics",
            "Sum": 149012.0,
            "U": "Bytes"
          },
          {
            "N": "BytesUploadedToKinesis",
            "Sum": 12192.0,
            "U": "Bytes"
          },
          {
            "N": "BytesUploadedToIoTSiteWise",
            "Sum": 13321.0,
            "U": "Bytes"
          },
          {
            "N": "BytesUploadedToS3",
            "Sum": 12213.0,
            "U": "Bytes"
          }
        ]
      }
    ]
  }
}
```

A matriz ADP contém uma lista de pontos de dados agregados que têm as seguintes propriedades:

`TS`

A data e hora de quando os dados foram coletados.

`NS`

O namespace métrico.

`M`

Lista de métricas. Uma métrica contém as seguintes propriedades:

`N`

O nome da métrica.

`Sum`

A soma dos valores da métrica nesse evento de telemetria.

`U`

A unidade do valor da métrica.

Para obter mais informações sobre cada métrica, consulteMétricas de telemetria.

Pré-requisitos para criar regras EventBridge

Antes de criar uma EventBridge regra paraAWS IoT Greengrass, você deve fazer o seguinte:

- Familiarize-se com eventos, regras e metas em EventBridge.
- Crie e configure os alvos invocados por suas EventBridge regras. As regras podem invocar vários tipos de destinos, como fluxos do Amazon Kinesis, perfis AWS Lambda, tópicos do Amazon SNS e filas do Amazon SQS.

  Sua EventBridge regra e os alvos associados devem estar no Região da AWS local em que você criou seus recursos do Greengrass. Para obter mais informações, consulte Endpoints e cotas do serviço na Referência geral da AWS.

Para obter mais informações, consulte O que é a Amazon EventBridge? e Introdução à Amazon EventBridge no Guia do EventBridge usuário da Amazon.

Crie uma regra de evento para obter dados de telemetria (console)

Use as etapas a seguir para usar o AWS Management Console para criar uma EventBridge regra que receba dados de telemetria publicados pelo dispositivo principal do Greengrass. Isso permite que servidores web, endereços de e-mail e outros assinantes de tópicos respondam ao evento. Para obter mais informações, consulte Criação de uma EventBridge regra que é acionada em um evento a partir de um AWS recurso no Guia do EventBridge usuário da Amazon.

1. Abra o EventBridgeconsole da Amazon e escolha Criar regra.
2. Em Name and description (Nome e descrição), insira um nome e uma descrição para a regra.
3. Em Define pattern (Definir padrão), configure o padrão de regra.
   a. Escolha Event pattern (Padrão de evento).
   b. Escolha Pre-defined pattern by service (Padrão predefinido por serviço).
   c. Em Service provider (Provedor de serviços), escolha AWS.
   d. Em Service name (Nome do serviço), escolha Greengrass.
   e. Em Tipo de evento, selecione Dados de telemetria do Greengrass.
4. Em Select event bus (Selecionar barramento de eventos), mantenha as opções de barramento de eventos padrão.
5. Em Select targets (Selecionar destinos), configure seu destino. O exemplo a seguir usa uma fila do Amazon SQS, mas você pode configurar outros tipos de destino.
   a. Em Target, escolha SQS queue.
   b. Para Fila*, escolha sua fila de destino.
6. Em Tags - optional (Tags - opcional), defina tags para a regra ou deixe os campos em branco.
7. Escolha Criar.

Crie uma regra de evento para obter dados de telemetria (CLI)

Use as etapas a seguir para usar o AWS CLI para criar uma EventBridge regra que receba dados de telemetria publicados pelos dispositivos principais do Greengrass. Isso permite que servidores web, endereços de e-mail e outros assinantes de tópicos respondam ao evento.

1. Crie a regra.
   - Substitua *thing-name* pelo nome da coisa do dispositivo principal.

Linux or Unix

```
aws events put-rule \
  --name MyGreengrassTelemetryEventRule \
  --event-pattern "{\"source\": [\"aws.greengrass\"], \"detail\": {\"ThingName
\": [\"thing-name\"]}}"
```

Windows Command Prompt (CMD)

```
aws events put-rule ^
  --name MyGreengrassTelemetryEventRule ^
  --event-pattern "{\"source\": [\"aws.greengrass\"], \"detail\": {\"ThingName
\": [\"thing-name\"]}}"
```

PowerShell

```
aws events put-rule `
  --name MyGreengrassTelemetryEventRule `
  --event-pattern "{\"source\": [\"aws.greengrass\"], \"detail\": {\"ThingName
\": [\"thing-name\"]}}"
```

As propriedades que são omitidas do padrão são ignoradas.

2. Adicione o tópico como um destino de regra. O exemplo a seguir usa o Amazon SQS, mas você pode configurar outros tipos de destino.

   - Substitua *queue-arn* pelo ARN da sua fila do Amazon SQS.

   Linux or Unix

   ```
   aws events put-targets \
     --rule MyGreengrassTelemetryEventRule \
     --targets "Id"="1","Arn"="queue-arn"
   ```

   Windows Command Prompt (CMD)

   ```
   aws events put-targets ^
     --rule MyGreengrassTelemetryEventRule ^
   ```

```
  --targets "Id"="1","Arn"="queue-arn"
```

PowerShell

```
aws events put-targets `
  --rule MyGreengrassTelemetryEventRule `
  --targets "Id"="1","Arn"="queue-arn"
```

> **Note**
>
> Para permitir que EventBridge a Amazon invoque sua fila de destino, você deve adicionar uma política baseada em recursos ao seu tópico. Para obter mais informações, consulte as permissões do Amazon SQS no Guia EventBridge do usuário da Amazon.

Para obter mais informações, consulte Eventos e padrões de eventos EventBridge no Guia do EventBridge usuário da Amazon.

Receba notificações de status de integridade de componentes e implantações

As regras de EventBridge eventos da Amazon fornecem notificações sobre mudanças de estado para suas implantações do Greengrass recebidas por seus dispositivos e para componentes instalados em seu dispositivo. EventBridge fornece um fluxo quase em tempo real de eventos do sistema que descreve as mudanças nos AWS recursos. AWS IoT Greengrass envia esses eventos com EventBridge base no melhor esforço. Isso significa que as AWS IoT Greengrass tentativas de enviar todos os eventos para, EventBridge mas, em alguns casos raros, um evento podem não ser entregues. Além disso, AWS IoT Greengrass pode enviar várias cópias de um determinado evento, o que significa que seus ouvintes do evento podem não receber os eventos na ordem em que os eventos ocorreram.

> **Note**
>
> EventBridge A Amazon é um serviço de ônibus de eventos que você pode usar para conectar seus aplicativos a dados de várias fontes, como os principais dispositivos do Greengrass e

notificações de componentes e de implantação. Para obter mais informações, consulte O que é a Amazon EventBridge? no Guia do EventBridge usuário da Amazon.

Tópicos

Evento de alteração do status de implantação

AWS IoT Greengrass emite um evento quando uma implantação entra nos seguintes estados: `FAILED`SUCCEEDED,`COMPLETED`,`REJECTED`, e. `CANCELED` Você pode criar uma EventBridge regra que seja executada para todas as transições de estado ou transições para estados que você especificar. Quando uma implantação entra em um estado que inicia uma regra, EventBridge invoca as ações de destino definidas na regra. Isso permite enviar notificações, capturar informações de eventos, executar ações corretivas ou iniciar outros eventos em resposta a uma alteração de estado. Por exemplo, você pode criar regras para os seguintes casos de uso:

- Iniciar operações pós-implantação, como fazer download de ativos e notificar a equipe.
- Enviar notificações após uma implantação bem-sucedida ou com falha.
- Publicar métricas personalizadas sobre eventos de implantação.

O evento para uma alteração no estado da implantação usa o seguinte formato:

```
{
    "version":"0",
    "id":" cd4d811e-ab12-322b-8255-EXAMPLEb1bc8",
    "detail-type":"Greengrass V2 Effective Deployment Status Change",
    "source":"aws.greengrass",
    "account":"123456789012",
```

```
    "region":"us-west-2",
    "time":"2018-03-22T00:38:11Z",
    "resources":["arn:aws:greengrass:us-
east-1:123456789012:coreDevices:MyGreengrassCore"],
    "detail":{
        "deploymentId": "4f38f1a7-3dd0-42a1-af48-EXAMPLE09681",
        "coreDeviceExecutionStatus": "FAILED|SUCCEEDED|COMPLETED|REJECTED|CANCELED",
        "statusDetails": {
          "errorStack": ["DEPLOYMENT_FAILURE", "ARTIFACT_DOWNLOAD_ERROR", "S3_ERROR",
 "S3_ACCESS_DENIED", "S3_HEAD_OBJECT_ACCESS_DENIED"],
          "errorTypes": ["DEPENDENCY_ERROR", "PERMISSION_ERROR"],
        },
        "reason": "S3_HEAD_OBJECT_ACCESS_DENIED: FAILED_NO_STATE_CHANGE: Failed to
 download artifact name: 's3://pentest27/nucleus/281/aws.greengrass.nucleus.zip' for
 component aws.greengrass.Nucleus-2.8.1, reason: S3 HeadObject returns 403 Access
 Denied. Ensure the IAM role associated with the core device has a policy granting
 s3:GetObject. null (Service: S3, Status Code: 403, Request ID: HR94ZNT2161DAR58,
 Extended Request ID: wTX4DDI+qigQt3uzwl9rlnQiYlBgwvPm/KJFWeFAn9t1mnGXTms/
luLCYANgq08RIH+x2H+hEKc=)"
     }
}
```

Você pode criar regras e eventos que o atualizarão sobre o status de uma implantação. Um evento é iniciado quando uma implantação é concluída como`FAILED`,`SUCCEEDED`, `COMPLETEDREJECTED`, ou`CANCELED`. Se a implantação falhar no dispositivo principal, você receberá uma resposta detalhada que explica por que a implantação falhou. Para obter mais informações sobre códigos de erro de implantação, consulte[Códigos de erro de implantação detalhados](#).

Estados de implantação

- `FAILED`. Houve falha na implantação.
- `SUCCEEDED`. A implantação direcionada a um grupo de coisas foi concluída com sucesso.
- `COMPLETED`. A implantação foi direcionada para uma coisa concluída com sucesso.
- `REJECTED`. A implantação foi rejeitada. Para obter mais informações, consulte o `statusDetails` campo.
- `CANCELED`. A implantação foi cancelada pelo usuário.

É possível que os eventos estejam duplicados ou fora de ordem. Para determinar a ordem dos eventos, use a propriedade `time`.

Para obter uma lista completa dos códigos de erro em `errorStacks` `eerrorTypes`, consulte Códigos de erro de implantação detalhados Códigos detalhados de status do componente e.

Evento de alteração do status do componente

AWS IoT Greengrass Nas versões 2.12.2 e anteriores, o Greengrass emite um evento quando um componente entra nos seguintes estados: e. `ERRORED` `BROKEN` Para as versões 2.12.3 e posteriores do Greengrass nucleus, o Greengrass emite um evento quando um componente entra nos seguintes estados:,, e. `ERRORED` `BROKEN` `RUNNING` `FINISHED` O Greengrass também emitirá um evento quando a implantação for concluída. Você pode criar uma EventBridge regra que seja executada para todas as transições de estado ou transições para estados que você especificar. Quando um componente instalado entra em um estado que inicia uma regra, EventBridge invoca as ações de destino definidas na regra. Isso permite enviar notificações, capturar informações de eventos, executar ações corretivas ou iniciar outros eventos em resposta a uma alteração de estado.

O evento para uma alteração do estado do componente usa os seguintes formatos:

Greengrass nucleus v2.12.2 and earlier

<title>Status do componente: ERRORED ou BROKEN</title>

```
{
    "version":"0",
    "id":" cd4d811e-ab12-322b-8255-EXAMPLEb1bc8",
    "detail-type":"Greengrass V2 Installed Component Status Change",
    "source":"aws.greengrass",
    "account":"123456789012",
    "region":"us-west-2",
    "time":"2018-03-22T00:38:11Z",
    "resources":["arn:aws:greengrass:us-
east-1:123456789012:coreDevices:MyGreengrassCore"],
    "detail": {
       "components": [
          {
            "componentName": "MyComponent",
            "componentVersion": "1.0.0",
            "root": true,
            "lifecycleState": "ERRORED|BROKEN",
            "lifecycleStatusCodes": ["STARTUP_ERROR"],
            "lifecycleStateDetails": "An error occurred during startup. The startup
 script exited with code 1."
```

```
          }
        ]
    }
}
```

Greengrass nucleus v2.12.3 and later

<title>Status do componente: ERRORED ou BROKEN</title>

```
{
    "version":"0",
    "id":" cd4d811e-ab12-322b-8255-EXAMPLEb1bc8",
    "detail-type":"Greengrass V2 Installed Component Status Change",
    "source":"aws.greengrass",
    "account":"123456789012",
    "region":"us-west-2",
    "time":"2018-03-22T00:38:11Z",
    "resources":["arn:aws:greengrass:us-
east-1:123456789012:coreDevices:MyGreengrassCore"],
    "detail": {
       "components": [
          {
            "componentName": "MyComponent",
            "componentVersion": "1.0.0",
            "root": true,
            "lifecycleState": "ERRORED|BROKEN",
            "lifecycleStatusCodes": ["STARTUP_ERROR"],
            "lifecycleStateDetails": "An error occurred during startup. The startup
 script exited with code 1."
          }
        ]
    }
}
```

<title>Status do componente: RUNNING ou FINISHED</title>

```
{
    "version":"0",
    "id":" cd4d811e-ab12-322b-8255-EXAMPLEb1bc8",
    "detail-type":"Greengrass V2 Installed Component Status Change",
    "source":"aws.greengrass",
    "account":"123456789012",
    "region":"us-west-2",
```

```
    "time":"2018-03-22T00:38:11Z",
    "resources":["arn:aws:greengrass:us-
east-1:123456789012:coreDevices:MyGreengrassCore"],
    "detail": {
       "components": [
          {
            "componentName": "MyComponent",
            "componentVersion": "1.0.0",
            "root": true,
            "lifecycleState": "RUNNING|FINISHED",
            "lifecycleStateDetails": null
          }
        ]
    }
}
```

Você pode criar regras e eventos que o atualizarão sobre o status de um componente instalado. Um evento é iniciado quando um componente muda de estado no dispositivo. Você receberá uma resposta detalhada que explica por que um componente está errado ou quebrado. Você também receberá um código de status que indicará o motivo da falha. Para obter mais informações sobre códigos de status de componentes, consulteCódigos detalhados de status do componente.

Pré-requisitos para criar regras EventBridge

Antes de criar uma EventBridge regra para AWS IoT Greengrass, faça o seguinte:

- Familiarize-se com eventos, regras e metas em EventBridge.
- Crie e configure os alvos invocados por suas EventBridge regras. As regras podem invocar muitos tipos de destinos, incluindo:
  - Amazon Simple Notification Service (Amazon SNS)
  - AWS Lambda funções
  - Amazon Kinesis Video Streams
  - Filas do Amazon Simple Queue Service (Amazon SQS)

Para obter mais informações, consulte O que é a Amazon EventBridge? e Introdução à Amazon EventBridge no Guia do EventBridge usuário da Amazon.

Configurar notificações de integridade do dispositivo (console)

Use as etapas a seguir para criar uma EventBridge regra que publique um tópico do Amazon SNS quando o estado de implantação mudar para um grupo. Isso permite que servidores web, endereços de e-mail e outros assinantes de tópicos respondam ao evento. Para obter mais informações, consulte Criação de uma EventBridge regra que é acionada em um evento a partir de um AWS recurso no Guia do EventBridge usuário da Amazon.

1. Abra o EventBridgeconsole da Amazon.
2. No painel de navegação, escolha Regras.
3. Escolha Create rule.
4. Insira um nome e uma descrição para a regra.

   Uma regra não pode ter o mesmo nome que outra na mesma Região e barramento de eventos.
5. Em Barramento de eventos, selecione o barramento de eventos que você deseja associar a essa regra. Se quiser que essa regra faça a correspondência com eventos provenientes da sua conta, escolha Barramento de eventos padrão da AWS . Quando um AWS serviço em sua conta emite um evento, ele sempre vai para o barramento de eventos padrão da sua conta.
6. Em Tipo de Regra, escolha Regra com Padrão de Evento.
7. Escolha Próximo.
8. Em Origem de eventos, escolha Eventos da AWS .
9. Em Padrão do evento, selecione Serviços da AWS .
10. Em Serviço da AWS , selecione Greengrass.
11. Para Tipo de evento, escolha entre as seguintes opções:

    - Para eventos de implantação, escolha Greengrass V2 Effective Deployment Status Change.
    - Para eventos de componentes, escolha Greengrass V2 Installed Component Status Change.
12. Selecione Next (Próximo).
13. Em Target types (Tipos de destinos), selecione AWS service (Serviço da).
14. Em Selecionar um destino, configure seu destino. Este exemplo usa um tópico do Amazon SNS, mas você pode configurar outros tipos de destino para enviar notificações.

    a. Em Target (Destino), selecione SNS topic (Tópico do SNS).
    b. Em Topic (Tópico), selecione o tópico de destino.
    c. Escolha Próximo.

15. Escolha Próximo.
16. Analise os detalhes da regra e selecione Criar regra.

Configurar notificações de integridade do dispositivo (CLI)

Use as etapas a seguir para criar uma EventBridge regra que publique um tópico do Amazon SNS quando houver um evento de mudança de status do Greengrass. Isso permite que servidores web, endereços de e-mail e outros assinantes de tópicos respondam ao evento.

1. Crie a regra.

   - Para eventos de mudança de status de implantação.

     ```
     aws events put-rule \
       --name TestRule \
       --event-pattern "{\"source\": [\"aws.greengrass\"], \"detail-type\":
      [\"Greengrass V2 Effective Deployment Status Change\"]}"
     ```

   - Para eventos de mudança de status de componentes.

     ```
     aws events put-rule \
       --name TestRule \
       --event-pattern "{\"source\": [\"aws.greengrass\"], \"detail-type\":
      [\"Greengrass V2 Installed Component Status Change\"]}"
     ```

   As propriedades que são omitidas do padrão são ignoradas.

2. Adicione o tópico como um destino de regra.

   - Substitua *topic-arn* pelo ARN do tópico do Amazon SNS.

   ```
   aws events put-targets \
     --rule TestRule \
     --targets "Id"="1","Arn"="topic-arn"
   ```

> **Note**
>
> Para permitir que EventBridge a Amazon ligue para seu tópico de destino, você deve adicionar uma política baseada em recursos ao seu tópico. Para obter mais informações, consulte as permissões do Amazon SNS no Guia EventBridge do usuário da Amazon.

Para obter mais informações, consulte Eventos e padrões de eventos EventBridge no Guia do EventBridge usuário da Amazon.

Configurar notificações de saúde do dispositivo (AWS CloudFormation)

Use AWS CloudFormation modelos para criar EventBridge regras que enviem notificações sobre mudanças de estado para suas implantações de grupos do Greengrass. Para obter mais informações, consulte a referência EventBridge de tipo de recurso da Amazon no Guia AWS CloudFormation do usuário.

Consulte também

- Verifique o status de implantação do dispositivo
- O que é a Amazon EventBridge? no Guia do EventBridge usuário da Amazon

Verifique o status do dispositivo principal do Greengrass

Os principais dispositivos do Greengrass reportam o status de seus componentes de software para. AWS IoT Greengrass Você pode verificar o resumo da integridade de cada dispositivo e verificar o status de cada componente em cada dispositivo.

Os dispositivos principais têm os seguintes status de integridade:

- `HEALTHY`— O software AWS IoT Greengrass principal e todos os componentes são executados sem problemas no dispositivo principal.
- `UNHEALTHY`— O software AWS IoT Greengrass principal ou um componente está em um estado de erro no dispositivo principal.

> **Note**
>
> AWS IoT Greengrass depende de dispositivos individuais para enviar atualizações de status para o. Nuvem AWS Se o software AWS IoT Greengrass Core não estiver em execução no dispositivo ou se o dispositivo não estiver conectado ao Nuvem AWS, o status relatado desse dispositivo pode não refletir seu status atual. A data e hora do status indica quando o status do dispositivo foi atualizado pela última vez.
> Os dispositivos principais enviam atualizações de status nos seguintes horários:
>
> - Quando o software AWS IoT Greengrass principal é iniciado
> - Quando o dispositivo principal recebe uma implantação do Nuvem AWS
> - Para o Greengrass nucleus 2.12.2 e versões anteriores, o dispositivo principal envia atualizações de status quando o status de qualquer componente no dispositivo principal se torna ou `ERRORED` `BROKEN`
> - Para o Greengrass nucleus 2.12.3 e versões posteriores, o dispositivo principal envia atualizações de status quando o status de qualquer componente no dispositivo principal se torna,,, ou `ERRORED` `BROKEN` `RUNNING` `FINISHED`
> - Em um intervalo regular que você pode configurar, cujo padrão é 24 horas
>
> Para o AWS IoT Greengrass Core v2.7.0 e versões posteriores, o dispositivo principal envia atualizações de status quando a implantação local e a implantação na nuvem ocorrem

Tópicos

Verifique a integridade de um dispositivo principal

Você pode verificar o status de dispositivos principais individuais.

Para verificar o status de um dispositivo principal (AWS CLI)

- Execute o comando a seguir para recuperar o status de um dispositivo. *`coreDeviceName`*Substitua pelo nome do dispositivo principal a ser consultado.

```
aws greengrassv2 get-core-device --core-device-thing-name coreDeviceName
```

A resposta contém informações sobre o dispositivo principal, incluindo seu status.

Verifique a integridade de um grupo de dispositivos principais

Você pode verificar o status de um grupo de dispositivos principais (um grupo de coisas).

Para verificar o status de um grupo de dispositivos (AWS CLI)

- Execute o comando a seguir para recuperar o status de vários dispositivos principais. Substitua o ARN no comando pelo ARN do grupo de coisas a ser consultado.

  ```
  aws greengrassv2 list-core-devices --thing-group-arn "arn:aws:iot:region:account-id:thinggroup/thingGroupName"
  ```

  A resposta contém a lista dos principais dispositivos do grupo de coisas. Cada entrada na lista contém o status do dispositivo principal.

Verifique o status do componente principal do dispositivo

Você pode verificar o status, como o estado do ciclo de vida, dos componentes de software em um dispositivo principal. Para obter mais informações sobre os estados do ciclo de vida dos componentes, consulte. Desenvolva AWS IoT Greengrass componentes

Para verificar o status dos componentes em um dispositivo principal (AWS CLI)

- Execute o comando a seguir para recuperar o status dos componentes em um dispositivo principal. *coreDeviceName*Substitua pelo nome do dispositivo principal a ser consultado.

  ```
  aws greengrassv2 list-installed-components --core-device-thing-name coreDeviceName
  ```

  A resposta contém a lista de componentes que são executados no dispositivo principal. Cada entrada na lista contém o estado do ciclo de vida do componente, incluindo o status atual dos dados e quando o dispositivo principal do Greengrass enviou pela última vez uma mensagem contendo um determinado componente para a nuvem. A resposta também incluirá a fonte de implantação mais recente que trouxe o componente para o dispositivo principal do Greengrass.

> **Note**
>
> Esse comando recupera uma lista paginada dos componentes que um dispositivo principal do Greengrass executa. Por padrão, essa lista não inclui componentes que são implantados como dependências de outros componentes. Você pode incluir dependências na resposta definindo o `topologyFilter` parâmetro como. `ALL`

Executar AWS Lambda funções

> **Note**
>
> AWS IoT Greengrassatualmente não oferece suporte a esse recurso nos dispositivos principais do Windows.

Você pode importar AWS Lambda funções como componentes que são executados nos dispositivos AWS IoT Greengrass principais. Talvez você queira fazer isso nos seguintes casos:

- Você tem o código do aplicativo nas funções do Lambda que deseja implantar nos dispositivos principais.
- Você tem aplicativos AWS IoT Greengrass V1 que deseja executar nos dispositivos AWS IoT Greengrass V2 principais. Para ter mais informações, consulte Etapa 2: criar e implantar AWS IoT Greengrass V2 componentes para migrar aplicativos AWS IoT Greengrass V1.

As funções Lambda incluem dependências nos seguintes componentes. Você não precisa definir esses componentes como dependências ao importar a função. Quando você implanta o componente da função Lambda, a implantação inclui essas dependências do componente Lambda.

- O componente Lambda launcher (`aws.greengrass.LambdaLauncher`) manipula processos e configurações do ambiente.
- O componente Lambda Manager (`aws.greengrass.LambdaManager`) lida com a comunicação e o escalonamento entre processos.
- O componente de tempos de execução do Lambda (`aws.greengrass.LambdaRuntimes`) fornece artefatos para cada tempo de execução do Lambda compatível.

Tópicos

Requisitos

Seus dispositivos principais e funções do Lambda devem atender aos seguintes requisitos para que você execute as funções no software AWS IoT Greengrass Core:

- Seu dispositivo principal deve atender aos requisitos para executar as funções do Lambda. Se você quiser que o dispositivo principal execute funções Lambda em contêineres, o dispositivo deve atender aos requisitos para fazer isso. Para ter mais informações, consulte Requisitos da função do Lambda.
- Você deve instalar as linguagens de programação que a função Lambda usa em seus dispositivos principais.

> **Tip**
>
> Você pode criar um componente que instala a linguagem de programação e, em seguida, especificar esse componente como uma dependência do seu componente da função Lambda. O Greengrass é compatível com todas as versões compatíveis com o Lambda dos tempos de execução Python, Node.js e Java. O Greengrass não aplica nenhuma restrição adicional às versões obsoletas de tempo de execução do Lambda. Você pode executar funções do Lambda que usam esses tempos de execução obsoletosAWS IoT Greengrass, mas não pode criá-las no. AWS Lambda Para obter mais informações sobre compatibilidade da AWS IoT Greengrass com runtime do Lambda, consulte Executar AWS Lambda funções.

Configurar o ciclo de vida da função Lambda

O ciclo de vida da função do Lambda do Greengrass determina quando uma função começa e como ela cria e usa contêineres. O ciclo de vida também determina como o software AWS IoT Greengrass Core retém as variáveis e a lógica de pré-processamento que estão fora do manipulador de funções.

AWS IoT Greengrasssuporta ciclos de vida sob demanda (padrão) e de longa duração:

- As funções sob demanda começam quando são invocadas e param quando não há mais tarefas para serem executadas. Cada invocação da função cria um contêiner separado, também chamado de sandbox, para processar invocações, a menos que um contêiner existente esteja disponível para reutilização. Qualquer um dos contêineres pode processar dados que você envia para a função.

Várias invocações de uma função sob demanda podem ser executadas simultaneamente.

As variáveis e a lógica de pré-processamento que você define fora do manipulador de funções não são retidas quando novos contêineres são criados.

- As funções de longa duração (ou fixas) começam quando o software AWS IoT Greengrass principal é iniciado e executado em um único contêiner. O mesmo contêiner processa todos os dados que você envia para a função.

  Várias invocações são colocadas em fila até que o software AWS IoT Greengrass Core execute invocações anteriores.

  As variáveis e a lógica de pré-processamento que você define fora do manipulador de funções são mantidas para cada invocação do manipulador.

  Use funções Lambda de longa duração quando precisar começar a trabalhar sem nenhuma entrada inicial. Por exemplo, uma função de longa duração pode carregar e começar a processar um modelo de aprendizado de máquina para ficar pronta quando a função receber dados do dispositivo.

  > **Note**
  >
  > Funções de longa duração têm tempos limite associados a cada invocação de seu manipulador. Se você quiser invocar um código que é executado indefinidamente, inicie-o fora do manipulador. Certifique-se de que não haja nenhum código de bloqueio fora do manipulador que possa impedir a inicialização da função.
  > Essas funções são executadas a menos que o software AWS IoT Greengrass principal pare, como durante uma implantação ou reinicialização. Essas funções não serão executadas se a função encontrar uma exceção não detectada, exceder seus limites de memória ou entrar em um estado de erro, como o tempo limite do manipulador.

Para obter mais informações sobre a reutilização de contêineres, consulte Entendendo a reutilização de contêineres AWS Lambda no blog de AWSComputação.

Configurar a conteinerização da função Lambda

Por padrão, as funções Lambda são executadas dentro de um AWS IoT Greengrass contêiner. Os contêineres do Greengrass fornecem isolamento entre suas funções e o host. Esse isolamento aumenta a segurança do host e das funções no contêiner.

Recomendamos que você execute funções do Lambda em um contêiner do Greengrass, a menos que seu caso de uso exija que elas sejam executadas sem conteinerização. Ao executar suas funções do Lambda em um contêiner do Greengrass, você tem mais controle sobre como restringir o acesso aos recursos.

Você pode executar uma função Lambda sem conteinerização nos seguintes casos:

- Você deseja executar AWS IoT Greengrass em um dispositivo que não seja compatível com o modo contêiner. Um exemplo seria se você quisesse usar uma distribuição Linux especial ou ter uma versão anterior do kernel que esteja desatualizada.
- Você deseja executar a função do Lambda em outro ambiente de contêiner com seu próprio OverlayFS, mas encontra conflitos de OverlayFS ao executar em um contêiner do Greengrass.
- Você precisa acessar recursos locais com caminhos que não podem ser determinados no momento da implantação ou cujos caminhos podem mudar após a implantação. Um exemplo desse recurso seria um dispositivo conectável.
- Você tem um aplicativo anterior que foi escrito como um processo e encontra problemas ao executá-lo em um contêiner do Greengrass.

Diferenças de conteinerização

Conteinerização	Observações
Contêiner do Greengrass	• Todos os atributos do AWS IoT Greengras s estão disponíveis quando você executa uma função do Lambda em um contêiner do Greengrass. • As funções do Lambda que são executada s em um contêiner do Greengrass não têm acesso ao código implantado de outras funções do Lambda, mesmo que sejam executadas com o mesmo grupo de

Conteinerização	Observações
	sistemas. Em outras palavras, suas funções do Lambda são executadas com maior isolamento umas das outras. • Como o software AWS IoT Greengrass Core executa todos os processos secundários no mesmo contêiner da função Lambda, os processos secundários param quando a função Lambda para.

Nenhum contêiner	• Os seguintes recursos não estão disponíveis para funções Lambda não conteinerizadas:
	  • Limites de memória de funções do Lambda.
	  • Recursos de volume e dispositivo locais. Você deve acessar esses recursos usando seus caminhos de arquivo no dispositivo principal em vez de como recursos da função Lambda.
	• Se sua função do Lambda não conteinerizada acessar um recurso de machine learning, você deverá identificar um proprietário do recurso e definir permissões de acesso no recurso, não na função do Lambda.
	• As funções Lambda não conteinerizadas têm acesso somente de leitura ao código implantado de outras funções do Lambda que são executadas com o mesmo grupo de sistemas.

Se você alterar a conteinerização de uma função Lambda ao implantá-la, a função pode não funcionar conforme o esperado. Se a função Lambda usar recursos locais que não estão mais disponíveis com a nova configuração de conteinerização, a implantação falhará.

- Quando você altera uma função Lambda de executada em um contêiner do Greengrass para execução sem conteinerização, os limites de memória da função são descartados. Você deve acessar o sistema de arquivos diretamente em vez de usar recursos locais anexados. Você deve remover todos os recursos anexados antes de implantar a função Lambda.
- Quando você altera uma função do Lambda da execução sem conteinerização para execução em um contêiner, sua função do Lambda perde o acesso direto ao sistema de arquivos. Você deve definir um limite de memória para cada função ou aceitar o limite de memória padrão de 16 MB. Você pode definir essas configurações para cada função do Lambda ao implantá-la.

Para alterar as configurações de conteinerização de um componente da função Lambda, defina o valor do parâmetro de `containerMode` configuração como uma das opções a seguir ao implantar o componente.

- `NoContainer`— O componente não é executado em um ambiente de execução isolado.
- `GreengrassContainer`— O componente é executado em um ambiente de execução isolado dentro do AWS IoT Greengrass contêiner.

Para obter mais informações sobre como implantar e configurar componentes, consulte Implemente AWS IoT Greengrass componentes em dispositivos Atualizar configurações de componentes e.

Importar uma função Lambda como componente (console)

Ao usar o AWS IoT Greengrassconsole para criar um componente de função Lambda, você importa uma AWS Lambda função existente e a configura para criar um componente que seja executado em seu dispositivo Greengrass.

Antes de começar, revise os requisitos para executar as funções do Lambda em dispositivos Greengrass.

Tarefas

- Etapa 1: Escolha uma função Lambda para importar
- Etapa 2: Configurar os parâmetros da função Lambda
- Etapa 3: (opcional) especificar plataformas suportadas para a função Lambda
- Etapa 4: (opcional) especificar dependências de componentes para a função Lambda
- Etapa 5: (opcional) executar a função Lambda em um contêiner

Etapa 1: Escolha uma função Lambda para importar

1. No menu de navegação AWS IoT Greengrassdo console, escolha Componentes.
2. Na página Componentes, escolha Criar componente.
3. Na página Criar componente, em Informações do componente, escolha Importar função Lambda.
4. Na função Lambda, pesquise e escolha a função Lambda que você deseja importar.

   AWS IoT Greengrasscria o componente com o nome da função Lambda.
5. Na versão da função Lambda, escolha a versão a ser importada. Você não pode escolher aliases do Lambda como. `$LATEST`

   AWS IoT Greengrasscria o componente com a versão da função Lambda como uma versão semântica válida. Por exemplo, se a versão da função for `3`, a versão do componente se tornará `3.0.0`.

Etapa 2: Configurar os parâmetros da função Lambda

Na página Criar componente, em Configuração da função Lambda, configure os seguintes parâmetros para usar na execução da função Lambda.

1. (Opcional) Adicione a lista de fontes de eventos nas quais a função Lambda se inscreve para receber mensagens de trabalho. Você pode especificar fontes de eventos para inscrever essa função em mensagens locais de publicação/assinatura e mensagens AWS IoT Core MQTT. A função Lambda é chamada quando recebe uma mensagem de uma fonte de eventos.

   > **Note**
   >
   > Para inscrever essa função em mensagens de outras funções ou componentes do Lambda, implante o componente legado do roteador de assinatura ao implantar esse componente da função Lambda. Ao implantar o componente legado do roteador de assinatura, especifique as assinaturas que a função Lambda usa.

Em Fontes de eventos, faça o seguinte para adicionar uma fonte de eventos:

a. Para cada fonte de evento que você adicionar, especifique as seguintes opções:

   - Tópico — O tópico para assinar mensagens.
   - Tipo — O tipo de origem do evento. Escolha uma das seguintes opções:
     - Publicação/assinatura local — Assine mensagens locais de publicação/assinatura.

       Se você usa o Greengrass nucleus v2.6.0 ou posterior e o Lambda manager v2.2.5 ou posterior, você pode usar **+** curingas de tópico MQTT (e) no Tópico ao especificar esse tipo. **#**
     - AWS IoT CoreMQTT — Assine as mensagens do AWS IoT Core MQTT.

       Você pode usar curingas de tópico MQTT (+e#) no Tópico ao especificar esse tipo.

b. Para adicionar outra fonte de eventos, escolha Adicionar fonte de eventos e repita a etapa anterior. Para remover uma fonte de eventos, escolha Remover ao lado da fonte de eventos que você deseja remover.

2. Em Tempo limite (segundos), insira o tempo máximo em segundos que uma função Lambda não fixada pode ser executada antes de atingir o tempo limite. O padrão é 3 segundos.
3. Em Fixado, escolha se o componente da função Lambda está fixado. O padrão é True.

   - Uma função Lambda fixa (ou de longa duração) AWS IoT Greengrass começa quando é iniciada e continua sendo executada em seu próprio contêiner.
   - Uma função Lambda não fixada (ou sob demanda) começa somente quando recebe um item de trabalho e sai depois de permanecer ociosa por um tempo de inatividade máximo especificado. Se a função tiver vários itens de trabalho, o software AWS IoT Greengrass Core criará várias instâncias da função.
4. (Opcional) Em Parâmetros adicionais, defina os seguintes parâmetros da função Lambda.

   - Tempo limite de status (segundos) — O intervalo em segundos no qual o componente da função Lambda envia atualizações de status para o componente gerenciador do Lambda. Esse parâmetro se aplica somente às funções fixadas. O padrão é 60 segundos.
   - Tamanho máximo da fila — O tamanho máximo da fila de mensagens para o componente da função Lambda. O software AWS IoT Greengrass Core armazena mensagens em uma fila

FIFO (primeiro a entrar, primeiro a sair) até poder executar a função Lambda para consumir cada mensagem. O padrão é 1.000 mensagens.

- Número máximo de instâncias — O número máximo de instâncias que uma função Lambda não fixada pode executar ao mesmo tempo. O padrão é 100 instâncias.
- Tempo máximo de inatividade (segundos) — A quantidade máxima de tempo em segundos que uma função Lambda não fixada pode ficar ociosa antes que AWS IoT Greengrass o software principal interrompa seu processo. O padrão é 60 segundos.
- Tipo de codificação — O tipo de carga útil que a função Lambda suporta. Escolha uma das seguintes opções:
  - JSON
  - Binário

  O padrão é JSON.

5. (Opcional) Especifique a lista de argumentos da linha de comando a serem transmitidos para a função Lambda quando ela for executada.

   a. Em Parâmetros adicionais, Processar argumentos, escolha Adicionar argumento.

   b. Para cada argumento adicionado, insira o argumento que você deseja passar para a função.

   c. Para remover um argumento, escolha Remover ao lado do argumento que você deseja remover.

6. (Opcional) Especifique as variáveis de ambiente que estão disponíveis para a função Lambda quando ela é executada. As variáveis de ambiente permitem que você armazene e atualize as configurações sem a necessidade de alterar o código da função.

   a. Em Parâmetros adicionais, Variáveis de ambiente, escolha Adicionar variável de ambiente.

   b. Para cada variável de ambiente que você adicionar, especifique as seguintes opções:

      - Chave — O nome da variável.
      - Valor — O valor padrão para essa variável.

   c. Para remover uma variável de ambiente, escolha Remover ao lado da variável de ambiente que você deseja remover.

Etapa 3: (opcional) especificar plataformas suportadas para a função Lambda

Todos os dispositivos principais têm atributos de sistema operacional e arquitetura. Quando você implanta o componente da função Lambda, o software AWS IoT Greengrass Core compara os valores da plataforma que você especifica com os atributos da plataforma no dispositivo principal para determinar se a função Lambda é suportada nesse dispositivo.

> **Note**
>
> Você também pode especificar atributos personalizados da plataforma ao implantar o componente nucleus do Greengrass em um dispositivo principal. Para obter mais informações, consulte o parâmetro de substituição da plataforma do componente do núcleo do Greengrass.

Em Configuração da função Lambda, Parâmetros adicionais, Plataformas, faça o seguinte para especificar as plataformas que essa função Lambda suporta.

1. Para cada plataforma, especifique as seguintes opções:
   - Sistema operacional — O nome do sistema operacional da plataforma. Atualmente, o único valor compatível é `linux`.
   - Arquitetura — A arquitetura do processador para a plataforma. Os valores compatíveis são:
     - `amd64`
     - `arm`
     - `aarch64`
     - `x86`
2. Para adicionar outra plataforma, escolha Adicionar plataforma e repita a etapa anterior. Para remover uma plataforma compatível, escolha Remover ao lado da plataforma que você deseja remover.

Etapa 4: (opcional) especificar dependências de componentes para a função Lambda

As dependências de componentes identificam componentes adicionais AWS fornecidos ou componentes personalizados que sua função usa. Quando você implanta o componente da função Lambda, a implantação inclui essas dependências para que sua função seja executada.

> **Important**
>
> Para importar uma função Lambda que você criou para ser executada na AWS IoT Greengrass V1, você deve definir dependências de componentes individuais para os recursos que sua função usa, como segredos, sombras locais e gerenciador de stream. Defina esses componentes como dependências rígidas para que seu componente da função Lambda seja reiniciado se a dependência mudar de estado. Para ter mais informações, consulte Importar funções V1 Lambda.

Em Configuração da função Lambda, Parâmetros adicionais, Dependências de componentes, conclua as etapas a seguir para especificar as dependências de componentes para sua função Lambda.

1. Escolha Adicionar dependência.
2. Para cada dependência de componente que você adicionar, especifique as seguintes opções:
   - Nome do componente — O nome do componente. Por exemplo, insira **`aws.greengrass.StreamManager`** para incluir o componente gerenciador de fluxo.
   - Requisito de versão — A restrição de versão semântica no estilo npm que identifica as versões compatíveis dessa dependência de componente. Você pode especificar uma única versão ou um intervalo de versões. Por exemplo, insira **`^1.0.0`** para especificar que essa função Lambda depende de qualquer versão na primeira versão principal do componente do gerenciador de fluxo. Para obter mais informações sobre restrições de versão semântica, consulte a calculadora npm semver.
   - Tipo — O tipo de dependência. Escolha uma das seguintes opções:
     - Difícil — O componente da função Lambda é reiniciado se a dependência mudar de estado. Esta é a ação padrão.
     - Soft — O componente da função Lambda não reinicia se a dependência mudar de estado.

3. Para remover uma dependência de componente, escolha Remover ao lado da dependência do componente

Etapa 5: (opcional) executar a função Lambda em um contêiner

Por padrão, as funções Lambda são executadas em um ambiente de execução isolado dentro do software AWS IoT Greengrass Core. Você também pode optar por executar a função Lambda como um processo sem nenhum isolamento (ou seja, no modo Sem contêiner).

Em Configuração do processo Linux, para o modo de isolamento, escolha entre as seguintes opções para selecionar a conteinerização para sua função Lambda:

- Contêiner Greengrass — A função Lambda é executada em um contêiner. Esta é a ação padrão.
- Sem contêiner — A função Lambda é executada como um processo sem nenhum isolamento.

Se você executar a função Lambda em um contêiner, conclua as etapas a seguir para definir a configuração do processo para a função Lambda.

1. Configure a quantidade de memória e os recursos do sistema, como volumes e dispositivos, a serem disponibilizados para o contêiner.

   Em Parâmetros do contêiner, faça o seguinte.

   a. Em Tamanho da memória, insira o tamanho da memória que você deseja alocar para o contêiner. Você pode especificar o tamanho da memória em MB ou kB.

   b. Em Pasta sys somente para leitura, escolha se o contêiner pode ou não ler as informações da pasta do `/sys` dispositivo. O padrão é False.

2. (Opcional) Configure os volumes locais que a função Lambda em contêiner pode acessar. Quando você define um volume, o software AWS IoT Greengrass Core monta os arquivos de origem no destino dentro do contêiner.

   a. Em Volumes, escolha Adicionar volume.

   b. Para cada volume que você adicionar, especifique as seguintes opções:

      - Volume físico — O caminho para a pasta de origem no dispositivo principal.
      - Volume lógico — O caminho para a pasta de destino no contêiner.

- Permissão — (Opcional) A permissão para acessar a pasta de origem a partir do contêiner. Escolha uma das seguintes opções:
  - Somente leitura — A função Lambda tem acesso somente de leitura à pasta de origem. Esta é a ação padrão.
  - Leitura/gravação — A função Lambda tem acesso de leitura/gravação à pasta de origem.
- Adicionar proprietário do grupo — (Opcional) Se deve ou não adicionar o grupo do sistema que executa o componente da função Lambda como proprietário da pasta de origem. O padrão é False.

c. Para remover um volume, escolha Remover ao lado do volume que você deseja remover.

3. (Opcional) Configure os dispositivos do sistema local que a função Lambda em contêiner pode acessar.

   a. Em Dispositivos, escolha Adicionar dispositivo.

   b. Para cada dispositivo que você adicionar, especifique as seguintes opções:

   - Caminho de montagem — O caminho para o dispositivo do sistema no dispositivo principal.
   - Permissão — (Opcional) A permissão para acessar o dispositivo do sistema a partir do contêiner. Escolha uma das seguintes opções:
     - Somente leitura — A função Lambda tem acesso somente de leitura ao dispositivo do sistema. Esta é a ação padrão.
     - Leitura/gravação — A função Lambda tem acesso de leitura/gravação à pasta de origem.
   - Adicionar proprietário do grupo — (Opcional) Se deve ou não adicionar o grupo do sistema que executa o componente da função Lambda como proprietário do dispositivo do sistema. O padrão é False.

Etapa 6: criar o componente da função Lambda

Depois de definir as configurações do componente da função Lambda, escolha Criar para concluir a criação do novo componente.

Para executar a função Lambda em seu dispositivo principal, você pode então implantar o novo componente em seus dispositivos principais. Para ter mais informações, consulte Implemente AWS IoT Greengrass componentes em dispositivos.

Importar uma função Lambda como componente () AWS CLI

Use a CreateComponentVersionoperação para criar componentes a partir das funções Lambda. Ao chamar essa operação, especifique `lambdaFunction` a importação de uma função Lambda.

Tarefas

- Etapa 1: Definir a configuração da função Lambda
- Etapa 2: criar o componente da função Lambda

Etapa 1: Definir a configuração da função Lambda

1. Crie um arquivo chamado `elambda-function-component.json`, em seguida, copie o seguinte objeto JSON para o arquivo. `lambdaArn`Substitua o pelo ARN da função Lambda a ser importada.

   ```
   {
     "lambdaFunction": {
       "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1"
     }
   }
   ```

   > **Important**
   >
   > Você deve especificar um ARN que inclua a versão da função a ser importada. Você não pode usar aliases de versão, como `$LATEST`.

2. (Opcional) Especifique o nome (`componentName`) do componente. Se você omitir esse parâmetro, AWS IoT Greengrass cria o componente com o nome da função Lambda.

   ```
   {
     "lambdaFunction": {
       "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
       "componentName": "com.example.HelloWorldLambda"
     }
   ```

```
}
```

3. (Opcional) Especifique a versão (`componentVersion`) do componente. Se você omitir esse parâmetro, AWS IoT Greengrass cria o componente com a versão da função Lambda como uma versão semântica válida. Por exemplo, se a versão da função for `3`, a versão do componente se tornará `3.0.0`.

   > **Note**
   >
   > Cada versão do componente que você carrega deve ser exclusiva. Certifique-se de fazer o upload da versão correta do componente, pois você não poderá editá-la depois de carregá-la.
   > AWS IoT Greengrassusa versões semânticas para componentes. As versões semânticas seguem um sistema de numeração principal.secundária.patch. Por exemplo, a versão `1.0.0` representa a primeira versão principal de um componente. Para obter mais informações, consulte a especificação da versão semântica.

   ```
   {
     "lambdaFunction": {
       "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
       "componentName": "com.example.HelloWorldLambda",
       "componentVersion": "1.0.0"
     }
   }
   ```

4. (Opcional) Especifique as plataformas suportadas por essa função Lambda. Cada plataforma contém um mapa de atributos que identificam uma plataforma. Todos os dispositivos principais têm atributos para sistema operacional (`os`) e arquitetura (`architecture`). O software AWS IoT Greengrass principal pode adicionar outros atributos da plataforma. Você também pode especificar atributos personalizados da plataforma ao implantar o componente nucleus do Greengrass em um dispositivo principal. Faça o seguinte:

   a. Adicione uma lista de plataformas (`componentPlatforms`) à função Lambda em. `lambda-function-component.json`

      ```
      {
        "lambdaFunction": {
          "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
      ```

```
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [

    ]
  }
}
```

b. Adicione cada plataforma compatível à lista. Cada plataforma tem uma interface amigável `name` para identificá-la e um mapa de atributos. O exemplo a seguir especifica que essa função oferece suporte a dispositivos x86 que executam Linux.

```
{
  "name": "Linux x86",
  "attributes": {
    "os": "linux",
    "architecture": "x86"
  }
}
```

Você `lambda-function-component.json` pode conter um documento semelhante ao exemplo a seguir.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ]
  }
}
```

5. (Opcional) Especifique as dependências do componente para sua função Lambda. Quando você implanta o componente da função Lambda, a implantação inclui essas dependências para que sua função seja executada.

   > **Important**
   >
   > Para importar uma função Lambda que você criou para ser executada na AWS IoT Greengrass V1, você deve definir dependências de componentes individuais para os recursos que sua função usa, como segredos, sombras locais e gerenciador de stream. Defina esses componentes como dependências rígidas para que seu componente da função Lambda seja reiniciado se a dependência mudar de estado. Para ter mais informações, consulte Importar funções V1 Lambda.

   Faça o seguinte:

   a. Adicione um mapa das dependências do componente (`componentDependencies`) à função Lambda em. `lambda-function-component.json`

   ```
   {
     "lambdaFunction": {
       "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
       "componentName": "com.example.HelloWorldLambda",
       "componentVersion": "1.0.0",
       "componentPlatforms": [
         {
           "name": "Linux x86",
           "attributes": {
             "os": "linux",
             "architecture": "x86"
           }
         }
       ],
       "componentDependencies": {

       }
     }
   }
   ```

   b. Adicione cada dependência de componente ao mapa. Especifique o nome do componente como chave e especifique um objeto com os seguintes parâmetros:

- `versionRequirement`— A restrição de versão semântica no estilo npm que identifica as versões compatíveis da dependência do componente. Você pode especificar uma única versão ou um intervalo de versões. Para obter mais informações sobre restrições de versão semântica, consulte a calculadora npm semver.
- `dependencyType`— (Opcional) O tipo da dependência. Escolha uma das seguintes opções:
  - `SOFT`— O componente da função Lambda não reinicia se a dependência mudar de estado.
  - `HARD`— O componente da função Lambda é reiniciado se a dependência mudar de estado.

  O padrão é `HARD`.

O exemplo a seguir especifica que essa função Lambda depende de qualquer versão na primeira versão principal do componente do gerenciador de stream. O componente da função Lambda é reiniciado quando o gerenciador de stream é reiniciado ou atualizado.

```
{
  "aws.greengrass.StreamManager": {
    "versionRequirement": "^1.0.0",
    "dependencyType": "HARD"
  }
}
```

Você `lambda-function-component.json` pode conter um documento semelhante ao exemplo a seguir.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
```

```
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    }
  }
}
```

6. (Opcional) Configure os parâmetros da função Lambda a serem usados para executar a função. Você pode configurar opções como variáveis de ambiente, fontes de eventos de mensagens, tempos limite e configurações de contêiner. Faça o seguinte:

   a. Adicione o objeto de parâmetros do Lambda (`componentLambdaParameters`) à função Lambda em. `lambda-function-component.json`

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {

    }
  }
```

```
}
```

b. (Opcional) Especifique as fontes de eventos nas quais a função Lambda se inscreve para receber mensagens de trabalho. Você pode especificar fontes de eventos para inscrever essa função em mensagens locais de publicação/assinatura e mensagens AWS IoT Core MQTT. A função Lambda é chamada quando recebe uma mensagem de uma fonte de eventos.

> **Note**
>
> Para inscrever essa função em mensagens de outras funções ou componentes do Lambda, implante o componente legado do roteador de assinatura ao implantar esse componente da função Lambda. Ao implantar o componente legado do roteador de assinatura, especifique as assinaturas que a função Lambda usa.

Faça o seguinte:

i. Adicione a lista de fontes de eventos (`eventSources`) aos parâmetros da função Lambda.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
```

```
    "componentLambdaParameters": {
      "eventSources": [

      ]
    }
  }
}
```

ii. Adicione cada fonte de evento à lista. Cada fonte de evento tem os seguintes parâmetros:

- `topic`— O tópico para se inscrever para receber mensagens.
- `type`— O tipo de origem do evento. Escolha uma das seguintes opções:
  - `PUB_SUB` – Assine mensagens locais de publicar/assinar.

    Se você usa o Greengrass nucleus v2.6.0 ou posterior e o Lambda manager v2.2.5 ou posterior, você pode usar + curingas de tópico do MQTT (e) no ao especificar esse tipo. # `topic`
  - `IOT_CORE`: assinar mensagens MQTT do AWS IoT Core.

    Você pode usar curingas de tópico do MQTT (+e#) no `topic` ao especificar esse tipo.

  O exemplo a seguir assina o AWS IoT Core MQTT em tópicos que correspondem ao filtro de `hello/world/+` tópicos.

```
{
  "topic": "hello/world/+",
  "type": "IOT_CORE"
}
```

Sua aparência `lambda-function-component.json` pode ser semelhante ao exemplo a seguir.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-
id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
```

```
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [
        {
          "topic": "hello/world/+",
          "type": "IOT_CORE"
        }
      ]
    }
  }
}
```

c. (Opcional) Especifique qualquer um dos seguintes parâmetros no objeto de parâmetros da função Lambda:

- `environmentVariables`— O mapa das variáveis de ambiente que estão disponíveis para a função Lambda quando ela é executada.
- `execArgs`— A lista de argumentos a serem passados para a função Lambda quando ela é executada.
- `inputPayloadEncodingType`— O tipo de carga útil que a função Lambda suporta. Escolha uma das seguintes opções:
  - `json`
  - `binary`

  Padrão: `json`
- `pinned`— Se a função Lambda está fixada ou não. O padrão é `true`.

  - Uma função Lambda fixa (ou de longa duração) AWS IoT Greengrass começa quando é iniciada e continua sendo executada em seu próprio contêiner.
  - Uma função Lambda não fixada (ou sob demanda) começa somente quando recebe um item de trabalho e sai depois de permanecer ociosa por um tempo de inatividade máximo especificado. Se a função tiver vários itens de trabalho, o software AWS IoT Greengrass Core criará várias instâncias da função.

  Use `maxIdleTimeInSeconds` para definir o tempo máximo de inatividade para sua função.
- `timeoutInSeconds`— O tempo máximo em segundos que a função Lambda pode ser executada antes de atingir o tempo limite. O padrão é 3 segundos.
- `statusTimeoutInSeconds`— O intervalo em segundos no qual o componente da função Lambda envia atualizações de status para o componente gerenciador do Lambda. Esse parâmetro se aplica somente às funções fixadas. O padrão é 60 segundos.
- `maxIdleTimeInSeconds`— O tempo máximo em segundos que uma função Lambda não fixada pode ficar ociosa antes que o software AWS IoT Greengrass Core interrompa seu processo. O padrão é 60 segundos.
- `maxInstancesCount`— O número máximo de instâncias que uma função Lambda não fixada pode executar ao mesmo tempo. O padrão é 100 instâncias.
- `maxQueueSize`— O tamanho máximo da fila de mensagens para o componente da função Lambda. O software AWS IoT Greengrass Core armazena mensagens em uma fila FIFO (first-in-first-out) até poder executar a função Lambda para consumir cada mensagem. O padrão é 1.000 mensagens.

Você `lambda-function-component.json` pode conter um documento semelhante ao exemplo a seguir.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
```

```
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [
        {
          "topic": "hello/world/+",
          "type": "IOT_CORE"
        }
      ],
      "environmentVariables": {
        "LIMIT": "300"
      },
      "execArgs": [
        "-d"
      ],
      "inputPayloadEncodingType": "json",
      "pinned": true,
      "timeoutInSeconds": 120,
      "statusTimeoutInSeconds": 30,
      "maxIdleTimeInSeconds": 30,
      "maxInstancesCount": 50,
      "maxQueueSize": 500
    }
  }
}
```

d. (Opcional) Defina as configurações do contêiner para a função Lambda. Por padrão, as funções Lambda são executadas em um ambiente de execução isolado dentro do software AWS IoT Greengrass Core. Você também pode optar por executar a função Lambda como um processo sem nenhum isolamento. Se você executar a função Lambda em um contêiner, você configura o tamanho da memória do contêiner e quais recursos do sistema estão disponíveis para a função Lambda. Faça o seguinte:

   i. Adicione o objeto de parâmetros de processo do Linux (`linuxProcessParams`) ao objeto de parâmetros Lambda em. `lambda-function-component.json`

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [
        {
          "topic": "hello/world/+",
          "type": "IOT_CORE"
        }
      ],
      "environmentVariables": {
        "LIMIT": "300"
      },
      "execArgs": [
        "-d"
      ],
      "inputPayloadEncodingType": "json",
      "pinned": true,
      "timeoutInSeconds": 120,
      "statusTimeoutInSeconds": 30,
      "maxIdleTimeInSeconds": 30,
      "maxInstancesCount": 50,
      "maxQueueSize": 500,
      "linuxProcessParams": {

      }
```

```
    }
  }
}
```

ii. (Opcional) Especifique se a função Lambda é executada ou não em um contêiner. Adicione o `isolationMode` parâmetro ao objeto de parâmetros do processo e escolha entre as seguintes opções:

- `GreengrassContainer`— A função Lambda é executada em um contêiner.
- `NoContainer`— A função Lambda é executada como um processo sem nenhum isolamento.

O padrão é `GreengrassContainer`.

iii. (Opcional) Se você executar a função Lambda em um contêiner, poderá configurar a quantidade de memória e os recursos do sistema, como volumes e dispositivos, a serem disponibilizados para o contêiner. Faça o seguinte:

A. Adicione o objeto de parâmetros do contêiner (`containerParams`) ao objeto de parâmetros do processo Linux em`lambda-function-component.json`.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-
id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
```

```
    "componentLambdaParameters": {
      "eventSources": [
        {
          "topic": "hello/world/+",
          "type": "IOT_CORE"
        }
      ],
      "environmentVariables": {
        "LIMIT": "300"
      },
      "execArgs": [
        "-d"
      ],
      "inputPayloadEncodingType": "json",
      "pinned": true,
      "timeoutInSeconds": 120,
      "statusTimeoutInSeconds": 30,
      "maxIdleTimeInSeconds": 30,
      "maxInstancesCount": 50,
      "maxQueueSize": 500,
      "linuxProcessParams": {
        "containerParams": {

        }
      }
    }
  }
}
```

B. (Opcional) Adicione o `memorySizeInKB` parâmetro para especificar o tamanho da memória do contêiner. O padrão é 16.384 KB (16 MB).

C. (Opcional) Adicione o `mountROSysfs` parâmetro para especificar se o contêiner pode ou não ler as informações da `/sys` pasta do dispositivo. O padrão é `false`.

D. (Opcional) Configure os volumes locais que a função Lambda em contêiner pode acessar. Quando você define um volume, o software AWS IoT Greengrass Core monta os arquivos de origem no destino dentro do contêiner. Faça o seguinte:

   I. Adicione a lista de volumes (`volumes`) aos parâmetros do contêiner.

```
{
  "lambdaFunction": {
```

```
    "lambdaArn": "arn:aws:lambda:region:account-
id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [
        {
          "topic": "hello/world/+",
          "type": "IOT_CORE"
        }
      ],
      "environmentVariables": {
        "LIMIT": "300"
      },
      "execArgs": [
        "-d"
      ],
      "inputPayloadEncodingType": "json",
      "pinned": true,
      "timeoutInSeconds": 120,
      "statusTimeoutInSeconds": 30,
      "maxIdleTimeInSeconds": 30,
      "maxInstancesCount": 50,
      "maxQueueSize": 500,
      "linuxProcessParams": {
        "containerParams": {
          "memorySizeInKB": 32768,
          "mountROSysfs": true,
          "volumes": [
```

```
          ]
        }
      }
    }
  }
}
```

II. Adicione cada volume à lista. Cada volume tem os seguintes parâmetros:

- `sourcePath`— O caminho para a pasta de origem no dispositivo principal.
- `destinationPath`— O caminho para a pasta de destino no contêiner.
- `permission`— (Opcional) A permissão para acessar a pasta de origem a partir do contêiner. Escolha uma das seguintes opções:
  - `ro`— A função Lambda tem acesso somente de leitura à pasta de origem.
  - `rw`— A função Lambda tem acesso de leitura e gravação à pasta de origem.

  O padrão é `ro`.
- `addGroupOwner`— (Opcional) Adicionar ou não o grupo do sistema que executa o componente da função Lambda como proprietário da pasta de origem. O padrão é `false`.

Você `lambda-function-component.json` pode conter um documento semelhante ao exemplo a seguir.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-
id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
```

```
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [
        {
          "topic": "hello/world/+",
          "type": "IOT_CORE"
        }
      ],
      "environmentVariables": {
        "LIMIT": "300"
      },
      "execArgs": [
        "-d"
      ],
      "inputPayloadEncodingType": "json",
      "pinned": true,
      "timeoutInSeconds": 120,
      "statusTimeoutInSeconds": 30,
      "maxIdleTimeInSeconds": 30,
      "maxInstancesCount": 50,
      "maxQueueSize": 500,
      "linuxProcessParams": {
        "containerParams": {
          "memorySizeInKB": 32768,
          "mountROSysfs": true,
          "volumes": [
            {
              "sourcePath": "/var/data/src",
              "destinationPath": "/var/data/dest",
              "permission": "rw",
              "addGroupOwner": true
            }
          ]
        }
      }
    }
  }
```

```
}
```

E. (Opcional) Configure os dispositivos do sistema local que a função Lambda em contêiner pode acessar. Faça o seguinte:

I. Adicione a lista de dispositivos do sistema (`devices`) aos parâmetros do contêiner.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-
id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [
        {
          "topic": "hello/world/+",
          "type": "IOT_CORE"
        }
      ],
      "environmentVariables": {
        "LIMIT": "300"
      },
      "execArgs": [
        "-d"
      ],
      "inputPayloadEncodingType": "json",
```

```
      "pinned": true,
      "timeoutInSeconds": 120,
      "statusTimeoutInSeconds": 30,
      "maxIdleTimeInSeconds": 30,
      "maxInstancesCount": 50,
      "maxQueueSize": 500,
      "linuxProcessParams": {
        "containerParams": {
          "memorySizeInKB": 32768,
          "mountROSysfs": true,
          "volumes": [
            {
              "sourcePath": "/var/data/src",
              "destinationPath": "/var/data/dest",
              "permission": "rw",
              "addGroupOwner": true
            }
          ],
          "devices": [

          ]
        }
      }
    }
  }
}
```

II. Adicione cada dispositivo do sistema à lista. Cada dispositivo do sistema tem os seguintes parâmetros:

- `path`— O caminho para o dispositivo do sistema no dispositivo principal.
- `permission`— (Opcional) A permissão para acessar o dispositivo do sistema a partir do contêiner. Escolha uma das seguintes opções:
  - `ro`— A função Lambda tem acesso somente de leitura ao dispositivo do sistema.
  - `rw`— A função Lambda tem acesso de leitura e gravação ao dispositivo do sistema.

  O padrão é `ro`.

- `addGroupOwner`— (Opcional) Adicionar ou não o grupo do sistema que executa o componente da função Lambda como proprietário do dispositivo do sistema. O padrão é `false`.

Você `lambda-function-component.json` pode conter um documento semelhante ao exemplo a seguir.

```
{
  "lambdaFunction": {
    "lambdaArn": "arn:aws:lambda:region:account-
id:function:HelloWorld:1",
    "componentName": "com.example.HelloWorldLambda",
    "componentVersion": "1.0.0",
    "componentPlatforms": [
      {
        "name": "Linux x86",
        "attributes": {
          "os": "linux",
          "architecture": "x86"
        }
      }
    ],
    "componentDependencies": {
      "aws.greengrass.StreamManager": {
        "versionRequirement": "^1.0.0",
        "dependencyType": "HARD"
      }
    },
    "componentLambdaParameters": {
      "eventSources": [
        {
          "topic": "hello/world/+",
          "type": "IOT_CORE"
        }
      ],
      "environmentVariables": {
        "LIMIT": "300"
      },
      "execArgs": [
        "-d"
      ],
      "inputPayloadEncodingType": "json",
```

```
      "pinned": true,
      "timeoutInSeconds": 120,
      "statusTimeoutInSeconds": 30,
      "maxIdleTimeInSeconds": 30,
      "maxInstancesCount": 50,
      "maxQueueSize": 500,
      "linuxProcessParams": {
        "containerParams": {
          "memorySizeInKB": 32768,
          "mountROSysfs": true,
          "volumes": [
            {
              "sourcePath": "/var/data/src",
              "destinationPath": "/var/data/dest",
              "permission": "rw",
              "addGroupOwner": true
            }
          ],
          "devices": [
            {
              "path": "/dev/sda3",
              "permission": "rw",
              "addGroupOwner": true
            }
          ]
        }
      }
    }
  }
}
```

7. (Opcional) Adicione tags (`tags`) para o componente. Para ter mais informações, consulte Marcar com tag os recursos do AWS IoT Greengrass Version 2.

Etapa 2: criar o componente da função Lambda

1. Execute o comando a seguir para criar o componente da função Lambda a partir de. `lambda-function-component.json`

```
aws greengrassv2 create-component-version --cli-input-json file://lambda-function-component.json
```

Se a solicitação for bem-sucedida, a resposta será semelhante ao exemplo a seguir.

```
{
  "arn":
 "arn:aws:greengrass:region:123456789012:components:com.example.HelloWorldLambda:versions:1
  "componentName": "com.example.HelloWorldLambda",
  "componentVersion": "1.0.0",
  "creationTimestamp": "Mon Dec 15 20:56:34 UTC 2020",
  "status": {
    "componentState": "REQUESTED",
    "message": "NONE",
    "errors": {}
  }
}
```

Copie o `arn` da saída para verificar o estado do componente na próxima etapa.

2. Quando você cria um componente, seu estado é`REQUESTED`. Em seguida, AWS IoT Greengrass valida se o componente é implantável. Você pode executar o comando a seguir para consultar o status do componente e verificar se ele é implantável. `arn`Substitua o pelo ARN da etapa anterior.

```
aws greengrassv2 describe-component \
  --arn "arn:aws:greengrass:region:account-
id:components:com.example.HelloWorldLambda:versions:1.0.0"
```

Se o componente for validado, a resposta indicará que o estado do componente é`DEPLOYABLE`.

```
{
  "arn": "arn:aws:greengrass:region:account-
id:components:com.example.HelloWorldLambda:versions:1.0.0",
  "componentName": "com.example.HelloWorldLambda",
  "componentVersion": "1.0.0",
  "creationTimestamp": "2020-12-15T20:56:34.376000-08:00",
  "publisher": "AWS Lambda",
  "status": {
    "componentState": "DEPLOYABLE",
    "message": "NONE",
    "errors": {}
  },
  "platforms": [
```

```
    {
      "name": "Linux x86",
      "attributes": {
        "architecture": "x86",
        "os": "linux"
      }
    }
  ]
}
```

Depois que o componente estiver `DEPLOYABLE` pronto, você poderá implantar a função Lambda em seus dispositivos principais. Para ter mais informações, consulte Implemente AWS IoT Greengrass componentes em dispositivos.

Use o AWS IoT Device SDK para se comunicar com o núcleo do Greengrass, outros componentes e AWS IoT Core

Os componentes executados em seu dispositivo principal podem usar a biblioteca de comunicação entre processos AWS IoT Greengrass principais (IPC) no AWS IoT Device SDK para se comunicar com o AWS IoT Greengrass núcleo e outros componentes do Greengrass. Para desenvolver e executar componentes personalizados que usam IPC, você deve usar o AWS IoT Device SDK para se conectar ao serviço AWS IoT Greengrass Core IPC e realizar operações de IPC.

A interface IPC suporta dois tipos de operações:

- Solicitação/resposta

  Os componentes enviam uma solicitação ao serviço IPC e recebem uma resposta que contém o resultado da solicitação.

- Assinatura

  Os componentes enviam uma solicitação de assinatura ao serviço IPC e esperam um fluxo de mensagens de eventos em resposta. Os componentes fornecem um manipulador de assinaturas que lida com mensagens de eventos, erros e encerramento de streams. AWS IoT Device SDK Isso inclui uma interface de manipulador com os tipos corretos de resposta e evento para cada operação de IPC. Para ter mais informações, consulte Inscreva-se nos streams de eventos do IPC.

Tópicos

- Interaja com a configuração do componente
- Recuperar valores secretos
- Interaja com sombras locais
- Gerencie implantações e componentes locais
- Autenticar e autorizar dispositivos clientes

Versões do cliente IPC

Nas versões posteriores dos SDKs de Java e Python, AWS IoT Greengrass fornece uma versão aprimorada do cliente IPC, chamada cliente IPC V2. Cliente IPC V2:

- Reduz a quantidade de código que você precisa escrever para usar operações IPC e ajuda a evitar erros comuns que podem ocorrer com o cliente IPC V1.
- Chama os retornos de chamada do manipulador de assinatura em um thread separado, para que agora você possa executar o código de bloqueio, incluindo chamadas de função IPC adicionais, nos retornos de chamada do manipulador de assinaturas. O cliente IPC V1 usa o mesmo encadeamento para se comunicar com o servidor IPC e chamar os retornos de chamada do manipulador de assinatura.
- Permite chamar operações de assinatura usando expressões Lambda (Java) ou funções (Python). O cliente IPC V1 exige que você defina classes de manipuladores de assinaturas.
- Fornece versões síncronas e assíncronas de cada operação IPC. O cliente IPC V1 fornece somente versões assíncronas de cada operação.

Recomendamos que você use o cliente IPC V2 para aproveitar essas melhorias. No entanto, muitos exemplos nesta documentação e em alguns conteúdos on-line demonstram somente como usar o cliente IPC V1. Você pode usar os exemplos e tutoriais a seguir para ver exemplos de componentes que usam o cliente IPC V2:

- PublishToTopicexemplos
- SubscribeToTopicexemplos
- Tutorial: Desenvolva um componente do Greengrass que adia as atualizações de componentes
- Tutorial: Interaja com dispositivos locais de IoT por meio do MQTT

Atualmente, o AWS IoT Device SDK for C++ v2 suporta somente o cliente IPC V1.

SDKs compatíveis para comunicação entre processos

As bibliotecas AWS IoT Greengrass principais do IPC estão incluídas nas seguintes AWS IoT Device SDK versões.

SDK	Versão mínima	Uso	
AWS IoT Device SDK para Java v2	v1.6.0	Consulte Use AWS IoT Device SDK para Java v2 (cliente IPC V2)	
AWS IoT Device SDK para Python v2	v1.9.0	Consulte Use AWS IoT Device SDK para Python v2 (cliente IPC V2)	
AWS IoT Device SDK para C++ v2	v1.17.0	Consulte Use AWS IoT Device SDK para C++ v2	
AWS IoT Device SDK para JavaScript v2	v1.12.0	Consulte Use AWS IoT Device SDK para JavaScript v2 (cliente IPC V1)	

Conecte-se ao serviço AWS IoT Greengrass Core IPC

Para usar a comunicação entre processos em seu componente personalizado, você deve criar uma conexão com um soquete de servidor IPC executado pelo software AWS IoT Greengrass Core. Conclua as tarefas a seguir para baixar e usar o AWS IoT Device SDK no idioma de sua escolha.

Use AWS IoT Device SDK para Java v2 (cliente IPC V2)

Para usar o AWS IoT Device SDK para Java v2 (cliente IPC V2)

1. Faça o download do AWS IoT Device SDK para Java v2 (v1.6.0 ou posterior).
2. Faça o seguinte para executar seu código personalizado em seu componente:

- Crie seu componente como um arquivo JAR que inclua AWS IoT Device SDK o. e execute esse arquivo JAR na receita do componente.
- Defina o AWS IoT Device SDK JAR como um artefato de componente e adicione esse artefato ao classpath ao executar seu aplicativo na receita do componente.

3. Use o código a seguir para criar o cliente IPC.

```
try (GreengrassCoreIPCClientV2 ipcClient =
 GreengrassCoreIPCClientV2.builder().build()) {
    // Use client.
} catch (Exception e) {
    LOGGER.log(Level.SEVERE, "Exception occurred when using IPC.", e);
    System.exit(1);
}
```

Use AWS IoT Device SDK para Python v2 (cliente IPC V2)

Para usar o AWS IoT Device SDK para Python v2 (cliente IPC V2)

1. Baixe o AWS IoT Device SDK para Python (v1.9.0 ou posterior).
2. Adicione as etapas de instalação do SDK ao ciclo de vida da instalação na receita do seu componente.
3. Crie uma conexão com o serviço AWS IoT Greengrass Core IPC. Use o código a seguir para criar o cliente IPC.

```
from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2

try:
    ipc_client = GreengrassCoreIPCClientV2()
    # Use IPC client.
except Exception:
    print('Exception occurred when using IPC.', file=sys.stderr)
    traceback.print_exc()
    exit(1)
```

Use AWS IoT Device SDK para C++ v2

Para criar a AWS IoT Device SDK v2 para C++, um dispositivo deve ter as seguintes ferramentas:

- C++ 11 ou posterior
- CMake 3.1 ou posterior
- Um dos seguintes compiladores:
  - GCC 4.8 ou posterior
  - Clang 3.9 ou posterior
  - MSVC 2015 ou posterior

Para usar o AWS IoT Device SDK para C++ v2

1. Faça o download do AWS IoT Device SDK para C++ v2 (v1.17.0 ou posterior).
2. Siga as instruções de instalação no README para criar o AWS IoT Device SDK para C++ v2 a partir do código-fonte.
3. Em sua ferramenta de criação de C++, vincule a biblioteca Greengrass IPC`AWS::GreengrassIpc-cpp`,, que você criou na etapa anterior. O `CMakeLists.txt` exemplo a seguir vincula a biblioteca Greengrass IPC a um projeto que você cria com o CMake.

   ```
   cmake_minimum_required(VERSION 3.1)
   project (greengrassv2_pubsub_subscriber)

   file(GLOB MAIN_SRC
           "*.h"
           "*.cpp"
           )
   add_executable(${PROJECT_NAME} ${MAIN_SRC})

   set_target_properties(${PROJECT_NAME} PROPERTIES
           LINKER_LANGUAGE CXX
           CXX_STANDARD 11)
   find_package(aws-crt-cpp PATHS ~/sdk-cpp-workspace/build)
   find_package(EventstreamRpc-cpp PATHS ~/sdk-cpp-workspace/build)
   find_package(GreengrassIpc-cpp PATHS ~/sdk-cpp-workspace/build)
   target_link_libraries(${PROJECT_NAME} AWS::GreengrassIpc-cpp)
   ```

4. No código do componente, crie uma conexão com o serviço AWS IoT Greengrass Core IPC para criar um cliente IPC ()`Aws::Greengrass::GreengrassCoreIpcClient`. Você deve definir um manipulador do ciclo de vida da conexão IPC que gerencie eventos de conexão, desconexão e erro IPC. O exemplo a seguir cria um cliente IPC e um manipulador do ciclo de

vida da conexão IPC que imprime quando o cliente IPC se conecta, desconecta e encontra erros.

```
#include <iostream>

#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        std::cout << "OnConnectCallback" << std::endl;
    }

    void OnDisconnectCallback(RpcError error) override {
        std::cout << "OnDisconnectCallback: " << error.StatusToString() <<
 std::endl;
        exit(-1);
    }

    bool OnErrorCallback(RpcError error) override {
        std::cout << "OnErrorCallback: " << error.StatusToString() << std::endl;
        return true;
    }
};

int main() {
    // Create the IPC client.
    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
 connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }
```

```
    // Use the IPC client to create an operation request.

    // Activate the operation request.
    auto activate = operation.Activate(request, nullptr);
    activate.wait();

    // Wait for Greengrass Core to respond to the request.
    auto responseFuture = operation.GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
 std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from
 Greengrass Core." << std::endl;
        exit(-1);
    }

    // Check the result of the request.
    auto response = responseFuture.get();
    if (response) {
        std::cout << "Successfully published to topic: " << topic << std::endl;
    } else {
        // An error occurred.
        std::cout << "Failed to publish to topic: " << topic << std::endl;
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
            std::cout << "Operation error: " << error->GetMessage().value() <<
 std::endl;
        } else {
            std::cout << "RPC error: " << response.GetRpcError() << std::endl;
        }
        exit(-1);
    }

    return 0;
}
```

5. Para executar seu código personalizado em seu componente, crie seu código como um artefato binário e execute o artefato binário em sua receita de componente. Defina a `Execute` permissão do artefato `OWNER` para permitir que o software AWS IoT Greengrass Core execute o artefato binário.

   A `Manifests` seção da receita do seu componente pode ser semelhante ao exemplo a seguir.

JSON

```
{
  ...
  "Manifests": [
    {
      "Lifecycle": {
        "run": "{artifacts:path}/greengrassv2_pubsub_subscriber"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.PubSubSubscriberCpp/1.0.0/greengrassv2_pubsub_subscriber",
          "Permission": {
            "Execute": "OWNER"
          }
        }
      ]
    }
  ]
}
```

YAML

```
...
Manifests:
  - Lifecycle:
      run: {artifacts:path}/greengrassv2_pubsub_subscriber
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.PubSubSubscriberCpp/1.0.0/greengrassv2_pubsub_subscriber
        Permission:
          Execute: OWNER
```

Use AWS IoT Device SDK para JavaScript v2 (cliente IPC V1)

Para criar o AWS IoT Device SDK for JavaScript v2 para uso com o NodeJS, um dispositivo deve ter as seguintes ferramentas:

- NodeJS 10.0 ou posterior

  - Execute `node -v` para verificar a versão do Node.
- CMake 3.1 ou posterior

Para usar o AWS IoT Device SDK for JavaScript v2 (cliente IPC V1)

1. Baixe o AWS IoT Device SDK para JavaScript v2 (v1.12.10 ou posterior).
2. Siga as instruções de instalação no README para criar o AWS IoT Device SDK for JavaScript v2 a partir do código-fonte.
3. Crie uma conexão com o serviço AWS IoT Greengrass Core IPC. Conclua as etapas a seguir para criar o cliente IPC e estabelecer uma conexão.
4. Use o código a seguir para criar o cliente IPC.

```
import * as greengrascoreipc from 'aws-iot-device-sdk-v2';

let client = greengrascoreipc.createClient();
```

5. Use o código a seguir para estabelecer uma conexão do seu componente com o núcleo do Greengrass.

```
await client.connect();
```

Autorize componentes a realizar operações de IPC

Para permitir que seus componentes personalizados usem algumas operações de IPC, você deve definir políticas de autorização que permitam que o componente execute a operação em determinados recursos. Cada política de autorização define uma lista de operações e uma lista de recursos que a política permite. Por exemplo, o serviço IPC de mensagens de publicação/assinatura define operações de publicação e assinatura para recursos de tópicos. Você pode usar o `*` caractere curinga para permitir o acesso a todas as operações ou a todos os recursos.

Você define políticas de autorização com o parâmetro de `accessControl` configuração, que pode ser definido na receita do componente ou ao implantar o componente. O `accessControl` objeto mapeia identificadores de serviço IPC para listas de políticas de autorização. Você pode definir várias políticas de autorização para cada serviço IPC para controlar o acesso. Cada política de autorização tem um ID de política, que deve ser exclusivo entre todos os componentes.

> **Tip**
>
> Para criar IDs de política exclusivos, você pode combinar o nome do componente, o nome do serviço IPC e um contador. Por exemplo, um componente chamado `com.example.HelloWorld` pode definir duas políticas de autorização de publicação/assinatura com as seguintes IDs:
>
> - `com.example.HelloWorld:pubsub:1`
> - `com.example.HelloWorld:pubsub:2`

As políticas de autorização usam o seguinte formato. Esse objeto é o parâmetro `accessControl` de configuração.

JSON

```
{
  "IPC service identifier": {
    "policyId": {
      "policyDescription": "description",
      "operations": [
        "operation1",
        "operation2"
      ],
      "resources": [
        "resource1",
        "resource2"
      ]
    }
  }
}
```

YAML

```
IPC service identifier:
  policyId:
    policyDescription: description
    operations:
      - operation1
      - operation2
    resources:
```

```
      - resource1
      - resource2
```

Caracteres curingas nas políticas de autorização

Você pode usar o * caractere curinga no `resources` elemento das políticas de autorização do IPC para permitir o acesso a vários recursos em uma única política de autorização.

- Em todas as versões do núcleo do Greengrass, você pode especificar um único * caractere como recurso para permitir o acesso a todos os recursos.
- No Greengrass nucleus v2.6.0 e versões posteriores, você pode especificar o * caractere em um recurso para corresponder a qualquer combinação de caracteres. Por exemplo, você pode especificar `factory/1/devices/Thermostat*/status` para permitir o acesso a um tópico de status para todos os dispositivos de termostato em uma fábrica, onde o nome de cada dispositivo começa com. `Thermostat`

Ao definir políticas de autorização para o serviço AWS IoT Core MQTT IPC, você também pode usar curingas do MQTT (+e#) para combinar vários recursos. Para obter mais informações, consulte Caracteres curinga do MQTT nas políticas de autorização do IPC do AWS IoT Core MQTT.

Variáveis de receita nas políticas de autorização

Se você usar o Greengrass nucleus v2.6.0 ou posterior e definir a opção de interpolateComponentConfigurationconfiguração do Greengrass nucleus como`true`, poderá usar a variável de receita nas políticas de autorização. `{iot:thingName}` Quando você precisar de uma política de autorização que inclua o nome do dispositivo principal, como para tópicos do MQTT ou sombras do dispositivo, você pode usar essa variável de receita para configurar uma única política de autorização para um grupo de dispositivos principais. Por exemplo, você pode permitir que um componente acesse o seguinte recurso para operações de IPC paralelas.

```
$aws/things/{iot:thingName}/shadow/
```

Caracteres especiais nas políticas de autorização

Para especificar um literal * ou ? caractere em uma política de autorização, você deve usar uma sequência de escape. As sequências de escape a seguir instruem o software AWS IoT Greengrass

Core a usar o valor literal em vez do significado especial do caractere. Por exemplo, o * caractere é um curinga que corresponde a qualquer combinação de caracteres.

Caráter literal	Sequência de escape	Observações
*	${*}	
?	${?}	AWS IoT Greengrass atualmente não suporta o ? curinga, que corresponde a um único caractere.
$	${$}	Use essa sequência de escape para correspon der a um recurso que contém${. Por exemplo, para corresponder a um recurso chamado${resourc eName} , você deve especific ar${$}{resourceName} . Caso contrário, para corresponder a um recurso que contém$, você pode usar um literal$, como para permitir acesso a um tópico que comece com$aws.

Exemplos de políticas de autorização

Você pode consultar os exemplos de políticas de autorização a seguir para ajudá-lo a configurar políticas de autorização para seus componentes.

Example Exemplo de receita de componente com uma política de autorização

O exemplo de receita de componente a seguir inclui um accessControl objeto que define uma política de autorização. Essa política autoriza o com.example.HelloWorld componente a publicar no test/topic tópico.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.HelloWorld",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that publishes messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.HelloWorld:pubsub:1": {
            "policyDescription": "Allows access to publish to test/topic.",
            "operations": [
              "aws.greengrass#PublishToTopic"
            ],
            "resources": [
              "test/topic"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Lifecycle": {
        "run": "java -jar {artifacts:path}/HelloWorld.jar"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.HelloWorld
ComponentVersion: '1.0.0'
ComponentDescription: A component that publishes messages.
ComponentPublisher: Amazon
ComponentConfiguration:
```

```
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.pubsub:
        "com.example.HelloWorld:pubsub:1":
          policyDescription: Allows access to publish to test/topic.
          operations:
            - "aws.greengrass#PublishToTopic"
          resources:
            - "test/topic"
Manifests:
  - Lifecycle:
      run: |-
        java -jar {artifacts:path}/HelloWorld.jar
```

Example Exemplo de atualização da configuração do componente com uma política de autorização

O exemplo de atualização de configuração a seguir em uma implantação especifica a configuração de um componente com um `accessControl` objeto que define uma política de autorização. Essa política autoriza o `com.example.HelloWorld` componente a publicar no `test/topic` tópico.

Console

Configuração para mesclar

```
{
  "accessControl": {
    "aws.greengrass.ipc.pubsub": {
      "com.example.HelloWorld:pubsub:1": {
        "policyDescription": "Allows access to publish to test/topic.",
        "operations": [
          "aws.greengrass#PublishToTopic"
        ],
        "resources": [
          "test/topic"
        ]
      }
    }
  }
}
```

AWS CLI

O comando a seguir cria uma implantação em um dispositivo principal.

```
aws greengrassv2 create-deployment --cli-input-json file://hello-world-
deployment.json
```

O `hello-world-deployment.json` arquivo contém o seguinte documento JSON.

```
{
  "targetArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
  "deploymentName": "Deployment for MyGreengrassCore",
  "components": {
    "com.example.HelloWorld": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "merge": "{\"accessControl\":{\"aws.greengrass.ipc.pubsub\":
{\"com.example.HelloWorld:pubsub:1\":{\"policyDescription\":\"Allows access to
 publish to test/topic.\",\"operations\":[\"aws.greengrass#PublishToTopic\"],
\"resources\":[\"test/topic\"]}}}}"
      }
    }
  }
}
```

Greengrass CLI

O comando CLI do Greengrass a seguir cria uma implantação local em um dispositivo principal.

```
sudo greengrass-cli deployment create \
  --recipeDir recipes \
  --artifactDir artifacts \
  --merge "com.example.HelloWorld=1.0.0" \
  --update-config hello-world-configuration.json
```

O `hello-world-configuration.json` arquivo contém o seguinte documento JSON.

```
{
  "com.example.HelloWorld": {
    "MERGE": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
```

```
          "com.example.HelloWorld:pubsub:1": {
            "policyDescription": "Allows access to publish to test/topic.",
            "operations": [
              "aws.greengrass#PublishToTopic"
            ],
            "resources": [
              "test/topic"
            ]
          }
        }
      }
    }
  }
}
```

Inscreva-se nos streams de eventos do IPC

Você pode usar as operações de IPC para assinar fluxos de eventos em um dispositivo principal do Greengrass. Para usar uma operação de assinatura, defina um manipulador de assinatura e crie uma solicitação para o serviço IPC. Em seguida, o cliente IPC executa as funções do manipulador de assinaturas sempre que o dispositivo principal transmite uma mensagem de evento para seu componente.

Você pode fechar uma assinatura para interromper o processamento de mensagens de eventos. Para fazer isso, chame `closeStream()` (Java), `close()` (Python) ou `Close()` (C++) no objeto de operação de assinatura que você usou para abrir a assinatura.

O serviço AWS IoT Greengrass Core IPC suporta as seguintes operações de assinatura:

- SubscribeToTopic
- SubscribeToIoTCore
- SubscribeToComponentUpdates
- SubscribeToConfigurationUpdate
- SubscribeToValidateConfigurationUpdates

Tópicos

- Defina gerenciadores de assinaturas
- Exemplos de gerenciadores de assinaturas

Defina gerenciadores de assinaturas

Para definir um manipulador de assinatura, defina funções de retorno de chamada que lidem com mensagens de eventos, erros e encerramento de fluxo. Se você usar o cliente IPC V1, deverá definir essas funções em uma classe. Se você usa o cliente IPC V2, que está disponível em versões posteriores dos SDKs para Java e Python, você pode definir essas funções sem criar uma classe de manipulador de assinaturas.

Java

Se você usar o cliente IPC V1, deverá implementar a interface genéricasoftware.amazon.awssdk.eventstreamrpc.StreamResponseHandler<*StreamEvent* *StreamEventType*é o tipo de mensagem de evento para a operação de assinatura. Defina as funções a seguir para lidar com mensagens de eventos, erros e encerramento de fluxo.

Se você usa o cliente IPC V2, pode definir essas funções fora de uma classe de manipulador de assinatura ou usar expressões lambda.

`void onStreamEvent(`*`StreamEventType`* `event)`

O retorno de chamada que o cliente IPC chama quando recebe uma mensagem de evento, como uma mensagem MQTT ou uma notificação de atualização de componente.

`boolean onStreamError(Throwable error)`

O retorno de chamada que o cliente IPC chama quando ocorre um erro de stream.

Retorne true para fechar o stream de assinatura como resultado do erro ou retorne false para manter o stream aberto.

`void onStreamClosed()`

O retorno de chamada que o cliente IPC chama quando o fluxo é fechado.

Python

Se você usar o cliente IPC V1, deverá estender a classe do manipulador de resposta de fluxo que corresponde à operação de assinatura. AWS IoT Device SDK Isso inclui uma classe de gerenciador de assinaturas para cada operação de assinatura. *`StreamEventType`*é o tipo de mensagem de evento para a operação de assinatura. Defina as funções a seguir para lidar com mensagens de eventos, erros e encerramento de fluxo.

Se você usa o cliente IPC V2, pode definir essas funções fora de uma classe de manipulador de assinatura ou usar expressões lambda.

```
def on_stream_event(self, event: StreamEventType) -> None
```

O retorno de chamada que o cliente IPC chama quando recebe uma mensagem de evento, como uma mensagem MQTT ou uma notificação de atualização de componente.

```
def on_stream_error(self, error: Exception) -> bool
```

O retorno de chamada que o cliente IPC chama quando ocorre um erro de stream.

Retorne true para fechar o stream de assinatura como resultado do erro ou retorne false para manter o stream aberto.

```
def on_stream_closed(self) -> None
```

O retorno de chamada que o cliente IPC chama quando o fluxo é fechado.

C++

Implemente uma classe derivada da classe do manipulador de resposta de fluxo que corresponde à operação de assinatura. AWS IoT Device SDK Isso inclui uma classe base de manipulador de assinaturas para cada operação de assinatura. *StreamEventType*é o tipo de mensagem de evento para a operação de assinatura. Defina as funções a seguir para lidar com mensagens de eventos, erros e encerramento de fluxo.

```
void OnStreamEvent(StreamEventType *event)
```

O retorno de chamada que o cliente IPC chama quando recebe uma mensagem de evento, como uma mensagem MQTT ou uma notificação de atualização de componente.

```
bool OnStreamError(OperationError *error)
```

O retorno de chamada que o cliente IPC chama quando ocorre um erro de stream.

Retorne true para fechar o stream de assinatura como resultado do erro ou retorne false para manter o stream aberto.

```
void OnStreamClosed()
```

O retorno de chamada que o cliente IPC chama quando o fluxo é fechado.

JavaScript

Implemente uma classe derivada da classe do manipulador de resposta de fluxo que corresponde à operação de assinatura. AWS IoT Device SDK Isso inclui uma classe base de manipulador de assinaturas para cada operação de assinatura. *`StreamEventType`*é o tipo de mensagem de evento para a operação de assinatura. Defina as funções a seguir para lidar com mensagens de eventos, erros e encerramento de fluxo.

`on(event: 'ended', listener: StreamingOperationEndedListener)`

O retorno de chamada que o cliente IPC chama quando o fluxo é fechado.

`on(event: 'streamError', listener: StreamingRpcErrorListener)`

O retorno de chamada que o cliente IPC chama quando ocorre um erro de stream.

Retorne true para fechar o stream de assinatura como resultado do erro ou retorne false para manter o stream aberto.

`on(event: 'message', listener: (message: InboundMessageType) => void)`

O retorno de chamada que o cliente IPC chama quando recebe uma mensagem de evento, como uma mensagem MQTT ou uma notificação de atualização de componente.

Exemplos de gerenciadores de assinaturas

O exemplo a seguir demonstra como usar a SubscribeToTopic operação e um manipulador de assinaturas para assinar mensagens locais de publicação/assinatura.

Java (IPC client V2)

Example Exemplo: Inscrever-se em mensagens locais de publicação/assinatura

```
package com.aws.greengrass.docs.samples.ipc;

import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClientV2;
import software.amazon.awssdk.aws.greengrass.SubscribeToTopicResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.*;

import java.nio.charset.StandardCharsets;
import java.util.Optional;

public class SubscribeToTopicV2 {
```

```
   public static void main(String[] args) {
       String topic = args[0];
       try (GreengrassCoreIPCClientV2 ipcClient =
GreengrassCoreIPCClientV2.builder().build()) {
           SubscribeToTopicRequest request = new
SubscribeToTopicRequest().withTopic(topic);
           GreengrassCoreIPCClientV2.StreamingResponse<SubscribeToTopicResponse,
                   SubscribeToTopicResponseHandler> response =
                   ipcClient.subscribeToTopic(request,
SubscribeToTopicV2::onStreamEvent,
                           Optional.of(SubscribeToTopicV2::onStreamError),
                           Optional.of(SubscribeToTopicV2::onStreamClosed));
           SubscribeToTopicResponseHandler responseHandler =
response.getHandler();
           System.out.println("Successfully subscribed to topic: " + topic);

           // Keep the main thread alive, or the process will exit.
           try {
               while (true) {
                   Thread.sleep(10000);
               }
           } catch (InterruptedException e) {
               System.out.println("Subscribe interrupted.");
           }

           // To stop subscribing, close the stream.
           responseHandler.closeStream();
       } catch (Exception e) {
           if (e.getCause() instanceof UnauthorizedError) {
               System.err.println("Unauthorized error while publishing to topic: "
+ topic);
           } else {
               System.err.println("Exception occurred when using IPC.");
           }
           e.printStackTrace();
           System.exit(1);
       }
   }

   public static void onStreamEvent(SubscriptionResponseMessage
subscriptionResponseMessage) {
       try {
```

```
            BinaryMessage binaryMessage =
 subscriptionResponseMessage.getBinaryMessage();
            String message = new String(binaryMessage.getMessage(),
 StandardCharsets.UTF_8);
            String topic = binaryMessage.getContext().getTopic();
            System.out.printf("Received new message on topic %s: %s%n", topic,
 message);
        } catch (Exception e) {
            System.err.println("Exception occurred while processing subscription
 response " +
                    "message.");
            e.printStackTrace();
        }
    }

    public static boolean onStreamError(Throwable error) {
        System.err.println("Received a stream error.");
        error.printStackTrace();
        return false; // Return true to close stream, false to keep stream open.
    }

    public static void onStreamClosed() {
        System.out.println("Subscribe to topic stream closed.");
    }
}
```

Python (IPC client V2)

Example Exemplo: Inscrever-se em mensagens locais de publicação/assinatura

```
import sys
import time
import traceback

from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2
from awsiot.greengrasscoreipc.model import (
    SubscriptionResponseMessage,
    UnauthorizedError
)


def main():
    args = sys.argv[1:]
    topic = args[0]
```

```
    try:
        ipc_client = GreengrassCoreIPCClientV2()
        # Subscription operations return a tuple with the response and the
 operation.
        _, operation = ipc_client.subscribe_to_topic(topic=topic,
 on_stream_event=on_stream_event,

 on_stream_error=on_stream_error, on_stream_closed=on_stream_closed)
        print('Successfully subscribed to topic: ' + topic)

        # Keep the main thread alive, or the process will exit.
        try:
            while True:
                time.sleep(10)
        except InterruptedError:
            print('Subscribe interrupted.')

        # To stop subscribing, close the stream.
        operation.close()
    except UnauthorizedError:
        print('Unauthorized error while subscribing to topic: ' +
              topic, file=sys.stderr)
        traceback.print_exc()
        exit(1)
    except Exception:
        print('Exception occurred', file=sys.stderr)
        traceback.print_exc()
        exit(1)


def on_stream_event(event: SubscriptionResponseMessage) -> None:
    try:
        message = str(event.binary_message.message, 'utf-8')
        topic = event.binary_message.context.topic
        print('Received new message on topic %s: %s' % (topic, message))
    except:
        traceback.print_exc()


def on_stream_error(error: Exception) -> bool:
    print('Received a stream error.', file=sys.stderr)
    traceback.print_exc()
    return False  # Return True to close stream, False to keep stream open.
```

```
def on_stream_closed() -> None:
    print('Subscribe to topic stream closed.')


if __name__ == '__main__':
    main()
```

C++

Example Exemplo: Inscrever-se em mensagens locais de publicação/assinatura

```
#include <iostream>

#include </crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class SubscribeResponseHandler : public SubscribeToTopicStreamHandler {
    public:
        virtual ~SubscribeResponseHandler() {}

    private:
        void OnStreamEvent(SubscriptionResponseMessage *response) override {
            auto jsonMessage = response->GetJsonMessage();
            if (jsonMessage.has_value() &&
 jsonMessage.value().GetMessage().has_value()) {
                auto messageString =
 jsonMessage.value().GetMessage().value().View().WriteReadable();
                // Handle JSON message.
            } else {
                auto binaryMessage = response->GetBinaryMessage();
                if (binaryMessage.has_value() &&
 binaryMessage.value().GetMessage().has_value()) {
                    auto messageBytes = binaryMessage.value().GetMessage().value();
                    std::string messageString(messageBytes.begin(),
 messageBytes.end());
                    // Handle binary message.
                }
            }
        }
```

```
        bool OnStreamError(OperationError *error) override {
            // Handle error.
            return false; // Return true to close stream, false to keep stream open.
        }

        void OnStreamClosed() override {
            // Handle close.
        }
};

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        // Handle connection to IPC service.
    }

    void OnDisconnectCallback(RpcError error) override {
        // Handle disconnection from IPC service.
    }

    bool OnErrorCallback(RpcError error) override {
        // Handle IPC service connection error.
        return true;
    }
};

int main() {
    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
 connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    String topic("my/topic");
    int timeout = 10;

    SubscribeToTopicRequest request;
```

```
    request.SetTopic(topic);

    //SubscribeResponseHandler streamHandler;
    auto streamHandler = MakeShared<SubscribeResponseHandler>(DefaultAllocator());
    auto operation = ipcClient.NewSubscribeToTopic(streamHandler);
    auto activate = operation->Activate(request, nullptr);
    activate.wait();

    auto responseFuture = operation->GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
 std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from Greengrass
 Core." << std::endl;
        exit(-1);
    }

    auto response = responseFuture.get();
    if (!response) {
        // Handle error.
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
            (void)error;
            // Handle operation error.
        } else {
            // Handle RPC error.
        }
        exit(-1);
    }

    // Keep the main thread alive, or the process will exit.
    while (true) {
        std::this_thread::sleep_for(std::chrono::seconds(10));
    }

    operation->Close();
    return 0;
}
```

JavaScript

Example Exemplo: Inscrever-se em mensagens locais de publicação/assinatura

```
import * as greengrasscoreipc from "aws-iot-device-sdk-v2/dist/greengrasscoreipc";
import {SubscribeToTopicRequest, SubscriptionResponseMessage} from "aws-iot-device-
sdk-v2/dist/greengrasscoreipc/model";
import {RpcError} from "aws-iot-device-sdk-v2/dist/eventstream_rpc";

class SubscribeToTopic {
    private ipcClient : greengrasscoreipc.Client
    private readonly topic : string;

    constructor() {
        // define your own constructor, e.g.
        this.topic = "<define_your_topic>";
        this.subscribeToTopic().then(r => console.log("Started workflow"));
    }

    private async subscribeToTopic() {
        try {
            this.ipcClient = await getIpcClient();

            const subscribeToTopicRequest : SubscribeToTopicRequest = {
                topic: this.topic,
            }

            const streamingOperation =
 this.ipcClient.subscribeToTopic(subscribeToTopicRequest, undefined); //
 conditionally apply options

            streamingOperation.on("message", (message: SubscriptionResponseMessage)
 => {
                // parse the message depending on your use cases, e.g.
                if(message.binaryMessage && message.binaryMessage.message) {
                    const receivedMessage =
 message.binaryMessage?.message.toString();
                }
            });

            streamingOperation.on("streamError", (error : RpcError) => {
                // define your own error handling logic
            })

            streamingOperation.on("ended", () => {
                // define your own logic
            })
```

```
            await streamingOperation.activate();

            // Keep the main thread alive, or the process will exit.
            await new Promise((resolve) => setTimeout(resolve, 10000))
        } catch (e) {
            // parse the error depending on your use cases
            throw e
        }
    }
}

export async function getIpcClient(){
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

// starting point
const subscribeToTopic = new SubscribeToTopic();
```

Melhores práticas do IPC

As melhores práticas para usar o IPC em componentes personalizados diferem entre o cliente IPC V1 e o cliente IPC V2. Siga as melhores práticas para a versão do cliente IPC que você usa.

IPC client V2

O cliente IPC V2 executa funções de retorno de chamada em um thread separado, portanto, em comparação com o cliente IPC V1, há menos diretrizes a serem seguidas ao usar IPC e escrever funções de manipulador de assinatura.

- Reutilize um cliente IPC

Depois de criar um cliente IPC, mantenha-o aberto e reutilize-o para todas as operações de IPC. A criação de vários clientes usa recursos extras e pode resultar em vazamentos de recursos.

- Lidar com exceções

  O cliente IPC V2 registra exceções não detectadas nas funções do manipulador de assinaturas. Você deve capturar exceções nas funções do manipulador para lidar com erros que ocorrem no seu código.

IPC client V1

O cliente IPC V1 usa um único thread que se comunica com o servidor IPC e chama os manipuladores de assinatura. Você deve considerar esse comportamento síncrono ao escrever funções de manipulador de assinaturas.

- Reutilize um cliente IPC

  Depois de criar um cliente IPC, mantenha-o aberto e reutilize-o para todas as operações de IPC. A criação de vários clientes usa recursos extras e pode resultar em vazamentos de recursos.

- Execute o código de bloqueio de forma assíncrona

  O cliente IPC V1 não pode enviar novas solicitações ou processar novas mensagens de eventos enquanto o thread está bloqueado. Você deve executar o código de bloqueio em um encadeamento separado, executado a partir da função de manipulador. O código de bloqueio inclui `sleep` chamadas, loops que são executados continuamente e solicitações de E/S síncronas que demoram para serem concluídas.

- Enviar novas solicitações de IPC de forma assíncrona

  O cliente IPC V1 não pode enviar uma nova solicitação de dentro das funções do manipulador de assinatura, porque a solicitação bloqueia a função do manipulador se você esperar por uma resposta. Você deve enviar solicitações de IPC em um thread separado que você executa a partir da função de manipulador.

- Lidar com exceções

  O cliente IPC V1 não lida com exceções não detectadas nas funções do manipulador de assinaturas. Se sua função de manipulador gerar uma exceção, a assinatura será encerrada

e a exceção não aparecerá nos registros do componente. Você deve capturar exceções nas funções do manipulador para manter a assinatura aberta e registrar os erros que ocorrem no seu código.

Publique/assine mensagens locais

As mensagens de publicação/assinatura (pubsub) permitem que você envie e receba mensagens sobre tópicos. Os componentes podem publicar mensagens em tópicos para enviar mensagens para outros componentes. Em seguida, os componentes inscritos nesse tópico podem agir nas mensagens que recebem.

> **Note**
>
> Você não pode usar esse serviço IPC de publicação/assinatura para publicar ou assinar o MQTT. AWS IoT Core Para obter mais informações sobre como trocar mensagens com o AWS IoT Core MQTT, consulte Publique/assine mensagens MQTT AWS IoT Core.

Tópicos

Versões mínimas do SDK

A tabela a seguir lista as versões mínimas do AWS IoT Device SDK que você deve usar para publicar e assinar mensagens de e para tópicos locais.

SDK	Versão mínima	
AWS IoT Device SDK para Java v2	v1.2.10	

SDK	Versão mínima	
AWS IoT Device SDK para Python v2	v1.5.3	
AWS IoT Device SDK para C++ v2	v1.17.0	
AWS IoT Device SDK for JavaScript v2	v1.12.0	

Autorização

Para usar mensagens locais de publicação/assinatura em um componente personalizado, você deve definir políticas de autorização que permitam que seu componente envie e receba mensagens para tópicos. Para obter informações sobre como definir políticas de autorização, consulteAutorize componentes a realizar operações de IPC.

As políticas de autorização para publicar/assinar mensagens têm as seguintes propriedades.

Identificador de serviço IPC: `aws.greengrass.ipc.pubsub`

Operation	Descrição	Recursos
`aws.greengrass#PublishToTopic`	Permite que um componente publique mensagens nos tópicos que você especificar.	Uma sequência de tópicos, como `test/topic` . Use um `*` para combinar qualquer combinação de caracteres em um tópico. Essa string de tópico não suporta caracteres curinga de tópico (#e+) do MQTT.
`aws.greengrass#SubscribeToTopic`	Permite que um componente assine mensagens para os tópicos que você especificar.	Uma sequência de tópicos, como `test/topic` . Use um `*` para combinar qualquer

Operation	Descrição	Recursos
		combinação de caracteres em um tópico. No Greengrass nucleus v2.6.0 e versões posteriores, você pode se inscrever em tópicos que contêm curingas de tópicos do MQTT (e). # + Essa string de tópico suporta curingas de tópico MQTT como caracteres literais. Por exemplo, se a política de autorização de um component e conceder acesso `atest/topic/#` , o component e pode se inscrever`test/topic/#` , mas não pode assinar`test/topic/filter`.

Operation	Descrição	Recursos
*	Permite que um componente publique e assine mensagens para os tópicos que você especificar.	Uma sequência de tópicos, como `test/topic` . Use um `*` para combinar qualquer combinação de caracteres em um tópico. No Greengrass nucleus v2.6.0 e versões posteriores, você pode se inscrever em tópicos que contêm curingas de tópicos do MQTT (e). # + Essa string de tópico suporta curingas de tópico MQTT como caracteres literais. Por exemplo, se a política de autorização de um component e conceder acesso `a test/topic/#` , o component e pode se inscrever `test/topic/#` , mas não pode assinar `test/topic/filter`.

Exemplos de políticas de autorização

Você pode consultar o exemplo de política de autorização a seguir para ajudá-lo a configurar políticas de autorização para seus componentes.

Example Exemplo de política de autorização

O exemplo de política de autorização a seguir permite que um componente publique e assine todos os tópicos.

```
{
  "accessControl": {
    "aws.greengrass.ipc.pubsub": {
```

```
      "com.example.MyLocalPubSubComponent:pubsub:1": {
        "policyDescription": "Allows access to publish/subscribe to all topics.",
        "operations": [
          "aws.greengrass#PublishToTopic",
          "aws.greengrass#SubscribeToTopic"
        ],
        "resources": [
          "*"
        ]
      }
    }
  }
}
```

PublishToTopic

Publique uma mensagem em um tópico.

Solicitação

A solicitação dessa operação tem os seguintes parâmetros:

`topic`

O tópico no qual publicar a mensagem.

`publishMessage`(Python:) `publish_message`

A mensagem a ser publicada. Esse objeto,`PublishMessage`, contém as seguintes informações. Você deve especificar um dos `jsonMessage` `binaryMessage` e.

`jsonMessage`(Python:) `json_message`

(Opcional) Uma mensagem JSON. Esse objeto,`JsonMessage`, contém as seguintes informações:

`message`

A mensagem JSON como um objeto.

`context`

O contexto da mensagem, como o tópico em que a mensagem foi publicada.

Esse recurso está disponível para a versão 2.6.0 e posterior do componente núcleo do Greengrass. A tabela a seguir lista as versões mínimas do AWS IoT Device SDK que você deve usar para acessar o contexto da mensagem.

SDK	Versão mínima
AWS IoT Device SDK para Java v2	v1.9.3
AWS IoT Device SDK para Python v2	v1.11.3
AWS IoT Device SDK para C++ v2	v1.18.4
AWS IoT Device SDK for JavaScript v2	v1.12.0

> **Note**
>
> O software AWS IoT Greengrass Core usa os mesmos objetos de mensagem nas `SubscribeToTopic` operações `PublishToTopic` e. O software AWS IoT Greengrass Core define esse objeto de contexto nas mensagens quando você se inscreve e ignora esse objeto de contexto nas mensagens que você publica.

Esse objeto,`MessageContext`, contém as seguintes informações:

`topic`

O tópico em que a mensagem foi publicada.

`binaryMessage`(Python:) `binary_message`

(Opcional) Uma mensagem binária. Esse objeto,`BinaryMessage`, contém as seguintes informações:

`message`

A mensagem binária como uma bolha.

`context`

O contexto da mensagem, como o tópico em que a mensagem foi publicada.

Esse recurso está disponível para a versão 2.6.0 e posterior do componente núcleo do Greengrass. A tabela a seguir lista as versões mínimas do AWS IoT Device SDK que você deve usar para acessar o contexto da mensagem.

SDK	Versão mínima	
AWS IoT Device SDK para Java v2	v1.9.3	
AWS IoT Device SDK para Python v2	v1.11.3	
AWS IoT Device SDK para C++ v2	v1.18.4	
AWS IoT Device SDK for JavaScript v2	v1.12.0	

> **Note**
>
> O software AWS IoT Greengrass Core usa os mesmos objetos de mensagem nas `SubscribeToTopic` operações `PublishToTopic` e. O software AWS IoT Greengrass Core define esse objeto de contexto nas mensagens quando você se inscreve e ignora esse objeto de contexto nas mensagens que você publica.

Esse objeto,`MessageContext`, contém as seguintes informações:

`topic`

O tópico em que a mensagem foi publicada.

Resposta

Essa operação não fornece nenhuma informação em sua resposta.

Exemplos

Os exemplos a seguir demonstram como chamar essa operação no código de componente personalizado.

Java (IPC client V2)

Example Exemplo: publicar uma mensagem binária

```
package com.aws.greengrass.docs.samples.ipc;

import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClientV2;
import software.amazon.awssdk.aws.greengrass.model.BinaryMessage;
import software.amazon.awssdk.aws.greengrass.model.PublishMessage;
import software.amazon.awssdk.aws.greengrass.model.PublishToTopicRequest;
import software.amazon.awssdk.aws.greengrass.model.PublishToTopicResponse;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;

import java.nio.charset.StandardCharsets;

public class PublishToTopicV2 {

    public static void main(String[] args) {
        String topic = args[0];
        String message = args[1];
        try (GreengrassCoreIPCClientV2 ipcClient =
 GreengrassCoreIPCClientV2.builder().build()) {
            PublishToTopicV2.publishBinaryMessageToTopic(ipcClient, topic,
 message);
            System.out.println("Successfully published to topic: " + topic);
        } catch (Exception e) {
            if (e.getCause() instanceof UnauthorizedError) {
                System.err.println("Unauthorized error while publishing to topic: "
 + topic);
            } else {
                System.err.println("Exception occurred when using IPC.");
            }
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static PublishToTopicResponse publishBinaryMessageToTopic(
```

```
            GreengrassCoreIPCClientV2 ipcClient, String topic, String message)
 throws InterruptedException {
        BinaryMessage binaryMessage =
                new
 BinaryMessage().withMessage(message.getBytes(StandardCharsets.UTF_8));
        PublishMessage publishMessage = new
 PublishMessage().withBinaryMessage(binaryMessage);
        PublishToTopicRequest publishToTopicRequest =
                new
 PublishToTopicRequest().withTopic(topic).withPublishMessage(publishMessage);
        return ipcClient.publishToTopic(publishToTopicRequest);
    }
}
```

Python (IPC client V2)

Example Exemplo: publicar uma mensagem binária

```
import sys
import traceback

from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2
from awsiot.greengrasscoreipc.model import (
    PublishMessage,
    BinaryMessage
)


def main():
    args = sys.argv[1:]
    topic = args[0]
    message = args[1]

    try:
        ipc_client = GreengrassCoreIPCClientV2()
        publish_binary_message_to_topic(ipc_client, topic, message)
        print('Successfully published to topic: ' + topic)
    except Exception:
        print('Exception occurred', file=sys.stderr)
        traceback.print_exc()
        exit(1)


def publish_binary_message_to_topic(ipc_client, topic, message):
```

```
    binary_message = BinaryMessage(message=bytes(message, 'utf-8'))
    publish_message = PublishMessage(binary_message=binary_message)
    return ipc_client.publish_to_topic(topic=topic,
 publish_message=publish_message)


if __name__ == '__main__':
    main()
```

C++

Example Exemplo: publicar uma mensagem binária

```
#include <iostream>

#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        // Handle connection to IPC service.
    }

    void OnDisconnectCallback(RpcError error) override {
        // Handle disconnection from IPC service.
    }

    bool OnErrorCallback(RpcError error) override {
        // Handle IPC service connection error.
        return true;
    }
};

int main() {
    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
```

```
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    String topic("my/topic");
    String message("Hello, World!");
    int timeout = 10;

    PublishToTopicRequest request;
    Vector<uint8_t> messageData({message.begin(), message.end()});
    BinaryMessage binaryMessage;
    binaryMessage.SetMessage(messageData);
    PublishMessage publishMessage;
    publishMessage.SetBinaryMessage(binaryMessage);
    request.SetTopic(topic);
    request.SetPublishMessage(publishMessage);

    auto operation = ipcClient.NewPublishToTopic();
    auto activate = operation->Activate(request, nullptr);
    activate.wait();

    auto responseFuture = operation->GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from Greengrass
Core." << std::endl;
        exit(-1);
    }

    auto response = responseFuture.get();
    if (!response) {
        // Handle error.
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
            (void)error;
            // Handle operation error.
        } else {
            // Handle RPC error.
        }
    }
    return 0;
```

```
}
```

JavaScript

Example Exemplo: publicar uma mensagem binária

```
import * as greengrasscoreipc from "aws-iot-device-sdk-v2/dist/greengrasscoreipc";
import {BinaryMessage, PublishMessage, PublishToTopicRequest} from "aws-iot-device-
sdk-v2/dist/greengrasscoreipc/model";

class PublishToTopic {
    private ipcClient : greengrasscoreipc.Client
    private readonly topic : string;
    private readonly messageString : string;

    constructor() {
        // define your own constructor, e.g.
        this.topic = "<define_your_topic>";
        this.messageString = "<define_your_message_string>";
        this.publishToTopic().then(r => console.log("Started workflow"));
    }

    private async publishToTopic() {
        try {
            this.ipcClient = await getIpcClient();

            const binaryMessage : BinaryMessage = {
                message: this.messageString
            }

            const publishMessage : PublishMessage = {
                binaryMessage: binaryMessage
            }

            const request : PublishToTopicRequest = {
                topic: this.topic,
                publishMessage: publishMessage
            }

            this.ipcClient.publishToTopic(request).finally(() =>
 console.log(`Published message ${publishMessage.binaryMessage?.message} to topic`))
```

```
        } catch (e) {
            // parse the error depending on your use cases
            throw e
        }
    }
}


export async function getIpcClient(){
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

// starting point
const publishToTopic = new PublishToTopic();
```

SubscribeToTopic

Assine mensagens sobre um tópico.

Essa operação é uma operação de assinatura em que você assina um fluxo de mensagens de eventos. Para usar essa operação, defina um manipulador de resposta de fluxo com funções que manipulam mensagens de eventos, erros e encerramento de fluxo. Para ter mais informações, consulte Inscreva-se nos streams de eventos do IPC.

Tipo de mensagem do evento: `SubscriptionResponseMessage`

Solicitação

A solicitação dessa operação tem os seguintes parâmetros:

`topic`

O tópico no qual se inscrever.

> **Note**
>
> No Greengrass nucleus v2.6.0 e versões posteriores, este tópico oferece suporte aos curingas do tópico MQTT (e). # +

`receiveMode`(Python:) `receive_mode`

(Opcional) O comportamento que especifica se o componente recebe mensagens de si mesmo. Você pode alterar esse comportamento para permitir que um componente atue em suas próprias mensagens. O comportamento padrão depende se o tópico contém um curinga MQTT. Escolha uma das seguintes opções:

- `RECEIVE_ALL_MESSAGES`— Receba todas as mensagens que correspondam ao tópico, incluindo mensagens do componente que se inscreve.

  Esse modo é a opção padrão quando você se inscreve em um tópico que não contém um curinga MQTT.
- `RECEIVE_MESSAGES_FROM_OTHERS`— Receba todas as mensagens que correspondam ao tópico, exceto as mensagens do componente que se inscreve.

  Esse modo é a opção padrão quando você se inscreve em um tópico que contém um curinga MQTT.

Esse recurso está disponível para a versão 2.6.0 e posterior do componente núcleo do Greengrass. A tabela a seguir lista as versões mínimas do AWS IoT Device SDK que você deve usar para definir o modo de recepção.

SDK	Versão mínima	
AWS IoT Device SDK para Java v2	v1.9.3	
AWS IoT Device SDK para Python v2	v1.11.3	

SDK	Versão mínima
AWS IoT Device SDK para C++ v2	v1.18.4
AWS IoT Device SDK para JavaScript v2	v1.12.0

Resposta

A resposta dessa operação tem as seguintes informações:

`messages`

O fluxo de mensagens. Esse objeto,`SubscriptionResponseMessage`, contém as seguintes informações. Cada mensagem contém `jsonMessage` ou`binaryMessage`.

`jsonMessage`(Python:) `json_message`

(Opcional) Uma mensagem JSON. Esse objeto,`JsonMessage`, contém as seguintes informações:

`message`

A mensagem JSON como um objeto.

`context`

O contexto da mensagem, como o tópico em que a mensagem foi publicada.

Esse recurso está disponível para a versão 2.6.0 e posterior do componente núcleo do Greengrass. A tabela a seguir lista as versões mínimas do AWS IoT Device SDK que você deve usar para acessar o contexto da mensagem.

SDK	Versão mínima
AWS IoT Device SDK para Java v2	v1.9.3
AWS IoT Device SDK para Python v2	v1.11.3

SDK	Versão mínima	
AWS IoT Device SDK para C++ v2	v1.18.4	
AWS IoT Device SDK for JavaScript v2	v1.12.0	

> **Note**
>
> O software AWS IoT Greengrass Core usa os mesmos objetos de mensagem nas `SubscribeToTopic` operações `PublishToTopic` e. O software AWS IoT Greengrass Core define esse objeto de contexto nas mensagens quando você se inscreve e ignora esse objeto de contexto nas mensagens que você publica.

Esse objeto,`MessageContext`, contém as seguintes informações:

`topic`

O tópico em que a mensagem foi publicada.

`binaryMessage`(Python:) `binary_message`

(Opcional) Uma mensagem binária. Esse objeto,`BinaryMessage`, contém as seguintes informações:

`message`

A mensagem binária como uma bolha.

`context`

O contexto da mensagem, como o tópico em que a mensagem foi publicada.

Esse recurso está disponível para a versão 2.6.0 e posterior do componente núcleo do Greengrass. A tabela a seguir lista as versões mínimas do AWS IoT Device SDK que você deve usar para acessar o contexto da mensagem.

SDK	Versão mínima	
AWS IoT Device SDK para Java v2	v1.9.3	
AWS IoT Device SDK para Python v2	v1.11.3	
AWS IoT Device SDK para C++ v2	v1.18.4	
AWS IoT Device SDK for JavaScript v2	v1.12.0	

> **Note**
>
> O software AWS IoT Greengrass Core usa os mesmos objetos de mensagem nas `SubscribeToTopic` operações `PublishToTopic` e. O software AWS IoT Greengrass Core define esse objeto de contexto nas mensagens quando você se inscreve e ignora esse objeto de contexto nas mensagens que você publica.

Esse objeto,`MessageContext`, contém as seguintes informações:

`topic`

O tópico em que a mensagem foi publicada.

`topicName`(Python:) `topic_name`

O tópico no qual a mensagem foi publicada.

> **Note**
>
> Essa propriedade não é usada atualmente. No Greengrass nucleus v2.6.0 e versões posteriores, você pode obter o `(jsonMessage|binaryMessage).context.topic` valor de a para obter o tópico em `SubscriptionResponseMessage` que a mensagem foi publicada.

Exemplos

Os exemplos a seguir demonstram como chamar essa operação no código de componente personalizado.

Java (IPC client V2)

Example Exemplo: Inscrever-se em mensagens locais de publicação/assinatura

```
package com.aws.greengrass.docs.samples.ipc;

import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClientV2;
import software.amazon.awssdk.aws.greengrass.SubscribeToTopicResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.*;

import java.nio.charset.StandardCharsets;
import java.util.Optional;

public class SubscribeToTopicV2 {

    public static void main(String[] args) {
        String topic = args[0];
        try (GreengrassCoreIPCClientV2 ipcClient =
 GreengrassCoreIPCClientV2.builder().build()) {
            SubscribeToTopicRequest request = new
 SubscribeToTopicRequest().withTopic(topic);
            GreengrassCoreIPCClientV2.StreamingResponse<SubscribeToTopicResponse,
                    SubscribeToTopicResponseHandler> response =
                    ipcClient.subscribeToTopic(request,
 SubscribeToTopicV2::onStreamEvent,
                            Optional.of(SubscribeToTopicV2::onStreamError),
                            Optional.of(SubscribeToTopicV2::onStreamClosed));
            SubscribeToTopicResponseHandler responseHandler =
 response.getHandler();
            System.out.println("Successfully subscribed to topic: " + topic);

            // Keep the main thread alive, or the process will exit.
            try {
                while (true) {
                    Thread.sleep(10000);
                }
            } catch (InterruptedException e) {
                System.out.println("Subscribe interrupted.");
```

```
            }

            // To stop subscribing, close the stream.
            responseHandler.closeStream();
        } catch (Exception e) {
            if (e.getCause() instanceof UnauthorizedError) {
                System.err.println("Unauthorized error while publishing to topic: "
+ topic);
            } else {
                System.err.println("Exception occurred when using IPC.");
            }
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static void onStreamEvent(SubscriptionResponseMessage
subscriptionResponseMessage) {
        try {
            BinaryMessage binaryMessage =
subscriptionResponseMessage.getBinaryMessage();
            String message = new String(binaryMessage.getMessage(),
StandardCharsets.UTF_8);
            String topic = binaryMessage.getContext().getTopic();
            System.out.printf("Received new message on topic %s: %s%n", topic,
message);
        } catch (Exception e) {
            System.err.println("Exception occurred while processing subscription
response " +
                    "message.");
            e.printStackTrace();
        }
    }

    public static boolean onStreamError(Throwable error) {
        System.err.println("Received a stream error.");
        error.printStackTrace();
        return false; // Return true to close stream, false to keep stream open.
    }

    public static void onStreamClosed() {
        System.out.println("Subscribe to topic stream closed.");
    }
```

```
}
```

Python (IPC client V2)

Example Exemplo: Inscrever-se em mensagens locais de publicação/assinatura

```
import sys
import time
import traceback

from awsiot.greengrasscoreipc.clientv2 import GreengrassCoreIPCClientV2
from awsiot.greengrasscoreipc.model import (
    SubscriptionResponseMessage,
    UnauthorizedError
)


def main():
    args = sys.argv[1:]
    topic = args[0]

    try:
        ipc_client = GreengrassCoreIPCClientV2()
        # Subscription operations return a tuple with the response and the
 operation.
        _, operation = ipc_client.subscribe_to_topic(topic=topic,
 on_stream_event=on_stream_event,

 on_stream_error=on_stream_error, on_stream_closed=on_stream_closed)
        print('Successfully subscribed to topic: ' + topic)

        # Keep the main thread alive, or the process will exit.
        try:
            while True:
                time.sleep(10)
        except InterruptedError:
            print('Subscribe interrupted.')

        # To stop subscribing, close the stream.
        operation.close()
    except UnauthorizedError:
        print('Unauthorized error while subscribing to topic: ' +
              topic, file=sys.stderr)
        traceback.print_exc()
```

```
        exit(1)
    except Exception:
        print('Exception occurred', file=sys.stderr)
        traceback.print_exc()
        exit(1)


def on_stream_event(event: SubscriptionResponseMessage) -> None:
    try:
        message = str(event.binary_message.message, 'utf-8')
        topic = event.binary_message.context.topic
        print('Received new message on topic %s: %s' % (topic, message))
    except:
        traceback.print_exc()


def on_stream_error(error: Exception) -> bool:
    print('Received a stream error.', file=sys.stderr)
    traceback.print_exc()
    return False  # Return True to close stream, False to keep stream open.


def on_stream_closed() -> None:
    print('Subscribe to topic stream closed.')


if __name__ == '__main__':
    main()
```

C++

Example Exemplo: Inscrever-se em mensagens locais de publicação/assinatura

```
#include <iostream>

#include </crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class SubscribeResponseHandler : public SubscribeToTopicStreamHandler {
    public:
        virtual ~SubscribeResponseHandler() {}
```

```
    private:
        void OnStreamEvent(SubscriptionResponseMessage *response) override {
            auto jsonMessage = response->GetJsonMessage();
            if (jsonMessage.has_value() &&
 jsonMessage.value().GetMessage().has_value()) {
                auto messageString =
 jsonMessage.value().GetMessage().value().View().WriteReadable();
                // Handle JSON message.
            } else {
                auto binaryMessage = response->GetBinaryMessage();
                if (binaryMessage.has_value() &&
 binaryMessage.value().GetMessage().has_value()) {
                    auto messageBytes = binaryMessage.value().GetMessage().value();
                    std::string messageString(messageBytes.begin(),
 messageBytes.end());
                    // Handle binary message.
                }
            }
        }

        bool OnStreamError(OperationError *error) override {
            // Handle error.
            return false; // Return true to close stream, false to keep stream open.
        }

        void OnStreamClosed() override {
            // Handle close.
        }
};

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        // Handle connection to IPC service.
    }

    void OnDisconnectCallback(RpcError error) override {
        // Handle disconnection from IPC service.
    }

    bool OnErrorCallback(RpcError error) override {
        // Handle IPC service connection error.
        return true;
    }
```

```
};

int main() {
    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
 connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    String topic("my/topic");
    int timeout = 10;

    SubscribeToTopicRequest request;
    request.SetTopic(topic);

    //SubscribeResponseHandler streamHandler;
    auto streamHandler = MakeShared<SubscribeResponseHandler>(DefaultAllocator());
    auto operation = ipcClient.NewSubscribeToTopic(streamHandler);
    auto activate = operation->Activate(request, nullptr);
    activate.wait();

    auto responseFuture = operation->GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
 std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from Greengrass
 Core." << std::endl;
        exit(-1);
    }

    auto response = responseFuture.get();
    if (!response) {
        // Handle error.
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
            (void)error;
            // Handle operation error.
```

```
        } else {
            // Handle RPC error.
        }
        exit(-1);
    }

    // Keep the main thread alive, or the process will exit.
    while (true) {
        std::this_thread::sleep_for(std::chrono::seconds(10));
    }

    operation->Close();
    return 0;
}
```

JavaScript

Example Exemplo: Inscrever-se em mensagens locais de publicação/assinatura

```
import * as greengrasscoreipc from "aws-iot-device-sdk-v2/dist/greengrasscoreipc";
import {SubscribeToTopicRequest, SubscriptionResponseMessage} from "aws-iot-device-
sdk-v2/dist/greengrasscoreipc/model";
import {RpcError} from "aws-iot-device-sdk-v2/dist/eventstream_rpc";

class SubscribeToTopic {
    private ipcClient : greengrasscoreipc.Client
    private readonly topic : string;

    constructor() {
        // define your own constructor, e.g.
        this.topic = "<define_your_topic>";
        this.subscribeToTopic().then(r => console.log("Started workflow"));
    }

    private async subscribeToTopic() {
        try {
            this.ipcClient = await getIpcClient();

            const subscribeToTopicRequest : SubscribeToTopicRequest = {
                topic: this.topic,
            }
```

```
            const streamingOperation =
 this.ipcClient.subscribeToTopic(subscribeToTopicRequest, undefined); //
 conditionally apply options

            streamingOperation.on("message", (message: SubscriptionResponseMessage)
 => {
                // parse the message depending on your use cases, e.g.
                if(message.binaryMessage && message.binaryMessage.message) {
                    const receivedMessage =
 message.binaryMessage?.message.toString();
                }
            });

            streamingOperation.on("streamError", (error : RpcError) => {
                // define your own error handling logic
            })

            streamingOperation.on("ended", () => {
                // define your own logic
            })

            await streamingOperation.activate();

            // Keep the main thread alive, or the process will exit.
            await new Promise((resolve) => setTimeout(resolve, 10000))
        } catch (e) {
            // parse the error depending on your use cases
            throw e
        }
    }
}

export async function getIpcClient(){
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
```

```
    }
}

// starting point
const subscribeToTopic = new SubscribeToTopic();
```

Exemplos

Use os exemplos a seguir para aprender a usar o serviço IPC de publicação/assinatura em seus componentes.

Exemplo de editor de publicação/assinatura (Java, cliente IPC V1)

O exemplo de receita a seguir permite que o componente seja publicado em todos os tópicos.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubPublisherJava",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that publishes messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.PubSubPublisherJava:pubsub:1": {
            "policyDescription": "Allows access to publish to all topics.",
            "operations": [
              "aws.greengrass#PublishToTopic"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
```

```
      "Lifecycle": {
        "run": "java -jar {artifacts:path}/PubSubPublisher.jar"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PubSubPublisherJava
ComponentVersion: '1.0.0'
ComponentDescription: A component that publishes messages.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.pubsub:
        'com.example.PubSubPublisherJava:pubsub:1':
          policyDescription: Allows access to publish to all topics.
          operations:
            - 'aws.greengrass#PublishToTopic'
          resources:
            - '*'
Manifests:
  - Lifecycle:
      run: |-
        java -jar {artifacts:path}/PubSubPublisher.jar
```

O exemplo de aplicativo Java a seguir demonstra como usar o serviço IPC de publicação/assinatura para publicar mensagens em outros componentes.

```
/* Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: Apache-2.0 */

package com.example.ipc.pubsub;

import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.model.*;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;
```

```
import java.nio.charset.StandardCharsets;
import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class PubSubPublisher {

    public static void main(String[] args) {
        String message = "Hello from the pub/sub publisher (Java).";
        String topic = "test/topic/java";

        try (EventStreamRPCConnection eventStreamRPCConnection =
 IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient = new
 GreengrassCoreIPCClient(eventStreamRPCConnection);

            while (true) {
                PublishToTopicRequest publishRequest = new PublishToTopicRequest();
                PublishMessage publishMessage = new PublishMessage();
                BinaryMessage binaryMessage = new BinaryMessage();
                binaryMessage.setMessage(message.getBytes(StandardCharsets.UTF_8));
                publishMessage.setBinaryMessage(binaryMessage);
                publishRequest.setPublishMessage(publishMessage);
                publishRequest.setTopic(topic);
                CompletableFuture<PublishToTopicResponse> futureResponse = ipcClient
                        .publishToTopic(publishRequest,
 Optional.empty()).getResponse();

                try {
                    futureResponse.get(10, TimeUnit.SECONDS);
                    System.out.println("Successfully published to topic: " + topic);
                } catch (TimeoutException e) {
                    System.err.println("Timeout occurred while publishing to topic: " +
 topic);
                } catch (ExecutionException e) {
                    if (e.getCause() instanceof UnauthorizedError) {
                        System.err.println("Unauthorized error while publishing to
 topic: " + topic);
                    } else {
                        System.err.println("Execution exception while publishing to
 topic: " + topic);
                    }
```

```
                    throw e;
                }
                Thread.sleep(5000);
            }
        } catch (InterruptedException e) {
            System.out.println("Publisher interrupted.");
        } catch (Exception e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
        }
    }
}
```

Exemplo de assinante de publicação/assinatura (Java, cliente IPC V1)

O exemplo de receita a seguir permite que o componente se inscreva em todos os tópicos.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubSubscriberJava",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that subscribes to messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.PubSubSubscriberJava:pubsub:1": {
            "policyDescription": "Allows access to subscribe to all topics.",
            "operations": [
              "aws.greengrass#SubscribeToTopic"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
```

```
    {
      "Lifecycle": {
        "run": "java -jar {artifacts:path}/PubSubSubscriber.jar"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PubSubSubscriberJava
ComponentVersion: '1.0.0'
ComponentDescription: A component that subscribes to messages.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.pubsub:
        'com.example.PubSubSubscriberJava:pubsub:1':
          policyDescription: Allows access to subscribe to all topics.
          operations:
            - 'aws.greengrass#SubscribeToTopic'
          resources:
            - '*'
Manifests:
  - Lifecycle:
      run: |-
        java -jar {artifacts:path}/PubSubSubscriber.jar
```

O exemplo de aplicativo Java a seguir demonstra como usar o serviço IPC de publicação/assinatura para assinar mensagens em outros componentes.

```
/* Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
 * SPDX-License-Identifier: Apache-2.0 */

package com.example.ipc.pubsub;

import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.SubscribeToTopicResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.SubscribeToTopicRequest;
```

```
import software.amazon.awssdk.aws.greengrass.model.SubscribeToTopicResponse;
import software.amazon.awssdk.aws.greengrass.model.SubscriptionResponseMessage;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;
import software.amazon.awssdk.eventstreamrpc.StreamResponseHandler;

import java.nio.charset.StandardCharsets;
import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class PubSubSubscriber {

    public static void main(String[] args) {
        String topic = "test/topic/java";

        try (EventStreamRPCConnection eventStreamRPCConnection =
 IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient = new
 GreengrassCoreIPCClient(eventStreamRPCConnection);

            SubscribeToTopicRequest subscribeRequest = new SubscribeToTopicRequest();
            subscribeRequest.setTopic(topic);
            SubscribeToTopicResponseHandler operationResponseHandler = ipcClient
                    .subscribeToTopic(subscribeRequest, Optional.of(new
 SubscribeResponseHandler()));
            CompletableFuture<SubscribeToTopicResponse> futureResponse =
 operationResponseHandler.getResponse();

            try {
                futureResponse.get(10, TimeUnit.SECONDS);
                System.out.println("Successfully subscribed to topic: " + topic);
            } catch (TimeoutException e) {
                System.err.println("Timeout occurred while subscribing to topic: " +
 topic);
                throw e;
            } catch (ExecutionException e) {
                if (e.getCause() instanceof UnauthorizedError) {
                    System.err.println("Unauthorized error while subscribing to topic:
 " + topic);
                } else {
```

```
                    System.err.println("Execution exception while subscribing to topic:
" + topic);
                }
                throw e;
            }

            // Keep the main thread alive, or the process will exit.
            try {
                while (true) {
                    Thread.sleep(10000);
                }
            } catch (InterruptedException e) {
                System.out.println("Subscribe interrupted.");
            }
        } catch (Exception e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
        }
    }

    private static class SubscribeResponseHandler implements
StreamResponseHandler<SubscriptionResponseMessage> {

        @Override
        public void onStreamEvent(SubscriptionResponseMessage
subscriptionResponseMessage) {
            try {
                String message = new
String(subscriptionResponseMessage.getBinaryMessage()
                        .getMessage(), StandardCharsets.UTF_8);
                System.out.println("Received new message: " + message);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }

        @Override
        public boolean onStreamError(Throwable error) {
            System.err.println("Received a stream error.");
            error.printStackTrace();
            return false; // Return true to close stream, false to keep stream open.
        }
```

```
        @Override
        public void onStreamClosed() {
            System.out.println("Subscribe to topic stream closed.");
        }
    }
}
```

Exemplo de editor de publicação/assinatura (Python, cliente IPC V1)

O exemplo de receita a seguir permite que o componente seja publicado em todos os tópicos.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubPublisherPython",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that publishes messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.PubSubPublisherPython:pubsub:1": {
            "policyDescription": "Allows access to publish to all topics.",
            "operations": [
              "aws.greengrass#PublishToTopic"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "python3 -m pip install --user awsiotsdk",
        "run": "python3 -u {artifacts:path}/pubsub_publisher.py"
```

```
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "install": "py -3 -m pip install --user awsiotsdk",
        "run": "py -3 -u {artifacts:path}/pubsub_publisher.py"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PubSubPublisherPython
ComponentVersion: 1.0.0
ComponentDescription: A component that publishes messages.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.pubsub:
        com.example.PubSubPublisherPython:pubsub:1:
          policyDescription: Allows access to publish to all topics.
          operations:
            - aws.greengrass#PublishToTopic
          resources:
            - "*"
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: python3 -m pip install --user awsiotsdk
      run: python3 -u {artifacts:path}/pubsub_publisher.py
  - Platform:
      os: windows
    Lifecycle:
      install: py -3 -m pip install --user awsiotsdk
      run: py -3 -u {artifacts:path}/pubsub_publisher.py
```

O exemplo de aplicativo Python a seguir demonstra como usar o serviço IPC de publicação/assinatura para publicar mensagens em outros componentes.

```
import concurrent.futures
import sys
import time
import traceback

import awsiot.greengrasscoreipc
from awsiot.greengrasscoreipc.model import (
    PublishToTopicRequest,
    PublishMessage,
    BinaryMessage,
    UnauthorizedError
)


topic = "test/topic/python"
message = "Hello from the pub/sub publisher (Python)."
TIMEOUT = 10

try:
    ipc_client = awsiot.greengrasscoreipc.connect()

    while True:
        request = PublishToTopicRequest()
        request.topic = topic
        publish_message = PublishMessage()
        publish_message.binary_message = BinaryMessage()
        publish_message.binary_message.message = bytes(message, "utf-8")
        request.publish_message = publish_message
        operation = ipc_client.new_publish_to_topic()
        operation.activate(request)
        future_response = operation.get_response()

        try:
            future_response.result(TIMEOUT)
            print('Successfully published to topic: ' + topic)
        except concurrent.futures.TimeoutError:
            print('Timeout occurred while publishing to topic: ' + topic,
 file=sys.stderr)
        except UnauthorizedError as e:
```

```
            print('Unauthorized error while publishing to topic: ' + topic,
 file=sys.stderr)
            raise e
        except Exception as e:
            print('Exception while publishing to topic: ' + topic, file=sys.stderr)
            raise e
        time.sleep(5)
except InterruptedError:
    print('Publisher interrupted.')
except Exception:
    print('Exception occurred when using IPC.', file=sys.stderr)
    traceback.print_exc()
    exit(1)
```

Exemplo de assinante de publicação/assinatura (Python, cliente IPC V1)

O exemplo de receita a seguir permite que o componente se inscreva em todos os tópicos.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubSubscriberPython",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that subscribes to messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.PubSubSubscriberPython:pubsub:1": {
            "policyDescription": "Allows access to subscribe to all topics.",
            "operations": [
              "aws.greengrass#SubscribeToTopic"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
```

```
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "python3 -m pip install --user awsiotsdk",
        "run": "python3 -u {artifacts:path}/pubsub_subscriber.py"
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "install": "py -3 -m pip install --user awsiotsdk",
        "run": "py -3 -u {artifacts:path}/pubsub_subscriber.py"
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PubSubSubscriberPython
ComponentVersion: 1.0.0
ComponentDescription: A component that subscribes to messages.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.pubsub:
        com.example.PubSubSubscriberPython:pubsub:1:
          policyDescription: Allows access to subscribe to all topics.
          operations:
            - aws.greengrass#SubscribeToTopic
          resources:
            - "*"
Manifests:
  - Platform:
      os: linux
    Lifecycle:
```

```
      install: python3 -m pip install --user awsiotsdk
      run: python3 -u {artifacts:path}/pubsub_subscriber.py
  - Platform:
      os: windows
    Lifecycle:
      install: py -3 -m pip install --user awsiotsdk
      run: py -3 -u {artifacts:path}/pubsub_subscriber.py
```

O exemplo de aplicativo Python a seguir demonstra como usar o serviço IPC de publicação/assinatura para assinar mensagens em outros componentes.

```
import concurrent.futures
import sys
import time
import traceback

import awsiot.greengrasscoreipc
import awsiot.greengrasscoreipc.client as client
from awsiot.greengrasscoreipc.model import (
    SubscribeToTopicRequest,
    SubscriptionResponseMessage,
    UnauthorizedError
)

topic = "test/topic/python"
TIMEOUT = 10


class StreamHandler(client.SubscribeToTopicStreamHandler):
    def __init__(self):
        super().__init__()

    def on_stream_event(self, event: SubscriptionResponseMessage) -> None:
        try:
            message = str(event.binary_message.message, "utf-8")
            print("Received new message: " + message)
        except:
            traceback.print_exc()

    def on_stream_error(self, error: Exception) -> bool:
        print("Received a stream error.", file=sys.stderr)
        traceback.print_exc()
```

```
        return False  # Return True to close stream, False to keep stream open.

    def on_stream_closed(self) -> None:
        print('Subscribe to topic stream closed.')


try:
    ipc_client = awsiot.greengrasscoreipc.connect()

    request = SubscribeToTopicRequest()
    request.topic = topic
    handler = StreamHandler()
    operation = ipc_client.new_subscribe_to_topic(handler)
    operation.activate(request)
    future_response = operation.get_response()

    try:
        future_response.result(TIMEOUT)
        print('Successfully subscribed to topic: ' + topic)
    except concurrent.futures.TimeoutError as e:
        print('Timeout occurred while subscribing to topic: ' + topic,
 file=sys.stderr)
        raise e
    except UnauthorizedError as e:
        print('Unauthorized error while subscribing to topic: ' + topic,
 file=sys.stderr)
        raise e
    except Exception as e:
        print('Exception while subscribing to topic: ' + topic, file=sys.stderr)
        raise e

    # Keep the main thread alive, or the process will exit.
    try:
        while True:
            time.sleep(10)
    except InterruptedError:
        print('Subscribe interrupted.')
except Exception:
    print('Exception occurred when using IPC.', file=sys.stderr)
    traceback.print_exc()
    exit(1)
```

Exemplo de editor de publicação/assinatura (C++)

O exemplo de receita a seguir permite que o componente seja publicado em todos os tópicos.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubPublisherCpp",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that publishes messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.PubSubPublisherCpp:pubsub:1": {
            "policyDescription": "Allows access to publish to all topics.",
            "operations": [
              "aws.greengrass#PublishToTopic"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Lifecycle": {
        "run": "{artifacts:path}/greengrassv2_pubsub_publisher"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.PubSubPublisherCpp/1.0.0/greengrassv2_pubsub_publisher",
          "Permission": {
            "Execute": "OWNER"
          }
        }
      ]
    }
```

```
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PubSubPublisherCpp
ComponentVersion: 1.0.0
ComponentDescription: A component that publishes messages.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.pubsub:
        com.example.PubSubPublisherCpp:pubsub:1:
          policyDescription: Allows access to publish to all topics.
          operations:
            - aws.greengrass#PublishToTopic
          resources:
            - "*"
Manifests:
  - Lifecycle:
      run: "{artifacts:path}/greengrassv2_pubsub_publisher"
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/com.example.PubSubPublisherCpp/1.0.0/
greengrassv2_pubsub_publisher
        Permission:
          Execute: OWNER
```

O exemplo de aplicativo C++ a seguir demonstra como usar o serviço IPC de publicação/assinatura para publicar mensagens em outros componentes.

```
#include <iostream>

#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
```

```
    void OnConnectCallback() override {
        std::cout << "OnConnectCallback" << std::endl;
    }

    void OnDisconnectCallback(RpcError error) override {
        std::cout << "OnDisconnectCallback: " << error.StatusToString() << std::endl;
        exit(-1);
    }

    bool OnErrorCallback(RpcError error) override {
        std::cout << "OnErrorCallback: " << error.StatusToString() << std::endl;
        return true;
    }
};

int main() {
    String message("Hello from the pub/sub publisher (C++).");
    String topic("test/topic/cpp");
    int timeout = 10;

    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
 connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    while (true) {
        PublishToTopicRequest request;
        Vector<uint8_t> messageData({message.begin(), message.end()});
        BinaryMessage binaryMessage;
        binaryMessage.SetMessage(messageData);
        PublishMessage publishMessage;
        publishMessage.SetBinaryMessage(binaryMessage);
        request.SetTopic(topic);
        request.SetPublishMessage(publishMessage);

        auto operation = ipcClient.NewPublishToTopic();
```

```
        auto activate = operation->Activate(request, nullptr);
        activate.wait();

        auto responseFuture = operation->GetResult();
        if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
 std::future_status::timeout) {
            std::cerr << "Operation timed out while waiting for response from
 Greengrass Core." << std::endl;
            exit(-1);
        }

        auto response = responseFuture.get();
        if (response) {
            std::cout << "Successfully published to topic: " << topic << std::endl;
        } else {
            // An error occurred.
            std::cout << "Failed to publish to topic: " << topic << std::endl;
            auto errorType = response.GetResultType();
            if (errorType == OPERATION_ERROR) {
                auto *error = response.GetOperationError();
                std::cout << "Operation error: " << error->GetMessage().value() <<
 std::endl;
            } else {
                std::cout << "RPC error: " << response.GetRpcError() << std::endl;
            }
            exit(-1);
        }

        std::this_thread::sleep_for(std::chrono::seconds(5));
    }

    return 0;
}
```

Exemplo de assinante de publicação/assinatura (C++)

O exemplo de receita a seguir permite que o componente se inscreva em todos os tópicos.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PubSubSubscriberCpp",
  "ComponentVersion": "1.0.0",
```

```
  "ComponentDescription": "A component that subscribes to messages.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.pubsub": {
          "com.example.PubSubSubscriberCpp:pubsub:1": {
            "policyDescription": "Allows access to subscribe to all topics.",
            "operations": [
              "aws.greengrass#SubscribeToTopic"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Lifecycle": {
        "run": "{artifacts:path}/greengrassv2_pub_sub_subscriber"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.PubSubSubscriberCpp/1.0.0/greengrassv2_pub_sub_subscriber",
          "Permission": {
            "Execute": "OWNER"
          }
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PubSubSubscriberCpp
ComponentVersion: 1.0.0
```

```
ComponentDescription: A component that subscribes to messages.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.pubsub:
        com.example.PubSubSubscriberCpp:pubsub:1:
          policyDescription: Allows access to subscribe to all topics.
          operations:
            - aws.greengrass#SubscribeToTopic
          resources:
            - "*"
Manifests:
  - Lifecycle:
      run: "{artifacts:path}/greengrassv2_pub_sub_subscriber"
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.PubSubSubscriberCpp/1.0.0/greengrassv2_pub_sub_subscriber
        Permission:
          Execute: OWNER
```

O exemplo de aplicativo C++ a seguir demonstra como usar o serviço IPC de publicação/assinatura para assinar mensagens em outros componentes.

```
#include <iostream>

#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class SubscribeResponseHandler : public SubscribeToTopicStreamHandler {
    public:
        virtual ~SubscribeResponseHandler() {}

    private:
        void OnStreamEvent(SubscriptionResponseMessage *response) override {
            auto jsonMessage = response->GetJsonMessage();
            if (jsonMessage.has_value() &&
 jsonMessage.value().GetMessage().has_value()) {
```

```
                auto messageString =
 jsonMessage.value().GetMessage().value().View().WriteReadable();
                std::cout << "Received new message: " << messageString << std::endl;
            } else {
                auto binaryMessage = response->GetBinaryMessage();
                if (binaryMessage.has_value() &&
 binaryMessage.value().GetMessage().has_value()) {
                    auto messageBytes = binaryMessage.value().GetMessage().value();
                    std::string messageString(messageBytes.begin(),
 messageBytes.end());
                    std::cout << "Received new message: " << messageString <<
 std::endl;
                }
            }
        }

        bool OnStreamError(OperationError *error) override {
            std::cout << "Received an operation error: ";
            if (error->GetMessage().has_value()) {
                std::cout << error->GetMessage().value();
            }
            std::cout << std::endl;
            return false; // Return true to close stream, false to keep stream open.
        }

        void OnStreamClosed() override {
            std::cout << "Subscribe to topic stream closed." << std::endl;
        }
};

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        std::cout << "OnConnectCallback" << std::endl;
    }

    void OnDisconnectCallback(RpcError error) override {
        std::cout << "OnDisconnectCallback: " << error.StatusToString() << std::endl;
        exit(-1);
    }

    bool OnErrorCallback(RpcError error) override {
        std::cout << "OnErrorCallback: " << error.StatusToString() << std::endl;
        return true;
    }
```

```
};

int main() {
    String topic("test/topic/cpp");
    int timeout = 10;

    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
 connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    SubscribeToTopicRequest request;
    request.SetTopic(topic);
    auto streamHandler = MakeShared<SubscribeResponseHandler>(DefaultAllocator());
    auto operation = ipcClient.NewSubscribeToTopic(streamHandler);
    auto activate = operation->Activate(request, nullptr);
    activate.wait();

    auto responseFuture = operation->GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
 std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from Greengrass
 Core." << std::endl;
        exit(-1);
    }

    auto response = responseFuture.get();
    if (response) {
        std::cout << "Successfully subscribed to topic: " << topic << std::endl;
    } else {
        // An error occurred.
        std::cout << "Failed to subscribe to topic: " << topic << std::endl;
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
```

```
            std::cout << "Operation error: " << error->GetMessage().value() <<
 std::endl;
        } else {
            std::cout << "RPC error: " << response.GetRpcError() << std::endl;
        }
        exit(-1);
    }

    // Keep the main thread alive, or the process will exit.
    while (true) {
        std::this_thread::sleep_for(std::chrono::seconds(10));
    }

    operation->Close();
    return 0;
}
```

Publique/assine mensagens MQTT AWS IoT Core

O serviço IPC de mensagens AWS IoT Core MQTT permite que você envie e receba mensagens MQTT de e para. AWS IoT Core Os componentes podem publicar mensagens AWS IoT Core e assinar tópicos para atuar nas mensagens MQTT de outras fontes. Para obter mais informações sobre a AWS IoT Core implementação do MQTT, consulte MQTT no Guia do AWS IoT Core Desenvolvedor.

> **Note**
>
> Esse serviço IPC de mensagens MQTT permite que você troque mensagens com. AWS IoT Core Para obter mais informações sobre como trocar mensagens entre componentes, consultePublique/assine mensagens locais.

Tópicos

Versões mínimas do SDK

A tabela a seguir lista as versões mínimas do AWS IoT Device SDK que você deve usar para publicar e assinar mensagens MQTT de AWS IoT Core e para.

SDK	Versão mínima	
AWS IoT Device SDK para Java v2	v1.2.10	
AWS IoT Device SDK para Python v2	v1.5.3	
AWS IoT Device SDK para C++ v2	v1.17.0	
AWS IoT Device SDK para JavaScript v2	v1.12.0	

Autorização

Para usar mensagens AWS IoT Core MQTT em um componente personalizado, você deve definir políticas de autorização que permitam que seu componente envie e receba mensagens sobre tópicos. Para obter informações sobre a definição de políticas de autorização, consulteAutorize componentes a realizar operações de IPC.

As políticas de autorização para mensagens AWS IoT Core MQTT têm as seguintes propriedades.

Identificador de serviço IPC: `aws.greengrass.ipc.mqttproxy`

Operation	Descrição	Recursos
`aws.greengrass#PublishToIoTCore`	Permite que um componente publique mensagens AWS IoT Core nos tópicos do MQTT que você especificar.	Uma sequência de tópicos, como`test/topic` , ou `*` para permitir o acesso a todos os tópicos. Você pode usar curingas de tópico do MQTT

Operation	Descrição	Recursos
		(#e+) para combinar vários recursos.
`aws.greengrass#SubscribeToIoTCore`	Permite que um componente assine mensagens dos AWS IoT Core tópicos que você especificar.	Uma sequência de tópicos, como`test/topic` , ou * para permitir o acesso a todos os tópicos. Você pode usar curingas de tópico do MQTT (#e+) para combinar vários recursos.
*	Permite que um componente publique e assine mensagens AWS IoT Core MQTT para os tópicos que você especificar.	Uma sequência de tópicos, como`test/topic` , ou * para permitir o acesso a todos os tópicos. Você pode usar curingas de tópico do MQTT (#e+) para combinar vários recursos.

Caracteres curingas do MQTT nas políticas de autorização do AWS IoT Core MQTT

Você pode usar curingas do MQTT nas políticas de autorização do AWS IoT Core MQTT IPC. Os componentes podem publicar e assinar tópicos que correspondam ao filtro de tópicos permitido em uma política de autorização. Por exemplo, se a política de autorização de um componente conceder acesso a `test/topic/#`test/topic/#, o componente pode se inscrever, publicar e assinar`test/topic/filter`.

Variáveis de receita nas políticas de autorização do AWS IoT Core MQTT

Se você usa a versão 2.6.0 ou posterior do núcleo do Greengrass, pode usar a variável de receita nas `{iot:thingName}` políticas de autorização. Esse recurso permite que você configure uma única política de autorização para um grupo de dispositivos principais, em que cada dispositivo principal pode acessar somente tópicos que contenham seu próprio nome. Por exemplo, você pode permitir que um componente acesse o seguinte recurso de tópico.

```
devices/{iot:thingName}/messages
```

Para obter mais informações, consulte Variáveis da receita e Use variáveis de receita em atualizações de mesclagem.

Exemplos de políticas de autorização

Você pode consultar os exemplos de políticas de autorização a seguir para ajudá-lo a configurar políticas de autorização para seus componentes.

Example Exemplo de política de autorização com acesso irrestrito

O exemplo de política de autorização a seguir permite que um componente publique e assine todos os tópicos.

JSON

```
{
  "accessControl": {
    "aws.greengrass.ipc.mqttproxy": {
      "com.example.MyIoTCorePubSubComponent:mqttproxy:1": {
        "policyDescription": "Allows access to publish/subscribe to all topics.",
        "operations": [
          "aws.greengrass#PublishToIoTCore",
          "aws.greengrass#SubscribeToIoTCore"
        ],
        "resources": [
          "*"
        ]
      }
    }
  }
}
```

YAML

```
---
accessControl:
  aws.greengrass.ipc.mqttproxy:
    com.example.MyIoTCorePubSubComponent:mqttproxy:1:
      policyDescription: Allows access to publish/subscribe to all topics.
      operations:
```

```
        - aws.greengrass#PublishToIoTCore
        - aws.greengrass#SubscribeToIoTCore
      resources:
        - "*"
```

Example Exemplo de política de autorização com acesso limitado

O exemplo de política de autorização a seguir permite que um componente publique e assine dois tópicos chamados `factory/1/events` `factory/1/actions` e.

JSON

```
{
  "accessControl": {
    "aws.greengrass.ipc.mqttproxy": {
      "com.example.MyIoTCorePubSubComponent:mqttproxy:1": {
        "policyDescription": "Allows access to publish/subscribe to factory 1
 topics.",
        "operations": [
          "aws.greengrass#PublishToIoTCore",
          "aws.greengrass#SubscribeToIoTCore"
        ],
        "resources": [
          "factory/1/actions",
          "factory/1/events"
        ]
      }
    }
  }
}
```

YAML

```
---
accessControl:
  aws.greengrass.ipc.mqttproxy:
    "com.example.MyIoTCorePubSubComponent:mqttproxy:1":
      policyDescription: Allows access to publish/subscribe to factory 1 topics.
      operations:
        - aws.greengrass#PublishToIoTCore
        - aws.greengrass#SubscribeToIoTCore
      resources:
```

```
        - factory/1/actions
        - factory/1/events
```

Example Exemplo de política de autorização para um grupo de dispositivos principais

> **Important**
>
> Este exemplo usa um recurso que está disponível para a versão 2.6.0 e posterior do componente núcleo do Greengrass. O Greengrass nucleus v2.6.0 adiciona suporte para a maioria das variáveis de receita, como configurações de componentes. `{iot:thingName}`

O exemplo de política de autorização a seguir permite que um componente publique e assine um tópico que contém o nome do dispositivo principal que executa o componente.

JSON

```
{
  "accessControl": {
    "aws.greengrass.ipc.mqttproxy": {
      "com.example.MyIoTCorePubSubComponent:mqttproxy:1": {
        "policyDescription": "Allows access to publish/subscribe to all topics.",
        "operations": [
          "aws.greengrass#PublishToIoTCore",
          "aws.greengrass#SubscribeToIoTCore"
        ],
        "resources": [
          "factory/1/devices/{iot:thingName}/controls"
        ]
      }
    }
  }
}
```

YAML

```
---
accessControl:
  aws.greengrass.ipc.mqttproxy:
    "com.example.MyIoTCorePubSubComponent:mqttproxy:1":
      policyDescription: Allows access to publish/subscribe to all topics.
```

```
      operations:
        - aws.greengrass#PublishToIoTCore
        - aws.greengrass#SubscribeToIoTCore
      resources:
        - factory/1/devices/{iot:thingName}/controls
```

PublishToIoTCore

Publica uma mensagem MQTT AWS IoT Core em um tópico.

Quando você publica mensagens MQTT no AWS IoT Core, há uma cota de 100 transações por segundo. Se você exceder essa cota, as mensagens serão enfileiradas para processamento no dispositivo Greengrass. Há também uma cota de 512 Kb de dados por segundo e uma cota de 20.000 publicações por segundo em toda a conta (2.000 em algumas). Regiões da AWS Para obter mais informações sobre os limites do agente de mensagens MQTT em AWS IoT Core, consulte limites e cotas do agente de AWS IoT Core mensagens e do protocolo.

Se você exceder essas cotas, o dispositivo Greengrass limitará a publicação de mensagens a. AWS IoT Core As mensagens são armazenadas em um spooler na memória. Por padrão, a memória alocada para o spooler é de 2,5 Mb. Se o spooler ficar cheio, novas mensagens serão rejeitadas. Você pode aumentar o tamanho do spooler. Para obter mais informações, consulte a Configuração documentação do Núcleo Greengrass. Para evitar preencher o spooler e precisar aumentar a memória alocada, limite as solicitações de publicação a no máximo 100 solicitações por segundo.

Quando seu aplicativo precisar enviar mensagens em uma taxa maior ou maiores, considere usar o Gerenciador de fluxo para enviar mensagens para o Kinesis Data Streams. O componente gerenciador de fluxo foi projetado para transferir dados de alto volume para o. Nuvem AWS Para ter mais informações, consulte Gerencie fluxos de dados nos dispositivos principais do Greengrass.

Solicitação

A solicitação dessa operação tem os seguintes parâmetros:

`topicName`(Python:) `topic_name`

O tópico no qual publicar a mensagem.

`qos`

O QoS do MQTT a ser usado. Essa enumeração,`QOS`, tem os seguintes valores:

- `AT_MOST_ONCE`— QoS 0. A mensagem MQTT é entregue no máximo uma vez.
- `AT_LEAST_ONCE`— QoS 1. A mensagem MQTT é entregue pelo menos uma vez.

`payload`

(Opcional) A carga útil da mensagem como um blob.

Os recursos a seguir estão disponíveis para a versão 2.10.0 e versões posteriores Núcleo Greengrass ao usar o MQTT 5. Esses recursos são ignorados quando você está usando o MQTT 3.1.1. A tabela a seguir lista a versão mínima do SDK do AWS IoT dispositivo que você deve usar para acessar esses recursos.

SDK	Versão mínima
AWS IoT Device SDK for Python v2	v1.15.0
AWS IoT Device SDK for Java v2	v1.13.0
AWS IoT Device SDK for C++ v2	v1.24.0
AWS IoT Device SDK for JavaScript v2	v1.13.0

`payloadFormat`

(Opcional) O formato da carga útil da mensagem. Se você não definir o`payloadFormat`, presume-se que o tipo sejaBYTES. A enumeração tem os seguintes valores:

- `BYTES`— O conteúdo da carga é um blob binário.
- `UTF8`— O conteúdo da carga útil é uma sequência de caracteres UTF-8.

`retain`

(Opcional) Indica se a opção de retenção do MQTT deve ser definida como `true` ao publicar.

`userProperties`

(Opcional) Uma lista de `UserProperty` objetos específicos do aplicativo a serem enviados. O `UserProperty` objeto é definido da seguinte forma:

```
UserProperty:
```

```
  key: string
  value: string
```

`messageExpiryIntervalSeconds`

(Opcional) O número de segundos antes de a mensagem expirar e ser excluída pelo servidor. Se esse valor não for definido, a mensagem não expirará.

`correlationData`

(Opcional) Informações adicionadas à solicitação que podem ser usadas para associar uma solicitação a uma resposta.

`responseTopic`

(Opcional) O tópico que deve ser usado para a mensagem de resposta.

`contentType`

(Opcional) Um identificador específico do aplicativo do tipo de conteúdo da mensagem.

Resposta

Essa operação não fornece nenhuma informação em sua resposta.

Exemplos

Os exemplos a seguir demonstram como chamar essa operação no código de componente personalizado.

Java (IPC client V2)

Example Exemplo: publicar uma mensagem

```
package com.aws.greengrass.docs.samples.ipc;

import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClientV2;
import software.amazon.awssdk.aws.greengrass.model.PublishToIoTCoreRequest;
import software.amazon.awssdk.aws.greengrass.model.QOS;
import java.nio.charset.StandardCharsets;

public class PublishToIoTCore {
```

```
    public static void main(String[] args) {
        String topic = args[0];
        String message = args[1];
        QOS qos = QOS.get(args[2]);

        try (GreengrassCoreIPCClientV2 ipcClientV2 =
 GreengrassCoreIPCClientV2.builder().build()) {
            ipcClientV2.publishToIoTCore(new PublishToIoTCoreRequest()
                    .withTopicName(topic)
                    .withPayload(message.getBytes(StandardCharsets.UTF_8))
                    .withQos(qos));
            System.out.println("Successfully published to topic: " + topic);
        } catch (Exception e) {
            System.err.println("Exception occurred.");
            e.printStackTrace();
            System.exit(1);
        }
    }
}
```

Python (IPC client V2)

Example Exemplo: publicar uma mensagem

> **Note**
>
> Este exemplo pressupõe que você esteja usando a versão 1.5.4 ou posterior do for AWS IoT Device SDK Python v2.

```
import awsiot.greengrasscoreipc.clientv2 as clientV2

topic = 'my/topic'
qos = '1'
payload = 'Hello, World'

ipc_client = clientV2.GreengrassCoreIPCClientV2()
resp = ipc_client.publish_to_iot_core(topic_name=topic, qos=qos, payload=payload)
ipc_client.close()
```

Java (IPC client V1)

Example Exemplo: publicar uma mensagem

> **Note**
>
> Este exemplo usa uma `IPCUtils` classe para criar uma conexão com o serviço AWS IoT Greengrass Core IPC. Para ter mais informações, consulte Conecte-se ao serviço AWS IoT Greengrass Core IPC.

```
package com.aws.greengrass.docs.samples.ipc;

import com.aws.greengrass.docs.samples.ipc.util.IPCUtils;
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.PublishToIoTCoreResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.PublishToIoTCoreRequest;
import software.amazon.awssdk.aws.greengrass.model.PublishToIoTCoreResponse;
import software.amazon.awssdk.aws.greengrass.model.QOS;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;

import java.nio.charset.StandardCharsets;
import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class PublishToIoTCore {

    public static final int TIMEOUT_SECONDS = 10;

    public static void main(String[] args) {
        String topic = args[0];
        String message = args[1];
        QOS qos = QOS.get(args[2]);
        try (EventStreamRPCConnection eventStreamRPCConnection =
                     IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient =
                    new GreengrassCoreIPCClient(eventStreamRPCConnection);
            PublishToIoTCoreResponseHandler responseHandler =
```

```
                    PublishToIoTCore.publishBinaryMessageToTopic(ipcClient, topic,
 message, qos);
            CompletableFuture<PublishToIoTCoreResponse> futureResponse =
                    responseHandler.getResponse();
            try {
                futureResponse.get(TIMEOUT_SECONDS, TimeUnit.SECONDS);
                System.out.println("Successfully published to topic: " + topic);
            } catch (TimeoutException e) {
                System.err.println("Timeout occurred while publishing to topic: " +
 topic);
            } catch (ExecutionException e) {
                if (e.getCause() instanceof UnauthorizedError) {
                    System.err.println("Unauthorized error while publishing to
 topic: " + topic);
                } else {
                    throw e;
                }
            }
        } catch (InterruptedException e) {
            System.out.println("IPC interrupted.");
        } catch (ExecutionException e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static PublishToIoTCoreResponseHandler
 publishBinaryMessageToTopic(GreengrassCoreIPCClient greengrassCoreIPCClient, String
 topic, String message, QOS qos) {
        PublishToIoTCoreRequest publishToIoTCoreRequest = new
 PublishToIoTCoreRequest();
        publishToIoTCoreRequest.setTopicName(topic);

 publishToIoTCoreRequest.setPayload(message.getBytes(StandardCharsets.UTF_8));
        publishToIoTCoreRequest.setQos(qos);
        return greengrassCoreIPCClient.publishToIoTCore(publishToIoTCoreRequest,
 Optional.empty());
    }
}
```

Python (IPC client V1)

Example Exemplo: publicar uma mensagem

> **Note**
>
> Este exemplo pressupõe que você esteja usando a versão 1.5.4 ou posterior do for AWS IoT Device SDK Python v2.

```
import awsiot.greengrasscoreipc
import awsiot.greengrasscoreipc.client as client
from awsiot.greengrasscoreipc.model import (
    QOS,
    PublishToIoTCoreRequest
)

TIMEOUT = 10

ipc_client = awsiot.greengrasscoreipc.connect()

topic = "my/topic"
message = "Hello, World"
qos = QOS.AT_LEAST_ONCE

request = PublishToIoTCoreRequest()
request.topic_name = topic
request.payload = bytes(message, "utf-8")
request.qos = qos
operation = ipc_client.new_publish_to_iot_core()
operation.activate(request)
future_response = operation.get_response()
future_response.result(TIMEOUT)
```

C++

Example Exemplo: publicar uma mensagem

```
#include <iostream>

#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>
```

```
using namespace Aws::Crt;
using namespace Aws::Greengrass;

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        // Handle connection to IPC service.
    }

    void OnDisconnectCallback(RpcError error) override {
        // Handle disconnection from IPC service.
    }

    bool OnErrorCallback(RpcError error) override {
        // Handle IPC service connection error.
        return true;
    }
};

int main() {
    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
 connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    String message("Hello, World!");
    String topic("my/topic");
    QOS qos = QOS_AT_MOST_ONCE;
    int timeout = 10;

    PublishToIoTCoreRequest request;
    Vector<uint8_t> messageData({message.begin(), message.end()});
    request.SetTopicName(topic);
    request.SetPayload(messageData);
    request.SetQos(qos);

    auto operation = ipcClient.NewPublishToIoTCore();
```

```
    auto activate = operation->Activate(request, nullptr);
    activate.wait();

    auto responseFuture = operation->GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
 std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from Greengrass
 Core." << std::endl;
        exit(-1);
    }

    auto response = responseFuture.get();
    if (!response) {
        // Handle error.
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
            (void)error;
            // Handle operation error.
        } else {
            // Handle RPC error.
        }
    }

    return 0;
}
```

JavaScript

Example Exemplo: publicar uma mensagem

```
import * as greengrasscoreipc from "aws-iot-device-sdk-v2/dist/greengrasscoreipc";
import {QOS, PublishToIoTCoreRequest} from "aws-iot-device-sdk-v2/dist/
greengrasscoreipc/model";

class PublishToIoTCore {
    private ipcClient: greengrasscoreipc.Client
    private readonly topic: string;

    constructor() {
        // define your own constructor, e.g.
        this.topic = "<define_your_topic>";
        this.publishToIoTCore().then(r => console.log("Started workflow"));
```

```
    }

    private async publishToIoTCore() {
        try {
            const request: PublishToIoTCoreRequest = {
                topicName: this.topic,
                qos: QOS.AT_LEAST_ONCE, // you can change this depending on your use
 case
            }

            this.ipcClient = await getIpcClient();

            await this.ipcClient.publishToIoTCore(request);
        } catch (e) {
            // parse the error depending on your use cases
            throw e
        }
    }
}


export async function getIpcClient(){
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

// starting point
const publishToIoTCore = new PublishToIoTCore();
```

SubscribeToIoTCore

Assine as mensagens do MQTT a partir AWS IoT Core de um tópico ou filtro de tópicos. O software AWS IoT Greengrass principal remove as assinaturas quando o componente chega ao fim de seu ciclo de vida.

Essa operação é uma operação de assinatura em que você assina um fluxo de mensagens de eventos. Para usar essa operação, defina um manipulador de resposta de fluxo com funções que manipulam mensagens de eventos, erros e encerramento de fluxo. Para ter mais informações, consulte Inscreva-se nos streams de eventos do IPC.

Tipo de mensagem do evento: `IoTCoreMessage`

Solicitação

A solicitação dessa operação tem os seguintes parâmetros:

`topicName`(Python:) `topic_name`

O tópico no qual se inscrever. Você pode usar curingas de tópicos do MQTT (#e+) para se inscrever em vários tópicos.

`qos`

O QoS do MQTT a ser usado. Essa enumeração,`QOS`, tem os seguintes valores:

- `AT_MOST_ONCE`— QoS 0. A mensagem MQTT é entregue no máximo uma vez.
- `AT_LEAST_ONCE`— QoS 1. A mensagem MQTT é entregue pelo menos uma vez.

Resposta

A resposta dessa operação tem as seguintes informações:

`messages`

O fluxo de mensagens do MQTT. Esse objeto,`IoTCoreMessage`, contém as seguintes informações:

`message`

A mensagem do MQTT. Esse objeto,`MQTTMessage`, contém as seguintes informações:

`topicName`(Python:) `topic_name`

O tópico no qual a mensagem foi publicada.

`payload`

(Opcional) A carga útil da mensagem como um blob.

Os recursos a seguir estão disponíveis para a versão 2.10.0 e versões posteriores Núcleo Greengrass ao usar o MQTT 5. Esses recursos são ignorados quando você está usando o MQTT 3.1.1. A tabela a seguir lista a versão mínima do SDK do AWS IoT dispositivo que você deve usar para acessar esses recursos.

SDK	Versão mínima
AWS IoT Device SDK for Python v2	v1.15.0
AWS IoT Device SDK for Java v2	v1.13.0
AWS IoT Device SDK for C++ v2	v1.24.0
AWS IoT Device SDK for JavaScript v2	v1.13.0

`payloadFormat`

(Opcional) O formato da carga útil da mensagem. Se você não definir o`payloadFormat`, presume-se que o tipo sejaBYTES. A enumeração tem os seguintes valores:

- `BYTES`— O conteúdo da carga é um blob binário.
- `UTF8`— O conteúdo da carga útil é uma sequência de caracteres UTF-8.

`retain`

(Opcional) Indica se a opção de retenção do MQTT deve ser definida como `true` ao publicar.

`userProperties`

(Opcional) Uma lista de `UserProperty` objetos específicos do aplicativo a serem enviados. O `UserProperty` objeto é definido da seguinte forma:

```
UserProperty:
  key: string
```

```
    value: string
```

`messageExpiryIntervalSeconds`

(Opcional) O número de segundos antes de a mensagem expirar e ser excluída pelo servidor. Se esse valor não for definido, a mensagem não expirará.

`correlationData`

(Opcional) Informações adicionadas à solicitação que podem ser usadas para associar uma solicitação a uma resposta.

`responseTopic`

(Opcional) O tópico que deve ser usado para a mensagem de resposta.

`contentType`

(Opcional) Um identificador específico do aplicativo do tipo de conteúdo da mensagem.

Exemplos

Os exemplos a seguir demonstram como chamar essa operação no código de componente personalizado.

Java (IPC client V2)

Example Exemplo: Inscrever-se em mensagens

```
package com.aws.greengrass.docs.samples.ipc;

import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClientV2;
import software.amazon.awssdk.aws.greengrass.SubscribeToIoTCoreResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.QOS;
import software.amazon.awssdk.aws.greengrass.model.IoTCoreMessage;
import software.amazon.awssdk.aws.greengrass.model.SubscribeToIoTCoreRequest;
import software.amazon.awssdk.aws.greengrass.model.SubscribeToIoTCoreResponse;

import java.nio.charset.StandardCharsets;
import java.util.Optional;
import java.util.function.Consumer;
import java.util.function.Function;
```

```
public class SubscribeToIoTCore {

    public static void main(String[] args) {
        String topic = args[0];
        QOS qos = QOS.get(args[1]);

        Consumer<IoTCoreMessage> onStreamEvent = ioTCoreMessage ->
                System.out.printf("Received new message on topic %s: %s%n",
                        ioTCoreMessage.getMessage().getTopicName(),
                        new String(ioTCoreMessage.getMessage().getPayload(),
 StandardCharsets.UTF_8));

        Optional<Function<Throwable, Boolean>> onStreamError =
                Optional.of(e -> {
                    System.err.println("Received a stream error.");
                    e.printStackTrace();
                    return false;
                });

        Optional<Runnable> onStreamClosed = Optional.of(() ->
                System.out.println("Subscribe to IoT Core stream closed."));

        try (GreengrassCoreIPCClientV2 ipcClientV2 =
 GreengrassCoreIPCClientV2.builder().build()) {
            SubscribeToIoTCoreRequest request = new SubscribeToIoTCoreRequest()
                    .withTopicName(topic)
                    .withQos(qos);

            GreengrassCoreIPCClientV2.StreamingResponse<SubscribeToIoTCoreResponse,
 SubscribeToIoTCoreResponseHandler>
                    streamingResponse = ipcClientV2.subscribeToIoTCore(request,
 onStreamEvent, onStreamError, onStreamClosed);

            streamingResponse.getResponse();
            System.out.println("Successfully subscribed to topic: " + topic);

            // Keep the main thread alive, or the process will exit.
            while (true) {
                Thread.sleep(10000);
            }

            // To stop subscribing, close the stream.
            streamingResponse.getHandler().closeStream();
        } catch (InterruptedException e) {
```

```
            System.out.println("Subscribe interrupted.");
        } catch (Exception e) {
            System.err.println("Exception occurred.");
            e.printStackTrace();
            System.exit(1);
        }
    }
}
```

Python (IPC client V2)

Example Exemplo: assinar mensagens

> **Note**
>
> Este exemplo pressupõe que você esteja usando a versão 1.5.4 ou posterior do for AWS IoT Device SDK Python v2.

```
import threading
import traceback

import awsiot.greengrasscoreipc.clientv2 as clientV2

topic = 'my/topic'
qos = '1'

def on_stream_event(event):
    try:
        topic_name = event.message.topic_name
        message = str(event.message.payload, 'utf-8')
        print(f'Received new message on topic {topic_name}:  {message}')
    except:
        traceback.print_exc()

def on_stream_error(error):
    # Return True to close stream, False to keep stream open.
    return True

def on_stream_closed():
    pass
```

```
ipc_client = clientV2.GreengrassCoreIPCClientV2()
resp, operation = ipc_client.subscribe_to_iot_core(
    topic_name=topic,
    qos=qos,
    on_stream_event=on_stream_event,
    on_stream_error=on_stream_error,
    on_stream_closed=on_stream_closed
)

# Keep the main thread alive, or the process will exit.
event = threading.Event()
event.wait()

# To stop subscribing, close the operation stream.
operation.close()
ipc_client.close()
```

Java (IPC client V1)

Example Exemplo: Inscrever-se em mensagens

> **Note**
>
> Este exemplo usa uma `IPCUtils` classe para criar uma conexão com o serviço AWS IoT Greengrass Core IPC. Para ter mais informações, consulte Conecte-se ao serviço AWS IoT Greengrass Core IPC.

```
package com.aws.greengrass.docs.samples.ipc;

import com.aws.greengrass.docs.samples.ipc.util.IPCUtils;
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.SubscribeToIoTCoreResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.*;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;
import software.amazon.awssdk.eventstreamrpc.StreamResponseHandler;

import java.nio.charset.StandardCharsets;
import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
```

```
import java.util.concurrent.TimeoutException;

public class SubscribeToIoTCore {

    public static final int TIMEOUT_SECONDS = 10;

    public static void main(String[] args) {
        String topic = args[0];
        QOS qos = QOS.get(args[1]);
        try (EventStreamRPCConnection eventStreamRPCConnection =
                     IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient =
                    new GreengrassCoreIPCClient(eventStreamRPCConnection);
            StreamResponseHandler<IoTCoreMessage> streamResponseHandler =
                    new SubscriptionResponseHandler();
            SubscribeToIoTCoreResponseHandler responseHandler =
                    SubscribeToIoTCore.subscribeToIoTCore(ipcClient, topic, qos,
                            streamResponseHandler);
            CompletableFuture<SubscribeToIoTCoreResponse> futureResponse =
                    responseHandler.getResponse();
            try {
                futureResponse.get(TIMEOUT_SECONDS, TimeUnit.SECONDS);
                System.out.println("Successfully subscribed to topic: " + topic);
            } catch (TimeoutException e) {
                System.err.println("Timeout occurred while subscribing to topic: " +
 topic);
            } catch (ExecutionException e) {
                if (e.getCause() instanceof UnauthorizedError) {
                    System.err.println("Unauthorized error while subscribing to
 topic: " + topic);
                } else {
                    throw e;
                }
            }

            // Keep the main thread alive, or the process will exit.
            try {
                while (true) {
                    Thread.sleep(10000);
                }
            } catch (InterruptedException e) {
                System.out.println("Subscribe interrupted.");
            }
```

```
            // To stop subscribing, close the stream.
            responseHandler.closeStream();
        } catch (InterruptedException e) {
            System.out.println("IPC interrupted.");
        } catch (ExecutionException e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static SubscribeToIoTCoreResponseHandler
subscribeToIoTCore(GreengrassCoreIPCClient greengrassCoreIPCClient, String topic,
QOS qos, StreamResponseHandler<IoTCoreMessage> streamResponseHandler) {
        SubscribeToIoTCoreRequest subscribeToIoTCoreRequest = new
SubscribeToIoTCoreRequest();
        subscribeToIoTCoreRequest.setTopicName(topic);
        subscribeToIoTCoreRequest.setQos(qos);
        return
greengrassCoreIPCClient.subscribeToIoTCore(subscribeToIoTCoreRequest,
                Optional.of(streamResponseHandler));
    }

    public static class SubscriptionResponseHandler implements
StreamResponseHandler<IoTCoreMessage> {

        @Override
        public void onStreamEvent(IoTCoreMessage ioTCoreMessage) {
            try {
                String topic = ioTCoreMessage.getMessage().getTopicName();
                String message = new
String(ioTCoreMessage.getMessage().getPayload(),
                        StandardCharsets.UTF_8);
                System.out.printf("Received new message on topic %s: %s%n", topic,
message);
            } catch (Exception e) {
                System.err.println("Exception occurred while processing subscription
response " +
                        "message.");
                e.printStackTrace();
            }
        }

        @Override
```

```
        public boolean onStreamError(Throwable error) {
            System.err.println("Received a stream error.");
            error.printStackTrace();
            return false;
        }

        @Override
        public void onStreamClosed() {
            System.out.println("Subscribe to IoT Core stream closed.");
        }
    }
}
```

Python (IPC client V1)

Example Exemplo: Inscrever-se em mensagens

> **Note**
>
> Este exemplo pressupõe que você esteja usando a versão 1.5.4 ou posterior do for AWS IoT Device SDK Python v2.

```
import time
import traceback

import awsiot.greengrasscoreipc
import awsiot.greengrasscoreipc.client as client
from awsiot.greengrasscoreipc.model import (
    IoTCoreMessage,
    QOS,
    SubscribeToIoTCoreRequest
)

TIMEOUT = 10

ipc_client = awsiot.greengrasscoreipc.connect()

class StreamHandler(client.SubscribeToIoTCoreStreamHandler):
    def __init__(self):
        super().__init__()

    def on_stream_event(self, event: IoTCoreMessage) -> None:
```

```
        try:
            message = str(event.message.payload, "utf-8")
            topic_name = event.message.topic_name
            # Handle message.
        except:
            traceback.print_exc()

    def on_stream_error(self, error: Exception) -> bool:
        # Handle error.
        return True  # Return True to close stream, False to keep stream open.

    def on_stream_closed(self) -> None:
        # Handle close.
        pass


topic = "my/topic"
qos = QOS.AT_MOST_ONCE

request = SubscribeToIoTCoreRequest()
request.topic_name = topic
request.qos = qos
handler = StreamHandler()
operation = ipc_client.new_subscribe_to_iot_core(handler)
operation.activate(request)
future_response = operation.get_response()
future_response.result(TIMEOUT)

# Keep the main thread alive, or the process will exit.
while True:
    time.sleep(10)

# To stop subscribing, close the operation stream.
operation.close()
```

C++

Example Exemplo: Inscrever-se em mensagens

```
#include <iostream>

#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>
```

```
using namespace Aws::Crt;
using namespace Aws::Greengrass;

class IoTCoreResponseHandler : public SubscribeToIoTCoreStreamHandler {

    public:
        virtual ~IoTCoreResponseHandler() {}

    private:
        void OnStreamEvent(IoTCoreMessage *response) override {
            auto message = response->GetMessage();
            if (message.has_value() && message.value().GetPayload().has_value()) {
                auto messageBytes = message.value().GetPayload().value();
                std::string messageString(messageBytes.begin(), messageBytes.end());
                std::string topicName =
 message.value().GetTopicName().value().c_str();
                // Handle message.
            }
        }

        bool OnStreamError(OperationError *error) override {
            // Handle error.
            return false; // Return true to close stream, false to keep stream open.
        }

        void OnStreamClosed() override {
            // Handle close.
        }
};

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        // Handle connection to IPC service.
    }

    void OnDisconnectCallback(RpcError error) override {
        // Handle disconnection from IPC service.
    }

    bool OnErrorCallback(RpcError error) override {
        // Handle IPC service connection error.
        return true;
    }
};
```

```
int main() {
    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
 connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    String topic("my/topic");
    QOS qos = QOS_AT_MOST_ONCE;
    int timeout = 10;

    SubscribeToIoTCoreRequest request;
    request.SetTopicName(topic);
    request.SetQos(qos);
    auto streamHandler = MakeShared<IoTCoreResponseHandler>(DefaultAllocator());
    auto operation = ipcClient.NewSubscribeToIoTCore(streamHandler);
    auto activate = operation->Activate(request, nullptr);
    activate.wait();

    auto responseFuture = operation->GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
 std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from Greengrass
 Core." << std::endl;
        exit(-1);
    }

    auto response = responseFuture.get();
    if (!response) {
        // Handle error.
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
            (void)error;
            // Handle operation error.
        } else {
```

```
            // Handle RPC error.
        }
        exit(-1);
    }

    // Keep the main thread alive, or the process will exit.
    while (true) {
        std::this_thread::sleep_for(std::chrono::seconds(10));
    }

    operation->Close();
    return 0;
}
```

JavaScript

Example Exemplo: Inscrever-se em mensagens

```
import * as greengrasscoreipc from "aws-iot-device-sdk-v2/dist/greengrasscoreipc";
import {IoTCoreMessage, QOS, SubscribeToIoTCoreRequest} from "aws-iot-device-sdk-v2/
dist/greengrasscoreipc/model";
import {RpcError} from "aws-iot-device-sdk-v2/dist/eventstream_rpc";

class SubscribeToIoTCore {
    private ipcClient: greengrasscoreipc.Client
    private readonly topic: string;

    constructor() {
        // define your own constructor, e.g.
        this.topic = "<define_your_topic>";
        this.subscribeToIoTCore().then(r => console.log("Started workflow"));
    }

    private async subscribeToIoTCore() {
        try {
            const request: SubscribeToIoTCoreRequest = {
                topicName: this.topic,
                qos: QOS.AT_LEAST_ONCE, // you can change this depending on your use
 case
            }

            this.ipcClient = await getIpcClient();
```

```
            const streamingOperation = this.ipcClient.subscribeToIoTCore(request);

            streamingOperation.on('message', (message: IoTCoreMessage) => {
                // parse the message depending on your use cases, e.g.
                if (message.message && message.message.payload) {
                    const receivedMessage = message.message.payload.toString();
                }
            });

            streamingOperation.on('streamError', (error : RpcError) => {
                // define your own error handling logic
            });

            streamingOperation.on('ended', () => {
                // define your own logic
            });

            await streamingOperation.activate();

            // Keep the main thread alive, or the process will exit.
            await new Promise((resolve) => setTimeout(resolve, 10000))
        } catch (e) {
            // parse the error depending on your use cases
            throw e
        }
    }
}

export async function getIpcClient(){
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

// starting point
```

```
const subscribeToIoTCore = new SubscribeToIoTCore();
```

Exemplos

Use os exemplos a seguir para aprender a usar o serviço AWS IoT Core MQTT IPC em seus componentes.

Exemplo de editor AWS IoT Core MQTT (C++)

O exemplo de receita a seguir permite que o componente seja publicado em todos os tópicos.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.IoTCorePublisherCpp",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that publishes MQTT messages to IoT Core.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.mqttproxy": {
          "com.example.IoTCorePublisherCpp:mqttproxy:1": {
            "policyDescription": "Allows access to publish to all topics.",
            "operations": [
              "aws.greengrass#PublishToIoTCore"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Lifecycle": {
        "run": "{artifacts:path}/greengrassv2_iotcore_publisher"
      },
      "Artifacts": [
```

```
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.IoTCorePublisherCpp/1.0.0/greengrassv2_iotcore_publisher",
          "Permission": {
            "Execute": "OWNER"
          }
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.IoTCorePublisherCpp
ComponentVersion: 1.0.0
ComponentDescription: A component that publishes MQTT messages to IoT Core.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.mqttproxy:
        com.example.IoTCorePublisherCpp:mqttproxy:1:
          policyDescription: Allows access to publish to all topics.
          operations:
            - aws.greengrass#PublishToIoTCore
          resources:
            - "*"
Manifests:
  - Lifecycle:
      run: "{artifacts:path}/greengrassv2_iotcore_publisher"
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.IoTCorePublisherCpp/1.0.0/greengrassv2_iotcore_publisher
        Permission:
          Execute: OWNER
```

O exemplo de aplicativo C++ a seguir demonstra como usar o serviço AWS IoT Core MQTT IPC para publicar mensagens no. AWS IoT Core

```
#include <iostream>

#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        std::cout << "OnConnectCallback" << std::endl;
    }

    void OnDisconnectCallback(RpcError error) override {
        std::cout << "OnDisconnectCallback: " << error.StatusToString() << std::endl;
        exit(-1);
    }

    bool OnErrorCallback(RpcError error) override {
        std::cout << "OnErrorCallback: " << error.StatusToString() << std::endl;
        return true;
    }
};

int main() {
    String message("Hello from the Greengrass IPC MQTT publisher (C++).");
    String topic("test/topic/cpp");
    QOS qos = QOS_AT_LEAST_ONCE;
    int timeout = 10;

    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
 connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }
```

```
    while (true) {
        PublishToIoTCoreRequest request;
        Vector<uint8_t> messageData({message.begin(), message.end()});
        request.SetTopicName(topic);
        request.SetPayload(messageData);
        request.SetQos(qos);

        auto operation = ipcClient.NewPublishToIoTCore();
        auto activate = operation->Activate(request, nullptr);
        activate.wait();

        auto responseFuture = operation->GetResult();
        if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
 std::future_status::timeout) {
            std::cerr << "Operation timed out while waiting for response from
 Greengrass Core." << std::endl;
            exit(-1);
        }

        auto response = responseFuture.get();
        if (response) {
            std::cout << "Successfully published to topic: " << topic << std::endl;
        } else {
            // An error occurred.
            std::cout << "Failed to publish to topic: " << topic << std::endl;
            auto errorType = response.GetResultType();
            if (errorType == OPERATION_ERROR) {
                auto *error = response.GetOperationError();
                std::cout << "Operation error: " << error->GetMessage().value() <<
 std::endl;
            } else {
                std::cout << "RPC error: " << response.GetRpcError() << std::endl;
            }
            exit(-1);
        }

        std::this_thread::sleep_for(std::chrono::seconds(5));
    }

    return 0;
}
```

Exemplo de assinante AWS IoT Core MQTT (C++)

O exemplo de receita a seguir permite que o componente se inscreva em todos os tópicos.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.IoTCoreSubscriberCpp",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "A component that subscribes to MQTT messages from IoT
 Core.",
  "ComponentPublisher": "Amazon",
  "ComponentConfiguration": {
    "DefaultConfiguration": {
      "accessControl": {
        "aws.greengrass.ipc.mqttproxy": {
          "com.example.IoTCoreSubscriberCpp:mqttproxy:1": {
            "policyDescription": "Allows access to subscribe to all topics.",
            "operations": [
              "aws.greengrass#SubscribeToIoTCore"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Lifecycle": {
        "run": "{artifacts:path}/greengrassv2_iotcore_subscriber"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.IoTCoreSubscriberCpp/1.0.0/greengrassv2_iotcore_subscriber",
          "Permission": {
            "Execute": "OWNER"
          }
        }
      ]
```

```
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.IoTCoreSubscriberCpp
ComponentVersion: 1.0.0
ComponentDescription: A component that subscribes to MQTT messages from IoT Core.
ComponentPublisher: Amazon
ComponentConfiguration:
  DefaultConfiguration:
    accessControl:
      aws.greengrass.ipc.mqttproxy:
        com.example.IoTCoreSubscriberCpp:mqttproxy:1:
          policyDescription: Allows access to subscribe to all topics.
          operations:
            - aws.greengrass#SubscribeToIoTCore
          resources:
            - "*"
Manifests:
  - Lifecycle:
      run: "{artifacts:path}/greengrassv2_iotcore_subscriber"
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.IoTCoreSubscriberCpp/1.0.0/greengrassv2_iotcore_subscriber
        Permission:
          Execute: OWNER
```

O exemplo de aplicativo C++ a seguir demonstra como usar o serviço AWS IoT Core MQTT IPC para assinar mensagens do. AWS IoT Core

```
#include <iostream>

#include <aws/crt/Api.h>
#include <aws/greengrass/GreengrassCoreIpcClient.h>

using namespace Aws::Crt;
using namespace Aws::Greengrass;
```

```
class IoTCoreResponseHandler : public SubscribeToIoTCoreStreamHandler {

    public:
        virtual ~IoTCoreResponseHandler() {}

    private:

        void OnStreamEvent(IoTCoreMessage *response) override {
            auto message = response->GetMessage();
            if (message.has_value() && message.value().GetPayload().has_value()) {
                auto messageBytes = message.value().GetPayload().value();
                std::string messageString(messageBytes.begin(), messageBytes.end());
                std::string messageTopic =
 message.value().GetTopicName().value().c_str();
                std::cout << "Received new message on topic: " << messageTopic <<
 std::endl;
                std::cout << "Message: " << messageString << std::endl;
            }
        }

        bool OnStreamError(OperationError *error) override {
            std::cout << "Received an operation error: ";
            if (error->GetMessage().has_value()) {
                std::cout << error->GetMessage().value();
            }
            std::cout << std::endl;
            return false; // Return true to close stream, false to keep stream open.
        }

        void OnStreamClosed() override {
            std::cout << "Subscribe to IoT Core stream closed." << std::endl;
        }
};

class IpcClientLifecycleHandler : public ConnectionLifecycleHandler {
    void OnConnectCallback() override {
        std::cout << "OnConnectCallback" << std::endl;
    }

    void OnDisconnectCallback(RpcError error) override {
        std::cout << "OnDisconnectCallback: " << error.StatusToString() << std::endl;
        exit(-1);
    }
```

```
    bool OnErrorCallback(RpcError error) override {
        std::cout << "OnErrorCallback: " << error.StatusToString() << std::endl;
        return true;
    }
};

int main() {
    String topic("test/topic/cpp");
    QOS qos = QOS_AT_LEAST_ONCE;
    int timeout = 10;

    ApiHandle apiHandle(g_allocator);
    Io::EventLoopGroup eventLoopGroup(1);
    Io::DefaultHostResolver socketResolver(eventLoopGroup, 64, 30);
    Io::ClientBootstrap bootstrap(eventLoopGroup, socketResolver);
    IpcClientLifecycleHandler ipcLifecycleHandler;
    GreengrassCoreIpcClient ipcClient(bootstrap);
    auto connectionStatus = ipcClient.Connect(ipcLifecycleHandler).get();
    if (!connectionStatus) {
        std::cerr << "Failed to establish IPC connection: " <<
 connectionStatus.StatusToString() << std::endl;
        exit(-1);
    }

    SubscribeToIoTCoreRequest request;
    request.SetTopicName(topic);
    request.SetQos(qos);
    auto streamHandler = MakeShared<IoTCoreResponseHandler>(DefaultAllocator());
    auto operation = ipcClient.NewSubscribeToIoTCore(streamHandler);
    auto activate = operation->Activate(request, nullptr);
    activate.wait();

    auto responseFuture = operation->GetResult();
    if (responseFuture.wait_for(std::chrono::seconds(timeout)) ==
 std::future_status::timeout) {
        std::cerr << "Operation timed out while waiting for response from Greengrass
 Core." << std::endl;
        exit(-1);
    }

    auto response = responseFuture.get();
    if (response) {
        std::cout << "Successfully subscribed to topic: " << topic << std::endl;
    } else {
```

```
        // An error occurred.
        std::cout << "Failed to subscribe to topic: " << topic << std::endl;
        auto errorType = response.GetResultType();
        if (errorType == OPERATION_ERROR) {
            auto *error = response.GetOperationError();
            std::cout << "Operation error: " << error->GetMessage().value() <<
 std::endl;
        } else {
            std::cout << "RPC error: " << response.GetRpcError() << std::endl;
        }
        exit(-1);
    }

    // Keep the main thread alive, or the process will exit.
    while (true) {
        std::this_thread::sleep_for(std::chrono::seconds(10));
    }

    operation->Close();
    return 0;
}
```

Interaja com o ciclo de vida dos componentes

Use o serviço IPC do ciclo de vida do componente para:

- Atualize o estado do componente no dispositivo principal.
- Assine as atualizações do estado do componente.
- Evite que o núcleo interrompa o componente para aplicar uma atualização durante uma implantação.
- Pausa e retoma os processos dos componentes.

Tópicos

- Versões mínimas do SDK
- Autorização
- UpdateState
- SubscribeToComponentUpdates
- DeferComponentUpdate

- PauseComponent
- ResumeComponent

Versões mínimas do SDK

A tabela a seguir lista as versões mínimas do AWS IoT Device SDK que você deve usar para interagir com o ciclo de vida do componente.

SDK	Versão mínima	
AWS IoT Device SDKpara Java v2	v1.2.10	
AWS IoT Device SDKpara Python v2	v1.5.3	
AWS IoT Device SDKpara C++ v2	v1.17.0	
AWS IoT Device SDKpara JavaScript v2	v1.12.0	

Autorização

Para pausar ou retomar outros componentes a partir de um componente personalizado, você deve definir políticas de autorização que permitam que seu componente gerencie outros componentes. Para obter informações sobre a definição de políticas de autorização, consulteAutorize componentes a realizar operações de IPC.

As políticas de autorização para o gerenciamento do ciclo de vida dos componentes têm as seguintes propriedades.

Identificador de serviço IPC: `aws.greengrass.ipc.lifecycle`

Operation	Descrição	Recursos
`aws.greengrass#PauseComponent`	Permite que um componente pause os componentes que você especificar.	Um nome de componente ou * para permitir o acesso a todos os componentes.
`aws.greengrass#ResumeComponent`	Permite que um componente retome os componentes que você especificar.	Um nome de componente ou * para permitir o acesso a todos os componentes.
*	Permite que um componente pause e retome os componentes que você especificar.	Um nome de componente ou * para permitir o acesso a todos os componentes.

Exemplos de políticas de autorização

Você pode consultar o exemplo de política de autorização a seguir para ajudá-lo a configurar políticas de autorização para seus componentes.

Example Exemplo de política de autorização

O exemplo de política de autorização a seguir permite que um componente pause e retome todos os componentes.

```
{
  "accessControl": {
    "aws.greengrass.ipc.lifecycle": {
      "com.example.MyLocalLifecycleComponent:lifecycle:1": {
        "policyDescription": "Allows access to pause/resume all components.",
        "operations": [
          "aws.greengrass#PauseComponent",
          "aws.greengrass#ResumeComponent"
        ],
        "resources": [
          "*"
        ]
      }
    }
  }
}
```

UpdateState

Atualize o estado do componente no dispositivo principal.

Solicitação

A solicitação dessa operação tem os seguintes parâmetros:

`state`

O estado a ser definido. Essa enumeração,`LifecycleState`, tem os seguintes valores:

- `RUNNING`
- `ERRORED`

Resposta

Essa operação não fornece nenhuma informação em sua resposta.

SubscribeToComponentUpdates

Inscreva-se para receber notificações antes que o software AWS IoT Greengrass Core atualize um componente. A notificação especifica se o núcleo será reiniciado ou não como parte da atualização.

O núcleo envia notificações de atualização somente se a política de atualização de componentes da implantação especificar a notificação dos componentes. O comportamento padrão é notificar os componentes. Para obter mais informações, consulte Criar implantações e o DeploymentComponentUpdatePolicyobjeto que você pode fornecer ao chamar a CreateDeploymentoperação.

> **Important**
>
> As implantações locais não notificam os componentes antes das atualizações.

Essa operação é uma operação de assinatura em que você assina um fluxo de mensagens de eventos. Para usar essa operação, defina um manipulador de resposta de fluxo com funções que manipulam mensagens de eventos, erros e encerramento de fluxo. Para ter mais informações, consulte Inscreva-se nos streams de eventos do IPC.

Tipo de mensagem do evento: `ComponentUpdatePolicyEvents`

> **Tip**
>
> Você pode seguir um tutorial para aprender como desenvolver um componente que adia condicionalmente as atualizações de componentes. Para ter mais informações, consulte Tutorial: Desenvolva um componente do Greengrass que adia as atualizações de componentes.

Solicitação

A solicitação dessa operação não tem nenhum parâmetro.

Resposta

A resposta dessa operação tem as seguintes informações:

`messages`

O fluxo de mensagens de notificação. Esse objeto,`ComponentUpdatePolicyEvents`, contém as seguintes informações:

`preUpdateEvent`(Python:) `pre_update_event`

(Opcional) Um evento que indica que o núcleo deseja atualizar um componente. Você pode responder com a DeferComponentUpdate operação para confirmar ou adiar a atualização até que seu componente esteja pronto para ser reiniciado. Esse objeto,`PreComponentUpdateEvent`, contém as seguintes informações:

`deploymentId`(Python:) `deployment_id`

O ID da AWS IoT Greengrass implantação que atualiza o componente.

`isGgcRestarting`(Python:) `is_ggc_restarting`

Se o núcleo precisa ou não ser reiniciado para aplicar a atualização.

`postUpdateEvent`(Python:) `post_update_event`

(Opcional) Um evento que indica que o núcleo atualizou um componente. Esse objeto,`PostComponentUpdateEvent`, contém as seguintes informações:

`deploymentId`(Python:) `deployment_id`

O ID da AWS IoT Greengrass implantação que atualizou o componente.

> **Note**
>
> Esse recurso requer a versão 2.7.0 ou posterior do componente do núcleo do Greengrass.

DeferComponentUpdate

Reconheça ou adie uma atualização de componente que você descobriu comSubscribeToComponentUpdates. Você especifica a quantidade de tempo de espera antes que o núcleo verifique novamente se seu componente está pronto para permitir que a atualização do componente continue. Você também pode usar essa operação para informar ao núcleo que seu componente está pronto para a atualização.

Se um componente não responder à notificação de atualização do componente, o núcleo aguardará a quantidade de tempo especificada na política de atualização de componentes da implantação. Após esse tempo limite, o núcleo prossegue com a implantação. O tempo limite padrão de atualização do componente é de 60 segundos. Para obter mais informações, consulte Criar implantações e o DeploymentComponentUpdatePolicyobjeto que você pode fornecer ao chamar a CreateDeploymentoperação.

> **Tip**
>
> Você pode seguir um tutorial para aprender como desenvolver um componente que adia condicionalmente as atualizações de componentes. Para ter mais informações, consulte Tutorial: Desenvolva um componente do Greengrass que adia as atualizações de componentes.

Solicitação

A solicitação dessa operação tem os seguintes parâmetros:

`deploymentId`(Python:) `deployment_id`

O ID da AWS IoT Greengrass implantação a ser adiada.

`message`

(Opcional) O nome do componente para o qual adiar as atualizações.

O padrão é o nome do componente que faz a solicitação.

`recheckAfterMs`(Python:) `recheck_after_ms`

A quantidade de tempo em milissegundos para adiar a atualização. O núcleo espera por esse período de tempo e depois envia outro `PreComponentUpdateEvent` que você pode descobrir com. SubscribeToComponentUpdates

Especifique `0` para confirmar a atualização. Isso informa ao núcleo que seu componente está pronto para a atualização.

O padrão é zero milissegundos, o que significa confirmar a atualização.

Resposta

Essa operação não fornece nenhuma informação em sua resposta.

PauseComponent

Esse recurso está disponível para a versão 2.4.0 e posterior do componente de núcleo do Greengrass. AWS IoT Greengrassatualmente não oferece suporte a esse recurso nos dispositivos principais do Windows.

Pausa os processos de um componente no dispositivo principal. Para retomar um componente, use a ResumeComponentoperação.

Você pode pausar somente componentes genéricos. Se você tentar pausar qualquer outro tipo de componente, essa operação lançará um. `InvalidRequestError`

> **Note**
>
> Essa operação não pode pausar processos em contêineres, como contêineres Docker. Para pausar e retomar um contêiner do Docker, você pode usar os comandos docker pause e docker unpause.

Essa operação não pausa dependências de componentes ou componentes que dependem do componente que você pausa. Considere esse comportamento ao pausar um componente que é uma dependência de outro componente, pois o componente dependente pode encontrar problemas quando sua dependência é pausada.

Quando você reinicia ou desliga um componente pausado, como por meio de uma implantação, o núcleo do Greengrass retoma o componente e executa seu ciclo de vida de desligamento. Para obter mais informações sobre como reiniciar um componente, consulteRestartComponent.

> **Important**
>
> Para usar essa operação, você deve definir uma política de autorização que conceda permissão para usar essa operação. Para ter mais informações, consulte Autorização.

Versões mínimas do SDK

A tabela a seguir lista as versões mínimas do AWS IoT Device SDK que você deve usar para pausar e retomar componentes.

SDK	Versão mínima
AWS IoT Device SDKpara Java v2	v1.4.3
AWS IoT Device SDKpara Python v2	v1.6.2
AWS IoT Device SDKpara C++ v2	v1.13.1
AWS IoT Device SDKpara JavaScript v2	v1.12.0

Solicitação

A solicitação dessa operação tem os seguintes parâmetros:

`componentName`(Python:) `component_name`

O nome do componente a ser pausado, que deve ser um componente genérico. Para ter mais informações, consulte Tipos de componentes.

Resposta

Essa operação não fornece nenhuma informação em sua resposta.

ResumeComponent

Esse recurso está disponível para a versão 2.4.0 e posterior do componente de núcleo do Greengrass. AWS IoT Greengrassatualmente não oferece suporte a esse recurso nos dispositivos principais do Windows.

Retoma os processos de um componente no dispositivo principal. Para pausar um componente, use a PauseComponentoperação.

Você pode retomar somente os componentes pausados. Se você tentar retomar um componente que não está pausado, essa operação gera um. `InvalidRequestError`

> **Important**
>
> Para usar essa operação, você deve definir uma política de autorização que conceda permissão para fazer isso. Para ter mais informações, consulte Autorização.

Versões mínimas do SDK

A tabela a seguir lista as versões mínimas do AWS IoT Device SDK que você deve usar para pausar e retomar componentes.

SDK	Versão mínima	
AWS IoT Device SDKpara Java v2	v1.4.3	
AWS IoT Device SDKpara Python v2	v1.6.2	
AWS IoT Device SDKpara C++ v2	v1.13.1	
AWS IoT Device SDKpara JavaScript v2	v1.12.0	

Solicitação

A solicitação dessa operação tem os seguintes parâmetros:

`componentName`(Python:) `component_name`

O nome do componente a ser retomado.

Resposta

Essa operação não fornece nenhuma informação em sua resposta.

Interaja com a configuração do componente

O serviço IPC de configuração do componente permite que você faça o seguinte:

- Obtenha e defina os parâmetros de configuração do componente.
- Inscreva-se para receber atualizações de configuração de componentes.
- Valide as atualizações de configuração dos componentes antes que o núcleo as aplique.

Tópicos

Versões mínimas do SDK

A tabela a seguir lista as versões mínimas do AWS IoT Device SDK que você deve usar para interagir com a configuração do componente.

SDK	Versão mínima	
AWS IoT Device SDKpara Java v2	v1.2.10	
AWS IoT Device SDKpara Python v2	v1.5.3	
AWS IoT Device SDKpara C++ v2	v1.17.0	
AWS IoT Device SDKpara JavaScript v2	v1.12.0	

GetConfiguration

Obtém um valor de configuração para um componente no dispositivo principal. Você especifica o caminho da chave para o qual obter um valor de configuração.

Solicitação

A solicitação dessa operação tem os seguintes parâmetros:

`componentName`(Python:) `component_name`

(Opcional) O nome do componente.

O padrão é o nome do componente que faz a solicitação.

`keyPath`(Python:) `key_path`

O caminho principal para o valor da configuração. Especifique uma lista em que cada entrada seja a chave para um único nível no objeto de configuração. Por exemplo, especifique `["mqtt", "port"]` para obter o valor de `port` na configuração a seguir.

```
{
  "mqtt": {
    "port": 443
  }
}
```

Para obter a configuração completa do componente, especifique uma lista vazia.

Resposta

A resposta dessa operação tem as seguintes informações:

`componentName`(Python:) `component_name`

O nome do componente.

`value`

A configuração solicitada como um objeto.

UpdateConfiguration

Atualiza um valor de configuração para esse componente no dispositivo principal.

Solicitação

A solicitação dessa operação tem os seguintes parâmetros:

`keyPath`(Python:) `key_path`

(Opcional) O caminho da chave para o nó do contêiner (o objeto) a ser atualizado. Especifique uma lista em que cada entrada seja a chave para um único nível no objeto de configuração. Por exemplo, especifique o caminho da chave `["mqtt"]` e o valor de mesclagem `{ "port": 443 }` para definir o valor `port` na configuração a seguir.

```
{
  "mqtt": {
    "port": 443
  }
}
```

O caminho da chave deve especificar um nó de contêiner (um objeto) na configuração. Se o nó não existir na configuração do componente, essa operação o cria e define seu valor para o objeto em`valueToMerge`.

O padrão é a raiz do objeto de configuração.

`timestamp`

O tempo atual da época do Unix em milissegundos. Essa operação usa esse carimbo de data/hora para resolver atualizações simultâneas na chave. Se a chave na configuração do componente tiver um carimbo de data/hora maior do que o carimbo de data/hora na solicitação, a solicitação falhará.

`valueToMerge`(Python:) `value_to_merge`

O objeto de configuração a ser mesclado no local especificado por você. `keyPath` Para ter mais informações, consulte Atualizar configurações de componentes.

Resposta

Essa operação não fornece nenhuma informação em sua resposta.

SubscribeToConfigurationUpdate

Inscreva-se para receber notificações quando a configuração de um componente for atualizada. Ao assinar uma chave, você recebe uma notificação quando algum filho dessa chave é atualizado.

Essa operação é uma operação de assinatura em que você assina um fluxo de mensagens de eventos. Para usar essa operação, defina um manipulador de resposta de fluxo com funções que manipulam mensagens de eventos, erros e encerramento de fluxo. Para ter mais informações, consulte Inscreva-se nos streams de eventos do IPC.

Tipo de mensagem do evento: `ConfigurationUpdateEvents`

Solicitação

A solicitação dessa operação tem os seguintes parâmetros:

`componentName`(Python:) `component_name`

(Opcional) O nome do componente.

O padrão é o nome do componente que faz a solicitação.

`keyPath`(Python:) `key_path`

O caminho principal para o valor de configuração para o qual se inscrever. Especifique uma lista em que cada entrada seja a chave para um único nível no objeto de configuração. Por exemplo, especifique `["mqtt", "port"]` para obter o valor de `port` na configuração a seguir.

```
{
  "mqtt": {
    "port": 443
  }
}
```

Para assinar as atualizações de todos os valores na configuração do componente, especifique uma lista vazia.

Resposta

A resposta dessa operação tem as seguintes informações:

`messages`

O fluxo de mensagens de notificação. Esse objeto,`ConfigurationUpdateEvents`, contém as seguintes informações:

`configurationUpdateEvent`(Python:) `configuration_update_event`

O evento de atualização da configuração. Esse objeto,`ConfigurationUpdateEvent`, contém as seguintes informações:

`componentName`(Python:) `component_name`

O nome do componente.

`keyPath`(Python:) `key_path`

O caminho principal para o valor de configuração que foi atualizado.

SubscribeToValidateConfigurationUpdates

Inscreva-se para receber notificações antes das atualizações de configuração desse componente. Isso permite que os componentes validem as atualizações de suas próprias configurações. Use a SendConfigurationValidityReport operação para dizer ao núcleo se a configuração é válida ou não.

> **Important**
>
> As implantações locais não notificam os componentes sobre atualizações.

Essa operação é uma operação de assinatura em que você assina um fluxo de mensagens de eventos. Para usar essa operação, defina um manipulador de resposta de fluxo com funções que manipulam mensagens de eventos, erros e encerramento de fluxo. Para ter mais informações, consulte Inscreva-se nos streams de eventos do IPC.

Tipo de mensagem do evento: `ValidateConfigurationUpdateEvents`

Solicitação

A solicitação dessa operação não tem nenhum parâmetro.

Resposta

A resposta dessa operação tem as seguintes informações:

`messages`

O fluxo de mensagens de notificação. Esse objeto,`ValidateConfigurationUpdateEvents`, contém as seguintes informações:

`validateConfigurationUpdateEvent`(Python:) `validate_configuration_update_event`

O evento de atualização da configuração. Esse objeto,`ValidateConfigurationUpdateEvent`, contém as seguintes informações:

`deploymentId`(Python:) `deployment_id`

O ID da AWS IoT Greengrass implantação que atualiza o componente.

`configuration`

O objeto que contém a nova configuração.

SendConfigurationValidityReport

Diga ao núcleo se uma atualização de configuração desse componente é válida ou não. A implantação falhará se você informar ao núcleo que a nova configuração não é válida. Use a SubscribeToValidateConfigurationUpdates operação para se inscrever para validar as atualizações de configuração.

Se um componente não responder a uma notificação de atualização de configuração de validação, o núcleo aguardará a quantidade de tempo especificada na política de validação de configuração

da implantação. Após esse tempo limite, o núcleo prossegue com a implantação. O tempo limite padrão de validação do componente é de 20 segundos. Para obter mais informações, consulte Criar implantações e o DeploymentConfigurationValidationPolicyobjeto que você pode fornecer ao chamar a CreateDeploymentoperação.

Solicitação

A solicitação dessa operação tem os seguintes parâmetros:

`configurationValidityReport`(Python:) `configuration_validity_report`

O relatório que informa ao núcleo se a atualização da configuração é válida ou não. Esse objeto,`ConfigurationValidityReport`, contém as seguintes informações:

`status`

O status de validade. Essa enumeração,`ConfigurationValidityStatus`, tem os seguintes valores:

- `ACCEPTED`— A configuração é válida e o núcleo pode aplicá-la a esse componente.
- `REJECTED`— A configuração não é válida e a implantação falha.

`deploymentId`(Python:) `deployment_id`

O ID da AWS IoT Greengrass implantação que solicitou a atualização da configuração.

`message`

(Opcional) Uma mensagem que informa por que a configuração não é válida.

Resposta

Essa operação não fornece nenhuma informação em sua resposta.

Recuperar valores secretos

Use o serviço IPC do gerenciador de segredos para recuperar valores secretos dos segredos no dispositivo principal. Você usa o componente gerenciador de segredos para implantar segredos criptografados nos dispositivos principais. Em seguida, você pode usar uma operação IPC para descriptografar o segredo e usar seu valor em seus componentes personalizados.

Tópicos

Versões mínimas do SDK

A tabela a seguir lista as versões mínimas do AWS IoT Device SDK que você deve usar para recuperar valores secretos de segredos no dispositivo principal.

SDK	Versão mínima
AWS IoT Device SDKpara Java v2	v1.2.10
AWS IoT Device SDKpara Python v2	v1.5.3
AWS IoT Device SDKpara C++ v2	v1.17.0
AWS IoT Device SDKpara JavaScript v2	v1.12.0

Autorização

Para usar o gerenciador de segredos em um componente personalizado, você deve definir políticas de autorização que permitam que seu componente obtenha o valor dos segredos que você armazena no dispositivo principal. Para obter informações sobre como definir políticas de autorização, consulteAutorize componentes a realizar operações de IPC.

As políticas de autorização para o gerenciador secreto têm as seguintes propriedades.

Identificador de serviço IPC: `aws.greengrass.SecretManager`

Operation	Descrição	Recursos
`aws.greengrass#GetSecretValue`	Permite que um componente obtenha o valor dos segredos criptografados no dispositivo principal.	Um ARN secreto do Secrets Manager, ou * para permitir acesso a todos os segredos.

Exemplos de políticas de autorização

Você pode consultar o exemplo de política de autorização a seguir para ajudá-lo a configurar políticas de autorização para seus componentes.

Example Exemplo de política de autorização

O exemplo de política de autorização a seguir permite que um componente obtenha o valor de qualquer segredo no dispositivo principal.

> **Note**
>
> Recomendamos que, em um ambiente de produção, você reduza o escopo da política de autorização para que o componente recupere somente os segredos que ele usa. Você pode alterar o * caractere curinga para uma lista de ARNs secretos ao implantar o componente.

```
{
  "accessControl": {
    "aws.greengrass.SecretManager": {
      "com.example.MySecretComponent:secrets:1": {
        "policyDescription": "Allows access to a secret.",
        "operations": [
          "aws.greengrass#GetSecretValue"
        ],
        "resources": [
          "*"
        ]
      }
    }
  }
}
```

GetSecretValue

Obtém o valor de um segredo que você armazena no dispositivo principal.

Essa operação é semelhante à operação do Secrets Manager que você pode usar para obter o valor de um segredo noNuvem AWS. Para obter mais informações, consulte GetSecretValue na Referência da API do AWS Secrets Manager.

Solicitação

A solicitação dessa operação tem os seguintes parâmetros:

`secretId`(Python:) `secret_id`

O nome do segredo a ser descoberto. Você pode especificar o Amazon Resource Name (ARN) ou o nome amigável do segredo.

`versionId`(Python:) `version_id`

(Opcional) O ID da versão a ser obtida.

Você pode especificar `versionId` ou `versionStage`.

Se você não especificar `versionId` ou`versionStage`, essa operação usará como padrão a versão com o `AWSCURRENT` rótulo.

`versionStage`(Python:) `version_stage`

(Opcional) O rótulo de teste da versão a ser obtida.

Você pode especificar `versionId` ou `versionStage`.

Se você não especificar `versionId` ou`versionStage`, essa operação usará como padrão a versão com o `AWSCURRENT` rótulo.

Resposta

A resposta dessa operação tem as seguintes informações:

`secretId`(Python:) `secret_id`

A identificação do segredo.

`versionId`(Python:) `version_id`

O ID dessa versão do segredo.

`versionStage`(Python:) `version_stage`

A lista de etiquetas de teste anexadas a esta versão do segredo.

`secretValue`(Python:) `secret_value`

O valor dessa versão do segredo. Esse objeto,`SecretValue`, contém as seguintes informações.

`secretString`(Python:) `secret_string`

A parte descriptografada das informações secretas protegidas que você forneceu ao Secrets Manager como uma string.

`secretBinary`(Python:) `secret_binary`

(Opcional) A parte descriptografada das informações secretas protegidas que você forneceu ao Secrets Manager como dados binários na forma de uma matriz de bytes. Essa propriedade contém os dados binários como uma string codificada em base64.

Essa propriedade não será usada se você criou o segredo no console do Secrets Manager.

Exemplos

Os exemplos a seguir demonstram como chamar essa operação no código de componente personalizado.

Java (IPC client V1)

Example Exemplo: Obter um valor secreto

> **Note**
>
> Este exemplo usa uma `IPCUtils` classe para criar uma conexão com o serviço AWS IoT Greengrass Core IPC. Para ter mais informações, consulte Conecte-se ao serviço AWS IoT Greengrass Core IPC.

```
package com.aws.greengrass.docs.samples.ipc;

import com.aws.greengrass.docs.samples.ipc.util.IPCUtils;
```

```
import software.amazon.awssdk.aws.greengrass.GetSecretValueResponseHandler;
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.model.GetSecretValueRequest;
import software.amazon.awssdk.aws.greengrass.model.GetSecretValueResponse;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;

import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class GetSecretValue {

    public static final int TIMEOUT_SECONDS = 10;

    public static void main(String[] args) {
        String secretArn = args[0];
        String versionStage = args[1];
        try (EventStreamRPCConnection eventStreamRPCConnection =
                     IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient =
                    new GreengrassCoreIPCClient(eventStreamRPCConnection);
            GetSecretValueResponseHandler responseHandler =
                    GetSecretValue.getSecretValue(ipcClient, secretArn,
 versionStage);
            CompletableFuture<GetSecretValueResponse> futureResponse =
                    responseHandler.getResponse();
            try {
                GetSecretValueResponse response =
 futureResponse.get(TIMEOUT_SECONDS, TimeUnit.SECONDS);
                response.getSecretValue().postFromJson();
                String secretString = response.getSecretValue().getSecretString();
                System.out.println("Successfully retrieved secret value: " +
 secretString);
            } catch (TimeoutException e) {
                System.err.println("Timeout occurred while retrieving secret: " +
 secretArn);
            } catch (ExecutionException e) {
                if (e.getCause() instanceof UnauthorizedError) {
                    System.err.println("Unauthorized error while retrieving secret:
 " + secretArn);
                } else {
```

```
                    throw e;
                }
            }
        } catch (InterruptedException e) {
            System.out.println("IPC interrupted.");
        } catch (ExecutionException e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static GetSecretValueResponseHandler
 getSecretValue(GreengrassCoreIPCClient greengrassCoreIPCClient, String secretArn,
 String versionStage) {
        GetSecretValueRequest getSecretValueRequest = new GetSecretValueRequest();
        getSecretValueRequest.setSecretId(secretArn);
        getSecretValueRequest.setVersionStage(versionStage);
        return greengrassCoreIPCClient.getSecretValue(getSecretValueRequest,
 Optional.empty());
    }
}
```

Python (IPC client V1)

Example Exemplo: Obter um valor secreto

> **Note**
>
> Este exemplo pressupõe que você esteja usando a versão 1.5.4 ou posterior do for AWS IoT Device SDK Python v2.

```
import json

import awsiot.greengrasscoreipc
from awsiot.greengrasscoreipc.model import (
    GetSecretValueRequest,
    GetSecretValueResponse,
    UnauthorizedError
)
```

```
secret_id = 'arn:aws:secretsmanager:us-
west-2:123456789012:secret:MyGreengrassSecret-abcdef'
TIMEOUT = 10

ipc_client = awsiot.greengrasscoreipc.connect()

request = GetSecretValueRequest()
request.secret_id = secret_id
request.version_stage = 'AWSCURRENT'
operation = ipc_client.new_get_secret_value()
operation.activate(request)
future_response = operation.get_response()
response = future_response.result(TIMEOUT)
secret_json = json.loads(response.secret_value.secret_string)
# Handle secret value.
```

JavaScript

Example Exemplo: Obter um valor secreto

```
import {
    GetSecretValueRequest,
} from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc/model';
import * as greengrasscoreipc from "aws-iot-device-sdk-v2/dist/greengrasscoreipc";

class GetSecretValue {
    private readonly secretId : string;
    private readonly versionStage : string;
    private ipcClient : greengrasscoreipc.Client

    constructor() {
        this.secretId = "<define_your_own_secretId>"
        this.versionStage = "<define_your_own_versionStage>"

        this.getSecretValue().then(r => console.log("Started workflow"));
    }

    private async getSecretValue() {
        try {
            this.ipcClient = await getIpcClient();
```

```
            const getSecretValueRequest : GetSecretValueRequest = {
                secretId: this.secretId,
                versionStage: this.versionStage,
            };

            const result = await
 this.ipcClient.getSecretValue(getSecretValueRequest);
            const secretString = result.secretValue.secretString;
            console.log("Successfully retrieved secret value: " + secretString)
        } catch (e) {
            // parse the error depending on your use cases
            throw e
        }
    }
}

export async function getIpcClient(){
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

const getSecretValue = new GetSecretValue();
```

Exemplos

Use os exemplos a seguir para aprender a usar o serviço IPC do gerenciador secreto em seus componentes.

Exemplo: segredo de impressão (Python, cliente IPC V1)

Esse componente de exemplo imprime o valor de um segredo que você implanta no dispositivo principal.

> **Important**
>
> Esse componente de exemplo imprime o valor de um segredo, portanto, use-o somente com segredos que armazenam dados de teste. Não use esse componente para imprimir o valor de um segredo que armazena informações importantes.

Tópicos

Fórmula

O exemplo de receita a seguir define um parâmetro secreto de configuração do ARN e permite que o componente obtenha o valor de qualquer segredo no dispositivo principal.

> **Note**
>
> Recomendamos que, em um ambiente de produção, você reduza o escopo da política de autorização para que o componente recupere somente os segredos que ele usa. Você pode alterar o * caractere curinga para uma lista de ARNs secretos ao implantar o componente.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.PrintSecret",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Prints the value of an AWS Secrets Manager secret.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.SecretManager": {
      "VersionRequirement": "^2.0.0",
      "DependencyType": "HARD"
    }
  },
  "ComponentConfiguration": {
```

```
    "DefaultConfiguration": {
      "SecretArn": "",
      "accessControl": {
        "aws.greengrass.SecretManager": {
          "com.example.PrintSecret:secrets:1": {
            "policyDescription": "Allows access to a secret.",
            "operations": [
              "aws.greengrass#GetSecretValue"
            ],
            "resources": [
              "*"
            ]
          }
        }
      }
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "python3 -m pip install --user awsiotsdk",
        "run": "python3 -u {artifacts:path}/print_secret.py \"{configuration:/
SecretArn}\""
      }
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "install": "py -3 -m pip install --user awsiotsdk",
        "run": "py -3 -u {artifacts:path}/print_secret.py \"{configuration:/
SecretArn}\""
      }
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.PrintSecret
ComponentVersion: 1.0.0
ComponentDescription: Prints the value of a Secrets Manager secret.
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.SecretManager:
    VersionRequirement: "^2.0.0"
    DependencyType: HARD
ComponentConfiguration:
  DefaultConfiguration:
    SecretArn: ''
    accessControl:
      aws.greengrass.SecretManager:
        com.example.PrintSecret:secrets:1:
          policyDescription: Allows access to a secret.
          operations:
            - aws.greengrass#GetSecretValue
          resources:
            - "*"
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: python3 -m pip install --user awsiotsdk
      run: python3 -u {artifacts:path}/print_secret.py "{configuration:/SecretArn}"
  - Platform:
      os: windows
    Lifecycle:
      install: py -3 -m pip install --user awsiotsdk
      run: py -3 -u {artifacts:path}/print_secret.py "{configuration:/SecretArn}"
```

Artefatos

O exemplo de aplicativo Python a seguir demonstra como usar o serviço IPC do gerenciador de segredos para obter o valor de um segredo no dispositivo principal.

```
import concurrent.futures
import json
```

```
import sys
import traceback

import awsiot.greengrasscoreipc
from awsiot.greengrasscoreipc.model import (
    GetSecretValueRequest,
    GetSecretValueResponse,
    UnauthorizedError
)

TIMEOUT = 10

if len(sys.argv) == 1:
    print('Provide SecretArn in the component configuration.', file=sys.stdout)
    exit(1)

secret_id = sys.argv[1]

try:
    ipc_client = awsiot.greengrasscoreipc.connect()

    request = GetSecretValueRequest()
    request.secret_id = secret_id
    operation = ipc_client.new_get_secret_value()
    operation.activate(request)
    future_response = operation.get_response()

    try:
        response = future_response.result(TIMEOUT)
        secret_json = json.loads(response.secret_value.secret_string)
        print('Successfully got secret: ' + secret_id)
        print('Secret value: ' + str(secret_json))
    except concurrent.futures.TimeoutError:
        print('Timeout occurred while getting secret: ' + secret_id, file=sys.stderr)
    except UnauthorizedError as e:
        print('Unauthorized error while getting secret: ' + secret_id,
 file=sys.stderr)
        raise e
    except Exception as e:
        print('Exception while getting secret: ' + secret_id, file=sys.stderr)
        raise e
except Exception:
    print('Exception occurred when using IPC.', file=sys.stderr)
    traceback.print_exc()
```

```
    exit(1)
```

Uso

Você pode usar esse componente de exemplo com o componente gerenciador de segredos para implantar e imprimir o valor de um segredo em seu dispositivo principal.

Para criar, implantar e imprimir um segredo de teste

1. Crie um segredo do Secrets Manager com dados de teste.

   Linux or Unix

   ```
   aws secretsmanager create-secret \
     --name MyTestGreengrassSecret \
     --secret-string '{"my-secret-key": "my-secret-value"}'
   ```

   Windows Command Prompt (CMD)

   ```
   aws secretsmanager create-secret ^
     --name MyTestGreengrassSecret ^
     --secret-string '{"my-secret-key": "my-secret-value"}'
   ```

   PowerShell

   ```
   aws secretsmanager create-secret `
     --name MyTestGreengrassSecret `
     --secret-string '{"my-secret-key": "my-secret-value"}'
   ```

   Salve o ARN do segredo para usar nas etapas a seguir.

   Para obter mais informações, consulte Criação de um segredo no Guia AWS Secrets Manager do usuário.

2. Implante o componente do gerenciador secreto (`aws.greengrass.SecretManager`) com a seguinte atualização de mesclagem de configuração. Especifique o ARN do segredo que você criou anteriormente.

   ```
   {
     "cloudSecrets": [
   ```

```
    {
      "arn": "arn:aws:secretsmanager:us-
west-2:123456789012:secret:MyTestGreengrassSecret-abcdef"
    }
  ]
}
```

Para obter mais informações, consulte Implemente AWS IoT Greengrass componentes em dispositivos ou o comando de implantação da CLI do Greengrass.

3. Crie e implante o componente de exemplo nesta seção com a seguinte atualização de mesclagem de configuração. Especifique o ARN do segredo que você criou anteriormente.

```
{
  "SecretArn": "arn:aws:secretsmanager:us-
west-2:123456789012:secret:MyTestGreengrassSecret",
  "accessControl": {
    "aws.greengrass.SecretManager": {
      "com.example.PrintSecret:secrets:1": {
        "policyDescription": "Allows access to a secret.",
        "operations": [
          "aws.greengrass#GetSecretValue"
        ],
        "resources": [
          "arn:aws:secretsmanager:us-
west-2:123456789012:secret:MyTestGreengrassSecret-abcdef"
        ]
      }
    }
  }
}
```

Para obter mais informações, consulte Crie AWS IoT Greengrass componentes.

4. Visualize os registros do software AWS IoT Greengrass principal para verificar se as implantações foram bem-sucedidas e veja o registro do `com.example.PrintSecret` componente para ver o valor secreto impresso. Para ter mais informações, consulte Monitore AWS IoT Greengrass os registros.

Interaja com sombras locais

Use o serviço IPC de sombra para interagir com sombras locais em um dispositivo. O dispositivo com o qual você escolhe interagir pode ser seu dispositivo principal ou um dispositivo cliente conectado.

Para usar essas operações de IPC, inclua o componente do gerenciador de sombras como uma dependência em seu componente personalizado. Em seguida, você pode usar as operações de IPC em seus componentes personalizados para interagir com sombras locais em seu dispositivo por meio do gerenciador de sombras. Para permitir que componentes personalizados reajam às mudanças nos estados de sombra locais, você também pode usar o serviço IPC de publicação/assinatura para assinar eventos paralelos. Para obter mais informações sobre como usar o serviço de publicação/assinatura, consulte o. Publique/assine mensagens locais

> **Note**
>
> Para permitir que um dispositivo principal interaja com as sombras do dispositivo cliente, você também deve configurar e implantar o componente de ponte MQTT. Para obter mais informações, consulte Habilitar o shadow manager para se comunicar com dispositivos clientes.

Tópicos

- Versões mínimas do SDK
- Autorização
- GetThingShadow
- UpdateThingShadow
- DeleteThingShadow
- ListNamedShadowsForThing

Versões mínimas do SDK

A tabela a seguir lista as versões mínimas do AWS IoT Device SDK que você deve usar para interagir com sombras locais.

SDK	Versão mínima	
AWS IoT Device SDKpara Java v2	v1.4.0	
AWS IoT Device SDKpara Python v2	v1.6.0	
AWS IoT Device SDKpara C++ v2	v1.17.0	
AWS IoT Device SDKpara JavaScript v2	v1.12.0	

Autorização

Para usar o serviço IPC paralelo em um componente personalizado, você deve definir políticas de autorização que permitam que seu componente interaja com sombras. Para obter informações sobre a definição de políticas de autorização, consulteAutorize componentes a realizar operações de IPC.

As políticas de autorização para interação paralela têm as seguintes propriedades.

Identificador de serviço IPC: `aws.greengrass.ShadowManager`

Operation	Descrição	Recursos
`aws.greengrass#GetThingShadow`	Permite que um component e recupere a sombra de uma coisa.	Uma das seguintes sequências de caracteres: • `$aws/things/`*`thingName`*`/shadow/`, para permitir o acesso à sombra clássica do dispositivo. • `$aws/things/`*`thingName`*`/shadow/n`

Operation	Descrição	Recursos
		ame/ *shadowName* , para permitir o acesso a uma sombra nomeada. • *para permitir o acesso a todas as sombras.
`aws.greengrass#UpdateThingShadow`	Permite que um componente atualize a sombra de uma coisa.	Uma das seguintes sequências de caracteres: • `$aws/things/` *thingName* `/shadow/`, para permitir o acesso à sombra clássica do dispositivo. • `$aws/things/` *thingName* `/shadow/name/` *shadowName* , para permitir o acesso a uma sombra nomeada. • *para permitir o acesso a todas as sombras.

Operation	Descrição	Recursos
`aws.greengrass#DeleteThingShadow`	Permite que um componente exclua a sombra de uma coisa.	Uma das seguintes sequências de caracteres: • `$aws/things/` *`thingName`* `/shadow/`, para permitir o acesso à sombra clássica do dispositivo • `$aws/things/` *`thingName`* `/shadow/name/` *`shadowName`* , para permitir o acesso a uma sombra nomeada • `*`, para permitir o acesso a todas as sombras.
`aws.greengrass#ListNamedShadowsForThing`	Permite que um componente recupere a lista de sombras nomeadas de uma coisa.	Uma string de nome de coisa que permite acessar a coisa para listar suas sombras. Use `*` para permitir o acesso a todas as coisas.

Identificador de serviço IPC: `aws.greengrass.ipc.pubsub`

Operation	Descrição	Recursos
`aws.greengrass#SubscribeToTopic`	Permite que um componente assine mensagens para os tópicos que você especificar.	Uma das seguintes sequências de tópicos: • *`shadowTopicPrefix`* `/get/accepted`

Operation	Descrição	Recursos
		• *shadowTopicPrefix* / get/rejected • *shadowTopicPrefix* / delete/accepted • *shadowTopicPrefix* / delete/rejected • *shadowTopicPrefix* / update/accepted • *shadowTopicPrefix* / update/delta • *shadowTopicPrefix* / update/rejected O valor do prefixo *shadowTopicPrefix* do tópico depende do tipo de sombra: • Sombra clássica: $aws/things/ *thingName* /shadow • Sombra nomeada: $aws/things/ *thingName* /shadow/name/ *shadowName* Use * para permitir o acesso a todos os tópicos. No Greengrass nucleus v2.6.0 e versões posteriores, você pode se inscrever em tópicos que contêm curingas de tópicos do MQTT (e). # +

Operation	Descrição	Recursos
		Essa string de tópico suporta curingas de tópico MQTT como caracteres literais. Por exemplo, se a política de autorização de um component e conceder acesso `atest/topic/#` , o component e pode se inscrever`test/topic/#` , mas não pode assinar`test/topic/filter`.

Variáveis de receita nas políticas locais de autorização paralela

Se você usar a versão 2.6.0 ou posterior do núcleo do Greengrass e definir a opção de interpolateComponentConfigurationconfiguração do núcleo do Greengrass como`true`, poderá usar a variável de receita nas políticas de autorização. `{iot:thingName}` Esse recurso permite que você configure uma única política de autorização para um grupo de dispositivos principais, em que cada dispositivo principal pode acessar somente sua própria sombra. Por exemplo, você pode permitir que um componente acesse o seguinte recurso para operações de IPC paralelas.

```
$aws/things/{iot:thingName}/shadow/
```

Exemplos de políticas de autorização

Você pode consultar os seguintes exemplos de políticas de autorização para ajudá-lo a configurar políticas de autorização para seus componentes.

Example Exemplo: permitir que um grupo de dispositivos principais interaja com sombras locais

> **Important**
>
> Este exemplo usa um recurso que está disponível para a versão 2.6.0 e posterior do componente núcleo do Greengrass. O Greengrass nucleus v2.6.0 adiciona suporte para a maioria das variáveis de receita, como configurações de componentes. `{iot:thingName}` Para ativar esse recurso, defina a opção de interpolateComponentConfigurationconfiguração

do núcleo Greengrass como. `true` Para ver um exemplo que funciona para todas as versões do núcleo do Greengrass, consulte o exemplo de política de autorização para um único dispositivo central.

O exemplo de política de autorização a seguir permite que o componente `com.example.MyShadowInteractionComponent` interaja com a sombra clássica do dispositivo e a sombra `myNamedShadow` nomeada do dispositivo principal que executa o componente. Essa política também permite que esse componente receba mensagens sobre tópicos locais para essas sombras.

JSON

```
{
  "accessControl": {
    "aws.greengrass.ShadowManager": {
      "com.example.MyShadowInteractionComponent:shadow:1": {
        "policyDescription": "Allows access to shadows",
        "operations": [
          "aws.greengrass#GetThingShadow",
          "aws.greengrass#UpdateThingShadow",
          "aws.greengrass#DeleteThingShadow"
        ],
        "resources": [
          "$aws/things/{iot:thingName}/shadow",
          "$aws/things/{iot:thingName}/shadow/name/myNamedShadow"
        ]
      },
      "com.example.MyShadowInteractionComponent:shadow:2": {
        "policyDescription": "Allows access to things with shadows",
        "operations": [
          "aws.greengrass#ListNamedShadowsForThing"
        ],
        "resources": [
          "{iot:thingName}"
        ]
      }
    },
    "aws.greengrass.ipc.pubsub": {
      "com.example.MyShadowInteractionComponent:pubsub:1": {
        "policyDescription": "Allows access to shadow pubsub topics",
        "operations": [
```

```
          "aws.greengrass#SubscribeToTopic"
        ],
        "resources": [
          "$aws/things/{iot:thingName}/shadow/get/accepted",
          "$aws/things/{iot:thingName}/shadow/name/myNamedShadow/get/accepted"
        ]
      }
    }
  }
}
```

YAML

```
accessControl:
  aws.greengrass.ShadowManager:
    'com.example.MyShadowInteractionComponent:shadow:1':
      policyDescription: 'Allows access to shadows'
      operations:
        - 'aws.greengrass#GetThingShadow'
        - 'aws.greengrass#UpdateThingShadow'
        - 'aws.greengrass#DeleteThingShadow'
      resources:
        - $aws/things/{iot:thingName}/shadow
        - $aws/things/{iot:thingName}/shadow/name/myNamedShadow
    'com.example.MyShadowInteractionComponent:shadow:2':
      policyDescription: 'Allows access to things with shadows'
      operations:
        - 'aws.greengrass#ListNamedShadowsForThing'
      resources:
        - '{iot:thingName}'
  aws.greengrass.ipc.pubsub:
    'com.example.MyShadowInteractionComponent:pubsub:1':
      policyDescription: 'Allows access to shadow pubsub topics'
      operations:
        - 'aws.greengrass#SubscribeToTopic'
      resources:
        - $aws/things/{iot:thingName}/shadow/get/accepted
        - $aws/things/{iot:thingName}/shadow/name/myNamedShadow/get/accepted
```

Example Exemplo: permitir que um grupo de dispositivos principais interaja com as sombras do dispositivo cliente

> **Important**
>
> Esse recurso requer o Greengrass nucleus v2.6.0 ou posterior, o shadow manager v2.2.0 ou posterior e o MQTT bridge v2.2.0 ou posterior. Você deve configurar a ponte MQTT para permitir que o shadow manager se comunique com os dispositivos do cliente.

O exemplo de política de autorização a seguir permite que `com.example.MyShadowInteractionComponent` o componente interaja com todas as sombras de dispositivos clientes cujos nomes começam com`MyClientDevice`.

> **Note**
>
> Para permitir que um dispositivo principal interaja com as sombras do dispositivo cliente, você também deve configurar e implantar o componente de ponte MQTT. Para obter mais informações, consulte Habilitar o shadow manager para se comunicar com dispositivos clientes.

JSON

```
{
  "accessControl": {
    "aws.greengrass.ShadowManager": {
      "com.example.MyShadowInteractionComponent:shadow:1": {
        "policyDescription": "Allows access to shadows",
        "operations": [
          "aws.greengrass#GetThingShadow",
          "aws.greengrass#UpdateThingShadow",
          "aws.greengrass#DeleteThingShadow"
        ],
        "resources": [
          "$aws/things/MyClientDevice*/shadow",
          "$aws/things/MyClientDevice*/shadow/name/*"
        ]
      },
      "com.example.MyShadowInteractionComponent:shadow:2": {
        "policyDescription": "Allows access to things with shadows",
```

```
        "operations": [
          "aws.greengrass#ListNamedShadowsForThing"
        ],
        "resources": [
          "MyClientDevice*"
        ]
      }
    }
  }
}
```

YAML

```
accessControl:
  aws.greengrass.ShadowManager:
    'com.example.MyShadowInteractionComponent:shadow:1':
      policyDescription: 'Allows access to shadows'
      operations:
        - 'aws.greengrass#GetThingShadow'
        - 'aws.greengrass#UpdateThingShadow'
        - 'aws.greengrass#DeleteThingShadow'
      resources:
        - $aws/things/MyClientDevice*/shadow
        - $aws/things/MyClientDevice*/shadow/name/*
    'com.example.MyShadowInteractionComponent:shadow:2':
      policyDescription: 'Allows access to things with shadows'
      operations:
        - 'aws.greengrass#ListNamedShadowsForThing'
      resources:
        - MyClientDevice*
```

Example Exemplo: permitir que um dispositivo de núcleo único interaja com sombras locais

O exemplo de política de autorização a seguir permite que o componente `com.example.MyShadowInteractionComponent` interaja com a sombra clássica do dispositivo e a sombra `myNamedShadow` nomeada do dispositivo`MyThingName`. Essa política também permite que esse componente receba mensagens sobre tópicos locais para essas sombras.

JSON

```
{
```

```
  "accessControl": {
    "aws.greengrass.ShadowManager": {
      "com.example.MyShadowInteractionComponent:shadow:1": {
        "policyDescription": "Allows access to shadows",
        "operations": [
          "aws.greengrass#GetThingShadow",
          "aws.greengrass#UpdateThingShadow",
          "aws.greengrass#DeleteThingShadow"
        ],
        "resources": [
          "$aws/things/MyThingName/shadow",
          "$aws/things/MyThingName/shadow/name/myNamedShadow"
        ]
      },
      "com.example.MyShadowInteractionComponent:shadow:2": {
        "policyDescription": "Allows access to things with shadows",
        "operations": [
          "aws.greengrass#ListNamedShadowsForThing"
        ],
        "resources": [
          "MyThingName"
        ]
      }
    },
    "aws.greengrass.ipc.pubsub": {
      "com.example.MyShadowInteractionComponent:pubsub:1": {
        "policyDescription": "Allows access to shadow pubsub topics",
        "operations": [
          "aws.greengrass#SubscribeToTopic"
        ],
        "resources": [
          "$aws/things/MyThingName/shadow/get/accepted",
          "$aws/things/MyThingName/shadow/name/myNamedShadow/get/accepted"
        ]
      }
    }
  }
}
```

YAML

```
accessControl:
  aws.greengrass.ShadowManager:
```

```
    'com.example.MyShadowInteractionComponent:shadow:1':
      policyDescription: 'Allows access to shadows'
      operations:
        - 'aws.greengrass#GetThingShadow'
        - 'aws.greengrass#UpdateThingShadow'
        - 'aws.greengrass#DeleteThingShadow'
      resources:
        - $aws/things/MyThingName/shadow
        - $aws/things/MyThingName/shadow/name/myNamedShadow
    'com.example.MyShadowInteractionComponent:shadow:2':
      policyDescription: 'Allows access to things with shadows'
      operations:
        - 'aws.greengrass#ListNamedShadowsForThing'
      resources:
        - MyThingName
  aws.greengrass.ipc.pubsub:
    'com.example.MyShadowInteractionComponent:pubsub:1':
      policyDescription: 'Allows access to shadow pubsub topics'
      operations:
        - 'aws.greengrass#SubscribeToTopic'
      resources:
        - $aws/things/MyThingName/shadow/get/accepted
        - $aws/things/MyThingName/shadow/name/myNamedShadow/get/accepted
```

Example Exemplo: permitir que um grupo de dispositivos principais reaja às mudanças locais do estado de sombra

> **Important**
>
> Este exemplo usa um recurso que está disponível para a versão 2.6.0 e posterior do componente núcleo do Greengrass. O Greengrass nucleus v2.6.0 adiciona suporte para a maioria das variáveis de receita, como configurações de componentes. `{iot:thingName}` Para ativar esse recurso, defina a opção de interpolateComponentConfigurationconfiguração do núcleo Greengrass como. `true` Para ver um exemplo que funciona para todas as versões do núcleo do Greengrass, consulte o exemplo de política de autorização para um único dispositivo central.

O exemplo de política de controle de acesso a seguir permite que o cliente receba mensagens sobre o `/update/delta` tópico da sombra clássica do dispositivo e da sombra

nomeada `myNamedShadow` em cada dispositivo principal que executa o componente. `com.example.MyShadowReactiveComponent`

JSON

```
{
  "accessControl": {
    "aws.greengrass.ipc.pubsub": {
      "com.example.MyShadowReactiveComponent:pubsub:1": {
        "policyDescription": "Allows access to shadow pubsub topics",
        "operations": [
          "aws.greengrass#SubscribeToTopic"
        ],
        "resources": [
          "$aws/things/{iot:thingName}/shadow/update/delta",
          "$aws/things/{iot:thingName}/shadow/name/myNamedShadow/update/delta"
        ]
      }
    }
  }
}
```

YAML

```
accessControl:
  aws.greengrass.ipc.pubsub:
    "com.example.MyShadowReactiveComponent:pubsub:1":
      policyDescription: Allows access to shadow pubsub topics
      operations:
        - 'aws.greengrass#SubscribeToTopic'
      resources:
        - $aws/things/{iot:thingName}/shadow/update/delta
        - $aws/things/{iot:thingName}/shadow/name/myNamedShadow/update/delta
```

Example Exemplo: permitir que um dispositivo de núcleo único reaja às mudanças locais do estado de sombra

O exemplo de política de controle de acesso `com.example.MyShadowReactiveComponent` a seguir permite que o cliente receba mensagens sobre o `/update/delta` tópico da sombra clássica do dispositivo e a sombra `myNamedShadow` nomeada do dispositivo`MyThingName`.

JSON

```
{
  "accessControl": {
    "aws.greengrass.ipc.pubsub": {
      "com.example.MyShadowReactiveComponent:pubsub:1": {
        "policyDescription": "Allows access to shadow pubsub topics",
        "operations": [
          "aws.greengrass#SubscribeToTopic"
        ],
        "resources": [
          "$aws/things/MyThingName/shadow/update/delta",
          "$aws/things/MyThingName/shadow/name/myNamedShadow/update/delta"
        ]
      }
    }
  }
}
```

YAML

```
accessControl:
  aws.greengrass.ipc.pubsub:
    "com.example.MyShadowReactiveComponent:pubsub:1":
      policyDescription: Allows access to shadow pubsub topics
      operations:
        - 'aws.greengrass#SubscribeToTopic'
      resources:
        - $aws/things/MyThingName/shadow/update/delta
        - $aws/things/MyThingName/shadow/name/myNamedShadow/update/delta
```

GetThingShadow

Obtenha a sombra de uma coisa específica.

Solicitação

A solicitação dessa operação tem os seguintes parâmetros:

`thingName`(Python:) `thing_name`

O nome da coisa.

Tipo: `string`

`shadowName`(Python:) `shadow_name`

O nome do shadow. Para especificar a sombra clássica da coisa, defina esse parâmetro como uma string vazia (`""`).

> **Warning**
>
> O AWS IoT Greengrass serviço usa a sombra `AWSManagedGreengrassV2Deployment` nomeada para gerenciar implantações direcionadas a dispositivos principais individuais. Essa sombra nomeada é reservada para uso pelo AWS IoT Greengrass serviço. Não atualize nem exclua essa sombra nomeada.

Tipo: `string`

Resposta

A resposta dessa operação tem as seguintes informações:

`payload`

O documento de estado da resposta como um blob.

Tipo: `object` que contém as seguintes informações:

`state`

As informações do estado.

Esse objeto contém as seguintes informações.

`desired`

As propriedades e os valores do estado solicitados para serem atualizados no dispositivo.

Tipo: `map` de pares de valores-chave

`reported`

As propriedades e os valores do estado relatados pelo dispositivo.

Tipo: `map` de pares de valores-chave

`delta`

A diferença entre as propriedades e valores do estado desejado e relatado. Essa propriedade está presente somente se os `reported` estados `desired` e forem diferentes.

Tipo: `map` de pares de valores-chave

`metadata`

Os registros de data e hora de cada atributo nas `reported` seções `desired` e para que você possa determinar quando o estado foi atualizado.

Tipo: `string`

`timestamp`

A data e a hora da época em que a resposta foi gerada.

Tipo: `integer`

`clientToken`(Python:) `clientToken`

O token usado para corresponder à solicitação e à resposta correspondente

Tipo: `string`

`version`

A versão do documento paralelo local.

Tipo: `integer`

Erros

Essa operação pode retornar os seguintes erros.

`InvalidArgumentsError`

O serviço paralelo local não consegue validar os parâmetros da solicitação. Isso pode ocorrer se a solicitação contiver caracteres JSON malformados ou sem suporte.

`ResourceNotFoundError`

O documento paralelo local solicitado não foi encontrado.

`ServiceError`

Ocorreu um erro de serviço interno ou o número de solicitações ao serviço IPC excedeu os limites especificados nos parâmetros de `maxTotalLocalRequestsRate` configuração `maxLocalRequestsPerSecondPerThing` e no componente do gerenciador de sombra.

`UnauthorizedError`

A política de autorização do componente não inclui as permissões necessárias para essa operação.

Exemplos

Os exemplos a seguir demonstram como chamar essa operação no código de componente personalizado.

Java (IPC client V1)

Example Exemplo: pegue uma sombra

> **Note**
>
> Este exemplo usa uma `IPCUtils` classe para criar uma conexão com o serviço AWS IoT Greengrass Core IPC. Para ter mais informações, consulte Conecte-se ao serviço AWS IoT Greengrass Core IPC.

```
package com.aws.greengrass.docs.samples.ipc;

import com.aws.greengrass.docs.samples.ipc.util.IPCUtils;
import software.amazon.awssdk.aws.greengrass.GetThingShadowResponseHandler;
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.model.GetThingShadowRequest;
import software.amazon.awssdk.aws.greengrass.model.GetThingShadowResponse;
import software.amazon.awssdk.aws.greengrass.model.ResourceNotFoundError;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;

import java.nio.charset.StandardCharsets;
import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
```

```
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class GetThingShadow {

    public static final int TIMEOUT_SECONDS = 10;

    public static void main(String[] args) {
        // Use the current core device's name if thing name isn't set.
        String thingName = args[0].isEmpty() ? System.getenv("AWS_IOT_THING_NAME") :
 args[0];
        String shadowName = args[1];
        try (EventStreamRPCConnection eventStreamRPCConnection =
                     IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient =
                    new GreengrassCoreIPCClient(eventStreamRPCConnection);
            GetThingShadowResponseHandler responseHandler =
                    GetThingShadow.getThingShadow(ipcClient, thingName,
 shadowName);
            CompletableFuture<GetThingShadowResponse> futureResponse =
                    responseHandler.getResponse();
            try {
                GetThingShadowResponse response =
 futureResponse.get(TIMEOUT_SECONDS,
                        TimeUnit.SECONDS);
                String shadowPayload = new String(response.getPayload(),
 StandardCharsets.UTF_8);
                System.out.printf("Successfully got shadow %s/%s: %s%n", thingName,
 shadowName,
                        shadowPayload);
            } catch (TimeoutException e) {
                System.err.printf("Timeout occurred while getting shadow: %s/%s%n",
 thingName,
                        shadowName);
            } catch (ExecutionException e) {
                if (e.getCause() instanceof UnauthorizedError) {
                    System.err.printf("Unauthorized error while getting shadow: %s/
%s%n",
                            thingName, shadowName);
                } else if (e.getCause() instanceof ResourceNotFoundError) {
                    System.err.printf("Unable to find shadow to get: %s/%s%n",
 thingName,
                            shadowName);
                } else {
```

```
                    throw e;
                }
            }
        } catch (InterruptedException e) {
            System.out.println("IPC interrupted.");
        } catch (ExecutionException e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static GetThingShadowResponseHandler
 getThingShadow(GreengrassCoreIPCClient greengrassCoreIPCClient, String thingName,
 String shadowName) {
        GetThingShadowRequest getThingShadowRequest = new GetThingShadowRequest();
        getThingShadowRequest.setThingName(thingName);
        getThingShadowRequest.setShadowName(shadowName);
        return greengrassCoreIPCClient.getThingShadow(getThingShadowRequest,
 Optional.empty());
    }
}
```

Python (IPC client V1)

Example Exemplo: pegue uma sombra

```
import awsiot.greengrasscoreipc
import awsiot.greengrasscoreipc.client as client
from awsiot.greengrasscoreipc.model import GetThingShadowRequest

TIMEOUT = 10

def sample_get_thing_shadow_request(thingName, shadowName):
    try:
        # set up IPC client to connect to the IPC server
        ipc_client = awsiot.greengrasscoreipc.connect()

        # create the GetThingShadow request
        get_thing_shadow_request = GetThingShadowRequest()
        get_thing_shadow_request.thing_name = thingName
        get_thing_shadow_request.shadow_name = shadowName
```

```
        # retrieve the GetThingShadow response after sending the request to the IPC
 server
        op = ipc_client.new_get_thing_shadow()
        op.activate(get_thing_shadow_request)
        fut = op.get_response()

        result = fut.result(TIMEOUT)
        return result.payload

    except InvalidArgumentsError as e:
        # add error handling
        ...
    # except ResourceNotFoundError | UnauthorizedError | ServiceError
```

JavaScript

Example Exemplo: pegue uma sombra

```
import {
    GetThingShadowRequest
} from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc/model';
import * as greengrasscoreipc from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc';

class GetThingShadow {
    private ipcClient: greengrasscoreipc.Client;
    private thingName: string;
    private shadowName: string;

    constructor() {
        // Define args parameters here
        this.thingName = "<define_your_own_thingName>";
        this.shadowName = "<define_your_own_shadowName>";
        this.bootstrap();
    }

    async bootstrap() {
        try {
            this.ipcClient = await getIpcClient();
        } catch (err) {
            // parse the error depending on your use cases
            throw err
        }
```

```
        try {
            await this.handleGetThingShadowOperation(this.thingName,
                this.shadowName);
        } catch (err) {
            // parse the error depending on your use cases
            throw err
        }
    }

    async handleGetThingShadowOperation(
        thingName: string,
        shadowName: string
    ) {
        const request: GetThingShadowRequest = {
            thingName: thingName,
            shadowName: shadowName
        };
        const response = await this.ipcClient.getThingShadow(request);
    }
}

export async function getIpcClient() {
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use caseså
        throw err
    }
}

const startScript = new GetThingShadow();
```

UpdateThingShadow

Atualize a sombra para a coisa especificada. Se uma sombra não existe, uma é criada.

Solicitação

A solicitação dessa operação tem os seguintes parâmetros:

`thingName`(Python:) `thing_name`

O nome da coisa.

Tipo: `string`

`shadowName`(Python:) `shadow_name`

O nome do shadow. Para especificar a sombra clássica da coisa, defina esse parâmetro como uma string vazia (`""`).

> **Warning**
>
> O AWS IoT Greengrass serviço usa a sombra `AWSManagedGreengrassV2Deployment` nomeada para gerenciar implantações direcionadas a dispositivos principais individuais. Essa sombra nomeada é reservada para uso pelo AWS IoT Greengrass serviço. Não atualize nem exclua essa sombra nomeada.

Tipo: `string`

`payload`

O documento de estado da solicitação como um blob.

Tipo: `object` que contém as seguintes informações:

`state`

As informações do estado a serem atualizadas. Essa operação de IPC afeta somente os campos especificados.

Esse objeto contém as seguintes informações. Normalmente, você usará a `desired` propriedade ou a `reported` propriedade, mas não as duas na mesma solicitação.

`desired`

As propriedades e os valores do estado solicitados para serem atualizados no dispositivo.

Tipo: `map` de pares de valores-chave

`reported`

As propriedades e os valores do estado relatados pelo dispositivo.

Tipo: `map` de pares de valores-chave

`clientToken`(Python:) `client_token`

(Opcional) O token usado para corresponder à solicitação e à resposta correspondente do token do cliente.

Tipo: `string`

`version`

(Opcional) A versão do documento paralelo local a ser atualizada. O serviço paralelo processa a atualização somente se a versão especificada corresponder à versão mais recente que ela tem.

Tipo: `integer`

Resposta

A resposta dessa operação tem as seguintes informações:

`payload`

O documento de estado da resposta como um blob.

Tipo: `object` que contém as seguintes informações:

`state`

As informações do estado.

Esse objeto contém as seguintes informações.

`desired`

As propriedades e os valores do estado solicitados para serem atualizados no dispositivo.

Tipo: `map` de pares de valores-chave

`reported`

As propriedades e os valores do estado relatados pelo dispositivo.

Tipo: `map` de pares de valores-chave

`delta`

As propriedades e os valores do estado relatados pelo dispositivo.

Tipo: `map` de pares de valores-chave

`metadata`

Os registros de data e hora de cada atributo nas `reported` seções `desired` e para que você possa determinar quando o estado foi atualizado.

Tipo: `string`

`timestamp`

A data e a hora da época em que a resposta foi gerada.

Tipo: `integer`

`clientToken`(Python:) `client_token`

O token usado para corresponder à solicitação e à resposta correspondente.

Tipo: `string`

`version`

A versão do documento paralelo local após a conclusão da atualização.

Tipo: `integer`

Erros

Essa operação pode retornar os seguintes erros.

`ConflictError`

O serviço paralelo local encontrou um conflito de versão durante a operação de atualização. Isso ocorre quando a versão na carga útil da solicitação não corresponde à versão no último documento paralelo local disponível.

`InvalidArgumentsError`

O serviço paralelo local não consegue validar os parâmetros da solicitação. Isso pode ocorrer se a solicitação contiver caracteres JSON malformados ou sem suporte.

Um válido `payload` tem as seguintes propriedades:

- O `state` nó existe e é um objeto que contém as informações do `reported` estado `desired` ou.
- Os `reported` nós `desired` e são objetos ou nulos. Pelo menos um desses objetos deve conter informações de estado válidas.
- A profundidade dos `reported` objetos `desired` e não pode exceder oito nós.
- O tamanho do `clientToken` valor não pode exceder 64 caracteres.
- O `version` valor deve ser igual 1 ou superior.

`ServiceError`

Ocorreu um erro de serviço interno ou o número de solicitações ao serviço IPC excedeu os limites especificados nos parâmetros de `maxTotalLocalRequestsRate` configuração `maxLocalRequestsPerSecondPerThing` e no componente do gerenciador de sombra.

`UnauthorizedError`

A política de autorização do componente não inclui as permissões necessárias para essa operação.

Exemplos

Os exemplos a seguir demonstram como chamar essa operação no código de componente personalizado.

Java (IPC client V1)

Example Exemplo: atualizar a sombra de uma coisa

> **Note**
>
> Este exemplo usa uma `IPCUtils` classe para criar uma conexão com o serviço AWS IoT Greengrass Core IPC. Para ter mais informações, consulte Conecte-se ao serviço AWS IoT Greengrass Core IPC.

```
package com.aws.greengrass.docs.samples.ipc;

import com.aws.greengrass.docs.samples.ipc.util.IPCUtils;
```

```
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.UpdateThingShadowResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;
import software.amazon.awssdk.aws.greengrass.model.UpdateThingShadowRequest;
import software.amazon.awssdk.aws.greengrass.model.UpdateThingShadowResponse;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;

import java.nio.charset.StandardCharsets;
import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class UpdateThingShadow {

    public static final int TIMEOUT_SECONDS = 10;

    public static void main(String[] args) {
        // Use the current core device's name if thing name isn't set.
        String thingName = args[0].isEmpty() ? System.getenv("AWS_IOT_THING_NAME") :
 args[0];
        String shadowName = args[1];
        byte[] shadowPayload = args[2].getBytes(StandardCharsets.UTF_8);
        try (EventStreamRPCConnection eventStreamRPCConnection =
                     IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient =
                    new GreengrassCoreIPCClient(eventStreamRPCConnection);
            UpdateThingShadowResponseHandler responseHandler =
                    UpdateThingShadow.updateThingShadow(ipcClient, thingName,
 shadowName,
                            shadowPayload);
            CompletableFuture<UpdateThingShadowResponse> futureResponse =
                    responseHandler.getResponse();
            try {
                futureResponse.get(TIMEOUT_SECONDS, TimeUnit.SECONDS);
                System.out.printf("Successfully updated shadow: %s/%s%n", thingName,
 shadowName);
            } catch (TimeoutException e) {
                System.err.printf("Timeout occurred while updating shadow: %s/%s%n",
 thingName,
                        shadowName);
            } catch (ExecutionException e) {
                if (e.getCause() instanceof UnauthorizedError) {
```

```
                    System.err.printf("Unauthorized error while updating shadow: %s/
%s%n",
                            thingName, shadowName);
                } else {
                    throw e;
                }
            }
        } catch (InterruptedException e) {
            System.out.println("IPC interrupted.");
        } catch (ExecutionException e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static UpdateThingShadowResponseHandler
 updateThingShadow(GreengrassCoreIPCClient greengrassCoreIPCClient, String
 thingName, String shadowName, byte[] shadowPayload) {
        UpdateThingShadowRequest updateThingShadowRequest = new
 UpdateThingShadowRequest();
        updateThingShadowRequest.setThingName(thingName);
        updateThingShadowRequest.setShadowName(shadowName);
        updateThingShadowRequest.setPayload(shadowPayload);
        return greengrassCoreIPCClient.updateThingShadow(updateThingShadowRequest,
                Optional.empty());
    }
}
```

Python (IPC client V1)

Example Exemplo: atualizar a sombra de uma coisa

```
import awsiot.greengrasscoreipc
import awsiot.greengrasscoreipc.client as client
from awsiot.greengrasscoreipc.model import UpdateThingShadowRequest

TIMEOUT = 10

def sample_update_thing_shadow_request(thingName, shadowName, payload):
    try:
        # set up IPC client to connect to the IPC server
        ipc_client = awsiot.greengrasscoreipc.connect()
```

```
        # create the UpdateThingShadow request
        update_thing_shadow_request = UpdateThingShadowRequest()
        update_thing_shadow_request.thing_name = thingName
        update_thing_shadow_request.shadow_name = shadowName
        update_thing_shadow_request.payload = payload

        # retrieve the UpdateThingShadow response after sending the request to the
 IPC server
        op = ipc_client.new_update_thing_shadow()
        op.activate(update_thing_shadow_request)
        fut = op.get_response()

        result = fut.result(TIMEOUT)
        return result.payload

    except InvalidArgumentsError as e:
        # add error handling
    ...
    # except ConflictError | UnauthorizedError | ServiceError
```

JavaScript

Example Exemplo: atualizar a sombra de uma coisa

```
import {
    UpdateThingShadowRequest
} from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc/model';
import * as greengrasscoreipc from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc';

class UpdateThingShadow {
    private ipcClient: greengrasscoreipc.Client;
    private thingName: string;
    private shadowName: string;
    private shadowDocumentStr: string;

    constructor() {
        // Define args parameters here

        this.thingName = "<define_your_own_thingName>";
        this.shadowName = "<define_your_own_shadowName>";
        this.shadowDocumentStr = "<define_your_own_payload>";

        this.bootstrap();
    }
```

```
    async bootstrap() {
        try {
            this.ipcClient = await getIpcClient();
        } catch (err) {
            // parse the error depending on your use cases
            throw err
        }

        try {
            await this.handleUpdateThingShadowOperation(
                this.thingName,
                this.shadowName,
                this.shadowDocumentStr);
        } catch (err) {
            // parse the error depending on your use cases
            throw err
        }
    }

    async handleUpdateThingShadowOperation(
        thingName: string,
        shadowName: string,
        payloadStr: string
    ) {
        const request: UpdateThingShadowRequest = {
            thingName: thingName,
            shadowName: shadowName,
            payload: payloadStr
        }
        // make the UpdateThingShadow request
        const response = await this.ipcClient.updateThingShadow(request);
    }
}

export async function getIpcClient() {
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
```

```
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

const startScript = new UpdateThingShadow();
```

DeleteThingShadow

Apaga o shadow do objeto especificada.

A partir do shadow manager v2.0.4, a exclusão de uma sombra incrementa o número da versão. Por exemplo, quando você exclui a sombra `MyThingShadow` na versão 1, a versão da sombra excluída é 2. Se você recriar uma sombra com o nome`MyThingShadow`, a versão dessa sombra será 3.

Solicitação

A solicitação dessa operação tem os seguintes parâmetros:

`thingName`(Python:) `thing_name`

O nome da coisa.

Tipo: `string`

`shadowName`(Python:) `shadow_name`

O nome do shadow. Para especificar a sombra clássica da coisa, defina esse parâmetro como uma string vazia (`""`).

> **Warning**
>
> O AWS IoT Greengrass serviço usa a sombra `AWSManagedGreengrassV2Deployment` nomeada para gerenciar implantações direcionadas a dispositivos principais individuais. Essa sombra nomeada é reservada para uso pelo AWS IoT Greengrass serviço. Não atualize nem exclua essa sombra nomeada.

Tipo: `string`

Resposta

A resposta dessa operação tem as seguintes informações:

`payload`

Um documento de estado de resposta vazio.

Erros

Essa operação pode retornar os seguintes erros.

`InvalidArgumentsError`

O serviço paralelo local não consegue validar os parâmetros da solicitação. Isso pode ocorrer se a solicitação contiver caracteres JSON malformados ou sem suporte.

`ResourceNotFoundError`

O documento paralelo local solicitado não foi encontrado.

`ServiceError`

Ocorreu um erro de serviço interno ou o número de solicitações ao serviço IPC excedeu os limites especificados nos parâmetros de `maxTotalLocalRequestsRate` configuração `maxLocalRequestsPerSecondPerThing` e no componente do gerenciador de sombra.

`UnauthorizedError`

A política de autorização do componente não inclui as permissões necessárias para essa operação.

Exemplos

Os exemplos a seguir demonstram como chamar essa operação no código de componente personalizado.

Java (IPC client V1)

Example Exemplo: excluir uma sombra de coisa

> **Note**
>
> Este exemplo usa uma `IPCUtils` classe para criar uma conexão com o serviço AWS IoT Greengrass Core IPC. Para ter mais informações, consulte Conecte-se ao serviço AWS IoT Greengrass Core IPC.

```
package com.aws.greengrass.docs.samples.ipc;

import com.aws.greengrass.docs.samples.ipc.util.IPCUtils;
import software.amazon.awssdk.aws.greengrass.DeleteThingShadowResponseHandler;
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import software.amazon.awssdk.aws.greengrass.model.DeleteThingShadowRequest;
import software.amazon.awssdk.aws.greengrass.model.DeleteThingShadowResponse;
import software.amazon.awssdk.aws.greengrass.model.ResourceNotFoundError;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;

import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class DeleteThingShadow {

    public static final int TIMEOUT_SECONDS = 10;

    public static void main(String[] args) {
        // Use the current core device's name if thing name isn't set.
        String thingName = args[0].isEmpty() ? System.getenv("AWS_IOT_THING_NAME") :
 args[0];
        String shadowName = args[1];
        try (EventStreamRPCConnection eventStreamRPCConnection =
                     IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient =
                    new GreengrassCoreIPCClient(eventStreamRPCConnection);
            DeleteThingShadowResponseHandler responseHandler =
```

```
                    DeleteThingShadow.deleteThingShadow(ipcClient, thingName,
 shadowName);
            CompletableFuture<DeleteThingShadowResponse> futureResponse =
                    responseHandler.getResponse();
            try {
                futureResponse.get(TIMEOUT_SECONDS, TimeUnit.SECONDS);
                System.out.printf("Successfully deleted shadow: %s/%s%n", thingName,
 shadowName);
            } catch (TimeoutException e) {
                System.err.printf("Timeout occurred while deleting shadow: %s/%s%n",
 thingName,
                        shadowName);
            } catch (ExecutionException e) {
                if (e.getCause() instanceof UnauthorizedError) {
                    System.err.printf("Unauthorized error while deleting shadow: %s/
%s%n",
                            thingName, shadowName);
                } else if (e.getCause() instanceof ResourceNotFoundError) {
                    System.err.printf("Unable to find shadow to delete: %s/%s%n",
 thingName,
                            shadowName);
                } else {
                    throw e;
                }
            }
        } catch (InterruptedException e) {
            System.out.println("IPC interrupted.");
        } catch (ExecutionException e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static DeleteThingShadowResponseHandler
 deleteThingShadow(GreengrassCoreIPCClient greengrassCoreIPCClient, String
 thingName, String shadowName) {
        DeleteThingShadowRequest deleteThingShadowRequest = new
 DeleteThingShadowRequest();
        deleteThingShadowRequest.setThingName(thingName);
        deleteThingShadowRequest.setShadowName(shadowName);
        return greengrassCoreIPCClient.deleteThingShadow(deleteThingShadowRequest,
                Optional.empty());
    }
```

```
}
```

Python (IPC client V1)

Example Exemplo: excluir uma sombra de coisa

```
import awsiot.greengrasscoreipc
import awsiot.greengrasscoreipc.client as client
from awsiot.greengrasscoreipc.model import DeleteThingShadowRequest

TIMEOUT = 10

def sample_delete_thing_shadow_request(thingName, shadowName):
    try:
        # set up IPC client to connect to the IPC server
        ipc_client = awsiot.greengrasscoreipc.connect()

        # create the DeleteThingShadow request
        delete_thing_shadow_request = DeleteThingShadowRequest()
        delete_thing_shadow_request.thing_name = thingName
        delete_thing_shadow_request.shadow_name = shadowName

        # retrieve the DeleteThingShadow response after sending the request to the
 IPC server
        op = ipc_client.new_delete_thing_shadow()
        op.activate(delete_thing_shadow_request)
        fut = op.get_response()

        result = fut.result(TIMEOUT)
        return result.payload

    except InvalidArgumentsError as e:
        # add error handling
    ...
    # except ResourceNotFoundError | UnauthorizedError | ServiceError
```

JavaScript

Example Exemplo: excluir uma sombra de coisa

```
import {
    DeleteThingShadowRequest
} from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc/model';
import * as greengrasscoreipc from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc';
```

```
class DeleteThingShadow {
    private ipcClient: greengrasscoreipc.Client;
    private thingName: string;
    private shadowName: string;

    constructor() {
        // Define args parameters here
        this.thingName = "<define_your_own_thingName>";
        this.shadowName = "<define_your_own_shadowName>";
        this.bootstrap();
    }

    async bootstrap() {
        try {
            this.ipcClient = await getIpcClient();
        } catch (err) {
            // parse the error depending on your use cases
            throw err
        }

        try {
            await this.handleDeleteThingShadowOperation(this.thingName,
 this.shadowName)
        } catch (err) {
            // parse the error depending on your use cases
            throw err
        }
    }

    async handleDeleteThingShadowOperation(thingName: string, shadowName: string) {
        const request: DeleteThingShadowRequest = {
            thingName: thingName,
            shadowName: shadowName
        }
        // make the DeleteThingShadow request
        const response = await this.ipcClient.deleteThingShadow(request);
    }
}

export async function getIpcClient() {
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
```

```
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

const startScript = new DeleteThingShadow();
```

ListNamedShadowsForThing

Liste as sombras nomeadas para a coisa especificada.

Solicitação

A solicitação dessa operação tem os seguintes parâmetros:

`thingName`(Python:) `thing_name`

O nome da coisa.

Tipo: `string`

`pageSize`(Python:) `page_size`

(Opcional) O número de nomes de sombra a serem retornados em cada chamada.

Tipo: `integer`

Padrão: 25

Maximum (Máximo): 100

`nextToken`(Python:) `next_token`

(Opcional) O token para recuperar o próximo conjunto de resultados. Esse valor é retornado nos resultados paginados e é usado na chamada que retorna a próxima página.

Tipo: `string`

Resposta

A resposta dessa operação tem as seguintes informações:

`results`

A lista de nomes de sombras.

Tipo: `array`

`timestamp`

(Opcional) A data e a hora em que a resposta foi gerada.

Tipo: `integer`

`nextToken`(Python:) `next_token`

(Opcional) O valor do token a ser usado em solicitações paginadas para recuperar a próxima página na sequência. Esse token não está presente quando não há mais nomes de sombra a serem retornados.

Tipo: `string`

> **Note**
>
> Se o tamanho da página solicitada corresponder exatamente ao número de nomes de sombra na resposta, esse token estará presente; no entanto, quando usado, ele retornará uma lista vazia.

Erros

Essa operação pode retornar os seguintes erros.

`InvalidArgumentsError`

O serviço paralelo local não consegue validar os parâmetros da solicitação. Isso pode ocorrer se a solicitação contiver caracteres JSON malformados ou sem suporte.

`ResourceNotFoundError`

O documento paralelo local solicitado não foi encontrado.

`ServiceError`

Ocorreu um erro de serviço interno ou o número de solicitações ao serviço IPC excedeu os limites especificados nos parâmetros de `maxTotalLocalRequestsRate` configuração `maxLocalRequestsPerSecondPerThing` e no componente do gerenciador de sombra.

`UnauthorizedError`

A política de autorização do componente não inclui as permissões necessárias para essa operação.

Exemplos

Os exemplos a seguir demonstram como chamar essa operação no código de componente personalizado.

Java (IPC client V1)

Example Exemplo: Listar sombras de uma coisa chamada

> **Note**
>
> Este exemplo usa uma `IPCUtils` classe para criar uma conexão com o serviço AWS IoT Greengrass Core IPC. Para ter mais informações, consulte Conecte-se ao serviço AWS IoT Greengrass Core IPC.

```
package com.aws.greengrass.docs.samples.ipc;

import com.aws.greengrass.docs.samples.ipc.util.IPCUtils;
import software.amazon.awssdk.aws.greengrass.GreengrassCoreIPCClient;
import
 software.amazon.awssdk.aws.greengrass.ListNamedShadowsForThingResponseHandler;
import software.amazon.awssdk.aws.greengrass.model.ListNamedShadowsForThingRequest;
import
 software.amazon.awssdk.aws.greengrass.model.ListNamedShadowsForThingResponse;
import software.amazon.awssdk.aws.greengrass.model.ResourceNotFoundError;
import software.amazon.awssdk.aws.greengrass.model.UnauthorizedError;
import software.amazon.awssdk.eventstreamrpc.EventStreamRPCConnection;

import java.util.ArrayList;
import java.util.List;
```

```
import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class ListNamedShadowsForThing {

    public static final int TIMEOUT_SECONDS = 10;

    public static void main(String[] args) {
        // Use the current core device's name if thing name isn't set.
        String thingName = args[0].isEmpty() ? System.getenv("AWS_IOT_THING_NAME") :
 args[0];
        try (EventStreamRPCConnection eventStreamRPCConnection =
                     IPCUtils.getEventStreamRpcConnection()) {
            GreengrassCoreIPCClient ipcClient =
                    new GreengrassCoreIPCClient(eventStreamRPCConnection);
            List<String> namedShadows = new ArrayList<>();
            String nextToken = null;
            try {
                // Send additional requests until there's no pagination token in the
 response.
                do {
                    ListNamedShadowsForThingResponseHandler responseHandler =

 ListNamedShadowsForThing.listNamedShadowsForThing(ipcClient, thingName,
                                    nextToken, 25);
                    CompletableFuture<ListNamedShadowsForThingResponse>
 futureResponse =
                            responseHandler.getResponse();
                    ListNamedShadowsForThingResponse response =
                            futureResponse.get(TIMEOUT_SECONDS, TimeUnit.SECONDS);
                    List<String> responseNamedShadows = response.getResults();
                    namedShadows.addAll(responseNamedShadows);
                    nextToken = response.getNextToken();
                } while (nextToken != null);
                System.out.printf("Successfully got named shadows for thing %s: %s
%n", thingName,
                        String.join(",", namedShadows));
            } catch (TimeoutException e) {
                System.err.println("Timeout occurred while listing named shadows for
 thing: " + thingName);
            } catch (ExecutionException e) {
```

```
                if (e.getCause() instanceof UnauthorizedError) {
                    System.err.println("Unauthorized error while listing named
 shadows for " +
                            "thing: " + thingName);
                } else if (e.getCause() instanceof ResourceNotFoundError) {
                    System.err.println("Unable to find thing to list named shadows:
 " + thingName);
                } else {
                    throw e;
                }
            }
        } catch (InterruptedException e) {
            System.out.println("IPC interrupted.");
        } catch (ExecutionException e) {
            System.err.println("Exception occurred when using IPC.");
            e.printStackTrace();
            System.exit(1);
        }
    }

    public static ListNamedShadowsForThingResponseHandler
 listNamedShadowsForThing(GreengrassCoreIPCClient greengrassCoreIPCClient, String
 thingName, String nextToken, int pageSize) {
        ListNamedShadowsForThingRequest listNamedShadowsForThingRequest =
                new ListNamedShadowsForThingRequest();
        listNamedShadowsForThingRequest.setThingName(thingName);
        listNamedShadowsForThingRequest.setNextToken(nextToken);
        listNamedShadowsForThingRequest.setPageSize(pageSize);
        return
 greengrassCoreIPCClient.listNamedShadowsForThing(listNamedShadowsForThingRequest,
                Optional.empty());
    }
}
```

Python (IPC client V1)

Example Exemplo: Listar sombras de uma coisa chamada

```
import awsiot.greengrasscoreipc
import awsiot.greengrasscoreipc.client as client
from awsiot.greengrasscoreipc.model import ListNamedShadowsForThingRequest

TIMEOUT = 10
```

```
def sample_list_named_shadows_for_thing_request(thingName, nextToken, pageSize):
    try:
        # set up IPC client to connect to the IPC server
        ipc_client = awsiot.greengrasscoreipc.connect()

        # create the ListNamedShadowsForThingRequest request
        list_named_shadows_for_thing_request = ListNamedShadowsForThingRequest()
        list_named_shadows_for_thing_request.thing_name = thingName
        list_named_shadows_for_thing_request.next_token = nextToken
        list_named_shadows_for_thing_request.page_size = pageSize

        # retrieve the ListNamedShadowsForThingRequest response after sending the
 request to the IPC server
        op = ipc_client.new_list_named_shadows_for_thing()
        op.activate(list_named_shadows_for_thing_request)
        fut = op.get_response()

        list_result = fut.result(TIMEOUT)

        # additional returned fields
        timestamp = list_result.timestamp
        next_token = result.next_token
        named_shadow_list = list_result.results

        return named_shadow_list, next_token, timestamp

    except InvalidArgumentsError as e:
        # add error handling
    ...
    # except ResourceNotFoundError | UnauthorizedError | ServiceError
```

JavaScript

Example Exemplo: Listar sombras de uma coisa chamada

```
import {
    ListNamedShadowsForThingRequest
} from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc/model';
import * as greengrasscoreipc from 'aws-iot-device-sdk-v2/dist/greengrasscoreipc';

class listNamedShadowsForThing {
    private ipcClient: greengrasscoreipc.Client;
    private thingName: string;
    private pageSizeStr: string;
```

```
    private nextToken: string;

    constructor() {
        // Define args parameters here
        this.thingName = "<define_your_own_thingName>";
        this.pageSizeStr = "<define_your_own_pageSize>";
        this.nextToken = "<define_your_own_token>";
        this.bootstrap();
    }

    async bootstrap() {
        try {
            this.ipcClient = await getIpcClient();
        } catch (err) {
            // parse the error depending on your use cases
            throw err
        }

        try {
            await this.handleListNamedShadowsForThingOperation(this.thingName,
                this.nextToken, this.pageSizeStr);
        } catch (err) {
            // parse the error depending on your use cases
            throw err
        }
    }

    async handleListNamedShadowsForThingOperation(
        thingName: string,
        nextToken: string,
        pageSizeStr: string
    ) {
        let request: ListNamedShadowsForThingRequest = {
            thingName: thingName,
            nextToken: nextToken,
        };
        if (pageSizeStr) {
            request.pageSize = parseInt(pageSizeStr);
        }
        // make the ListNamedShadowsForThing request
        const response = await this.ipcClient.listNamedShadowsForThing(request);
        const shadowNames = response.results;
    }
}
```

```
export async function getIpcClient(){
    try {
        const ipcClient = greengrasscoreipc.createClient();
        await ipcClient.connect()
            .catch(error => {
                // parse the error depending on your use cases
                throw error;
            });
        return ipcClient
    } catch (err) {
        // parse the error depending on your use cases
        throw err
    }
}

const startScript = new listNamedShadowsForThing();
```

Gerencie implantações e componentes locais

> **Note**
>
> Esse recurso está disponível para a versão 2.6.0 e posterior do componente núcleo do Greengrass.

Use o serviço Greengrass CLI IPC para gerenciar implantações locais e componentes do Greengrass no dispositivo principal.

Para usar essas operações de IPC, inclua a versão 2.6.0 ou posterior do componente Greengrass CLI como uma dependência em seu componente personalizado. Em seguida, você pode usar as operações de IPC em seus componentes personalizados para fazer o seguinte:

- Crie implantações locais para modificar e configurar os componentes do Greengrass no dispositivo principal.
- Reinicie e interrompa os componentes do Greengrass no dispositivo principal.
- Gere uma senha que você possa usar para entrar no console de depuração local.

Tópicos

Versões mínimas do SDK

A tabela a seguir lista as versões mínimas do AWS IoT Device SDK que você deve usar para interagir com o serviço Greengrass CLI IPC.

SDK	Versão mínima
AWS IoT Device SDKpara Java v2	v1.2.10
AWS IoT Device SDKpara Python v2	v1.5.3
AWS IoT Device SDKpara C++ v2	v1.17.0
AWS IoT Device SDKpara JavaScript v2	v1.12.0

Autorização

Para usar o serviço Greengrass CLI IPC em um componente personalizado, você deve definir políticas de autorização que permitam que seu componente gerencie implantações e componentes

locais. Para obter informações sobre a definição de políticas de autorização, consulteAutorize componentes a realizar operações de IPC.

As políticas de autorização para a CLI do Greengrass têm as seguintes propriedades.

Identificador de serviço IPC: `aws.greengrass.Cli`

Operation	Descrição	Recursos
`aws.greengrass#CreateLocalDeployment`	Permite que um componente crie uma implantação local no dispositivo principal.	*
`aws.greengrass#ListLocalDeployments`	Permite que um componente liste implantações locais no dispositivo principal.	*
`aws.greengrass#GetLocalDeploymentStatus`	Permite que um componente obtenha o status de uma implantação local no dispositivo principal.	Um ID de implantação local ou * para permitir o acesso a todas as implantações locais.
`aws.greengrass#ListComponents`	Permite que um componente liste componentes no dispositivo principal.	*
`aws.greengrass#GetComponentDetails`	Permite que um componente obtenha detalhes sobre um componente no dispositivo principal.	Um nome de componente, como `com.example.HelloWorld`, ou * para permitir o acesso a todos os componentes.
`aws.greengrass#RestartComponent`	Permite que um componente reinicie um componente no dispositivo principal.	Um nome de componente, como `com.example.HelloWorld`, ou * para permitir o acesso a todos os componentes.

Operation	Descrição	Recursos
`aws.greengrass#StopComponent`	Permite que um componente interrompa um componente no dispositivo principal.	Um nome de componente, como `com.example.HelloWorld`, ou * para permitir o acesso a todos os componentes.
`aws.greengrass#CreateDebugPassword`	Permite que um componente gere uma senha para usar para entrar no componente do console de depuração local.	*

Example Exemplo de política de autorização

O exemplo de políticas de autorização a seguir permite que um componente crie implantações locais, visualize todas as implantações e componentes locais e reinicie e interrompa um componente chamado. `com.example.HelloWorld`

```
{
  "accessControl": {
    "aws.greengrass.Cli": {
      "com.example.MyLocalManagerComponent:cli:1": {
        "policyDescription": "Allows access to create local deployments and view
 deployments and components.",
        "operations": [
          "aws.greengrass#CreateLocalDeployment",
          "aws.greengrass#ListLocalDeployments",
          "aws.greengrass#GetLocalDeploymentStatus",
          "aws.greengrass#ListComponents",
          "aws.greengrass#GetComponentDetails"
        ],
        "resources": [
          "*"
        ]
      }
    },
    "aws.greengrass.Cli": {
      "com.example.MyLocalManagerComponent:cli:2": {
```

```
        "policyDescription": "Allows access to restart and stop the Hello World
 component.",
        "operations": [
          "aws.greengrass#RestartComponent",
          "aws.greengrass#StopComponent"
        ],
        "resources": [
          "com.example.HelloWorld"
        ]
      }
    }
  }
}
```

CreateLocalDeployment

Crie ou atualize uma implantação local usando receitas de componentes, artefatos e argumentos de tempo de execução especificados.

Essa operação fornece a mesma funcionalidade do comando deployment create na CLI do Greengrass.

Solicitação

A solicitação dessa operação tem os seguintes parâmetros:

`recipeDirectoryPath`(Python:) `recipe_directory_path`

(Opcional) O caminho absoluto para a pasta que contém os arquivos de receitas dos componentes.

`artifactDirectoryPath`(Python:) `artifact_directory_path`

(Opcional) O caminho absoluto para a pasta que contém os arquivos de artefatos a serem incluídos na implantação. A pasta de artefatos deve conter a seguinte estrutura de pastas:

```
/path/to/artifact/folder/component-name/component-version/artifacts
```

`rootComponentVersionsToAdd`(Python:) `root_component_versions_to_add`

(Opcional) As versões dos componentes a serem instaladas no dispositivo principal. Esse objeto,`ComponentToVersionMap`, é um mapa que contém os seguintes pares de valores-chave:

`key`

O nome do componente.

`value`

A versão do componente.

`rootComponentsToRemove`(Python:) `root_components_to_remove`

(Opcional) Os componentes a serem desinstalados do dispositivo principal. Especifique uma lista em que cada entrada seja o nome de um componente.

`componentToConfiguration`(Python:) `component_to_configuration`

(Opcional) As atualizações de configuração para cada componente na implantação. Esse objeto,`ComponentToConfiguration`, é um mapa que contém os seguintes pares de valores-chave:

`key`

O nome do componente.

`value`

O objeto JSON de atualização de configuração do componente. O objeto JSON deve ter o seguinte formato.

```
{
  "MERGE": {
    "config-key": "config-value"
  },
  "RESET": [
    "path/to/reset/"
  ]
}
```

Para obter mais informações sobre atualizações de configuração, consulteAtualizar configurações de componentes.

`componentToRunWithInfo`(Python:) `component_to_run_with_info`

(Opcional) A configuração de tempo de execução para cada componente na implantação. Essa configuração inclui o usuário do sistema que possui os processos de cada

componente e os limites do sistema a serem aplicados a cada componente. Esse objeto,`ComponentToRunWithInfo`, é um mapa que contém os seguintes pares de valores-chave:

`key`

O nome do componente.

`value`

A configuração de tempo de execução do componente. Se você omitir um parâmetro de configuração de tempo de execução, o software AWS IoT Greengrass Core usará os valores padrão que você configura no núcleo do Greengrass. Esse objeto,`RunWithInfo`, contém as seguintes informações:

`posixUser`(Python:) `posix_user`

(Opcional) O usuário do sistema POSIX e, opcionalmente, o grupo a ser usado para executar esse componente nos dispositivos principais do Linux. O usuário e o grupo, se especificados, devem existir em cada dispositivo principal do Linux. Especifique o usuário e o grupo separando-os por dois pontos (`:`), no seguinte formato: `user:group`. O grupo é opcional. Se você não especificar um grupo, o software AWS IoT Greengrass Core usará o grupo primário para o usuário. Para obter mais informações, consulte Configurar o usuário que executa os componentes.

`windowsUser`(Python:) `windows_user`

(Opcional) O usuário do Windows a ser usado para executar esse componente nos dispositivos principais do Windows. O usuário deve existir em cada dispositivo principal do Windows e seu nome e senha devem ser armazenados na instância do Gerenciador de Credenciais da LocalSystem conta. Para obter mais informações, consulte Configurar o usuário que executa os componentes.

`systemResourceLimits`(Python:) `system_resource_limits`

(Opcional) Os limites de recursos do sistema a serem aplicados aos processos desse componente. Você pode aplicar limites de recursos do sistema a componentes Lambda genéricos e não conteinerizados. Para obter mais informações, consulte Configurar limites de recursos do sistema para componentes.

AWS IoT Greengrassatualmente não oferece suporte a esse recurso nos dispositivos principais do Windows.

Esse objeto,`SystemResourceLimits`, contém as seguintes informações:

`cpus`

(Opcional) A quantidade máxima de tempo de CPU que os processos desse componente podem usar no dispositivo principal. O tempo total da CPU de um dispositivo essencial é equivalente ao número de núcleos da CPU do dispositivo. Por exemplo, em um dispositivo principal com 4 núcleos de CPU, você pode definir esse valor 2 para limitar os processos desse componente a 50% de uso de cada núcleo da CPU. Em um dispositivo com 1 núcleo de CPU, você pode definir esse valor `0.25` para limitar os processos desse componente a 25% de uso da CPU. Se você definir esse valor como um número maior que o número de núcleos de CPU, o software AWS IoT Greengrass Core não limitará o uso da CPU do componente.

`memory`

(Opcional) A quantidade máxima de RAM (em kilobytes) que os processos desse componente podem usar no dispositivo principal.

`groupName`(Python:) `group_name`

(Opcional) O nome do grupo de coisas a ser direcionado com essa implantação.

Resposta

A resposta dessa operação tem as seguintes informações:

`deploymentId`(Python:) `deployment_id`

O ID da implantação local que a solicitação criou.

ListLocalDeployments

Obtém o status das últimas 10 implantações locais.

Essa operação fornece a mesma funcionalidade do comando de lista de implantação na CLI do Greengrass.

Solicitação

A solicitação dessa operação não tem nenhum parâmetro.

Resposta

A resposta dessa operação tem as seguintes informações:

`localDeployments`(Python:) `local_deployments`

A lista de implantações locais. Cada objeto nessa lista é um `LocalDeployment` objeto, que contém as seguintes informações:

`deploymentId`(Python:) `deployment_id`

O ID da implantação local.

`status`

O status da implantação local. Essa enumeração,`DeploymentStatus`, tem os seguintes valores:

- `QUEUED`
- `IN_PROGRESS`
- `SUCCEEDED`
- `FAILED`

GetLocalDeploymentStatus

Obtém o status de uma implantação local.

Essa operação fornece a mesma funcionalidade do comando de status de implantação na CLI do Greengrass.

Solicitação

A solicitação dessa operação tem os seguintes parâmetros:

`deploymentId`(Python:) `deployment_id`

O ID da implantação local a ser obtida.

Resposta

A resposta dessa operação tem as seguintes informações:

`deployment`

A implantação local. Esse objeto,`LocalDeployment`, contém as seguintes informações:

`deploymentId`(Python:) `deployment_id`

O ID da implantação local.

`status`

O status da implantação local. Essa enumeração,`DeploymentStatus`, tem os seguintes valores:

- `QUEUED`
- `IN_PROGRESS`
- `SUCCEEDED`
- `FAILED`

ListComponents

Obtém o nome, a versão, o status e a configuração de cada componente raiz no dispositivo principal. Um componente raiz é um componente que você especifica em uma implantação. Essa resposta não inclui componentes que são instalados como dependências de outros componentes.

Essa operação fornece a mesma funcionalidade do comando de lista de componentes na CLI do Greengrass.

Solicitação

A solicitação dessa operação não tem nenhum parâmetro.

Resposta

A resposta dessa operação tem as seguintes informações:

`components`

A lista de componentes raiz no dispositivo principal. Cada objeto nessa lista é um `ComponentDetails` objeto, que contém as seguintes informações:

`componentName`(Python:) `component_name`

O nome do componente.

`version`

A versão do componente.

`state`

O estado do componente. Esse estado pode ser um dos seguintes:

- `BROKEN`
- `ERRORED`
- `FINISHED`
- `INSTALLED`
- `NEW`
- `RUNNING`
- `STARTING`
- `STOPPING`

`configuration`

A configuração do componente como um objeto JSON.

GetComponentDetails

Obtém a versão, o status e a configuração de um componente no dispositivo principal.

Essa operação fornece a mesma funcionalidade do comando de detalhes do componente na CLI do Greengrass.

Solicitação

A solicitação dessa operação tem os seguintes parâmetros:

`componentName`(Python:) `component_name`

O nome do componente a ser obtido.

Resposta

A resposta dessa operação tem as seguintes informações:

`componentDetails`(Python:) `component_details`

Os detalhes do componente. Esse objeto,`ComponentDetails`, contém as seguintes informações:

`componentName`(Python:) `component_name`

O nome do componente.

`version`

A versão do componente.

`state`

O estado do componente. Esse estado pode ser um dos seguintes:

- `BROKEN`
- `ERRORED`
- `FINISHED`
- `INSTALLED`
- `NEW`
- `RUNNING`
- `STARTING`
- `STOPPING`

`configuration`

A configuração do componente como um objeto JSON.

RestartComponent

Reinicia um componente no dispositivo principal.

> **Note**
>
> Embora você possa reiniciar qualquer componente, recomendamos que você reinicie somente componentes genéricos.

Essa operação fornece a mesma funcionalidade do comando de reinicialização do componente na CLI do Greengrass.

Solicitação

A solicitação dessa operação tem os seguintes parâmetros:

`componentName`(Python:) `component_name`

O nome do componente.

Resposta

A resposta dessa operação tem as seguintes informações:

`restartStatus`(Python:) `restart_status`

O status da solicitação de reinicialização. O status da solicitação pode ser um dos seguintes:

- `SUCCEEDED`
- `FAILED`

`message`

Uma mensagem sobre por que o componente falhou na reinicialização, se a solicitação falhou.

StopComponent

Interrompe os processos de um componente no dispositivo principal.

> **Note**
>
> Embora você possa interromper qualquer componente, recomendamos que você interrompa somente os componentes genéricos.

Essa operação fornece a mesma funcionalidade do comando de parada do componente na CLI do Greengrass.

Solicitação

A solicitação dessa operação tem os seguintes parâmetros:

`componentName`(Python:) `component_name`

O nome do componente.

Resposta

A resposta dessa operação tem as seguintes informações:

`stopStatus`(Python:) `stop_status`

O status da solicitação de parada. O status da solicitação pode ser um dos seguintes:

- `SUCCEEDED`
- `FAILED`

`message`

Uma mensagem sobre o motivo pelo qual o componente não foi interrompido, se a solicitação falhou.

CreateDebugPassword

Gera uma senha aleatória que você pode usar para entrar no componente local do console de depuração. A senha expira 8 horas depois de ser gerada.

Essa operação fornece a mesma funcionalidade do get-debug-password comando na CLI do Greengrass.

Solicitação

A solicitação dessa operação não tem nenhum parâmetro.

Resposta

A resposta dessa operação tem as seguintes informações:

`username`

O nome de usuário a ser usado para fazer login.

`password`

A senha a ser usada para fazer login.

`passwordExpiration`(Python:) `password_expiration`

A hora em que a senha expira.

`certificateSHA256Hash`(Python:) `certificate_sha256_hash`

A impressão digital SHA-256 do certificado autoassinado que o console de depuração local usa quando o HTTPS está ativado. Ao abrir o console de depuração local, use essa impressão digital para verificar se o certificado é legítimo e se a conexão é segura.

`certificateSHA1Hash`(Python:) `certificate_sha1_hash`

A impressão digital SHA-1 do certificado autoassinado que o console de depuração local usa quando o HTTPS está ativado. Ao abrir o console de depuração local, use essa impressão digital para verificar se o certificado é legítimo e se a conexão é segura.

Autenticar e autorizar dispositivos clientes

> **Note**
>
> Esse recurso está disponível para a versão 2.6.0 e posterior do componente núcleo do Greengrass.

Use o serviço IPC de autenticação de dispositivo cliente para desenvolver um componente de intermediário local personalizado no qual dispositivos IoT locais, como dispositivos cliente, possam se conectar.

Para usar essas operações de IPC, inclua a versão 2.2.0 ou posterior do componente de autenticação do dispositivo cliente como uma dependência em seu componente personalizado. Em seguida, você pode usar as operações de IPC em seus componentes personalizados para fazer o seguinte:

- Verifique a identidade dos dispositivos clientes que se conectam ao dispositivo principal.
- Crie uma sessão para que um dispositivo cliente se conecte ao dispositivo principal.
- Verifique se um dispositivo cliente tem permissão para realizar uma ação.
- Receba uma notificação quando o certificado do servidor do dispositivo principal for alterado.

Tópicos

- Versões mínimas do SDK
- Autorização
- VerifyClientDeviceIdentity
- GetClientDeviceAuthToken
- AuthorizeClientDeviceAction
- SubscribeToCertificateUpdates

Versões mínimas do SDK

A tabela a seguir lista as versões mínimas do AWS IoT Device SDK que você deve usar para interagir com o serviço IPC de autenticação do dispositivo cliente.

SDK	Versão mínima
AWS IoT Device SDKpara Java v2	v1.9.3
AWS IoT Device SDKpara Python v2	v1.11.3
AWS IoT Device SDKpara C++ v2	v1.18.3
AWS IoT Device SDKpara JavaScript v2	v1.12.0

Autorização

Para usar o serviço IPC de autenticação do dispositivo cliente em um componente personalizado, você deve definir políticas de autorização que permitam que seu componente execute essas operações. Para obter informações sobre a definição de políticas de autorização, consulteAutorize componentes a realizar operações de IPC.

As políticas de autorização para autenticação e autorização do dispositivo cliente têm as seguintes propriedades.

Identificador de serviço IPC: `aws.greengrass.clientdevices.Auth`

Operation	Descrição	Recursos
`aws.greengrass#VerifyClientDeviceIdentity`	Permite que um componente verifique a identidade de um dispositivo cliente.	*
`aws.greengrass#GetClientDeviceAuthToken`	Permite que um componente valide as credenciais de um dispositivo cliente e crie uma sessão para esse dispositivo cliente.	*
`aws.greengrass#AuthorizeClientDeviceAction`	Permite que um componente verifique se um dispositivo cliente tem permissão para realizar uma ação.	*
`aws.greengrass#SubscribeToCertificateUpdates`	Permite que um componente receba notificações quando o certificado do servidor do dispositivo principal é alterado.	*
*	Permite que um componente execute todas as operações do serviço IPC de autenticação do dispositivo cliente.	*

Exemplos de políticas de autorização

Você pode consultar o exemplo de política de autorização a seguir para ajudá-lo a configurar políticas de autorização para seus componentes.

Example Exemplo de política de autorização

O exemplo de política de autorização a seguir permite que um componente execute todas as operações IPC de autenticação do dispositivo cliente.

```
{
  "accessControl": {
    "aws.greengrass.clientdevices.Auth": {
      "com.example.MyLocalBrokerComponent:clientdevices:1": {
        "policyDescription": "Allows access to authenticate and authorize client
 devices.",
        "operations": [
          "aws.greengrass#VerifyClientDeviceIdentity",
          "aws.greengrass#GetClientDeviceAuthToken",
          "aws.greengrass#AuthorizeClientDeviceAction",
          "aws.greengrass#SubscribeToCertificateUpdates"
        ],
        "resources": [
          "*"
        ]
      }
    }
  }
}
```

VerifyClientDeviceIdentity

Verifique a identidade de um dispositivo cliente. Essa operação verifica se o dispositivo cliente AWS IoT é válido.

Solicitação

A solicitação dessa operação tem os seguintes parâmetros:

`credential`

As credenciais do dispositivo cliente. Esse objeto,`ClientDeviceCredential`, contém as seguintes informações:

`clientDeviceCertificate`(Python:) `client_device_certificate`

O certificado do dispositivo X.509 do dispositivo cliente.

Resposta

A resposta dessa operação tem as seguintes informações:

`isValidClientDevice`(Python:) `is_valid_client_device`

Se a identidade do dispositivo cliente é válida.

GetClientDeviceAuthToken

Valida as credenciais de um dispositivo cliente e cria uma sessão para o dispositivo cliente. Essa operação retorna um token de sessão que você pode usar em solicitações subsequentes para autorizar ações do dispositivo cliente.

Para conectar com êxito um dispositivo cliente, o componente de autenticação do dispositivo cliente deve conceder a `mqtt:connect` permissão para o ID do cliente que o dispositivo cliente usa.

Solicitação

A solicitação dessa operação tem os seguintes parâmetros:

`credential`

As credenciais do dispositivo cliente. Esse objeto,`CredentialDocument`, contém as seguintes informações:

`mqttCredential`(Python:) `mqtt_credential`

As credenciais MQTT do dispositivo cliente. Especifique o ID do cliente e o certificado que o dispositivo cliente usa para se conectar. Esse objeto,`MQTTCredential`, contém as seguintes informações:

`clientId`(Python:) `client_id`

O ID do cliente a ser usado para se conectar.

`certificatePem`(Python:) `certificate_pem`

O certificado do dispositivo X.509 a ser usado para se conectar.

`username`

> **Note**
>
> Essa propriedade não é usada atualmente.

`password`

> **Note**
>
> Essa propriedade não é usada atualmente.

Resposta

A resposta dessa operação tem as seguintes informações:

`clientDeviceAuthToken`(Python:) `client_device_auth_token`

O token da sessão para o dispositivo cliente. Você pode usar esse token de sessão em solicitações subsequentes para autorizar as ações desse dispositivo cliente.

AuthorizeClientDeviceAction

Verifique se um dispositivo cliente tem permissão para realizar uma ação em um recurso. As políticas de autorização do dispositivo cliente especificam as permissões que os dispositivos cliente podem executar enquanto conectados a um dispositivo principal. Você define as políticas de autorização do dispositivo cliente ao configurar o componente de autenticação do dispositivo cliente.

Solicitação

A solicitação dessa operação tem os seguintes parâmetros:

`clientDeviceAuthToken`(Python:) `client_device_auth_token`

O token da sessão para o dispositivo cliente.

`operation`

A operação a ser autorizada.

`resource`

O recurso em que o dispositivo cliente executa a operação.

Resposta

A resposta dessa operação tem as seguintes informações:

`isAuthorized`(Python:) `is_authorized`

Se o dispositivo cliente está autorizado a realizar a operação no recurso.

SubscribeToCertificateUpdates

Inscreva-se para receber o novo certificado de servidor do dispositivo principal sempre que ele for rotacionado. Quando o certificado do servidor muda, os corretores devem recarregar usando o novo certificado do servidor.

Por padrão, o componente de autenticação do dispositivo cliente alterna os certificados do servidor a cada 7 dias. Você pode configurar o intervalo de rotação entre 2 e 10 dias.

Essa operação é uma operação de assinatura em que você assina um fluxo de mensagens de eventos. Para usar essa operação, defina um manipulador de resposta de fluxo com funções que manipulam mensagens de eventos, erros e encerramento de fluxo. Para ter mais informações, consulte Inscreva-se nos streams de eventos do IPC.

Tipo de mensagem do evento: `CertificateUpdateEvent`

Solicitação

A solicitação dessa operação tem os seguintes parâmetros:

`certificateOptions`(Python:) `certificate_options`

Os tipos de atualizações de certificado a serem assinadas. Esse objeto,`CertificateOptions`, contém as seguintes informações:

`certificateType`(Python:) `certificate_type`

O tipo de atualizações de certificado a serem assinadas. Escolha a seguinte opção:

- `SERVER`

Resposta

A resposta dessa operação tem as seguintes informações:

`messages`

O fluxo de mensagens. Esse objeto,`CertificateUpdateEvent`, contém as seguintes informações:

`certificateUpdate`(Python:) `certificate_update`

As informações sobre o novo certificado. Esse objeto,`CertificateUpdate`, contém as seguintes informações:

`certificate`

O certificado.

`privateKey`(Python:) `private_key`

A chave privada do certificado.

`publicKey`(Python:) `public_key`

A chave pública do certificado.

`caCertificates`(Python:) `ca_certificates`

A lista de certificados de autoridade de certificação (CA) na cadeia de certificados CA do certificado.

Interaja com dispositivos IoT locais

Os dispositivos cliente são dispositivos IoT locais que se conectam e se comunicam com um dispositivo principal do Greengrass por meio do MQTT. Você pode conectar dispositivos cliente aos dispositivos principais para fazer o seguinte:

- Interaja com mensagens do MQTT nos componentes do Greengrass.
- Retransmita mensagens e dados entre dispositivos clientes e. AWS IoT Core
- Interaja com as sombras do dispositivo cliente nos componentes do Greengrass.
- Sincronize as sombras dos dispositivos clientes com o. AWS IoT Core

Para se conectar a um dispositivo principal, os dispositivos cliente podem usar a descoberta na nuvem. Os dispositivos cliente se conectam ao serviço de AWS IoT Greengrass nuvem para recuperar informações sobre os dispositivos principais aos quais eles podem se conectar. Em seguida, eles podem se conectar a um dispositivo principal para processar suas mensagens e sincronizar seus dados com o serviço de AWS IoT Core nuvem.

Você pode seguir um tutorial que explica como configurar um dispositivo principal para se conectar e se comunicar com AWS IoT algo. Este tutorial também explora como desenvolver um componente personalizado do Greengrass que interage com dispositivos clientes. Para ter mais informações, consulte Tutorial: Interaja com dispositivos locais de IoT por meio do MQTT.

Tópicos

AWS- componentes do dispositivo cliente fornecidos

AWS IoT Greengrassfornece os seguintes componentes públicos que você pode implantar nos dispositivos principais. Esses componentes permitem que dispositivos cliente se conectem e se comuniquem com um dispositivo principal.

> **Note**
>
> Vários componentes AWS fornecidos dependem de versões secundárias específicas do núcleo do Greengrass. Por causa dessa dependência, você precisa atualizar esses componentes ao atualizar o núcleo do Greengrass para uma nova versão secundária. Para obter informações sobre as versões específicas do núcleo das quais cada componente depende, consulte o tópico do componente correspondente. Para obter mais informações sobre a atualização do núcleo, consulteAtualize o software AWS IoT Greengrass principal (OTA).

Quando um componente tem um tipo de componente genérico e Lambda, a versão atual do componente é do tipo genérico e uma versão anterior do componente é do tipo Lambda.

Componente	Descrição	Tipo de componente	SO com suporte	Código aberto
Autenticação do dispositivo cliente	Permite que dispositivos IoT locais, chamados de dispositivos cliente, se conectem ao dispositivo principal.	Plugin	Linux, Windows	Sim
Detector IP	Relata as informações de conectivi	Plugin	Linux, Windows	Sim

Componente	Descrição	Tipo de componente	SO com suporte	Código aberto
	dade do agente MQTT para AWS IoT Greengrass que os dispositivos do cliente possam descobrir como se conectar.			
Ponte MQTT	Retransmite mensagens MQTT entre dispositivos clientes, AWS IoT Greengrass publicação/assinatura local e. AWS IoT Core	Plugin	Linux, Windows	Sim
Corretor MQTT 3.1.1 (Moquette)	Executa um broker MQTT 3.1.1 que manipula mensagens entre dispositivos cliente e o dispositivo principal.	Plugin	Linux, Windows	Sim

Componente	Descrição	Tipo de componente	SO com suporte	Código aberto
Corretora MQTT 5 (EMQX)	Executa um broker MQTT 5 que manipula mensagens entre dispositivos cliente e o dispositivo principal.	Genérico	Linux, Windows	Não
Gerenciador de sombras	Permite a interação com sombras no dispositivo principal. Ele gerencia o armazenamento de documentos paralelos e também a sincronização dos estados paralelos locais com o serviço AWS IoT Device Shadow.	Plugin	Linux, Windows	Sim

Conecte dispositivos cliente aos dispositivos principais

Você pode configurar a descoberta na nuvem para conectar dispositivos cliente aos dispositivos principais. Quando você configura a descoberta na nuvem, os dispositivos cliente podem se conectar ao serviço de AWS IoT Greengrass nuvem para recuperar informações sobre os dispositivos principais aos quais eles podem se conectar. Em seguida, os dispositivos cliente podem tentar se conectar a cada dispositivo principal até que se conectem com sucesso.

Para usar a descoberta na nuvem, você deve fazer o seguinte:

- Associe dispositivos clientes aos dispositivos principais aos quais eles podem se conectar.
- Especifique os endpoints do agente MQTT nos quais os dispositivos cliente podem se conectar a cada dispositivo principal.
- Implante componentes no dispositivo principal que permitam o suporte para dispositivos clientes.

  Você também pode implantar componentes opcionais para fazer o seguinte:

  - Retransmita mensagens entre dispositivos cliente, componentes do Greengrass e AWS IoT Core o serviço de nuvem.
  - Gerencie automaticamente os endpoints do broker MQTT do dispositivo principal para você.
  - Gerencie as sombras do dispositivo cliente local e sincronize as sombras com o AWS IoT Core serviço de nuvem.

Você também deve revisar e atualizar a AWS IoT política do dispositivo principal para verificar se ele tem as permissões necessárias para conectar dispositivos clientes. Para ter mais informações, consulte Requisitos.

Depois de configurar a descoberta na nuvem, você pode testar as comunicações entre um dispositivo cliente e um dispositivo principal. Para ter mais informações, consulte Teste as comunicações entre dispositivos clientes.

Tópicos

- Autenticação de clientes enquanto estiver off-line
- Gerencie os endpoints principais do dispositivo
- Escolha um corretor MQTT
- Conectando dispositivos cliente a um dispositivoAWS IoT Greengrass Core com um corretor MQTT
- Teste as comunicações entre dispositivos clientes
- API RESTful de descoberta do Greengrass

Requisitos

Para conectar dispositivos cliente a um dispositivo principal, você deve ter o seguinte:

- O dispositivo principal deve executar o Greengrass nucleus v2.2.0 ou posterior.
- A função de serviço do Greengrass associada AWS IoT Greengrass a você Conta da AWS na AWS região em que o dispositivo principal opera. Para ter mais informações, consulte Configurar a função de serviço do Greengrass.
- A AWS IoT política do dispositivo principal deve permitir as seguintes permissões:
  - `greengrass:PutCertificateAuthorities`
  - `greengrass:VerifyClientDeviceIdentity`
  - `greengrass:VerifyClientDeviceIoTCertificateAssociation`
  - `greengrass:GetConnectivityInfo`
  - `greengrass:UpdateConnectivityInfo`— (Opcional) Essa permissão é necessária para usar o componente detector de IP, que reporta as informações de conectividade de rede do dispositivo principal ao serviço de AWS IoT Greengrass nuvem.
  - `iot:GetThingShadow`,`iot:UpdateThingShadow`, e `iot:DeleteThingShadow` — (Opcional) Essas permissões são necessárias para usar o componente gerenciador de sombras com AWS IoT Core o qual sincronizar as sombras do dispositivo cliente. Esse recurso requer o Greengrass nucleus v2.6.0 ou posterior, o shadow manager v2.2.0 ou posterior e o MQTT bridge v2.2.0 ou posterior.

  Para ter mais informações, consulte Configure a política da AWS IoT coisa.

> **Note**
>
> Se você usou a AWS IoT política padrão ao instalar o software AWS IoT Greengrass Core, o dispositivo principal tem uma AWS IoT política que permite acesso a todas as AWS IoT Greengrass ações (`greengrass:*`).

- AWS IoTcoisas que você pode conectar como dispositivos clientes. Para obter mais informações, consulte Criar AWS IoT recursos no Guia do AWS IoT Core desenvolvedor.
- O dispositivo cliente deve se conectar usando uma ID de cliente. Um ID de cliente é um nome de coisa. Nenhum outro ID de cliente será aceito.
- A AWS IoT política de cada dispositivo cliente deve permitir a `greengrass:Discover` permissão. Para ter mais informações, consulte AWS IoTPolítica mínima para dispositivos clientes.

Tópicos

- Configurar a função de serviço do Greengrass
- Configure a política da AWS IoT coisa

Configurar a função de serviço do Greengrass

perfil de serviçoO perfil de serviço do Greengrass é um perfil de serviço do (IAM) AWS Identity and Access Management que autoriza o AWS IoT Greengrass a acessar recursos de serviços da AWS em seu nome. Essa função possibilita verificar AWS IoT Greengrass a identidade dos dispositivos clientes e gerenciar as principais informações de conectividade do dispositivo.

Se você ainda não configurou a função de serviço do Greengrass nesta região, você deve associar uma função de serviço do Greengrass AWS IoT Greengrass à sua Conta da AWS nesta região.

Ao usar a página Configure Core Device Discovery no AWS IoT Greengrassconsole, AWS IoT Greengrass configura a função de serviço do Greengrass para você. Caso contrário, você pode configurá-lo manualmente usando o AWS IoTconsole ou a AWS IoT Greengrass API.

Nesta seção, você verifica se a função de serviço do Greengrass está configurada. Se não estiver configurado, você cria uma nova função de serviço do Greengrass para se AWS IoT Greengrass associar à sua Conta da AWS nesta região.

Configurar a função de serviço do Greengrass (console)

1. Verifique se a função de serviço do Greengrass está associada AWS IoT Greengrass à sua Conta da AWS nesta região. Faça o seguinte:

   a. Navegue até o console do AWS IoT.

   b. No painel de navegação, selecione Configurações.

   c. Na seção Função de serviço do Greengrass, encontre Função de serviço atual para ver se uma função de serviço do Greengrass está associada.

      Se você tiver uma função de serviço do Greengrass associada, você atende a esse requisito para usar o componente detector de IP. Vá para Configure a política da AWS IoT coisa.

2. Se a função de serviço do Greengrass não estiver associada AWS IoT Greengrass à sua Conta da AWS nesta região, crie uma função de serviço do Greengrass e associe-a. Faça o seguinte:

   a. Navegue até o console do IAM.

   b. Escolha Roles.

   c. Selecione Criar perfil.

   d. Na página Criar função, faça o seguinte:

      i. Em Tipo de entidade confiável, escolha AWS service (Serviço da AWS).

      ii. Em Caso de uso, Casos de uso para outros Serviços da AWS, escolha Greengrass, selecione Greengrass. Essa opção especifica a adição AWS IoT Greengrass como uma entidade confiável que pode assumir essa função.

      iii. Escolha Próximo.

      iv. Em Políticas de permissões, selecione a AWSGreengrassResourceAccessRolePolicypara anexar à função.

      v. Escolha Próximo.

      vi. Em Nome da função, insira um nome para a função, como**`Greengrass_ServiceRole`**.

      vii. Selecione Criar perfil.

   e. Navegue até o console do AWS IoT.

   f. No painel de navegação, selecione Configurações.

   g. Na seção Função de serviço do Greengrass, escolha Anexar função.

h. No modal Atualizar função de serviço do Greengrass, selecione a função do IAM que você criou e, em seguida, escolha Anexar função.

Configurar a função de serviço do Greengrass () AWS CLI

1. Verifique se a função de serviço do Greengrass está associada AWS IoT Greengrass à sua Conta da AWS nesta região.

```
aws greengrassv2 get-service-role-for-account
```

Se a função de serviço do Greengrass estiver associada, a operação retornará uma resposta que contém informações sobre a função.

Se você tiver uma função de serviço do Greengrass associada, você atende a esse requisito para usar o componente detector de IP. Vá para Configure a política da AWS IoT coisa.

2. Se a função de serviço do Greengrass não estiver associada AWS IoT Greengrass à sua Conta da AWS nesta região, crie uma função de serviço do Greengrass e associe-a. Faça o seguinte:

a. Crie uma função com uma política de confiança que permita que o AWS IoT Greengrass assuma a função. Este exemplo cria uma função chamada `Greengrass_ServiceRole`, mas você pode usar um nome diferente. Recomendamos que você também inclua as chaves de contexto de condição global `aws:SourceArn` e `aws:SourceAccount` em sua política de confiança para ajudar a evitar o problema de segurança confused deputy. As chaves de contexto de condição restringem o acesso para permitir somente as solicitações provenientes da conta especificada e do espaço de trabalho do Greengrass. Para obter mais informações sobre o problema confused deputy, consulte Prevenção do problema do substituto confuso entre serviços.

Linux or Unix

```
aws iam create-role --role-name Greengrass_ServiceRole --assume-role-policy-
document '{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "greengrass.amazonaws.com"
      },
```

```
      "Action": "sts:AssumeRole",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:greengrass:region:account-id:*"
        },
        "StringEquals": {
          "aws:SourceAccount": "account-id"
        }
      }
    }
  ]
}'
```

Windows Command Prompt (CMD)

```
aws iam create-role --role-name Greengrass_ServiceRole --assume-role-policy-
document "{\\"Version\\":\\"2012-10-17\\",\\"Statement\\":[{\\"Effect\
\":\\"Allow\\",\\"Principal\\":{\\"Service\\":\\"greengrass.amazonaws.com
\\"},\\"Action\\":\\"sts:AssumeRole\\",\\"Condition\\":{\\"ArnLike\\":
{\\"aws:SourceArn\\":\\"arn:aws:greengrass:region:account-id:*\\"},\
\"StringEquals\\":{\\"aws:SourceAccount\\":\\"account-id\\"}}}]}"
```

PowerShell

```
aws iam create-role --role-name Greengrass_ServiceRole --assume-role-policy-
document '{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "greengrass.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:greengrass:region:account-id:*"
        },
        "StringEquals": {
          "aws:SourceAccount": "account-id"
        }
      }
    }
```

```
  ]
}'
```

b. Copie o ARN da função dos metadados da função na saída. Você usará o ARN para associar a função à sua conta.

c. Anexe a política do `AWSGreengrassResourceAccessRolePolicy` à função.

```
aws iam attach-role-policy --role-name Greengrass_ServiceRole --policy-arn
 arn:aws:iam::aws:policy/service-role/AWSGreengrassResourceAccessRolePolicy
```

d. Associe a função de serviço do Greengrass com AWS IoT Greengrass for your. Conta da AWS Substitua *`role-arn`* pelo ARN da função de serviço.

```
aws greengrassv2 associate-service-role-to-account --role-arn role-arn
```

A operação retornará a seguinte resposta se for bem-sucedida.

```
{
  "associatedAt": "timestamp"
}
```

Configure a política da AWS IoT coisa

Os dispositivos principais usam certificados de dispositivo X.509 para autorizar conexões com o. AWS Você anexa AWS IoT políticas aos certificados do dispositivo para definir as permissões para um dispositivo principal. Para obter mais informações, consulte AWS IoTPolítica mínima para oferecer suporte aos dispositivos do cliente e Políticas do AWS IoT para operações de plano de dados.

Para conectar dispositivos cliente a um dispositivo principal, a AWS IoT política do dispositivo principal deve permitir as seguintes permissões:

- `greengrass:PutCertificateAuthorities`
- `greengrass:VerifyClientDeviceIdentity`
- `greengrass:VerifyClientDeviceIoTCertificateAssociation`
- `greengrass:GetConnectivityInfo`

- `greengrass:UpdateConnectivityInfo`— (Opcional) Essa permissão é necessária para usar o componente detector de IP, que reporta as informações de conectividade de rede do dispositivo principal ao serviço de AWS IoT Greengrass nuvem.
- `iot:GetThingShadow`,`iot:UpdateThingShadow`, e `iot:DeleteThingShadow` — (Opcional) Essas permissões são necessárias para usar o componente gerenciador de sombras com AWS IoT Core o qual sincronizar as sombras do dispositivo cliente. Esse recurso requer o Greengrass nucleus v2.6.0 ou posterior, o shadow manager v2.2.0 ou posterior e o MQTT bridge v2.2.0 ou posterior.

Nesta seção, você revisa as AWS IoT políticas do seu dispositivo principal e adiciona as permissões necessárias que estão faltando. Se você usou o instalador de software AWS IoT Greengrass Core para provisionar recursos, seu dispositivo principal tem uma AWS IoT política que permite acesso a todas as AWS IoT Greengrass ações (`greengrass:*`). Nesse caso, você deve atualizar a AWS IoT política somente se planeja implantar o componente do gerenciador de sombras para sincronizar as sombras do dispositivo. AWS IoT Core Caso contrário, você pode pular esta seção.

Configurar a política da AWS IoT coisa (console)

1. No menu de navegação AWS IoT Greengrassdo console, escolha Dispositivos principais.
2. Na página Dispositivos principais, escolha o dispositivo principal a ser atualizado.
3. Na página de detalhes do dispositivo principal, escolha o link para o item do dispositivo principal. Esse link abre a página de detalhes do item no AWS IoT console.
4. Na página de detalhes do item, escolha Certificados.
5. Na guia Certificados, escolha o certificado ativo do item.
6. Na página de detalhes do certificado, escolha Políticas.
7. Na guia Políticas, escolha a AWS IoT política a ser revisada e atualizada. Você pode adicionar as permissões necessárias a qualquer política anexada ao certificado ativo do dispositivo principal.

> **Note**
>
> Se você usou o instalador de software AWS IoT Greengrass Core para provisionar recursos, você tem duas AWS IoT políticas. Recomendamos que você escolha a política nomeada GreengrassV2IoTThingPolicy, se ela existir. Os dispositivos principais que você cria com o instalador rápido usam esse nome de política por padrão. Se você

adicionar permissões a essa política, também estará concedendo essas permissões a outros dispositivos principais que usam essa política.

8. Na visão geral da política, escolha Editar versão ativa.
9. Revise a política para obter as permissões necessárias e adicione as permissões necessárias que estejam faltando.

   - `greengrass:PutCertificateAuthorities`
   - `greengrass:VerifyClientDeviceIdentity`
   - `greengrass:VerifyClientDeviceIoTCertificateAssociation`
   - `greengrass:GetConnectivityInfo`
   - `greengrass:UpdateConnectivityInfo`— (Opcional) Essa permissão é necessária para usar o componente detector de IP, que reporta as informações de conectividade de rede do dispositivo principal ao serviço de AWS IoT Greengrass nuvem.
   - `iot:GetThingShadow`,`iot:UpdateThingShadow`, e `iot:DeleteThingShadow` — (Opcional) Essas permissões são necessárias para usar o componente gerenciador de sombras com AWS IoT Core o qual sincronizar as sombras do dispositivo cliente. Esse recurso requer o Greengrass nucleus v2.6.0 ou posterior, o shadow manager v2.2.0 ou posterior e o MQTT bridge v2.2.0 ou posterior.
10. (Opcional) Para permitir que o dispositivo principal sincronize sombras comAWS IoT Core, adicione a seguinte declaração à política. Se você planeja interagir com as sombras do dispositivo cliente, mas não sincronizá-las com elasAWS IoT Core, pule esta etapa. Substitua a *`região`* e *`o ID da conta`* pela região que você usa e pelo seu Conta da AWS número.

    - Esse exemplo de declaração permite o acesso às sombras do dispositivo de todas as coisas. Para seguir as melhores práticas de segurança, você pode restringir o acesso somente ao dispositivo principal e aos dispositivos cliente conectados ao dispositivo principal. Para ter mais informações, consulte AWS IoTPolítica mínima para oferecer suporte aos dispositivos do cliente.

```
{
  "Effect": "Allow",
  "Action": [
    "iot:GetThingShadow",
    "iot:UpdateThingShadow",
    "iot:DeleteThingShadow"
```

```
  ],
  "Resource": [
    "arn:aws:iot:region:account-id:thing/*"
  ]
}
```

Depois de adicionar essa declaração, o documento de política pode ser semelhante ao exemplo a seguir.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Connect",
        "iot:Publish",
        "iot:Subscribe",
        "iot:Receive",
        "greengrass:*"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:GetThingShadow",
        "iot:UpdateThingShadow",
        "iot:DeleteThingShadow"
      ],
      "Resource": [
        "arn:aws:iot:region:account-id:thing/*"
      ]
    }
  ]
}
```

11. Para definir uma nova versão da política como a versão ativa, em Status da versão da política, selecione Definir a versão editada como a versão ativa desta política.
12. Selecione Salvar como nova versão.

Configurar a política da AWS IoT coisa (AWS CLI)

1. Liste os princípios básicos do AWS IoT dispositivo principal. Os principais podem ser certificados de dispositivos X.509 ou outros identificadores. Execute o comando a seguir e *MyGreengrassCore*substitua pelo nome do dispositivo principal.

   ```
   aws iot list-thing-principals --thing-name MyGreengrassCore
   ```

   A operação retorna uma resposta que lista os princípios básicos do dispositivo principal.

   ```
   {
       "principals": [
           "arn:aws:iot:us-west-2:123456789012:cert/certificateId"
       ]
   }
   ```

2. Identifique o certificado ativo do dispositivo principal. *Execute o comando a seguir e substitua o CertificateID pelo ID de cada certificado da etapa anterior até encontrar o certificado ativo.* O ID do certificado é a string hexadecimal no final do ARN do certificado. O `--query` argumento especifica a saída somente do status do certificado.

   ```
   aws iot describe-certificate --certificate-id certificateId --query
    'certificateDescription.status'
   ```

   A operação retorna o status do certificado como uma string. Por exemplo, se o certificado estiver ativo, essa operação será exibida. `"ACTIVE"`

3. Liste as AWS IoT políticas anexadas ao certificado. Execute o comando a seguir e substitua o ARN do certificado pelo ARN do certificado.

   ```
   aws iot list-principal-policies --principal arn:aws:iot:us-
   west-2:123456789012:cert/certificateId
   ```

   A operação retorna uma resposta que lista as AWS IoT políticas anexadas ao certificado.

   ```
   {
       "policies": [
           {
   ```

```
            "policyName":
 "GreengrassTESCertificatePolicyMyGreengrassCoreTokenExchangeRoleAlias",
            "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassTESCertificatePolicyMyGreengrassCoreTokenExchangeRoleAlias"
        },
        {
            "policyName": "GreengrassV2IoTThingPolicy",
            "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassV2IoTThingPolicy"
        }
    ]
}
```

4. Escolha a política a ser visualizada e atualizada.

> **Note**
>
> Se você usou o instalador de software AWS IoT Greengrass Core para provisionar recursos, você tem duas AWS IoT políticas. Recomendamos que você escolha a política nomeada GreengrassV2IoTThingPolicy, se ela existir. Os dispositivos principais que você cria com o instalador rápido usam esse nome de política por padrão. Se você adicionar permissões a essa política, também estará concedendo essas permissões a outros dispositivos principais que usam essa política.

5. Obtenha o documento da política. Execute o comando a seguir e substitua *GreengrassV2IoTThingPolicy* pelo nome da política.

```
aws iot get-policy --policy-name GreengrassV2IoTThingPolicy
```

A operação retorna uma resposta que contém o documento da política e outras informações sobre a política. O documento de política é um objeto JSON serializado como uma string.

```
{
    "policyName": "GreengrassV2IoTThingPolicy",
    "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassV2IoTThingPolicy",
    "policyDocument": "{\
  \\"Version\\": \\"2012-10-17\\",\
  \\"Statement\\": [\
    {\
      \\"Effect\\": \\"Allow\\",\
```

```
      \\"Action\\": [\
                \\"iot:Connect\\",\
                \\"iot:Publish\\",\
                \\"iot:Subscribe\\",\
                \\"iot:Receive\\",\
                \\"greengrass:*\\"\
],\
      \\"Resource\\": \\"*\\"\
    }\
  ]\
}",
    "defaultVersionId": "1",
    "creationDate": "2021-02-05T16:03:14.098000-08:00",
    "lastModifiedDate": "2021-02-05T16:03:14.098000-08:00",
    "generationId":
 "f19144b798534f52c619d44f771a354f1b957dfa2b850625d9f1d0fde530e75f"
}
```

6. Use um conversor on-line ou outra ferramenta para converter a string do documento de política em um objeto JSON e, em seguida, salve-a em um arquivo chamado`iot-policy.json`.

   Por exemplo, se você tiver a ferramenta jq instalada, poderá executar o comando a seguir para obter o documento de política, convertê-lo em um objeto JSON e salvá-lo como um objeto JSON.

   ```
   aws iot get-policy --policy-name GreengrassV2IoTThingPolicy --query
    'policyDocument' | jq fromjson >> iot-policy.json
   ```

7. Revise a política para obter as permissões necessárias e adicione as permissões necessárias que estejam faltando.

   Por exemplo, em um sistema baseado em Linux, você pode executar o seguinte comando para usar o GNU nano para abrir o arquivo.

   ```
   nano iot-policy.json
   ```

   - `greengrass:PutCertificateAuthorities`
   - `greengrass:VerifyClientDeviceIdentity`
   - `greengrass:VerifyClientDeviceIoTCertificateAssociation`
   - `greengrass:GetConnectivityInfo`

- `greengrass:UpdateConnectivityInfo`— (Opcional) Essa permissão é necessária para usar o componente detector de IP, que reporta as informações de conectividade de rede do dispositivo principal ao serviço de AWS IoT Greengrass nuvem.
- `iot:GetThingShadow`,`iot:UpdateThingShadow`, e `iot:DeleteThingShadow` — (Opcional) Essas permissões são necessárias para usar o componente gerenciador de sombras com AWS IoT Core o qual sincronizar as sombras do dispositivo cliente. Esse recurso requer o Greengrass nucleus v2.6.0 ou posterior, o shadow manager v2.2.0 ou posterior e o MQTT bridge v2.2.0 ou posterior.

8. Salve as alterações como uma nova versão da política. Execute o comando a seguir e substitua *`GreengrassV2IoT ThingPolicy`* pelo nome da política.

```
aws iot create-policy-version --policy-name GreengrassV2IoTThingPolicy --policy-
document file://iot-policy.json --set-as-default
```

Se for bem-sucedida, a operação retornará uma resposta semelhante à do exemplo a seguir.

```
{
    "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassV2IoTThingPolicy",
    "policyDocument": "{\
  \\"Version\\": \\"2012-10-17\\",\
  \\"Statement\\": [\
    {\
      \\"Effect\\": \\"Allow\\",\
      \\"Action\\": [\
\\t\\t\\"iot:Connect\\",\
\\t\\t\\"iot:Publish\\",\
\\t\\t\\"iot:Subscribe\\",\
\\t\\t\\"iot:Receive\\",\
\\t\\t\\"greengrass:*\\"\
      ],\
      \\"Resource\\": \\"*\\"\
    }\
  ]\
}",
    "policyVersionId": "2",
    "isDefaultVersion": true
}
```

Componentes do Greengrass para suporte a dispositivos clientes

> **Important**
>
> O dispositivo principal deve executar o Greengrass nucleus v2.2.0 ou posterior para oferecer suporte aos dispositivos clientes.

Para permitir que dispositivos cliente se conectem e se comuniquem com um dispositivo principal, você implanta os seguintes componentes do Greengrass no dispositivo principal:

- Autenticação do dispositivo cliente (`aws.greengrass.clientdevices.Auth`)

  Implante o componente de autenticação do dispositivo cliente para autenticar dispositivos cliente e autorizar ações do dispositivo cliente. Esse componente permite que suas AWS IoT coisas se conectem a um dispositivo principal.

  Esse componente requer alguma configuração para ser usado. Você deve especificar grupos de dispositivos clientes e as operações que cada grupo está autorizado a realizar, como se conectar e se comunicar pelo MQTT. Para obter mais informações, consulte Configuração do componente de autenticação do dispositivo cliente.
- Corretor MQTT 3.1.1 (Moquette) (`aws.greengrass.clientdevices.mqtt.Moquette`)

  Implante o componente Moquette MQTT broker para executar um broker MQTT leve. O agente Moquette MQTT é compatível com o MQTT 3.1.1 e inclui suporte local para QoS 0, QoS 1, QoS 2, mensagens retidas, mensagens de última vontade e assinaturas persistentes.

  Você não precisa configurar esse componente para usá-lo. No entanto, você pode configurar a porta em que esse componente opera o agente MQTT. Por padrão, ele usa a porta 8883.
- Corretora MQTT 5 (EMQX) (`aws.greengrass.clientdevices.mqtt.EMQX`)

  > **Note**
  >
  > Para usar o agente EMQX MQTT 5, você deve usar o Greengrass nucleus v2.6.0 ou posterior e a autenticação do dispositivo cliente v2.2.0 ou posterior.

Implante o componente EMQX MQTT broker para usar os recursos do MQTT 5.0 na comunicação entre dispositivos cliente e o dispositivo principal. O broker EMQX MQTT é compatível com o MQTT 5.0 e inclui suporte para intervalos de expiração de sessões e mensagens, propriedades do usuário, assinaturas compartilhadas, aliases de tópicos e muito mais.

Você não precisa configurar esse componente para usá-lo. No entanto, você pode configurar a porta em que esse componente opera o agente MQTT. Por padrão, ele usa a porta 8883.

- Ponte MQTT (`aws.greengrass.clientdevices.mqtt.Bridge`)

  (Opcional) Implante o componente de ponte MQTT para retransmitir mensagens entre dispositivos cliente (MQTT local), publicação/assinatura local e MQTT. AWS IoT Core Configure esse componente para sincronizar dispositivos cliente AWS IoT Core e interagir com dispositivos clientes a partir dos componentes do Greengrass.

  Esse componente requer configuração para ser usado. Você deve especificar os mapeamentos de tópicos em que esse componente retransmite mensagens. Para obter mais informações, consulte Configuração do componente de ponte MQTT.
- Detector IP (`aws.greengrass.clientdevices.IPDetector`)

  (Opcional) Implante o componente detector de IP para reportar automaticamente os endpoints do broker MQTT do dispositivo principal ao serviço de AWS IoT Greengrass nuvem. Você não pode usar esse componente se tiver uma configuração de rede complexa, como aquela em que um roteador encaminha a porta do agente MQTT para o dispositivo principal.

  Você não precisa configurar esse componente para usá-lo.
- Gerenciador de sombras (`aws.greengrass.ShadowManager`)

  > **Note**
  >
  > Para gerenciar as sombras do dispositivo cliente, você deve usar o Greengrass nucleus v2.6.0 ou posterior, o shadow manager v2.2.0 ou posterior e o MQTT bridge v2.2.0 ou posterior.

  (Opcional) Implante o componente do gerenciador de sombras para gerenciar as sombras do dispositivo cliente no dispositivo principal. Os componentes do Greengrass podem obter, atualizar e excluir sombras do dispositivo cliente para interagir com os dispositivos do cliente. Você também

pode configurar o componente do gerenciador de sombras para sincronizar as sombras do dispositivo cliente com o serviço de AWS IoT Core nuvem.

Para usar esse componente com sombras do dispositivo cliente, você deve configurar o componente de ponte MQTT para retransmitir mensagens entre dispositivos cliente e o gerenciador de sombras, que usa publicação/assinatura local. Caso contrário, esse componente não exige configuração para ser usado, mas exige configuração para sincronizar as sombras do dispositivo.

> **Note**
>
> Recomendamos que você implante somente um componente do broker MQTT. A ponte MQTT e os componentes do detector IP funcionam com apenas um componente intermediário MQTT por vez. Se você implantar vários componentes do MQTT broker, deverá configurá-los para usar portas diferentes.

Configurar a descoberta na nuvem (console)

Você pode usar o AWS IoT Greengrass console para associar dispositivos cliente, gerenciar endpoints do dispositivo principal e implantar componentes para habilitar o suporte ao dispositivo cliente. Para ter mais informações, consulte Etapa 2: ativar o suporte ao dispositivo cliente.

Configurar a descoberta na nuvem (AWS CLI)

Você pode usar o AWS Command Line Interface (AWS CLI) para associar dispositivos cliente, gerenciar endpoints principais do dispositivo e implantar componentes para habilitar o suporte ao dispositivo cliente. Para ver mais informações, consulte:

- Gerenciar associações de dispositivos clientes (AWS CLI)
- Gerencie os endpoints principais do dispositivo
- AWS- componentes do dispositivo cliente fornecidos
- Criar implantações

Associe dispositivos do cliente

Para usar a descoberta na nuvem, associe dispositivos cliente a um dispositivo principal para que eles possam descobrir o dispositivo principal. Em seguida, eles podem usar a API de descoberta do Greengrass para recuperar informações de conectividade e certificados para seus dispositivos principais associados.

Da mesma forma, desassocie os dispositivos cliente de um dispositivo principal para impedir que eles descubram o dispositivo principal.

Tópicos

- Gerenciar associações de dispositivos clientes (console)
- Gerenciar associações de dispositivos clientes (AWS CLI)
- Gerenciar associações de dispositivos clientes (API)

Gerenciar associações de dispositivos clientes (console)

Você pode usar o AWS IoT Greengrass console para visualizar, adicionar e excluir associações de dispositivos clientes.

Para visualizar associações de dispositivos clientes para um dispositivo principal (console)

1. Navegue até o console do AWS IoT Greengrass.
2. Escolha dispositivos principais.
3. Escolha o dispositivo principal a ser gerenciado.
4. Na página de detalhes do dispositivo principal, escolha a guia Dispositivos clientes.
5. Na seção Dispositivos cliente associados, você pode ver quais dispositivos cliente (AWS IoTitens) estão associados ao dispositivo principal.

Para associar dispositivos cliente a um dispositivo principal (console)

1. Navegue até o console do AWS IoT Greengrass.
2. Escolha dispositivos principais.
3. Escolha o dispositivo principal a ser gerenciado.
4. Na página de detalhes do dispositivo principal, escolha a guia Dispositivos clientes.

5. Na seção Dispositivos cliente associados, escolha Associar dispositivos cliente.
6. No modal Associar dispositivos cliente ao dispositivo principal, faça o seguinte para cada dispositivo cliente a ser associado:

   a. Digite o nome da AWS IoT coisa a ser associada como dispositivo cliente.

   b. Escolha Adicionar.
7. Selecione Associar.

   Os dispositivos cliente que você associou agora podem usar a API de descoberta do Greengrass para descobrir esse dispositivo principal.

Para dissociar dispositivos cliente de um dispositivo principal (console)

1. Navegue até o console do AWS IoT Greengrass.
2. Escolha dispositivos principais.
3. Escolha o dispositivo principal a ser gerenciado.
4. Na página de detalhes do dispositivo principal, escolha a guia Dispositivos clientes.
5. Na seção Dispositivos cliente associados, selecione cada dispositivo cliente a ser desassociado.
6. Escolha Desassociar.
7. No modal de confirmação, escolha Desassociar.

   Os dispositivos cliente que você desassociou não podem mais usar a API de descoberta do Greengrass para descobrir esse dispositivo principal.

Gerenciar associações de dispositivos clientes (AWS CLI)

Você pode usar o AWS Command Line Interface (AWS CLI) para gerenciar associações de dispositivos clientes para um dispositivo principal.

Para visualizar associações de dispositivos clientes para um dispositivo principal (AWS CLI)

- Use o seguinte comando: list-client-devices-associated- with-core-device.

Para associar dispositivos cliente a um dispositivo principal (AWS CLI)

- Use o seguinte comando: batch-associate-client-device- with-core-device.

Para desassociar dispositivos cliente de um dispositivo principal () AWS CLI

- Use o seguinte comando: batch-disassociate-client-device- from-core-device.

Gerenciar associações de dispositivos clientes (API)

Você pode usar a AWS API para gerenciar associações de dispositivos clientes para um dispositivo principal.

Para visualizar associações de dispositivos clientes para um dispositivo principal (AWSAPI)

- Use a seguinte operação: ListClientDevicesAssociatedWithCoreDevice.

Para associar dispositivos cliente a um dispositivo principal (AWSAPI)

- Use a seguinte operação: BatchAssociateClientDeviceWithCoreDevice.

Para dissociar dispositivos cliente de um dispositivo principal (AWSAPI)

- Use a seguinte operação: BatchDisassociateClientDeviceFromCoreDevice.

Autenticação de clientes enquanto estiver off-line

Com a autenticação offline, você pode configurar seu dispositivo AWS IoT Greengrass Core para que os dispositivos cliente possam se conectar a um dispositivo principal, mesmo quando o dispositivo principal não estiver conectado à nuvem. Quando você usa a autenticação off-line, seus dispositivos Greengrass podem continuar funcionando em um ambiente parcialmente off-line.

Para usar a autenticação off-line para um dispositivo cliente com uma conexão com a nuvem, você precisa do seguinte:

- Um dispositivo AWS IoT Greengrass principal com o Autenticação do dispositivo cliente componente implantado. Você deve usar a versão 2.3.0 ou superior para autenticação offline.
- Uma conexão em nuvem para o dispositivo principal durante a conexão inicial dos dispositivos clientes.

Armazenamento de credenciais do cliente

Quando um dispositivo cliente se conecta a um dispositivo principal pela primeira vez, o dispositivo principal chama o AWS IoT Greengrass serviço. Quando chamado, o Greengrass valida o registro do dispositivo cliente como uma coisa. AWS IoT Também valida se o dispositivo tem um certificado válido. O dispositivo principal então armazena essas informações localmente.

Na próxima vez que o dispositivo se conectar, o dispositivo principal do Greengrass tentará validar o dispositivo cliente com o serviço. AWS IoT Greengrass Se não conseguir se conectar AWS IoT Greengrass, o dispositivo principal usa as informações do dispositivo armazenadas localmente para validar o dispositivo cliente.

Você pode configurar por quanto tempo o dispositivo principal do Greengrass armazena as credenciais. Você pode definir o tempo limite de um minuto para 2.147.483.647 minutos definindo a opção de `clientDeviceTrustDurationMinutes` configuração na configuração do componente de autenticação do dispositivo cliente. O padrão é um minuto, o que efetivamente desativa a autenticação offline. Ao definir esse tempo limite, recomendamos que você considere suas necessidades de segurança. Você também deve considerar por quanto tempo espera que os dispositivos principais funcionem enquanto estão desconectados da nuvem.

O dispositivo principal atualiza seu armazenamento de credenciais três vezes:

1. Quando um dispositivo se conecta ao dispositivo principal pela primeira vez.
2. Se o dispositivo principal estiver conectado à nuvem, quando um dispositivo cliente se reconectar ao dispositivo principal.
3. Se o dispositivo principal estiver conectado à nuvem, uma vez por dia para atualizar todo o armazenamento de credenciais.

Quando o dispositivo principal do Greengrass atualiza seu armazenamento de credenciais, ele usa a operação. ListClientDevicesAssociatedWithCoreDevice O Greengrass atualiza somente os dispositivos retornados por essa operação. Para associar um dispositivo cliente a um dispositivo principal, consulteAssocie dispositivos do cliente.

Para usar a `ListClientDevicesAssociatedWithCoreDevice` operação, você deve adicionar permissão para a operação à função AWS Identity and Access Management (IAM) associada à Conta da AWS que é executada AWS IoT Greengrass. Para ter mais informações, consulte Autorize os dispositivos principais a interagir com os serviços AWS.

Gerencie os endpoints principais do dispositivo

Ao usar a descoberta na nuvem, você armazena os endpoints do broker MQTT para dispositivos principais no serviço de AWS IoT Greengrass nuvem. Os dispositivos cliente se conectam AWS IoT Greengrass para recuperar esses endpoints e outras informações para seus dispositivos principais associados.

Para cada dispositivo principal, você pode gerenciar endpoints automática ou manualmente.

- Gerencie terminais automaticamente com detector de IP

  Você pode implantar o componente detector de IP para gerenciar automaticamente os endpoints do dispositivo principal se tiver uma configuração de rede não complexa, como quando os dispositivos cliente estão na mesma rede do dispositivo principal. Você não pode usar o componente detector de IP se o dispositivo principal estiver atrás de um roteador que encaminha a porta do agente MQTT para o dispositivo principal, por exemplo.

  O componente detector de IP também é útil se você implantar em grupos de coisas, porque ele gerencia os endpoints de todos os dispositivos principais do grupo de coisas. Para ter mais informações, consulte Use o detector de IP para gerenciar automaticamente os endpoints.
- Gerencie manualmente os endpoints

  Se você não puder usar o componente detector de IP, deverá gerenciar manualmente os endpoints principais do dispositivo. Você pode atualizar esses endpoints com o console ou a API. Para ter mais informações, consulte Gerencie manualmente os endpoints.

Tópicos

Use o detector de IP para gerenciar automaticamente os endpoints

Se você tiver uma configuração de rede simples, como os dispositivos cliente na mesma rede do dispositivo principal, poderá implantar o componente detector de IP para fazer o seguinte:

- Monitore as informações de conectividade de rede local do dispositivo principal do Greengrass. Essas informações incluem os endpoints de rede do dispositivo principal e a porta em que o agente MQTT opera.

- Relate as informações de conectividade do dispositivo principal ao serviço de AWS IoT Greengrass nuvem.

O componente detector de IP sobrescreve os endpoints que você define manualmente.

> **Important**
>
> A AWS IoT política do dispositivo principal deve permitir a `greengrass:UpdateConnectivityInfo` permissão para usar o componente detector de IP. Para obter mais informações, consulte Configure a política da AWS IoT coisa e Políticas do AWS IoT para operações de plano de dados.

Você pode fazer o seguinte para implantar o componente detector de IP:

- Use a página Configurar descoberta no console. Para ter mais informações, consulte Configurar a descoberta na nuvem (console).
- Crie e revise implantações para incluir o detector de IP. Você pode usar o console ou AWS CLI a AWS API para gerenciar implantações. Para ter mais informações, consulte Criar implantações.

Implemente o componente detector de IP (console)

1. No menu de navegação AWS IoT Greengrassdo console, escolha Componentes.
2. Na página Componentes, escolha a guia Componentes públicos e, em seguida, escolha aws.greengrass.clientdevices.IPDetector.
3. Na página aws.greengrass.clientdevices.IPDetector, escolha Implantar.
4. Em Adicionar à implantação, escolha uma implantação existente para revisar ou escolha criar uma nova implantação e, em seguida, escolha Avançar.
5. Se você optar por criar uma nova implantação, escolha o dispositivo principal ou grupo de itens de destino para a implantação. Na página Especificar destino, em Destino de implantação, escolha um dispositivo principal ou grupo de itens e, em seguida, escolha Avançar.
6. Na página Selecionar componentes, verifique se o aws.greengrass.clientdevices.IPDetectorcomponente está selecionado e escolha Avançar.
7. Na página Configurar componentes aws.greengrass.clientdevices.IPDetector, selecione e faça o seguinte:

a. Escolha Configurar componente.

b. No aws.greengrass.clientdevices.IPDetector modal Configurar, em Atualização de configuração, em Configuração para mesclar, você pode inserir uma atualização de configuração para configurar o componente do detector de IP. Você pode especificar qualquer uma das seguintes opções de configuração:

- `defaultPort`— (Opcional) A porta do agente MQTT para relatar quando esse componente detecta endereços IP. Você deve especificar esse parâmetro se configurar o agente MQTT para usar uma porta diferente da porta padrão 8883.
- `includeIPv4LoopbackAddrs`— (Opcional) Você pode ativar essa opção para detectar e relatar endereços de loopback IPv4. Esses são endereços IP, como, por exemplo`localhost`, onde um dispositivo pode se comunicar consigo mesmo. Use essa opção em ambientes de teste em que o dispositivo principal e o dispositivo cliente são executados no mesmo sistema.
- `includeIPv4LinkLocalAddrs`— (Opcional) Você pode ativar essa opção para detectar e relatar endereços locais de links IPv4. Use essa opção se a rede do dispositivo principal não tiver o Dynamic Host Configuration Protocol (DHCP) ou endereços IP atribuídos estaticamente.

A atualização da configuração pode ser semelhante ao exemplo a seguir.

```
{
  "defaultPort": "8883",
  "includeIPv4LoopbackAddrs": false,
  "includeIPv4LinkLocalAddrs": false
}
```

c. Escolha Confirmar para fechar o modal e, em seguida, escolha Avançar.

8. Na página Definir configurações avançadas, mantenha as configurações padrão e escolha Avançar.

9. Na página Review, escolha Deploy.

A implantação pode levar até um minuto para ser concluída.

Implemente o componente detector de IP (AWS CLI)

Para implantar o componente detector de IP, crie um documento de implantação que inclua `aws.greengrass.clientdevices.IPDetector` no `components` objeto e especifique a atualização de configuração do componente. Siga as instruções Criar implantações para criar uma nova implantação ou revisar uma implantação existente.

Você pode especificar qualquer uma das seguintes opções para configurar o componente do detector de IP ao criar o documento de implantação:

- `defaultPort`— (Opcional) A porta do agente MQTT para relatar quando esse componente detecta endereços IP. Você deve especificar esse parâmetro se configurar o agente MQTT para usar uma porta diferente da porta padrão 8883.
- `includeIPv4LoopbackAddrs`— (Opcional) Você pode ativar essa opção para detectar e relatar endereços de loopback IPv4. Esses são endereços IP, como, por exemplo`localhost`, onde um dispositivo pode se comunicar consigo mesmo. Use essa opção em ambientes de teste em que o dispositivo principal e o dispositivo cliente são executados no mesmo sistema.
- `includeIPv4LinkLocalAddrs`— (Opcional) Você pode ativar essa opção para detectar e relatar endereços locais de links IPv4. Use essa opção se a rede do dispositivo principal não tiver o Dynamic Host Configuration Protocol (DHCP) ou endereços IP atribuídos estaticamente.

O exemplo de documento de implantação parcial a seguir especifica o relatório da porta 8883 como a porta do agente MQTT.

```
{
  ...,
  "components": {
    ...,
    "aws.greengrass.clientdevices.IPDetector": {
      "componentVersion": "2.1.1",
      "configurationUpdate": {
        "merge": "{\"defaultPort\":\"8883\",}"
      }
    }
  }
}
```

Gerencie manualmente os endpoints

Você pode gerenciar manualmente os endpoints do broker MQTT para dispositivos principais.

Cada endpoint do broker MQTT tem as seguintes informações:

Ponto final () `HostAddress`

Um endereço IP ou endereço DNS em que os dispositivos clientes podem se conectar a um agente MQTT no dispositivo principal.

Port (Porta (`PortNumber`)

A porta em que o agente MQTT opera no dispositivo principal.

Você pode configurar essa porta no componente de agente Moquette MQTT, cujo padrão é usar a porta 8883.

Metadados () `Metadata`

Metadados adicionais a serem fornecidos aos dispositivos clientes que se conectam a esse endpoint.

Tópicos

- Gerenciar endpoints (console)
- Gerenciar endpoints () AWS CLI
- Gerenciar endpoints (API)

Gerenciar endpoints (console)

Você pode usar o AWS IoT Greengrass console para visualizar, atualizar e remover endpoints de um dispositivo principal.

Para gerenciar endpoints para um dispositivo principal (console)

1. Navegue até o console do AWS IoT Greengrass.
2. Escolha dispositivos principais.
3. Escolha o dispositivo principal a ser gerenciado.
4. Na página de detalhes do dispositivo principal, escolha a guia Dispositivos clientes.

5. Na seção de endpoints do broker MQTT, você pode ver os endpoints do broker MQTT do dispositivo principal. Escolha Gerenciar endpoints.
6. No modal Gerenciar endpoints, adicione ou remova os endpoints do broker MQTT para o dispositivo principal.
7. Escolha Atualizar.

Gerenciar endpoints () AWS CLI

Você pode usar o AWS Command Line Interface (AWS CLI) para gerenciar endpoints de um dispositivo principal.

> **Note**
>
> Como o suporte ao dispositivo cliente AWS IoT Greengrass V2 é compatível com versões anterioresAWS IoT Greengrass V1, você pode usar AWS IoT Greengrass V2 nossas operações de AWS IoT Greengrass V1 API para gerenciar os endpoints principais do dispositivo.

Para obter endpoints para um dispositivo principal () AWS CLI

- Use um dos seguintes comandos:
  - greengrass v2: get-connectivity-info
  - capim verde: get-connectivity-info

Para atualizar os endpoints de um dispositivo principal () AWS CLI

- Use um dos seguintes comandos:
  - greengrass v2: update-connectivity-info
  - capim verde: update-connectivity-info

Gerenciar endpoints (API)

Você pode usar a AWS API para gerenciar endpoints para um dispositivo principal.

> **Note**
>
> Como o suporte ao dispositivo cliente AWS IoT Greengrass V2 é compatível com versões anterioresAWS IoT Greengrass V1, você pode usar AWS IoT Greengrass V2 nossas operações de AWS IoT Greengrass V1 API para gerenciar os endpoints principais do dispositivo.

Para obter endpoints para um dispositivo principal (AWSAPI)

- Use uma das seguintes operações:
  - V2: GetConnectivityInfo
  - V1: GetConnectivityInfo

Para atualizar os endpoints de um dispositivo principal (AWSAPI)

- Use uma das seguintes operações:
  - V2: UpdateConnectivityInfo
  - V1: UpdateConnectivityInfo

Escolha um corretor MQTT

AWS IoT Greengrassfornece opções para você escolher qual broker MQTT local executar em seus dispositivos principais. Os dispositivos cliente se conectam ao agente MQTT que é executado em um dispositivo principal, então escolha um agente MQTT que seja compatível com os dispositivos cliente que você deseja conectar.

> **Note**
>
> Recomendamos que você implante somente um componente do broker MQTT. A ponte MQTT e os componentes do detector IP funcionam com apenas um componente intermediário MQTT por vez. Se você implantar vários componentes do MQTT broker, deverá configurá-los para usar portas diferentes.

Você pode escolher entre os seguintes corretores MQTT:

- Corretor MQTT 3.1.1 (Moquette) — `aws.greengrass.clientdevices.mqtt.Moquette`

  Escolha essa opção para um broker MQTT leve que seja compatível com o padrão MQTT 3.1.1. O corretor AWS IoT Core MQTT e também AWS IoT Device SDK são compatíveis com o padrão MQTT 3.1.1, portanto, você pode usar esses recursos para criar um aplicativo que usa o MQTT 3.1.1 em seus dispositivos e no. Nuvem AWS

- Corretor MQTT 5 (EMQX) — `aws.greengrass.clientdevices.mqtt.EMQX`

  Escolha essa opção para usar os recursos do MQTT 5 na comunicação entre dispositivos principais e dispositivos clientes. Esse componente usa mais recursos do que o broker Moquette MQTT 3.1.1 e, nos dispositivos principais do Linux, ele requer o Docker.

  O MQTT 5 é compatível com versões anteriores do MQTT 3.1.1, então você pode conectar dispositivos clientes que usam o MQTT 3.1.1 a esse broker. Se você executar o broker Moquette MQTT 3.1.1, poderá substituí-lo pelo broker EMQX MQTT 5, e os dispositivos clientes poderão continuar se conectando e operando normalmente.

- Implemente um corretor personalizado

  Escolha essa opção para criar um componente de agente local personalizado para se comunicar com os dispositivos do cliente. Você pode criar um broker local personalizado que usa um protocolo diferente do MQTT. AWS IoT Greengrassfornece um componente SDK que você pode usar para autenticar e autorizar dispositivos clientes. Para ter mais informações, consulte Use o AWS IoT Device SDK para se comunicar com o núcleo do Greengrass, outros componentes e AWS IoT Core e Autenticar e autorizar dispositivos clientes.

Conectando dispositivos cliente a um dispositivoAWS IoT Greengrass Core com um corretor MQTT

Quando você usa um agente MQTT em seu dispositivoAWS IoT Greengrass Core, o dispositivo usa uma autoridade de certificação de dispositivo central (CA) exclusiva do dispositivo para emitir um certificado ao corretor para fazer conexões TLS mútuas com clientes.

AWS IoT Greengrassgerados pelo, ou fornecer o próprio domínio. O dispositivo principal com o qual a CA é registradaAWS IoT Greengrass quando oAutenticação do dispositivo cliente componente é conectado. A CA do dispositivo principal gerada automaticamente é persistente. O dispositivo continuará usando a mesma CA enquanto o componente de autenticação do dispositivo cliente estiver configurado.

Quando o corretor MQTT é iniciado, ele solicita um certificado. O componente de autenticação do dispositivo cliente emite um certificado X.509 usando a CA do dispositivo principal. O certificado é alternado quando o corretor é iniciado, quando o certificado expira ou quando as informações de conectividade, como o endereço IP, mudam. Para obter mais informações, consulte Rotação de certificados no corretor MQTT local.

Para conectar um cliente ao corretor MQTT, você precisa do seguinte domínio:

- O dispositivo cliente deve ter a CA do dispositivoAWS IoT Greengrass Core. Você pode obter essa CA por meio da descoberta na nuvem ou fornecendo a CA manualmente. Para obter mais informações, consulte Usando sua própria autoridade de certificação.
- O nome de domínio totalmente qualificado (FQDN) ou o endereço IP do dispositivo principal devem estar presentes no certificado do corretor emitido pela CA do dispositivo principal. Você garante isso usando oDetector IP componente ou configurando manualmente o endereço IP. Para obter mais informações, consulte Gerencie os endpoints principais do dispositivo.
- O componente de autenticação do dispositivo cliente deve permitir que o dispositivo cliente se conecte ao dispositivo principal do Greengrass. Para obter mais informações, consulte Autenticação do dispositivo cliente.

Usando sua própria autoridade de certificação

Se seus dispositivos cliente não conseguirem acessar a nuvem para descobrir seu dispositivo principal, você poderá fornecer uma autoridade de certificação (CA) de dispositivo principal. Seu dispositivo principal Greengrass usa o dispositivo principal CA para emitir certificados para seu corretor MQTT. Depois de configurar o dispositivo principal e provisionar seu dispositivo cliente com sua CA, seus dispositivos cliente podem se conectar ao endpoint e verificar o handshake do TLS usando a CA do dispositivo principal (CA fornecida pela própria ou gerada automaticamente).

Para configurar oAutenticação do dispositivo cliente componente para usar o CA do dispositivo principal, defina o parâmetro `decertificateAuthority` configuração ao implantar o componente. Você deve fornecer os seguintes detalhes durante a configuração:

- A localização de um certificado CA do dispositivo principal.
- A chave privada do certificado CA do dispositivo principal.
- (Opcional) A cadeia de certificados para o certificado raiz se a CA do dispositivo principal for uma CA intermediária.

Se você fornecer uma CA de dispositivo principal,AWS IoT Greengrass registra a CA na nuvem.

Você pode armazenar seus certificados em um módulo de segurança de hardware ou no sistema de arquivos. O exemplo a seguir mostra um`certificateAuthority` configuração para uma CA intermediária armazenada usando HSM/TPM. Observe que a cadeia de certificados só pode ser armazenada em disco.

```
"certificateAuthority": {
    "certificateUri": "pkcs11:object=CustomerIntermediateCA;type=cert",
    "privateKeyUri": "pkcs11:object=CustomerIntermediateCA;type=private"
    "certificateChainUri": "file:///home/ec2-user/creds/certificateChain.pem",
  }
```

Neste exemplo, o parâmetro`certificateAuthority` de configuração configura o componente de autenticação do dispositivo cliente para usar uma CA intermediária do sistema de arquivos:

```
"certificateAuthority": {
    "certificateUri": "file:///home/ec2-user/creds/intermediateCA.pem",
    "privateKeyUri": "file:///home/ec2-user/creds/intermediateCA.privateKey.pem",
    "certificateChainUri": "file:///home/ec2-user/creds/certificateChain.pem",
  }
```

Para conectar os dispositivos ao seu dispositivoAWS IoT Greengrass Core, faça o seguinte:

1. Crie uma autoridade de certificação (CA) intermediária para o dispositivo principal do Greengrass usando a CA raiz da sua organização. É recomendável usar uma CA intermediária como prática recomendada de segurança.
2. Forneça o certificado CA intermediário, a chave privada e a cadeia de certificados para sua CA raiz e para o dispositivo principal do Greengrass. Para obter mais informações, consulte Autenticação do dispositivo cliente. A CA intermediária se torna a CA do dispositivo principal do Greengrass, e o dispositivo registra a CA comAWS IoT Greengrass.
3. Registre o dispositivo cliente como qualquerAWS IoT coisa. Para obter mais informações, consulte Criar um objeto no Guia doAWS IoT Core desenvolvedor. Adicione a chave privada, a chave pública, o certificado do dispositivo e o certificado CA raiz ao seu dispositivo cliente. A forma como você adiciona as informações depende do seu dispositivo e software.

Depois de configurar seu dispositivo, você pode usar o certificado e a cadeia de chaves públicas para se conectar ao dispositivo principal do Greengrass. Seu software é responsável por encontrar

os principais terminais do dispositivo. Você pode definir o endpoint manualmente para o dispositivo principal. Para obter mais informações, consulte Gerencie manualmente os endpoints.

Teste as comunicações entre dispositivos clientes

Os dispositivos cliente podem usar o AWS IoT Device SDK para descobrir, conectar e se comunicar com um dispositivo principal. Você pode usar o cliente de descoberta do Greengrass no AWS IoT Device SDK para usar a API de descoberta do Greengrass, que retorna informações sobre os dispositivos principais aos quais um dispositivo cliente pode se conectar. A resposta da API inclui endpoints do broker MQTT para conexão e certificados a serem usados para verificar a identidade de cada dispositivo principal. Em seguida, o dispositivo cliente pode testar cada endpoint até se conectar com êxito a um dispositivo principal.

Os dispositivos cliente podem descobrir somente os dispositivos principais aos quais você os associa. Antes de testar as comunicações entre um dispositivo cliente e um dispositivo principal, você deve associar o dispositivo cliente ao dispositivo principal. Para ter mais informações, consulte Associe dispositivos do cliente.

A API de descoberta do Greengrass retorna os endpoints do broker MQTT do dispositivo principal que você especifica. Você pode usar o componente detector de IP para gerenciar esses endpoints para você ou pode gerenciá-los manualmente para cada dispositivo principal. Para ter mais informações, consulte Gerencie os endpoints principais do dispositivo.

> **Note**
>
> Para usar a API de descoberta do Greengrass, um dispositivo cliente deve ter a `greengrass:Discover` permissão. Para ter mais informações, consulte AWS IoTPolítica mínima para dispositivos clientes.

O AWS IoT Device SDK está disponível em várias linguagens de programação. Para obter mais informações, consulte Dispositivos SDK AWS IoT no Guia do desenvolvedor do AWS IoT Core.

Tópicos

- Comunicações de teste (Python)
- Comunicações de teste (C++)
- Comunicações de teste (JavaScript)
- Comunicações de teste (Java)

Comunicações de teste (Python)

Nesta seção, você usa a amostra de descoberta do Greengrass na AWS IoT Device SDKv2 para Python para testar a comunicação entre um dispositivo cliente e um dispositivo principal.

> **Important**
>
> Para usar a AWS IoT Device SDK v2 para Python, um dispositivo deve executar o Python 3.6 ou posterior.

Para testar as comunicações (AWS IoT Device SDKv2 para Python)

1. Baixe e instale a AWS IoT Device SDKv2 para Python AWS IoT na coisa para se conectar como um dispositivo cliente.

   No dispositivo cliente, faça o seguinte:

   a. Clone o repositório AWS IoT Device SDK v2 for Python para baixá-lo.

   ```
   git clone https://github.com/aws/aws-iot-device-sdk-python-v2.git
   ```

   b. Instale a AWS IoT Device SDK v2 para Python.

   ```
   python3 -m pip install --user ./aws-iot-device-sdk-python-v2
   ```

2. Mude para a pasta de amostras na AWS IoT Device SDK v2 para Python.

   ```
   cd aws-iot-device-sdk-python-v2/samples
   ```

3. Execute o aplicativo de descoberta Greengrass de amostra. Esse aplicativo espera argumentos que especifiquem o nome do dispositivo cliente, o tópico e a mensagem do MQTT a serem usados e os certificados que autenticam e protegem a conexão. O exemplo a seguir envia uma mensagem Hello World para o `clients/`*`MyClientDevice1`*`/hello/world` tópico.

   - Substitua *`MyClientDevice1`* pelo nome do item do dispositivo cliente.
   - Substitua *`~/certs/ AmazonRoot CA1.pem`* pelo caminho para o certificado CA raiz da Amazon no dispositivo cliente.
   - Substitua *`~/certs/device.pem.crt`* pelo caminho para o certificado do dispositivo no dispositivo cliente.

- Substitua *~/certs/private.pem.key pelo caminho para o arquivo de chave* privada no dispositivo cliente.
- Substitua *us-east-1* pela AWS região em que seu dispositivo cliente e dispositivo principal operam.

```
python3 basic_discovery.py \\
  --thing_name MyClientDevice1 \\
  --topic 'clients/MyClientDevice1/hello/world' \\
  --message 'Hello World!' \\
  --ca_file ~/certs/AmazonRootCA1.pem \\
  --cert ~/certs/device.pem.crt \\
  --key ~/certs/private.pem.key \\
  --region us-east-1 \\
  --verbosity Warn
```

O aplicativo de amostra de descoberta envia a mensagem 10 vezes e se desconecta. Ele também se inscreve no mesmo tópico em que publica mensagens. Se a saída indicar que o aplicativo recebeu mensagens MQTT sobre o tópico, o dispositivo cliente poderá se comunicar com êxito com o dispositivo principal.

```
Performing greengrass discovery...
awsiot.greengrass_discovery.DiscoverResponse(gg_groups=[awsiot.greengrass_discovery.GGGroup
coreDevice-MyGreengrassCore',
 cores=[awsiot.greengrass_discovery.GGCore(thing_arn='arn:aws:iot:us-
east-1:123456789012:thing/MyGreengrassCore',
 connectivity=[awsiot.greengrass_discovery.ConnectivityInfo(id='203.0.113.0',
 host_address='203.0.113.0', metadata='', port=8883)])],
 certificate_authorities=['-----BEGIN CERTIFICATE-----\
MIICiT...EXAMPLE=\
-----END CERTIFICATE-----\
'])])
Trying core arn:aws:iot:us-east-1:123456789012:thing/MyGreengrassCore at host
 203.0.113.0 port 8883
Connected!
Published topic clients/MyClientDevice1/hello/world: {"message": "Hello World!",
 "sequence": 0}

Publish received on topic clients/MyClientDevice1/hello/world
b'{"message": "Hello World!", "sequence": 0}'
```

```
Published topic clients/MyClientDevice1/hello/world: {"message": "Hello World!",
 "sequence": 1}

Publish received on topic clients/MyClientDevice1/hello/world
b'{"message": "Hello World!", "sequence": 1}'

...

Published topic clients/MyClientDevice1/hello/world: {"message": "Hello World!",
 "sequence": 9}

Publish received on topic clients/MyClientDevice1/hello/world
b'{"message": "Hello World!", "sequence": 9}'
```

Se, em vez disso, o aplicativo gerar um erro, consulte Solução de problemas de descoberta do Greengrass.

Você também pode ver os registros do Greengrass no dispositivo principal para verificar se o dispositivo cliente se conecta e envia mensagens com sucesso. Para ter mais informações, consulte Monitore AWS IoT Greengrass os registros.

Comunicações de teste (C++)

Nesta seção, você usa a amostra de descoberta do Greengrass na AWS IoT Device SDKv2 para C++ para testar as comunicações entre um dispositivo cliente e um dispositivo principal.

Para criar a AWS IoT Device SDK v2 para C++, um dispositivo deve ter as seguintes ferramentas:

- C++ 11 ou posterior
- CMake 3.1 ou posterior
- Um dos seguintes compiladores:
  - GCC 4.8 ou posterior
  - Clang 3.9 ou posterior
  - MSVC 2015 ou posterior

Para testar as comunicações (AWS IoT Device SDKv2 para C++)

1. Baixe e crie a AWS IoT Device SDKversão 2 para C++ para AWS IoT conectar como um dispositivo cliente.

No dispositivo cliente, faça o seguinte:

a. Crie uma pasta para o espaço de trabalho AWS IoT Device SDK v2 for C++ e altere para ela.

```
cd
mkdir iot-device-sdk-cpp
cd iot-device-sdk-cpp
```

b. Clone o repositório AWS IoT Device SDK v2 para C++ para baixá-lo. O `--recursive` sinalizador especifica o download de submódulos.

```
git clone --recursive https://github.com/aws/aws-iot-device-sdk-cpp-v2.git
```

c. Crie uma pasta para a saída de compilação AWS IoT Device SDK v2 for C++ e altere-a para ela.

```
mkdir aws-iot-device-sdk-cpp-v2-build
cd aws-iot-device-sdk-cpp-v2-build
```

d. Crie a AWS IoT Device SDK v2 para C++.

```
cmake -DCMAKE_INSTALL_PREFIX="~/iot-device-sdk-cpp" -
DCMAKE_BUILD_TYPE="Release" ../aws-iot-device-sdk-cpp-v2
cmake --build . --target install
```

2. Crie o aplicativo de amostra Greengrass discovery na AWS IoT Device SDK v2 para C++. Faça o seguinte:

a. Mude para a pasta de amostra do Greengrass discovery na AWS IoT Device SDK v2 para C ++.

```
cd ../aws-iot-device-sdk-cpp-v2/samples/greengrass/basic_discovery
```

b. Crie uma pasta para a saída de compilação de amostra do Greengrass discovery e altere-a para ela.

```
mkdir build
cd build
```

c. Crie o aplicativo de amostra de descoberta do Greengrass.

```
cmake -DCMAKE_PREFIX_PATH="~/iot-device-sdk-cpp" -
DCMAKE_BUILD_TYPE="Release" ..
cmake --build . --config "Release"
```

3. Execute o aplicativo de descoberta Greengrass de amostra. Esse aplicativo espera argumentos que especifiquem o nome do dispositivo cliente, o tópico do MQTT a ser usado e os certificados que autenticam e protegem a conexão. O exemplo a seguir se inscreve no `clients/MyClientDevice1/hello/world` tópico e publica uma mensagem que você insere na linha de comando para o mesmo tópico.

   - Substitua *MyClientDevice1* pelo nome do item do dispositivo cliente.
   - Substitua *~/certs/ AmazonRoot CA1.pem* pelo caminho para o certificado CA raiz da Amazon no dispositivo cliente.
   - Substitua *~/certs/device.pem.crt* pelo caminho para o certificado do dispositivo no dispositivo cliente.
   - Substitua *~/certs/private.pem.key pelo caminho para o arquivo de chave* privada no dispositivo cliente.
   - Substitua *us-east-1* pela AWS região em que seu dispositivo cliente e dispositivo principal operam.

```
./basic-discovery \
  --thing_name MyClientDevice1 \
  --topic 'clients/MyClientDevice1/hello/world' \
  --ca_file ~/certs/AmazonRootCA1.pem \
  --cert ~/certs/device.pem.crt \
  --key ~/certs/private.pem.key \
  --region us-east-1
```

   O aplicativo de amostra de descoberta se inscreve no tópico e solicita que você insira uma mensagem para publicar.

```
Connecting to group greengrassV2-coreDevice-MyGreengrassCore with thing arn
 arn:aws:iot:us-east-1:123456789012:thing/MyGreengrassCore, using endpoint
 203.0.113.0:8883
Connected to group greengrassV2-coreDevice-MyGreengrassCore, using connection to
 203.0.113.0:8883
```

```
Successfully subscribed to clients/MyClientDevice1/hello/world
Enter the message you want to publish to topic clients/MyClientDevice1/hello/world
 and press enter. Enter 'exit' to exit this program.
```

Se, em vez disso, o aplicativo gerar um erro, consulte Solução de problemas de descoberta do Greengrass.

4. Insira uma mensagem, como **`Hello World!`**.

```
Enter the message you want to publish to topic clients/MyClientDevice1/hello/world
 and press enter. Enter 'exit' to exit this program.
Hello World!
```

Se a saída indicar que o aplicativo recebeu a mensagem MQTT sobre o tópico, o dispositivo cliente poderá se comunicar com êxito com o dispositivo principal.

```
Operation on packetId 2 Succeeded
Publish received on topic clients/MyClientDevice1/hello/world
Message:
Hello World!
```

Você também pode ver os registros do Greengrass no dispositivo principal para verificar se o dispositivo cliente se conecta e envia mensagens com sucesso. Para ter mais informações, consulte Monitore AWS IoT Greengrass os registros.

Comunicações de teste (JavaScript)

Nesta seção, você usa a amostra de descoberta do Greengrass na AWS IoT Device SDKv2 JavaScript para testar as comunicações entre um dispositivo cliente e um dispositivo principal.

> **Important**
>
> Para usar a AWS IoT Device SDK v2 JavaScript, um dispositivo deve executar o Node v10.0 ou posterior.

Para testar as comunicações (AWS IoT Device SDKv2 para JavaScript)

1. Baixe e instale a AWS IoT Device SDKv2 para JavaScript que a AWS IoT coisa se conecte como um dispositivo cliente.

   No dispositivo cliente, faça o seguinte:

   a. Clone a AWS IoT Device SDK v2 do JavaScript repositório para baixá-la.

   ```
   git clone https://github.com/aws/aws-iot-device-sdk-js-v2.git
   ```

   b. Instale a AWS IoT Device SDK v2 para JavaScript.

   ```
   cd aws-iot-device-sdk-js-v2
   npm install
   ```

2. Vá para a pasta de amostra do Greengrass discovery na AWS IoT Device SDK v2 para. JavaScript

   ```
   cd samples/node/basic_discovery
   ```

3. Instale o aplicativo de amostra Greengrass discovery.

   ```
   npm install
   ```

4. Execute o aplicativo de descoberta Greengrass de amostra. Esse aplicativo espera argumentos que especifiquem o nome do dispositivo cliente, o tópico e a mensagem do MQTT a serem usados e os certificados que autenticam e protegem a conexão. O exemplo a seguir envia uma mensagem Hello World para o `clients/`*`MyClientDevice1`*`/hello/world` tópico.

   - Substitua *`MyClientDevice1`* pelo nome do item do dispositivo cliente.
   - Substitua *`~/certs/ AmazonRoot CA1.pem`* pelo caminho para o certificado CA raiz da Amazon no dispositivo cliente.
   - Substitua *`~/certs/device.pem.crt`* pelo caminho para o certificado do dispositivo no dispositivo cliente.
   - Substitua *`~/certs/private.pem.key pelo caminho para o arquivo de chave`* privada no dispositivo cliente.
   - Substitua *`us-east-1`* pela AWS região em que seu dispositivo cliente e dispositivo principal operam.

```
node dist/index.js \
  --thing_name MyClientDevice1 \
  --topic 'clients/MyClientDevice1/hello/world' \
  --message 'Hello World!' \
  --ca_file ~/certs/AmazonRootCA1.pem \
  --cert ~/certs/device.pem.crt \
  --key ~/certs/private.pem.key \
  --region us-east-1 \
  --verbose warn
```

O aplicativo de amostra de descoberta envia a mensagem 10 vezes e se desconecta. Ele também se inscreve no mesmo tópico em que publica mensagens. Se a saída indicar que o aplicativo recebeu mensagens MQTT sobre o tópico, o dispositivo cliente poderá se comunicar com êxito com o dispositivo principal.

```
Discovery Response:
{"gg_groups":[{"gg_group_id":"greengrassV2-coreDevice-
MyGreengrassCore","cores":[{"thing_arn":"arn:aws:iot:us-
east-1:123456789012:thing/MyGreengrassCore","connectivity":
[{"id":"203.0.113.0","host_address":"203.0.113.0","port":8883,"metadata":""}]}],"certificat
["-----BEGIN CERTIFICATE-----\nMIICiT...EXAMPLE=\n-----END CERTIFICATE-----\n"]}]}
Trying
 endpoint={"id":"203.0.113.0","host_address":"203.0.113.0","port":8883,"metadata":""}
[WARN] [2021-06-12T00:46:45Z] [00007f90c0e8d700] [socket] - id=0x7f90b8018710
 fd=26: setsockopt() for NO_SIGNAL failed with errno 92. If you are having SIGPIPE
 signals thrown, you may want to install a signal trap in your application layer.
Connected to
 endpoint={"id":"203.0.113.0","host_address":"203.0.113.0","port":8883,"metadata":""}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":1}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":2}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":3}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":4}
```

```
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":5}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":6}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":7}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":8}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":9}
Publish received. topic:"clients/MyClientDevice1/hello/world" dup:false qos:0
 retain:false
{"message":"Hello World!","sequence":10}
Complete!
```

Se, em vez disso, o aplicativo gerar um erro, consulte Solução de problemas de descoberta do Greengrass.

Você também pode ver os registros do Greengrass no dispositivo principal para verificar se o dispositivo cliente se conecta e envia mensagens com sucesso. Para ter mais informações, consulte Monitore AWS IoT Greengrass os registros.

Comunicações de teste (Java)

Nesta seção, você usa a amostra de descoberta do Greengrass na AWS IoT Device SDKv2 para Java para testar as comunicações entre um dispositivo cliente e um dispositivo principal.

Important

Para criar a AWS IoT Device SDK v2 para Java, um dispositivo deve ter as seguintes ferramentas:

- Java 8 ou posterior, `JAVA_HOME` apontando para a pasta Java.
- Apache Maven

Para testar as comunicações (AWS IoT Device SDKv2 para Java)

1. Baixe e crie a AWS IoT Device SDKversão 2 para Java para AWS IoT conectar-se como um dispositivo cliente.

   No dispositivo cliente, faça o seguinte:

   a. Clone o repositório AWS IoT Device SDK v2 for Java para baixá-lo.

   ```
   git clone https://github.com/aws/aws-iot-device-sdk-java-v2.git
   ```

   b. Mude para a pasta AWS IoT Device SDK v2 for Java.

   c. Crie a AWS IoT Device SDK v2 para Java.

   ```
   cd aws-iot-device-sdk-java-v2
   mvn versions:use-latest-versions -Dincludes="software.amazon.awssdk.crt*"
   mvn clean install
   ```

2. Execute o aplicativo de descoberta Greengrass de amostra. Esse aplicativo espera argumentos que especifiquem o nome do dispositivo cliente, o tópico do MQTT a ser usado e os certificados que autenticam e protegem a conexão. O exemplo a seguir se inscreve no `clients/MyClientDevice1/hello/world` tópico e publica uma mensagem que você insere na linha de comando para o mesmo tópico.

   - Substitua as duas instâncias de *MyClientDevice1* pelo nome da coisa do dispositivo cliente.
   - Substitua *$HOME/certs/ AmazonRoot CA1.pem* pelo caminho para o certificado CA raiz da Amazon no dispositivo cliente.
   - Substitua *$HOME/certs/device.pem.crt* pelo caminho para o certificado do dispositivo no dispositivo cliente.
   - Substitua *$HOME/certs/private.pem.key pelo caminho para o arquivo de chave* privada no dispositivo cliente.
   - Substitua *us-east-1* pelo Região da AWS local em que seu dispositivo cliente e dispositivo principal operam.

   ```
   DISCOVERY_SAMPLE_ARGS="--thing_name MyClientDevice1 \
     --topic 'clients/MyClientDevice1/hello/world' \
     --ca_file $HOME/certs/AmazonRootCA1.pem \
   ```

```
  --cert $HOME/certs/device.pem.crt \
  --key $HOME/certs/private.pem.key \
  --region us-east-1"

mvn exec:java -pl samples/Greengrass \
  -Dexec.mainClass=greengrass.BasicDiscovery \
  -Dexec.args="$DISCOVERY_SAMPLE_ARGS"
```

O aplicativo de amostra de descoberta se inscreve no tópico e solicita que você insira uma mensagem para publicar.

```
Connecting to group ID greengrassV2-coreDevice-MyGreengrassCore, with thing
 arn arn:aws:iot:us-east-1:123456789012:thing/MyGreengrassCore, using endpoint
 203.0.113.0:8883
Started a clean session
Enter the message you want to publish to topic clients/MyClientDevice1/hello/world
 and press Enter. Type 'exit' or 'quit' to exit this program:
```

Se, em vez disso, o aplicativo gerar um erro, consulte Solução de problemas de descoberta do Greengrass.

3. Insira uma mensagem, como **`Hello World!`**.

```
Enter the message you want to publish to topic clients/MyClientDevice1/hello/world
 and press Enter. Type 'exit' or 'quit' to exit this program:
Hello World!
```

Se a saída indicar que o aplicativo recebeu a mensagem MQTT sobre o tópico, o dispositivo cliente poderá se comunicar com êxito com o dispositivo principal.

```
Message received on topic clients/MyClientDevice1/hello/world: Hello World!
```

Você também pode ver os registros do Greengrass no dispositivo principal para verificar se o dispositivo cliente se conecta e envia mensagens com sucesso. Para ter mais informações, consulte Monitore AWS IoT Greengrass os registros.

API RESTful de descoberta do Greengrass

AWS IoT Greengrassfornece a operação de `Discover` API que os dispositivos cliente podem usar para identificar os principais dispositivos do Greengrass aos quais eles podem se conectar. Os dispositivos cliente usam essa operação de plano de dados para recuperar as informações necessárias para se conectar aos dispositivos principais do Greengrass, onde você os BatchAssociateClientDeviceWithCoreDeviceassocia à operação da API. Quando um dispositivo cliente fica on-line, ele pode se conectar ao serviço de AWS IoT Greengrass nuvem e usar a API de descoberta para encontrar:

- O endereço IP e a porta de cada dispositivo principal do Greengrass associado.
- O certificado CA do dispositivo principal, que os dispositivos clientes podem usar para autenticar o dispositivo principal do Greengrass.

> **Note**
>
> Os dispositivos cliente também podem usar o cliente de descoberta no AWS IoT Device SDK para descobrir informações de conectividade dos dispositivos principais do Greengrass. O cliente de descoberta usa a API de descoberta. Para ver mais informações, consulte:
>
> - Teste as comunicações entre dispositivos clientes
> - API RESTful do Greengrass Discovery no Guia do AWS IoT Greengrass Version 1 desenvolvedor.

Para usar essa operação de API, envie solicitações HTTP para a API de descoberta no endpoint do plano de dados do Greengrass. Esse endpoint da API tem o seguinte formato.

```
https://greengrass-ats.iot.region.amazonaws.com:port/greengrass/discover/thing/thing-name
```

Para obter uma lista de endpoints compatíveis Regiões da AWS e de extremidade para a API de AWS IoT Greengrass descoberta, consulte AWS IoT Greengrass V2endpoints e cotas no. Referência geral da AWS Essa operação de API está disponível somente no endpoint do plano de dados do Greengrass. O endpoint do plano de controle que você usa para gerenciar componentes e implantações é diferente do endpoint do plano de dados.

> **Note**
>
> A API de descoberta é a mesma para AWS IoT Greengrass V1 AWS IoT Greengrass V2 e. Se você tiver dispositivos cliente que se conectam a um AWS IoT Greengrass V1 núcleo, você pode conectá-los aos dispositivos AWS IoT Greengrass V2 principais sem alterar o código nos dispositivos cliente. Para obter mais informações, consulte a API RESTful do Greengrass Discovery no Guia do AWS IoT Greengrass Version 1 desenvolvedor.

Tópicos

Autenticação e autorização de descoberta

Para usar a API de descoberta para recuperar informações de conectividade, um dispositivo cliente deve usar a autenticação mútua TLS com um certificado de cliente X.509 para se autenticar. Para obter mais informações, consulte certificados de cliente X.509 no Guia do AWS IoT Coredesenvolvedor.

Um dispositivo cliente também deve ter permissão para realizar a `greengrass:Discover` ação. O exemplo de AWS IoT política a seguir permite que uma AWS IoT coisa chamada `MyClientDevice1` funcione `Discover` por si mesma.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "greengrass:Discover",
      "Resource": [
        "arn:aws:iot:us-west-2:123456789012:thing/MyClientDevice1"
      ]
    }
  ]
}
```

> **Important**
>
> As variáveis de política Thing (`iot:Connection.Thing.*`) não são suportadas em AWS IoT políticas para dispositivos principais ou operações de plano de dados do Greengrass. Em vez disso, você pode usar um caractere curinga que corresponda a vários dispositivos com nomes semelhantes. Por exemplo, você pode especificar `MyGreengrassDevice*` para corresponder `MyGreengrassDevice1MyGreengrassDevice2`, e assim por diante.

Para obter mais informações, consulte AWS IoT Coreas políticas no Guia do AWS IoT Core desenvolvedor.

Solicitação

A solicitação contém os cabeçalhos HTTP padrão e é enviada para o endpoint de descoberta do Greengrass, conforme mostrado nos exemplos a seguir.

O número da porta depende se o dispositivo principal está configurado para enviar tráfego HTTPS pela porta 8443 ou pela porta 443. Para ter mais informações, consulte the section called “Conectar-se à porta 443 ou por meio de um proxy de rede”.

> **Note**
>
> Esses exemplos usam o endpoint Amazon Trust Services (ATS), que funciona com os certificados de CA raiz ATS recomendados. Os endpoints devem corresponder ao tipo de certificado da CA raiz.

Porta 8443

```
HTTP GET https://greengrass-ats.iot.region.amazonaws.com:8443/greengrass/discover/
thing/thing-name
```

Porta 443

```
HTTP GET https://greengrass-ats.iot.region.amazonaws.com:443/greengrass/discover/
thing/thing-name
```

> **Note**
>
> Os clientes que se conectam na porta 443 devem implementar a extensão TLS do Application Layer Protocol Negotiation (ALPN) e passar `x-amzn-http-ca` como o `ProtocolName` no `ProtocolNameList`. Para obter mais informações, consulte Protocolos no Guia do desenvolvedor do AWS IoT.

Resposta

Em caso de sucesso, o cabeçalho da resposta inclui o código de status HTTP 200 e o corpo da resposta contém o documento de resposta de descoberta.

> **Note**
>
> Como AWS IoT Greengrass V2 usa a mesma API de descoberta queAWS IoT Greengrass V1, a resposta organiza as informações de acordo com AWS IoT Greengrass V1 conceitos, como grupos do Greengrass. A resposta contém uma lista de grupos do Greengrass. EmAWS IoT Greengrass V2, cada dispositivo principal está em seu próprio grupo, onde o grupo contém somente esse dispositivo principal e suas informações de conectividade.

Exemplos de documentos de resposta de descoberta

O documento a seguir mostra a resposta de um dispositivo cliente associado a um dispositivo principal do Greengrass. O dispositivo principal tem um endpoint e um certificado CA.

```
{
  "GGGroups": [
    {
      "GGGroupId": "greengrassV2-coreDevice-core-device-01-thing-name",
      "Cores": [
        {
          "thingArn": "core-device-01-thing-arn",
          "Connectivity": [
            {
              "id": "core-device-01-connection-id",
              "hostAddress": "core-device-01-address",
              "portNumber": core-device-01-port,
              "metadata": "core-device-01-description"
```

```
            }
          ]
        }
      ],
      "CAs": [
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----"
      ]
    }
  ]
}
```

O documento a seguir mostra a resposta de um dispositivo cliente associado a dois dispositivos principais. Os dispositivos principais têm vários endpoints e vários certificados CA de grupo.

```
{
  "GGGroups": [
    {
      "GGGroupId": "greengrassV2-coreDevice-core-device-01-thing-name",
      "Cores": [
        {
          "thingArn": "core-device-01-thing-arn",
          "Connectivity": [
            {
              "id": "core-device-01-connection-id",
              "hostAddress": "core-device-01-address",
              "portNumber": core-device-01-port,
              "metadata": "core-device-01-connection-1-description"
            },
            {
              "id": "core-device-01-connection-id-2",
              "hostAddress": "core-device-01-address-2",
              "portNumber": core-device-01-port-2,
              "metadata": "core-device-01-connection-2-description"
            }
          ]
        }
      ],
      "CAs": [
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----"
      ]
    },
```

```
    {
      "GGGroupId": "greengrassV2-coreDevice-core-device-02-thing-name",
      "Cores": [
        {
          "thingArn":"core-device-02-thing-arn",
          "Connectivity" : [
            {
              "id": "core-device-02-connection-id",
              "hostAddress": "core-device-02-address",
              "portNumber": core-device-02-port,
              "metadata": "core-device-02-connection-1-description"
            }
          ]
        }
      ],
      "CAs": [
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----"
      ]
    }
  ]
}
```

Teste a API de descoberta com cURL

Se você cURL instalou, pode testar a API de descoberta. O exemplo a seguir especifica os certificados de um dispositivo cliente para autenticar uma solicitação no endpoint da API de descoberta do Greengrass.

```
curl -i \
  --cert 1a23bc4d56.cert.pem \
  --key 1a23bc4d56.private.key \
  https://greengrass-ats.iot.us-west-2.amazonaws.com:8443/greengrass/discover/
thing/MyClientDevice1
```

> **Note**
>
> O `-i` argumento especifica a saída de cabeçalhos de resposta HTTP. Você pode usar essa opção para ajudar a identificar erros.

Se a solicitação for bem-sucedida, esse comando gerará uma resposta semelhante ao exemplo a seguir.

```
{
  "GGGroups": [
    {
      "GGGroupId": "greengrassV2-coreDevice-MyGreengrassCore",
      "Cores": [
        {
          "thingArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
          "Connectivity": [
            {
              "Id": "AUTOIP_192.168.1.4_1",
              "HostAddress": "192.168.1.5",
              "PortNumber": 8883,
              "Metadata": ""
            }
          ]
        }
      ],
      "CAs": [
        "-----BEGIN CERTIFICATE-----\ncert-contents\n-----END CERTIFICATE-----\n"
      ]
    }
  ]
}
```

Se o comando gerar um erro, consulte Solução de problemas de descoberta do Greengrass.

Retransmita mensagens MQTT entre dispositivos clientes e AWS IoT Core

Você pode retransmitir mensagens MQTT e outros dados entre dispositivos clientes e. AWS IoT Core Os dispositivos cliente se conectam ao componente do agente MQTT que é executado no dispositivo principal. Por padrão, os dispositivos principais não retransmitem mensagens ou dados do MQTT entre dispositivos clientes e. AWS IoT Core Por padrão, os dispositivos cliente podem se comunicar somente entre si pelo MQTT.

Para retransmitir mensagens MQTT entre dispositivos cliente eAWS IoT Core, configure o componente de ponte MQTT para fazer o seguinte:

- Retransmita mensagens dos dispositivos do cliente para o. AWS IoT Core
- Retransmita mensagens AWS IoT Core para os dispositivos do cliente.

> **Note**
>
> A ponte MQTT usa QoS 1 para publicar e assinarAWS IoT Core, mesmo quando um dispositivo cliente usa QoS 0 para publicar e assinar o broker MQTT local. Como resultado, você pode observar latência adicional ao retransmitir mensagens MQTT de dispositivos clientes no agente MQTT local para o. AWS IoT Core Para obter mais informações sobre a configuração do MQTT nos dispositivos principais, consulteDefina os tempos limite do MQTT e as configurações de cache.

Tópicos

Configurar e implantar o componente de ponte MQTT

O componente de ponte MQTT consome uma lista de mapeamentos de tópicos, cada um especificando a origem e o destino da mensagem. Para retransmitir mensagens entre dispositivos cliente eAWS IoT Core, implante o componente de ponte MQTT e especifique cada tópico de origem e destino na configuração do componente.

Para implantar o componente de ponte MQTT em um dispositivo principal ou grupo de dispositivos principais, crie uma implantação que inclua o `aws.greengrass.clientdevices.mqtt.Bridge` componente. Especifique os mapeamentos de tópicos,`mqttTopicMapping`, na configuração do componente de ponte MQTT na implantação.

O exemplo a seguir define uma implantação que configura o componente de ponte MQTT para retransmitir mensagens sobre tópicos que correspondem ao filtro de `clients/+/hello/world` tópicos dos dispositivos do cliente para o. AWS IoT Core A atualização da `merge` configuração requer um objeto JSON serializado. Para ter mais informações, consulte Atualizar configurações de componentes.

Console

```
{
  "mqttTopicMapping": {
    "HelloWorldIotCore": {
      "topic": "clients/+/hello/world",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

AWS CLI

```
{
  "components": {
    "aws.greengrass.clientdevices.mqtt.Bridge": {
      "version": "2.0.0",
      "configurationUpdate": {
        "merge": "{\"mqttTopicMapping\":{\"HelloWorldIotCore\":{\"topic\":\"clients/
+/hello/world\",\"source\":\"LocalMqtt\",\"target\":\"IotCore\"}}}"
      }
    }
    ...
  }
}
```

Retransmitir mensagens MQTT

Para retransmitir mensagens MQTT entre dispositivos clientesAWS IoT Core, configure e implante o componente MQTT Bridge e especifique os tópicos a serem retransmitidos.

Example Exemplo: retransmitir mensagens sobre um tópico dos dispositivos do cliente para AWS IoT Core

A configuração do componente de ponte MQTT a seguir especifica a retransmissão de mensagens em tópicos que correspondem ao filtro de `clients/+/hello/world/event` tópicos dos dispositivos cliente para o. AWS IoT Core

```
{
  "mqttTopicMapping": {
```

```
    "HelloWorldEvent": {
      "topic": "clients/+/hello/world/event",
      "source": "LocalMqtt",
      "target": "IotCore"
    }
  }
}
```

Example Exemplo: retransmitir mensagens sobre um tópico AWS IoT Core para dispositivos clientes

A configuração do componente de ponte MQTT a seguir especifica a retransmissão de mensagens em tópicos que correspondem ao filtro de `clients/+/hello/world/event/response` tópicos dos dispositivos clientesAWS IoT Core.

```
{
  "mqttTopicMapping": {
    "HelloWorldEventConfirmation": {
      "topic": "clients/+/hello/world/event/response",
      "source": "IotCore",
      "target": "LocalMqtt"
    }
  }
}
```

Interaja com dispositivos clientes em componentes

Você pode desenvolver componentes personalizados do Greengrass que interagem com dispositivos clientes conectados a um dispositivo principal. Por exemplo, você pode desenvolver componentes que façam o seguinte:

- Atue nas mensagens MQTT dos dispositivos do cliente e envie dados para os Nuvem AWS destinos.
- Envie mensagens MQTT aos dispositivos do cliente para iniciar ações.

Os dispositivos cliente se conectam e se comunicam com um dispositivo principal por meio do componente intermediário MQTT que é executado no dispositivo principal. Por padrão, os dispositivos cliente só podem se comunicar uns com os outros pelo MQTT, e os componentes do Greengrass não podem receber essas mensagens do MQTT nem enviar mensagens aos dispositivos do cliente.

Os componentes do Greengrass usam a interface local de publicação/assinatura para se comunicarem em um dispositivo principal. Para se comunicar com dispositivos clientes nos componentes do Greengrass, configure o componente de ponte MQTT para fazer o seguinte:

- Retransmita mensagens MQTT dos dispositivos do cliente para publicação/assinatura local.
- Retransmita mensagens MQTT da publicação/assinatura local para dispositivos clientes.

Você também pode interagir com as sombras do dispositivo cliente nos componentes do Greengrass. Para ter mais informações, consulte Interaja e sincronize as sombras do dispositivo cliente.

Tópicos

Configurar e implantar o componente de ponte MQTT

O componente de ponte MQTT consome uma lista de mapeamentos de tópicos, cada um especificando a origem e o destino da mensagem. Para se comunicar com dispositivos clientes, implante o componente de ponte MQTT e especifique cada tópico de origem e destino na configuração do componente.

Para implantar o componente de ponte MQTT em um dispositivo principal ou grupo de dispositivos principais, crie uma implantação que inclua o `aws.greengrass.clientdevices.mqtt.Bridge` componente. Especifique os mapeamentos de tópicos,`mqttTopicMapping`, na configuração do componente de ponte MQTT na implantação.

O exemplo a seguir define uma implantação que configura o componente de ponte MQTT para retransmitir o `clients/MyClientDevice1/hello/world` tópico dos dispositivos do cliente para o agente local de publicação/assinatura. A atualização da `merge` configuração requer um objeto JSON serializado. Para ter mais informações, consulte Atualizar configurações de componentes.

Console

```
{
  "mqttTopicMapping": {
    "HelloWorldPubsub": {
```

```
      "topic": "clients/MyClientDevice1/hello/world",
      "source": "LocalMqtt",
      "target": "Pubsub"
    }
  }
}
```

AWS CLI

```
{
  "components": {
    "aws.greengrass.clientdevices.mqtt.Bridge": {
      "version": "2.0.0",
      "configurationUpdate": {
        "merge": "\"mqttTopicMapping\":{\"HelloWorldPubsub\":{\"topic\":\"clients/
MyClientDevice1/hello/world\",\"source\":\"LocalMqtt\",\"target\":\"Pubsub\"}}}"
      }
    }
    ...
  }
}
```

Você pode usar curingas de tópicos do MQTT para retransmitir mensagens sobre tópicos que correspondam a um filtro de tópicos. Se você usar o MQTT bridge v2.2.0 ou posterior, poderá usar curingas de tópicos do MQTT nos filtros de tópicos quando o agente de origem for publicar/assinar local. Para obter mais informações, consulte Configuração do componente de ponte MQTT.

Receba mensagens MQTT de dispositivos clientes

Você pode se inscrever nos tópicos locais de publicação/assinatura que você configura para que o componente de ponte MQTT receba mensagens de dispositivos clientes.

Para receber mensagens MQTT de dispositivos clientes em componentes personalizados

1. Configure e implante o componente de ponte do MQTT para retransmitir mensagens de um tópico do MQTT em que os dispositivos do cliente publicam em um tópico local de publicação/assinatura.
2. Use a interface IPC local de publicação/assinatura para assinar o tópico em que a ponte MQTT retransmite mensagens. Para obter mais informações, consulte SubscribeToTopic e Publique/assine mensagens locais.

O tutorial Conectar e testar dispositivos cliente inclui uma seção em que você desenvolve um componente que assina mensagens de um dispositivo cliente. Para ter mais informações, consulte Etapa 4: desenvolver um componente que se comunique com os dispositivos do cliente.

Envie mensagens MQTT para dispositivos clientes

Você pode publicar nos tópicos locais de publicação/assinatura que você configura para o componente de ponte MQTT para enviar mensagens aos dispositivos clientes.

Para publicar mensagens MQTT em dispositivos clientes em componentes personalizados

1. Configure e implante o componente de ponte do MQTT para retransmitir mensagens de um tópico local de publicação/assinatura para um tópico do MQTT no qual os dispositivos do cliente se inscrevem.
2. Use a interface IPC local de publicação/assinatura para publicar no tópico em que a ponte MQTT retransmite mensagens. Para ter mais informações, consulte Publique/assine mensagens locais e PublishToTopic.

Interaja e sincronize as sombras do dispositivo cliente

Você pode usar o componente do gerenciador de sombras para gerenciar sombras locais, incluindo sombras do dispositivo cliente. Você pode usar o gerenciador de sombras para fazer o seguinte:

- Interaja com as sombras do dispositivo cliente nos componentes do Greengrass.
- Sincronize as sombras do dispositivo cliente com o. AWS IoT Core

> **Note**
>
> Por padrão, o componente gerenciador de sombras não sincroniza sombras comAWS IoT Core. Você deve configurar o componente do gerenciador de sombras para especificar quais sombras do dispositivo cliente devem ser sincronizadas.

Tópicos

- Pré-requisitos
- Ative o Shadow Manager para se comunicar com os dispositivos do cliente
- Interaja com as sombras do dispositivo cliente nos componentes

- Sincronize as sombras do dispositivo cliente com AWS IoT Core

Pré-requisitos

Para interagir com as sombras do dispositivo cliente e sincronizar as sombras do dispositivo cliente comAWS IoT Core, um dispositivo principal deve atender aos seguintes requisitos:

- O dispositivo principal deve executar os seguintes componentes, além dos componentes do Greengrass para suporte ao dispositivo cliente:
  - Greengrass nucleus v2.6.0 ou posterior
  - Shadow manager v2.2.0 ou posterior
  - Ponte MQTT v2.2.0 ou posterior
- O componente de autenticação do dispositivo cliente deve ser configurado para permitir que os dispositivos do cliente se comuniquem sobre tópicos paralelos do dispositivo.

Ative o Shadow Manager para se comunicar com os dispositivos do cliente

Por padrão, o componente do gerenciador de sombras não gerencia as sombras do dispositivo cliente. Para ativar esse recurso, você deve retransmitir mensagens MQTT entre dispositivos cliente e o componente do gerenciador de sombras. Os dispositivos clientes usam mensagens MQTT para receber e enviar atualizações de sombra do dispositivo. O componente shadow manager assina a interface local de publicação/assinatura do Greengrass, para que você possa configurar o componente bridge do MQTT para retransmitir mensagens do MQTT sobre tópicos paralelos do dispositivo.

O componente de ponte MQTT consome uma lista de mapeamentos de tópicos, cada um especificando a origem e o destino da mensagem. Para permitir que o componente do gerenciador de sombras gerencie as sombras do dispositivo cliente, implante o componente de ponte MQTT e especifique os tópicos de sombra para as sombras do dispositivo cliente. Você deve configurar a ponte para retransmitir mensagens em ambas as direções entre o MQTT local e a publicação/assinatura local.

Para implantar o componente de ponte MQTT em um dispositivo principal ou grupo de dispositivos principais, crie uma implantação que inclua o `aws.greengrass.clientdevices.mqtt.Bridge` componente. Especifique os mapeamentos de tópicos,`mqttTopicMapping`, na configuração do componente de ponte MQTT na implantação.

Use os exemplos a seguir para configurar o componente de ponte MQTT para permitir a comunicação entre dispositivos cliente e o componente do gerenciador de sombras.

> **Note**
>
> Você pode usar esses exemplos de configuração no AWS IoT Greengrass console. Se você usa a AWS IoT Greengrass API, a atualização de `merge` configuração requer um objeto JSON serializado, portanto, você deve serializar os seguintes objetos JSON em strings. Para ter mais informações, consulte Atualizar configurações de componentes.

Example Exemplo: gerenciar todas as sombras do dispositivo cliente

O exemplo de configuração da ponte MQTT a seguir permite que o gerenciador de sombras gerencie todas as sombras de todos os dispositivos cliente.

```
{
  "mqttTopicMapping": {
    "ShadowsLocalMqttToPubsub": {
      "topic": "$aws/things/+/shadow/#",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ShadowsPubsubToLocalMqtt": {
      "topic": "$aws/things/+/shadow/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    }
  }
}
```

Example Exemplo: gerenciar sombras para um dispositivo cliente

O exemplo de configuração da ponte MQTT a seguir permite que o gerenciador de sombras gerencie todas as sombras de um dispositivo cliente chamado. `MyClientDevice`

```
{
  "mqttTopicMapping": {
    "ShadowsLocalMqttToPubsub": {
      "topic": "$aws/things/MyClientDevice/shadow/#",
      "source": "LocalMqtt",
      "target": "Pubsub"
```

```
    },
    "ShadowsPubsubToLocalMqtt": {
      "topic": "$aws/things/MyClientDevice/shadow/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    }
  }
}
```

Example Exemplo: Gerenciar uma sombra nomeada para todos os dispositivos cliente

O exemplo de configuração da ponte MQTT a seguir permite que o gerenciador de sombras gerencie uma sombra nomeada `DeviceConfiguration` para todos os dispositivos cliente.

```
{
  "mqttTopicMapping": {
    "ShadowsLocalMqttToPubsub": {
      "topic": "$aws/things/+/shadow/name/DeviceConfiguration/#",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "ShadowsPubsubToLocalMqtt": {
      "topic": "$aws/things/+/shadow/name/DeviceConfiguration/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    }
  }
}
```

Example Exemplo: gerencie as sombras sem nome de todos os dispositivos do cliente

O exemplo de configuração de ponte MQTT a seguir permite que o gerenciador de sombras gerencie sombras sem nome, mas não sombras nomeadas, para todos os dispositivos clientes.

```
{
  "mqttTopicMapping": {
    "DeleteShadowLocalMqttToPubsub": {
      "topic": "$aws/things/+/shadow/delete",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "DeleteShadowPubsubToLocalMqtt": {
      "topic": "$aws/things/+/shadow/delete/#",
```

```
      "source": "Pubsub",
      "target": "LocalMqtt"
    },
    "GetShadowLocalMqttToPubsub": {
      "topic": "$aws/things/+/shadow/get",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "GetShadowPubsubToLocalMqtt": {
      "topic": "$aws/things/+/shadow/get/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    },
    "UpdateShadowLocalMqttToPubsub": {
      "topic": "$aws/things/+/shadow/update",
      "source": "LocalMqtt",
      "target": "Pubsub"
    },
    "UpdateShadowPubsubToLocalMqtt": {
      "topic": "$aws/things/+/shadow/update/#",
      "source": "Pubsub",
      "target": "LocalMqtt"
    }
  }
}
```

Interaja com as sombras do dispositivo cliente nos componentes

Você pode desenvolver componentes personalizados que usam o serviço paralelo local para ler e modificar os documentos paralelos locais dos dispositivos clientes. Para ter mais informações, consulte Interaja com sombras em componentes.

Sincronize as sombras do dispositivo cliente com AWS IoT Core

Você pode configurar o componente do gerenciador de sombras para sincronizar os estados de sombra do dispositivo cliente local comAWS IoT Core. Para ter mais informações, consulte Sincronize sombras do dispositivo local com AWS IoT Core.

Solução de problemas de dispositivos cliente

Use as informações e soluções de solução de problemas desta seção para ajudar a resolver problemas com os dispositivos e componentes do dispositivo cliente do Greengrass.

Tópicos

Problemas de descoberta do Greengrass

Use as informações a seguir para solucionar problemas com o Greengrass discovery. Esses problemas podem ocorrer quando dispositivos clientes usam a API de descoberta do Greengrass para identificar um dispositivo principal do Greengrass ao qual eles podem se conectar.

Tópicos

Problemas de descoberta do Greengrass (API HTTP)

Use as informações a seguir para solucionar problemas com o Greengrass discovery. Talvez você veja esses erros se testar a API de descoberta com cURL.

Tópicos

curl: (52) Empty reply from server

Você pode ver esse erro se especificar um AWS IoT certificado inativo na solicitação.

Verifique se o dispositivo cliente tem um certificado anexado e se o certificado está ativo. Para obter mais informações, consulte Anexar algo ou política a um certificado de cliente e Ativar ou desativar um certificado de cliente no Guia do AWS IoT Core desenvolvedor.

HTTP 403: {"message":null,"traceId":"a1b2c3d4-5678-90ab-cdef-11111EXAMPLE"}

Você pode ver esse erro se o dispositivo cliente não tiver permissão para ligar `greengrass:Discover` por si mesmo.

Verifique se o certificado do dispositivo cliente tem uma política que permita`greengrass:Discover`. Você não pode usar variáveis de política de coisas (`iot:Connection.Thing.*`) na `Resource` seção para essa permissão. Para ter mais informações, consulte Autenticação e autorização de descoberta.

HTTP 404: {"errorMessage":"The thing provided for discovery was not found"}

Você pode ver esse erro nos seguintes casos:

- O dispositivo cliente não está associado a nenhum dispositivo ou AWS IoT Greengrass V1 grupo principal do Greengrass.
- Nenhum dos dispositivos ou AWS IoT Greengrass V1 grupos principais do Greengrass associados ao dispositivo cliente tem um endpoint de agente MQTT.
- Nenhum dos dispositivos principais do Greengrass associados ao dispositivo cliente executa o componente de autenticação do dispositivo cliente.

Verifique se o dispositivo cliente está associado ao dispositivo principal ao qual você deseja se conectar. Em seguida, verifique se o dispositivo principal executa o componente de autenticação do dispositivo cliente e tem pelo menos um endpoint do agente MQTT. Para ver mais informações, consulte:

- Associe dispositivos do cliente
- Gerencie os endpoints principais do dispositivo
- Configurar a descoberta na nuvem (console)

Problemas de descoberta do Greengrass (AWS IoT Device SDKv2 para Python)

Use as informações a seguir para solucionar problemas com a descoberta do Greengrass na v2 para AWS IoT Device SDK Python.

Tópicos

- awscrt.exceptions.AwsCrtError: AWS_ERROR_HTTP_CONNECTION_CLOSED: The connection has closed or is closing.

- awsiot.greengrass_discovery.DiscoveryException: ('Error during discover call: response_code=403', 403)
- awsiot.greengrass_discovery.DiscoveryException: ('Error during discover call: response_code=404', 404)

awscrt.exceptions.AwsCrtError: AWS_ERROR_HTTP_CONNECTION_CLOSED: The connection has closed or is closing.

Você pode ver esse erro se especificar um AWS IoT certificado inativo na solicitação.

Verifique se o dispositivo cliente tem um certificado anexado e se o certificado está ativo. Para obter mais informações, consulte Anexar algo ou política a um certificado de cliente e Ativar ou desativar um certificado de cliente no Guia do AWS IoT Core desenvolvedor.

awsiot.greengrass_discovery.DiscoveryException: ('Error during discover call: response_code=403', 403)

Você pode ver esse erro se o dispositivo cliente não tiver permissão para ligar `greengrass:Discover` por si mesmo.

Verifique se o certificado do dispositivo cliente tem uma política que permita`greengrass:Discover`. Você não pode usar variáveis de política de coisas (`iot:Connection.Thing.*`) na `Resource` seção para essa permissão. Para ter mais informações, consulte Autenticação e autorização de descoberta.

awsiot.greengrass_discovery.DiscoveryException: ('Error during discover call: response_code=404', 404)

Você pode ver esse erro nos seguintes casos:

- O dispositivo cliente não está associado a nenhum dispositivo ou AWS IoT Greengrass V1 grupo principal do Greengrass.
- Nenhum dos dispositivos ou AWS IoT Greengrass V1 grupos principais do Greengrass associados ao dispositivo cliente tem um endpoint de agente MQTT.
- Nenhum dos dispositivos principais do Greengrass associados ao dispositivo cliente executa o componente de autenticação do dispositivo cliente.

Verifique se o dispositivo cliente está associado ao dispositivo principal ao qual você deseja se conectar. Em seguida, verifique se o dispositivo principal executa o componente de autenticação

do dispositivo cliente e tem pelo menos um endpoint do agente MQTT. Para ver mais informações, consulte:

- Associe dispositivos do cliente
- Gerencie os endpoints principais do dispositivo
- Configurar a descoberta na nuvem (console)

Problemas de descoberta do Greengrass (AWS IoT Device SDKv2 para C++)

Use as informações a seguir para solucionar problemas com a descoberta do Greengrass na v2 para AWS IoT Device SDK C++.

Tópicos

- aws-c-http: AWS_ERROR_HTTP_CONNECTION_CLOSED, The connection has closed or is closing.
- aws-c-common: AWS_ERROR_UNKNOWN, Unknown error. (HTTP 403)
- aws-c-common: AWS_ERROR_UNKNOWN, Unknown error. (HTTP 404)

aws-c-http: AWS_ERROR_HTTP_CONNECTION_CLOSED, The connection has closed or is closing.

Você pode ver esse erro se especificar um AWS IoT certificado inativo na solicitação.

Verifique se o dispositivo cliente tem um certificado anexado e se o certificado está ativo. Para obter mais informações, consulte Anexar algo ou política a um certificado de cliente e Ativar ou desativar um certificado de cliente no Guia do AWS IoT Core desenvolvedor.

aws-c-common: AWS_ERROR_UNKNOWN, Unknown error. (HTTP 403)

Você pode ver esse erro se o dispositivo cliente não tiver permissão para ligar `greengrass:Discover` por si mesmo.

Verifique se o certificado do dispositivo cliente tem uma política que permita`greengrass:Discover`. Você não pode usar variáveis de política de coisas (`iot:Connection.Thing.*`) na `Resource` seção para essa permissão. Para ter mais informações, consulte Autenticação e autorização de descoberta.

aws-c-common: AWS_ERROR_UNKNOWN, Unknown error. (HTTP 404)

Você pode ver esse erro nos seguintes casos:

- O dispositivo cliente não está associado a nenhum dispositivo ou AWS IoT Greengrass V1 grupo principal do Greengrass.
- Nenhum dos dispositivos ou AWS IoT Greengrass V1 grupos principais do Greengrass associados ao dispositivo cliente tem um endpoint de agente MQTT.
- Nenhum dos dispositivos principais do Greengrass associados ao dispositivo cliente executa o componente de autenticação do dispositivo cliente.

Verifique se o dispositivo cliente está associado ao dispositivo principal ao qual você deseja se conectar. Em seguida, verifique se o dispositivo principal executa o componente de autenticação do dispositivo cliente e tem pelo menos um endpoint do agente MQTT. Para ver mais informações, consulte:

- Associe dispositivos do cliente
- Gerencie os endpoints principais do dispositivo
- Configurar a descoberta na nuvem (console)

Problemas de descoberta do Greengrass (versão 2 AWS IoT Device SDK para) JavaScript

Use as informações a seguir para solucionar problemas com a descoberta do Greengrass na AWS IoT Device SDK versão 2 para. JavaScript

Tópicos

- Error: aws-c-http: AWS_ERROR_HTTP_CONNECTION_CLOSED, The connection has closed or is closing.
- Error: Discovery failed (headers: [object Object]) { response_code: 403 }
- Error: Discovery failed (headers: [object Object]) { response_code: 404 }
- Error: Discovery failed (headers: [object Object])

Error: aws-c-http: AWS_ERROR_HTTP_CONNECTION_CLOSED, The connection has closed or is closing.

Você pode ver esse erro se especificar um AWS IoT certificado inativo na solicitação.

Verifique se o dispositivo cliente tem um certificado anexado e se o certificado está ativo. Para obter mais informações, consulte Anexar algo ou política a um certificado de cliente e Ativar ou desativar um certificado de cliente no Guia do AWS IoT Core desenvolvedor.

Error: Discovery failed (headers: [object Object]) { response_code: 403 }

Você pode ver esse erro se o dispositivo cliente não tiver permissão para ligar `greengrass:Discover` por si mesmo.

Verifique se o certificado do dispositivo cliente tem uma política que permita`greengrass:Discover`. Você não pode usar variáveis de política de coisas (`iot:Connection.Thing.*`) na `Resource` seção para essa permissão. Para ter mais informações, consulte Autenticação e autorização de descoberta.

Error: Discovery failed (headers: [object Object]) { response_code: 404 }

Você pode ver esse erro nos seguintes casos:

- O dispositivo cliente não está associado a nenhum dispositivo ou AWS IoT Greengrass V1 grupo principal do Greengrass.
- Nenhum dos dispositivos ou AWS IoT Greengrass V1 grupos principais do Greengrass associados ao dispositivo cliente tem um endpoint de agente MQTT.
- Nenhum dos dispositivos principais do Greengrass associados ao dispositivo cliente executa o componente de autenticação do dispositivo cliente.

Verifique se o dispositivo cliente está associado ao dispositivo principal ao qual você deseja se conectar. Em seguida, verifique se o dispositivo principal executa o componente de autenticação do dispositivo cliente e tem pelo menos um endpoint do agente MQTT. Para ver mais informações, consulte:

- Associe dispositivos do cliente
- Gerencie os endpoints principais do dispositivo
- Configurar a descoberta na nuvem (console)

Error: Discovery failed (headers: [object Object])

Você pode ver esse erro (sem um código de resposta HTTP) ao executar a amostra de descoberta do Greengrass. Esse erro pode ocorrer por vários motivos.

- Você pode ver esse erro se o dispositivo cliente não tiver permissão para ligar `greengrass:Discover` por si mesmo.

  Verifique se o certificado do dispositivo cliente tem uma política que permita`greengrass:Discover`. Você não pode usar variáveis de política de coisas (`iot:Connection.Thing.*`) na `Resource` seção para essa permissão. Para ter mais informações, consulte Autenticação e autorização de descoberta.
- Você pode ver esse erro nos seguintes casos:
  - O dispositivo cliente não está associado a nenhum dispositivo ou AWS IoT Greengrass V1 grupo principal do Greengrass.
  - Nenhum dos dispositivos ou AWS IoT Greengrass V1 grupos principais do Greengrass associados ao dispositivo cliente tem um endpoint de agente MQTT.
  - Nenhum dos dispositivos principais do Greengrass associados ao dispositivo cliente executa o componente de autenticação do dispositivo cliente.

  Verifique se o dispositivo cliente está associado ao dispositivo principal ao qual você deseja se conectar. Em seguida, verifique se o dispositivo principal executa o componente de autenticação do dispositivo cliente e tem pelo menos um endpoint do agente MQTT. Para ver mais informações, consulte:
  - Associe dispositivos do cliente
  - Gerencie os endpoints principais do dispositivo
  - Configurar a descoberta na nuvem (console)

Problemas de descoberta do Greengrass (AWS IoT Device SDKv2 para Java)

Use as informações a seguir para solucionar problemas com a descoberta do Greengrass na v2 para AWS IoT Device SDK Java.

Tópicos

- software.amazon.awssdk.crt.CrtRuntimeException: Error Getting Response Status Code from HttpStream. (aws_last_error: AWS_ERROR_HTTP_DATA_NOT_AVAILABLE(2062), This data is not yet available.)
- java.lang.RuntimeException: Error x-amzn-ErrorType(403)
- java.lang.RuntimeException: Error x-amzn-ErrorType(404)

software.amazon.awssdk.crt.CrtRuntimeException: Error Getting Response Status Code from HttpStream. (aws_last_error: AWS_ERROR_HTTP_DATA_NOT_AVAILABLE(2062), This data is not yet available.)

Você pode ver esse erro se especificar um AWS IoT certificado inativo na solicitação.

Verifique se o dispositivo cliente tem um certificado anexado e se o certificado está ativo. Para obter mais informações, consulte Anexar algo ou política a um certificado de cliente e Ativar ou desativar um certificado de cliente no Guia do AWS IoT Core desenvolvedor.

java.lang.RuntimeException: Error x-amzn-ErrorType(403)

Você pode ver esse erro se o dispositivo cliente não tiver permissão para ligar `greengrass:Discover` por si mesmo.

Verifique se o certificado do dispositivo cliente tem uma política que permita`greengrass:Discover`. Você não pode usar variáveis de política de coisas (`iot:Connection.Thing.*`) na `Resource` seção para essa permissão. Para ter mais informações, consulte Autenticação e autorização de descoberta.

java.lang.RuntimeException: Error x-amzn-ErrorType(404)

Você pode ver esse erro nos seguintes casos:

- O dispositivo cliente não está associado a nenhum dispositivo ou AWS IoT Greengrass V1 grupo principal do Greengrass.
- Nenhum dos dispositivos ou AWS IoT Greengrass V1 grupos principais do Greengrass associados ao dispositivo cliente tem um endpoint de agente MQTT.
- Nenhum dos dispositivos principais do Greengrass associados ao dispositivo cliente executa o componente de autenticação do dispositivo cliente.

Verifique se o dispositivo cliente está associado ao dispositivo principal ao qual você deseja se conectar. Em seguida, verifique se o dispositivo principal executa o componente de autenticação do dispositivo cliente e tem pelo menos um endpoint do agente MQTT. Para ver mais informações, consulte:

- Associe dispositivos do cliente
- Gerencie os endpoints principais do dispositivo
- Configurar a descoberta na nuvem (console)

Problemas de conexão com o MQTT

Use as informações a seguir para solucionar problemas com as conexões MQTT do dispositivo cliente. Esses problemas podem ocorrer quando dispositivos clientes tentam se conectar a um dispositivo principal por meio do MQTT.

Tópicos

io.moquette.broker.Authorizator: Client does not have read permissions on the topic

Você pode ver esse erro nos registros do Greengrass quando um dispositivo cliente tenta se inscrever em um tópico do MQTT sem permissão. A mensagem de erro inclui o tópico.

Verifique se a configuração do componente de autenticação do dispositivo cliente inclui o seguinte:

- Um grupo de dispositivos que corresponde ao dispositivo cliente.
- Uma política de autorização de dispositivo cliente para esse grupo de dispositivos que concede a `mqtt:subscribe` permissão para o tópico.

Para obter mais informações sobre como implantar e configurar o componente de autenticação do dispositivo cliente, consulte o seguinte:

- Configurar a descoberta na nuvem (console)
- Autenticação do dispositivo cliente
- Criar implantações

Problemas de conexão com o MQTT (Python)

Use as informações a seguir para solucionar problemas com conexões MQTT do dispositivo cliente ao usar a AWS IoT Device SDKv2 para Python.

Tópicos

AWS_ERROR_MQTT_PROTOCOL_ERROR: Protocol error occurred

Você pode ver esse erro se o componente de autenticação do dispositivo cliente não definir uma política de autorização do dispositivo cliente que conceda permissão ao dispositivo cliente para se conectar.

Verifique se a configuração do componente de autenticação do dispositivo cliente inclui o seguinte:

- Um grupo de dispositivos que corresponde ao dispositivo cliente.
- Uma política de autorização de dispositivo cliente para esse grupo de dispositivos que concede a `mqtt:connect` permissão para o dispositivo cliente.

Para obter mais informações sobre como implantar e configurar o componente de autenticação do dispositivo cliente, consulte o seguinte:

- Configurar a descoberta na nuvem (console)
- Autenticação do dispositivo cliente
- Criar implantações

AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

Você pode ver esse erro se o componente de autenticação do dispositivo cliente não definir uma política de autorização do dispositivo cliente que conceda permissão ao dispositivo cliente para se conectar.

Verifique se a configuração do componente de autenticação do dispositivo cliente inclui o seguinte:

- Um grupo de dispositivos que corresponde ao dispositivo cliente.
- Uma política de autorização de dispositivo cliente para esse grupo de dispositivos que concede a `mqtt:connect` permissão para o dispositivo cliente.

Para obter mais informações sobre como implantar e configurar o componente de autenticação do dispositivo cliente, consulte o seguinte:

- Configurar a descoberta na nuvem (console)
- Autenticação do dispositivo cliente
- Criar implantações

Problemas de conexão com o MQTT (C++)

Use as informações a seguir para solucionar problemas com conexões MQTT do dispositivo cliente ao usar a AWS IoT Device SDKv2 para C++.

Tópicos

- AWS_ERROR_MQTT_PROTOCOL_ERROR: Protocol error occurred
- AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

AWS_ERROR_MQTT_PROTOCOL_ERROR: Protocol error occurred

Você pode ver esse erro se o componente de autenticação do dispositivo cliente não definir uma política de autorização do dispositivo cliente que conceda permissão ao dispositivo cliente para se conectar.

Verifique se a configuração do componente de autenticação do dispositivo cliente inclui o seguinte:

- Um grupo de dispositivos que corresponde ao dispositivo cliente.
- Uma política de autorização de dispositivo cliente para esse grupo de dispositivos que concede a `mqtt:connect` permissão para o dispositivo cliente.

Para obter mais informações sobre como implantar e configurar o componente de autenticação do dispositivo cliente, consulte o seguinte:

- Configurar a descoberta na nuvem (console)
- Autenticação do dispositivo cliente
- Criar implantações

AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

Você pode ver esse erro se o componente de autenticação do dispositivo cliente não definir uma política de autorização do dispositivo cliente que conceda permissão ao dispositivo cliente para se conectar.

Verifique se a configuração do componente de autenticação do dispositivo cliente inclui o seguinte:

- Um grupo de dispositivos que corresponde ao dispositivo cliente.
- Uma política de autorização de dispositivo cliente para esse grupo de dispositivos que concede a `mqtt:connect` permissão para o dispositivo cliente.

Para obter mais informações sobre como implantar e configurar o componente de autenticação do dispositivo cliente, consulte o seguinte:

- Configurar a descoberta na nuvem (console)
- Autenticação do dispositivo cliente
- Criar implantações

Problemas de conexão com o MQTT (Java)

Use as informações a seguir para solucionar problemas com conexões MQTT do dispositivo cliente ao usar a AWS IoT Device SDKv2 para Java.

Tópicos

software.amazon.awssdk.crt.mqtt.MqttException: Protocol error occurred

Você pode ver esse erro se o componente de autenticação do dispositivo cliente não definir uma política de autorização do dispositivo cliente que conceda permissão ao dispositivo cliente para se conectar.

Verifique se a configuração do componente de autenticação do dispositivo cliente inclui o seguinte:

- Um grupo de dispositivos que corresponde ao dispositivo cliente.

- Uma política de autorização de dispositivo cliente para esse grupo de dispositivos que concede a `mqtt:connect` permissão para o dispositivo cliente.

Para obter mais informações sobre como implantar e configurar o componente de autenticação do dispositivo cliente, consulte o seguinte:

- Configurar a descoberta na nuvem (console)
- Autenticação do dispositivo cliente
- Criar implantações

AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

Você pode ver esse erro se o componente de autenticação do dispositivo cliente não definir uma política de autorização do dispositivo cliente que conceda permissão ao dispositivo cliente para se conectar.

Verifique se a configuração do componente de autenticação do dispositivo cliente inclui o seguinte:

- Um grupo de dispositivos que corresponde ao dispositivo cliente.
- Uma política de autorização de dispositivo cliente para esse grupo de dispositivos que concede a `mqtt:connect` permissão para o dispositivo cliente.

Para obter mais informações sobre como implantar e configurar o componente de autenticação do dispositivo cliente, consulte o seguinte:

- Configurar a descoberta na nuvem (console)
- Autenticação do dispositivo cliente
- Criar implantações

Problemas de conexão com o MQTT () JavaScript

Use as informações a seguir para solucionar problemas com conexões MQTT do dispositivo cliente ao usar a AWS IoT Device SDKv2 para. JavaScript

Tópicos

- AWS_ERROR_MQTT_PROTOCOL_ERROR: Protocol error occurred

- AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

AWS_ERROR_MQTT_PROTOCOL_ERROR: Protocol error occurred

Você pode ver esse erro se o componente de autenticação do dispositivo cliente não definir uma política de autorização do dispositivo cliente que conceda permissão ao dispositivo cliente para se conectar.

Verifique se a configuração do componente de autenticação do dispositivo cliente inclui o seguinte:

- Um grupo de dispositivos que corresponde ao dispositivo cliente.
- Uma política de autorização de dispositivo cliente para esse grupo de dispositivos que concede a `mqtt:connect` permissão para o dispositivo cliente.

Para obter mais informações sobre como implantar e configurar o componente de autenticação do dispositivo cliente, consulte o seguinte:

- Configurar a descoberta na nuvem (console)
- Autenticação do dispositivo cliente
- Criar implantações

AWS_ERROR_MQTT_UNEXPECTED_HANGUP: Unexpected hangup occurred

Você pode ver esse erro se o componente de autenticação do dispositivo cliente não definir uma política de autorização do dispositivo cliente que conceda permissão ao dispositivo cliente para se conectar.

Verifique se a configuração do componente de autenticação do dispositivo cliente inclui o seguinte:

- Um grupo de dispositivos que corresponde ao dispositivo cliente.
- Uma política de autorização de dispositivo cliente para esse grupo de dispositivos que concede a `mqtt:connect` permissão para o dispositivo cliente.

Para obter mais informações sobre como implantar e configurar o componente de autenticação do dispositivo cliente, consulte o seguinte:

- Configurar a descoberta na nuvem (console)

- Autenticação do dispositivo cliente
- Criar implantações

Interaja com as sombras do dispositivo

Os principais dispositivos do Greengrass podem interagir com as sombras AWS IoT do dispositivo usando componentes. Uma sombra é um documento JSON que armazena as informações de estado atuais ou desejadas de qualquer AWS IoT coisa. As sombras podem disponibilizar o estado de um dispositivo para outros AWS IoT Greengrass componentes, independentemente de o dispositivo estar conectado AWS IoT ou não. Cada AWS IoT dispositivo tem sua própria sombra clássica sem nome. Você também pode criar várias sombras nomeadas para cada dispositivo.

Dispositivos e serviços podem criar, atualizar e excluir sombras na nuvem usando o MQTT e os tópicos de sombra reservados do MQTT, o HTTP usando a API REST Device Shadow e o for. AWS CLIAWS IoT

O componente de gerenciamento de sombras permite que seus componentes do Greengrass criem, atualizem e excluam sombras locais usando o serviço paralelo local e os tópicos paralelos locais de publicação/assinatura. O gerenciador de sombras também gerencia o armazenamento desses documentos paralelos locais em seu dispositivo principal e gerencia a sincronização das informações do estado da sombra com as sombras da nuvem.

Você também pode usar o componente gerenciador de sombras para gerenciar sombras locais para dispositivos cliente que se conectam ao dispositivo principal. Para permitir que o shadow manager gerencie as sombras do dispositivo cliente, você configura o componente de ponte MQTT para retransmitir mensagens entre o agente MQTT local e o serviço local de publicação/assinatura. Para ter mais informações, consulte Interaja e sincronize as sombras do dispositivo cliente.

Para obter mais informações sobre os conceitos AWS IoT do Device Shadow, consulte AWS IoTo serviço Device Shadow no AWS IoTDeveloper Guide.

Tópicos

- Interaja com sombras em componentes
- Sincronize sombras do dispositivo local com AWS IoT Core

Interaja com sombras em componentes

Você pode desenvolver componentes personalizados, incluindo componentes da função Lambda, que usam o serviço paralelo local para ler e modificar documentos paralelos locais e documentos paralelos do dispositivo cliente.

Os componentes personalizados interagem com o serviço paralelo local usando as bibliotecas AWS IoT Greengrass Core IPC noAWS IoT Device SDK. O componente do gerenciador de sombras ativa o serviço paralelo local em seu dispositivo principal.

Para implantar o componente shadow manager em um dispositivo principal do Greengrass, crie uma implantação que inclua o `aws.greengrass.ShadowManager` componente.

> **Note**
>
> Por padrão, a implantação do componente do gerenciador de sombras permite somente operações de sombra locais. AWS IoT GreengrassPara permitir a sincronização das informações do estado da sombra das sombras do dispositivo principal ou de quaisquer sombras dos dispositivos cliente com os documentos de sombra da nuvem correspondentes emAWS IoT Core, você deve criar uma atualização de configuração para o componente do gerenciador de sombras que inclua o `synchronize` parâmetro. Para ter mais informações, consulte Sincronize sombras do dispositivo local com AWS IoT Core.

Tópicos

- Recupere e modifique estados de sombra
- Reaja às mudanças do estado da sombra

Recupere e modifique estados de sombra

As operações paralelas do IPC recuperam e atualizam informações de estado em documentos paralelos locais. O componente do gerenciador de sombras gerencia o armazenamento desses documentos paralelos em seu dispositivo principal.

Para modificar os estados de sombra locais

1. Adicione políticas de autorização à receita do seu componente personalizado para permitir que o componente receba mensagens sobre tópicos paralelos locais.

   Por exemplo, políticas de autorização, consulte Exemplos de políticas de autorização local de IPC paralelo.
2. Use as operações de IPC de sombra para recuperar e modificar as informações do estado de sombra. Para obter mais informações sobre o uso de operações de IPC de sombra no código do componente, consulteInteraja com sombras locais.

> **Note**
>
> Para permitir que um dispositivo principal interaja com as sombras do dispositivo cliente, você também deve configurar e implantar o componente de ponte MQTT. Para obter mais informações, consulte Habilitar o shadow manager para se comunicar com dispositivos clientes.

Reaja às mudanças do estado da sombra

Os componentes do Greengrass usam a interface local de publicação/assinatura para se comunicarem em um dispositivo principal. Para permitir que um componente personalizado reaja às mudanças do estado de sombra, você pode se inscrever nos tópicos locais de publicação/assinatura. Isso permite que o componente receba mensagens sobre os tópicos paralelos locais e, em seguida, atue de acordo com essas mensagens.

Os tópicos paralelos locais usam o mesmo formato dos tópicos MQTT de sombra do AWS IoT dispositivo. Para obter mais informações sobre tópicos paralelos, consulte os tópicos do Device Shadow MQTT no Guia do AWS IoT desenvolvedor.

Para reagir às mudanças locais do estado da sombra

1. Adicione políticas de controle de acesso à receita do seu componente personalizado para permitir que o componente receba mensagens sobre tópicos paralelos locais.

   Por exemplo, políticas de autorização, consulte Exemplos de políticas de autorização local de IPC paralelo.
2. Para iniciar uma ação personalizada em um componente, use as operações `SubscribeToTopic` IPC para assinar os tópicos paralelos nos quais você deseja receber mensagens. Para obter mais informações sobre o uso de operações IPC locais de publicação/assinatura no código do componente, consulte. Publique/assine mensagens locais
3. Para invocar uma função Lambda, use a configuração da fonte de eventos para fornecer o nome do tópico paralelo e especificar que é um tópico local de publicação/assinatura. Para obter informações sobre a criação de componentes da função Lambda, consulte. Executar AWS Lambda funções

> **Note**
>
> Para permitir que um dispositivo principal interaja com as sombras do dispositivo cliente, você também deve configurar e implantar o componente de ponte MQTT. Para obter mais informações, consulte Habilitar o shadow manager para se comunicar com dispositivos clientes.

Sincronize sombras do dispositivo local com AWS IoT Core

O componente do gerenciador de sombras permite AWS IoT Greengrass sincronizar os estados de sombra do dispositivo local com AWS IoT Core o. Você deve modificar a configuração do componente do gerenciador de sombras para incluir o parâmetro de `synchronization` configuração e especificar os AWS IoT nomes dos dispositivos e as sombras que você deseja sincronizar.

Quando você configura o gerenciador de sombras para sincronizar sombras, ele sincroniza todas as alterações de estado das sombras especificadas, independentemente de as alterações ocorrerem em documentos de sombra locais ou em documentos de sombra na nuvem.

Você também pode especificar se o componente gerenciador de sombras sincroniza sombras em tempo real ou em um intervalo periódico. Por padrão, o componente do gerenciador de sombras sincroniza as sombras em tempo real, para que o dispositivo principal envie e receba atualizações de sombra de e para AWS IoT Core quando cada atualização ocorre. Você pode configurar intervalos periódicos para reduzir o uso da largura de banda e as cobranças.

Tópicos

Pré-requisitos

Para sincronizar sombras locais comAWS IoT Core, você deve configurar a política do dispositivo principal AWS IoT do Greengrass para permitir as AWS IoT Core seguintes ações de política paralela.

- `iot:GetThingShadow`
- `iot:UpdateThingShadow`
- `iot:DeleteThingShadow`

Para ver mais informações, consulte:

- AWS IoT Coreações políticas no Guia do AWS IoT desenvolvedor
- AWS IoTPolítica mínima para dispositivos AWS IoT Greengrass V2 principais
- Atualizar a AWS IoT política de um dispositivo principal

Configurar o componente do gerenciador de sombras

O gerenciador de sombra requer uma lista de mapeamentos de nomes de sombra para sincronizar informações de estado de sombra em documentos de sombra locais com documentos de sombra em nuvem. AWS IoT Core

Para sincronizar estados de sombra, crie uma implantação que inclua o `aws.greengrass.ShadowManager` componente e especifique as sombras que você deseja sincronizar no parâmetro de `synchronize` configuração na configuração do gerenciador de sombras na implantação.

> **Note**
>
> Para permitir que um dispositivo principal interaja com as sombras do dispositivo cliente, você também deve configurar e implantar o componente de ponte MQTT. Para obter mais informações, consulte Habilitar o shadow manager para se comunicar com dispositivos clientes.

O exemplo de atualização de configuração a seguir instrui o componente do gerenciador de sombras a sincronizar as seguintes sombras com: AWS IoT Core

- A sombra clássica para o dispositivo principal
- O nome `MyCoreShadow` do dispositivo principal
- A sombra clássica de uma coisa de IoT chamada `MyDevice2`
- As sombras nomeadas `MyShadowA` e `MyShadowB` para uma coisa de IoT chamada `MyDevice1`

Essa atualização de configuração especifica a sincronização de sombras AWS IoT Core em tempo real. Se você usar o gerenciador de sombras v2.1.0 ou posterior, poderá configurar o componente do gerenciador de sombras para sincronizar sombras em um intervalo periódico. Para configurar esse recurso, altere a estratégia de sincronização para `periodic` e especifique um `delay` em segundos para o intervalo. Para obter mais informações, consulte o parâmetro de configuração da estratégia do componente shadow manager.

Essa atualização de configuração especifica a sincronização de sombras em ambas as direções entre o dispositivo principal AWS IoT Core e o dispositivo principal. Se você usar o gerenciador de sombras v2.2.0 ou posterior, poderá configurar o componente do gerenciador de sombras para sincronizar sombras em apenas uma direção. Para configurar esse recurso, altere a sincronização `direction` para `deviceToCloud` ou`cloudToDevice`. Para obter mais informações, consulte o parâmetro de configuração de direção do componente gerenciador de sombras.

```
{
  "strategy": {
    "type": "realTime"
  },
  "synchronize": {
    "coreThing": {
      "classic": true,
      "namedShadows": [
        "MyCoreShadow"
      ]
    },
    "shadowDocuments": [
      {
        "thingName": "MyDevice1",
        "classic": false,
        "namedShadows": [
          "MyShadowA",
          "MyShadowB"
        ]
      },
      {
        "thingName": "MyDevice2",
        "classic": true,
        "namedShadows": [ ]
      }
    ],
    "direction": "betweenDeviceAndCloud"
  }
```

```
}
```

Sincronizar sombras locais

Quando o dispositivo principal do Greengrass está conectado à AWS IoT nuvem, o gerenciador de sombras executa as seguintes tarefas para as sombras que você especifica na configuração do componente. O comportamento depende da opção de configuração da direção de sincronização de sombras especificada por você. Por padrão, o gerenciador de sombras usa a `betweenDeviceAndCloud` opção de sincronizar sombras nas duas direções. Se você usa o gerenciador de sombras v2.2.0 ou posterior, pode configurar o dispositivo principal para sincronizar sombras em apenas uma direção, que pode ser ou. `cloudToDevice` `deviceToCloud`

- Se a configuração da direção da sincronização de sombra for `betweenDeviceAndCloud` ou`cloudToDevice`, o gerenciador de sombras recuperará as informações de estado relatadas do documento de sombra na nuvem emAWS IoT Core. Em seguida, ele atualiza os documentos-sombra armazenados localmente para sincronizar o estado do dispositivo.
- Se a configuração da direção da sincronização de sombra for `betweenDeviceAndCloud` ou`deviceToCloud`, o gerenciador de sombras publicará o estado atual do dispositivo no documento de sombra na nuvem.

Comportamento de conflito do Shadow Merge

Em alguns casos, como quando o dispositivo principal está desconectado da Internet, uma sombra pode mudar no serviço paralelo local e na AWS IoT nuvem antes que o gerenciador de sombra sincronize as alterações. Como resultado, os estados desejados e reportados diferem entre o serviço paralelo local e a AWS IoT nuvem.

Quando o gerenciador de sombras sincroniza a sombra, ele mescla as alterações de acordo com o seguinte comportamento:

- Se você usa uma versão do gerenciador de sombras anterior à v2.2.0 ou quando especifica a direção da sincronização de `betweenDeviceAndCloud` sombras, o seguinte comportamento se aplica:
  - Quando há um conflito de mesclagem no estado desejado de uma sombra, o gerenciador de sombra substitui a seção conflitante do documento paralelo local pelo valor da nuvem. AWS IoT

  - Quando há um conflito de mesclagem no estado relatado de uma sombra, o gerenciador de sombra substitui a seção conflitante da sombra na AWS IoT nuvem pelo valor do documento de sombra local.
- Quando você especifica a `deviceToCloud` direção da sincronização de sombras, o gerenciador de sombras substitui a seção conflitante da sombra na AWS IoT nuvem pelo valor do documento de sombra local.
- Quando você especifica a `cloudToDevice` direção da sincronização de sombras, o gerenciador de sombras substitui a seção conflitante do documento de sombra local pelo valor da AWS IoT nuvem.

Gerencie fluxos de dados nos dispositivos principais do Greengrass

AWS IoT GreengrassO gerenciador de fluxo torna mais eficiente e confiável transferir dados de IoT de alto volume para o. Nuvem AWS O gerenciador de fluxo processa fluxos de dados no AWS IoT Greengrass Core antes de exportá-los para o. Nuvem AWS O Stream Manager se integra a cenários de ponta comuns, como inferência de aprendizado de máquina (ML), em que o dispositivo AWS IoT Greengrass principal processa e analisa os dados antes de exportá-los para os destinos de armazenamento locais Nuvem AWS ou para os destinos de armazenamento.

O Stream Manager fornece uma interface comum para simplificar o desenvolvimento de componentes personalizados para que você não precise criar uma funcionalidade personalizada de gerenciamento de stream. Seus componentes podem usar um mecanismo padronizado para processar fluxos de alto volume e gerenciar políticas locais de retenção de dados. Você pode definir políticas para tipo, tamanho e retenção de dados de armazenamento para cada stream para controlar como o stream manager processa e exporta dados.

O Stream Manager funciona em ambientes com conectividade intermitente ou limitada. Você pode definir o uso da largura de banda, o comportamento do tempo limite e como o AWS IoT Greengrass Core manipula os dados de streaming quando estão conectados ou desconectados. Você também pode definir prioridades para controlar a ordem na qual o AWS IoT Greengrass Core exporta fluxos para o. Nuvem AWS Isso possibilita que você manipule dados críticos mais cedo do que outros dados.

Você pode configurar o gerenciador de fluxo para exportar dados automaticamente Nuvem AWS para armazenamento ou processamento e análise adicionais. O Stream Manager suporta exportações para os seguintes Nuvem AWS destinos:

- Canais emAWS IoT Analytics. AWS IoT Analyticspermite realizar análises avançadas em seus dados para ajudar a tomar decisões comerciais e melhorar os modelos de aprendizado de máquina. Para obter mais informações, consulte O que é o AWS IoT Analytics? no Guia do usuário do AWS IoT Analytics.
- Streams no Amazon Kinesis Data Streams. Você pode usar o Kinesis Data Streams para agregar dados de alto volume e carregá-los em um data warehouse ou cluster. MapReduce Para obter mais informações, consulte O que é o Amazon Kinesis Data Streams? no Guia do desenvolvedor do Amazon Kinesis Data Streams.

- Propriedades de ativos emAWS IoT SiteWise. AWS IoT SiteWisepermite coletar, organizar e analisar dados de equipamentos industriais em grande escala. Para obter mais informações, consulte O que é o AWS IoT SiteWise? no Guia do usuário do AWS IoT SiteWise.
- Objetos no Amazon Simple Storage Service Amazon S3. Você pode utilizar o Amazon S3 para armazenar e recuperar grandes volumes de dados. Para obter mais informações, consulte O que é o Amazon S3? no Guia do desenvolvedor do Amazon Simple Storage Service.

Fluxo de trabalho do gerenciamento de streams

Seus aplicativos de IoT interagem com o stream manager por meio do Stream Manager SDK.

Em um fluxo de trabalho simples, um componente no AWS IoT Greengrass núcleo consome dados de IoT, como métricas de temperatura e pressão de séries temporais. O componente pode filtrar ou compactar os dados e, em seguida, chamar o SDK do Stream Manager para gravar os dados em um stream no stream manager. O gerenciador de fluxo pode exportar o fluxo para o Nuvem AWS automaticamente com base nas políticas que você define para o fluxo. Os componentes também podem enviar dados diretamente para bancos de dados ou repositórios de armazenamento locais.

Seus aplicativos de IoT podem incluir vários componentes personalizados que leem ou gravam em fluxos. Esses componentes podem ler e gravar em fluxos para filtrar, agregar e analisar dados no dispositivo AWS IoT Greengrass principal. Isso possibilita responder rapidamente a eventos locais e extrair informações valiosas antes que os dados sejam transferidos do núcleo para os Nuvem AWS destinos locais.

Para começar, implante o componente stream manager em seu dispositivo AWS IoT Greengrass principal. Na implantação, configure os parâmetros do componente stream manager para definir as configurações que se aplicam a todos os streams no dispositivo principal do Greengrass. Use esses parâmetros para controlar como o gerenciador de fluxos armazena, processa e exporta fluxos com base nas necessidades da sua empresa e nas restrições do ambiente.

Depois de configurar o gerenciador de fluxo, você pode criar e implantar seus aplicativos de IoT. Normalmente, esses são componentes personalizados usados `StreamManagerClient` no SDK do Stream Manager para criar e interagir com fluxos. Ao criar um stream, você pode definir políticas por stream, como destinos de exportação, prioridade e persistência.

Requisitos

Os seguintes requisitos são aplicados para usar o gerenciador de fluxo:

- O gerenciador de streaming requer um mínimo de 70 MB de RAM, além do software AWS IoT Greengrass Core. O requisito de memória total depende da sua carga de trabalho.
- AWS IoT Greengrassos componentes devem usar o Stream Manager SDK para interagir com o stream manager. O SDK do Stream Manager está disponível nos seguintes idiomas:
  - SDK do Stream Manager para Java (v1.1.0 ou posterior)
  - SDK do Stream Manager para Node.js (v1.1.0 ou posterior)
  - SDK do Stream Manager para Python (v1.1.0 ou posterior)
- AWS IoT Greengrassos componentes devem especificar o componente do gerenciador de fluxo (`aws.greengrass.StreamManager`) como uma dependência em sua receita para usar o gerenciador de fluxo.

  > **Note**
  >
  > Se você usa o gerenciador de stream para exportar dados para a nuvem, não poderá atualizar a versão 2.0.7 do componente stream manager para uma versão entre v2.0.8 e v2.0.11. Se você estiver implantando o gerenciador de fluxo pela primeira vez, é altamente recomendável implantar a versão mais recente do componente do gerenciador de fluxo.

- Se você definir destinos de Nuvem AWS exportação para um stream, deverá criar seus destinos de exportação e conceder permissões de acesso na função de dispositivo do Greengrass. Dependendo do destino, outros requisitos também podem ser aplicados. Para obter mais informações, consulte:
  - the section called “Canais do AWS IoT Analytics”
  - the section called “Amazon Kinesis Data Streams”
  - the section called “Propriedades do ativo AWS IoT SiteWise”
  - the section called “Objetos do Amazon S3”

  Você é responsável pela manutenção desses recursos da Nuvem AWS.

Segurança de dados

Ao usar o gerenciador de fluxo, esteja ciente das seguintes considerações de segurança.

Segurança de dados locais

AWS IoT Greengrassnão criptografa dados de fluxo em repouso ou em trânsito entre componentes locais no dispositivo principal.

- Dados em repouso. Os dados de fluxo são armazenados localmente em um diretório de armazenamento. Para segurança de dados, AWS IoT Greengrass depende de permissões de arquivo e criptografia de disco inteiro, se ativada. Você pode usar o parâmetro STREAM_MANAGER_STORE_ROOT_DIR opcional para especificar o diretório de armazenamento. Se você alterar esse parâmetro posteriormente de modo a usar um diretório de armazenamento diferente, o AWS IoT Greengrass não excluirá o diretório de armazenamento anterior nem seu conteúdo.
- Dados em trânsito localmente. AWS IoT Greengrassnão criptografa dados de fluxo em trânsito local entre fontes de dados, AWS IoT Greengrass componentes, o SDK do Stream Manager e o gerenciador de fluxo.
- Dados em trânsito para a Nuvem AWS. Os fluxos de dados exportados pelo gerenciador de fluxo para a Nuvem AWS usam criptografia de cliente do serviço da AWS padrão com Transport Layer Security (TLS).

Autenticação de cliente

Os clientes do Stream Manager usam o Stream Manager SDK para se comunicarem com o Stream Manager. Quando a autenticação do cliente está ativada, somente os componentes do Greengrass podem interagir com os fluxos no gerenciador de fluxos. Quando a autenticação do cliente está desativada, qualquer processo em execução no dispositivo principal do Greengrass pode interagir com os fluxos no gerenciador de fluxos. Você só deve desabilitar a autenticação se o seu caso de negócios exigir.

Use o parâmetro STREAM_MANAGER_AUTHENTICATE_CLIENT para definir o modo de autenticação do cliente. Você pode configurar esse parâmetro ao implantar o componente do gerenciador de fluxo nos dispositivos principais.

	Habilitado	Desabilitado
Valor do parâmetro	`true` (padrão e recomendado)	`false`

	Habilitado	Desabilitado
Clientes permitidos	Componentes do Greengrass no dispositivo principal	Componentes do Greengrass no dispositivo principal Outros processos em execução no dispositivo de núcleo do Greengrass

Consulte também

- the section called “Configurar o gerenciador de fluxo”
- the section called “Use StreamManagerClient para trabalhar com streams”
- the section called “Configurações de exportação para destinos de nuvem compatíveis”

Crie componentes personalizados que usam o gerenciador de fluxo

Use o gerenciador de streams em componentes personalizados do Greengrass para armazenar, processar e exportar dados de dispositivos de IoT. Use os procedimentos e exemplos desta seção para criar receitas de componentes, artefatos e aplicativos que funcionem com o gerenciador de fluxo. Para obter mais informações sobre como desenvolver e testar componentes, consulteCrie AWS IoT Greengrass componentes.

Tópicos

- Defina receitas de componentes que usam o gerenciador de fluxo
- Conecte-se ao gerenciador de streaming no código do aplicativo

Defina receitas de componentes que usam o gerenciador de fluxo

Para usar o stream manager em um componente personalizado, você deve definir o `aws.greengrass.StreamManager` componente como uma dependência. Você também deve fornecer o SDK do Stream Manager. Conclua as tarefas a seguir para baixar e usar o SDK do Stream Manager no idioma de sua escolha.

Use o SDK do Stream Manager para Java

O SDK do Stream Manager para Java está disponível como um arquivo JAR que você pode usar para compilar seu componente. Em seguida, você pode criar um aplicativo JAR que inclua o SDK do Stream Manager, definir o JAR do aplicativo como um artefato do componente e executar o JAR do aplicativo no ciclo de vida do componente.

Para usar o SDK do Stream Manager para Java

1. Baixe o arquivo JAR do Stream Manager SDK for Java.
2. Siga um destes procedimentos para criar artefatos de componentes do seu aplicativo Java e do arquivo JAR do SDK do Stream Manager:
   - Crie seu aplicativo como um arquivo JAR que inclua o JAR do SDK do Stream Manager e execute esse arquivo JAR na receita do componente.
   - Defina o JAR do SDK do Stream Manager como um artefato de componente. Adicione esse artefato ao classpath ao executar seu aplicativo na receita do componente.

   Sua receita de componente pode ser parecida com o exemplo a seguir. Esse componente executa uma versão modificada do exemplo StreamManagerS3.java, que `StreamManagerS3.jar` inclui o Stream Manager SDK JAR.

   JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.StreamManagerS3Java",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Uses stream manager to upload a file to an S3
 bucket.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.StreamManager": {
      "VersionRequirement": "^2.0.0"
    }
  },
  "Manifests": [
    {
      "Lifecycle": {
        "run": "java -jar {artifacts:path}/StreamManagerS3.jar"
      },
      "Artifacts": [
```

```
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Java/1.0.0/StreamManagerS3.jar"
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.StreamManagerS3Java
ComponentVersion: 1.0.0
ComponentDescription: Uses stream manager to upload a file to an S3 bucket.
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.StreamManager:
    VersionRequirement: "^2.0.0"
Manifests:
  - Lifecycle:
      run: java -jar {artifacts:path}/StreamManagerS3.jar
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Java/1.0.0/StreamManagerS3.jar
```

Para obter mais informações sobre como desenvolver e testar componentes, consulteCrie AWS IoT Greengrass componentes.

Use o SDK do Stream Manager para Python

O SDK do Stream Manager para Python está disponível como código-fonte que você pode incluir em seu componente. Crie um arquivo ZIP do SDK do Stream Manager, defina o arquivo ZIP como um artefato do componente e instale os requisitos do SDK no ciclo de vida do componente.

Para usar o SDK do Stream Manager para Python

1. Clone ou baixe o repositório aws-greengrass-stream-manager-sdk-python.

```
git clone git@github.com:aws-greengrass/aws-greengrass-stream-manager-sdk-
python.git
```

2. Crie um arquivo ZIP que contenha a `stream_manager` pasta, que contém o código-fonte do SDK do Stream Manager para Python. Você pode fornecer esse arquivo ZIP como um artefato de componente que o software AWS IoT Greengrass Core descompacta ao instalar seu componente. Faça o seguinte:

   a. Abra a pasta que contém o repositório que você clonou ou baixou na etapa anterior.

   ```
   cd aws-greengrass-stream-manager-sdk-python
   ```

   b. Compacte a `stream_manager` pasta em um arquivo ZIP chamado`stream_manager_sdk.zip`.

   Linux or Unix

   ```
   zip -rv stream_manager_sdk.zip stream_manager
   ```

   Windows Command Prompt (CMD)

   ```
   tar -acvf stream_manager_sdk.zip stream_manager
   ```

   PowerShell

   ```
   Compress-Archive stream_manager stream_manager_sdk.zip
   ```

   c. Verifique se o `stream_manager_sdk.zip` arquivo contém a `stream_manager` pasta e seu conteúdo. Execute o comando a seguir para listar o conteúdo do arquivo ZIP.

   Linux or Unix

   ```
   unzip -l stream_manager_sdk.zip
   ```

   Windows Command Prompt (CMD)

   ```
   tar -tf stream_manager_sdk.zip
   ```

A saída deve ser semelhante à seguinte.

```
Archive:  aws-greengrass-stream-manager-sdk-python/stream_manager.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
        0  02-24-2021 20:45   stream_manager/
      913  02-24-2021 20:45   stream_manager/__init__.py
     9719  02-24-2021 20:45   stream_manager/utilinternal.py
     1412  02-24-2021 20:45   stream_manager/exceptions.py
     1004  02-24-2021 20:45   stream_manager/util.py
        0  02-24-2021 20:45   stream_manager/data/
   254463  02-24-2021 20:45   stream_manager/data/__init__.py
    26515  02-24-2021 20:45   stream_manager/streammanagerclient.py
---------                     -------
   294026                     8 files
```

3. Copie os artefatos do SDK do Stream Manager para a pasta de artefatos do seu componente. Além do arquivo ZIP do SDK do Stream Manager, seu componente usa o arquivo do SDK para instalar as dependências do SDK do Stream Manager. `requirements.txt` Substitua *~/greengrass-components pelo* caminho para a pasta que você usa para desenvolvimento local.

   Linux or Unix

   ```
   cp {stream_manager_sdk.zip,requirements.txt} ~/greengrass-components/artifacts/
   com.example.StreamManagerS3Python/1.0.0/
   ```

   Windows Command Prompt (CMD)

   ```
   robocopy . %USERPROFILE%\greengrass-components\artifacts
   \com.example.StreamManagerS3Python\1.0.0 stream_manager_sdk.zip
   robocopy . %USERPROFILE%\greengrass-components\artifacts
   \com.example.StreamManagerS3Python\1.0.0 requirements.txt
   ```

   PowerShell

   ```
   cp .\stream_manager_sdk.zip,.\requirements.txt ~\greengrass-components\artifacts
   \com.example.StreamManagerS3Python\1.0.0\
   ```

4. Crie sua receita de componentes. Na receita, faça o seguinte:

a. Defina `stream_manager_sdk.zip` e `requirements.txt` como artefatos.

b. Defina seu aplicativo Python como um artefato.

c. No ciclo de vida da instalação, instale os requisitos do SDK do Stream Manager a partir de. `requirements.txt`

d. No ciclo de vida de execução, anexe o SDK do Stream Manager `PYTHONPATH` e execute seu aplicativo Python.

Sua receita de componente pode ser parecida com o exemplo a seguir. Esse componente executa o exemplo stream_manager_s3.py.

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.StreamManagerS3Python",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Uses stream manager to upload a file to an S3
 bucket.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.StreamManager": {
      "VersionRequirement": "^2.0.0"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "pip3 install --user -r {artifacts:path}/requirements.txt",
        "run": "export PYTHONPATH=$PYTHONPATH:{artifacts:decompressedPath}/
stream_manager_sdk; python3 {artifacts:path}/stream_manager_s3.py"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_sdk.zip",
          "Unarchive": "ZIP"
        },
```

```
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_s3.py"
        },
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/requirements.txt"
        }
      ]
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "install": "pip3 install --user -r {artifacts:path}/requirements.txt",
        "run": "set \"PYTHONPATH=%PYTHONPATH%;{artifacts:decompressedPath}/
stream_manager_sdk\" & py -3 {artifacts:path}/stream_manager_s3.py"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_sdk.zip",
          "Unarchive": "ZIP"
        },
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_s3.py"
        },
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/requirements.txt"
        }
      ]
    }
  ]
}
```

YAML

```
---
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.StreamManagerS3Python
```

```
ComponentVersion: 1.0.0
ComponentDescription: Uses stream manager to upload a file to an S3 bucket.
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.StreamManager:
    VersionRequirement: "^2.0.0"
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: pip3 install --user -r {artifacts:path}/requirements.txt
      run: |
        export PYTHONPATH=$PYTHONPATH:{artifacts:decompressedPath}/
stream_manager_sdk
        python3 {artifacts:path}/stream_manager_s3.py
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_sdk.zip
        Unarchive: ZIP
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_s3.py
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/requirements.txt
  - Platform:
      os: windows
    Lifecycle:
      install: pip3 install --user -r {artifacts:path}/requirements.txt
      run: |
        set "PYTHONPATH=%PYTHONPATH%;{artifacts:decompressedPath}/
stream_manager_sdk"
        py -3 {artifacts:path}/stream_manager_s3.py
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_sdk.zip
        Unarchive: ZIP
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/stream_manager_s3.py
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3Python/1.0.0/requirements.txt
```

Para obter mais informações sobre como desenvolver e testar componentes, consulteCrie AWS IoT Greengrass componentes.

Use o Stream Manager SDK para JavaScript

O Stream Manager SDK para JavaScript está disponível como código-fonte que você pode incluir em seu componente. Crie um arquivo ZIP do SDK do Stream Manager, defina o arquivo ZIP como um artefato do componente e instale o SDK no ciclo de vida do componente.

Para usar o SDK do Stream Manager para JavaScript

1. Clone ou baixe o repositório aws-greengrass-stream-manager-sdk-js.

   ```
   git clone git@github.com:aws-greengrass/aws-greengrass-stream-manager-sdk-js.git
   ```

2. Crie um arquivo ZIP que contenha a `aws-greengrass-stream-manager-sdk` pasta, que contém o código-fonte do SDK do Stream Manager para JavaScript. Você pode fornecer esse arquivo ZIP como um artefato de componente que o software AWS IoT Greengrass Core descompacta ao instalar seu componente. Faça o seguinte:

   a. Abra a pasta que contém o repositório que você clonou ou baixou na etapa anterior.

      ```
      cd aws-greengrass-stream-manager-sdk-js
      ```

   b. Compacte a `aws-greengrass-stream-manager-sdk` pasta em um arquivo ZIP chamado`stream-manager-sdk.zip`.

      Linux or Unix

      ```
      zip -rv stream-manager-sdk.zip aws-greengrass-stream-manager-sdk
      ```

      Windows Command Prompt (CMD)

      ```
      tar -acvf stream-manager-sdk.zip aws-greengrass-stream-manager-sdk
      ```

      PowerShell

      ```
      Compress-Archive aws-greengrass-stream-manager-sdk stream-manager-sdk.zip
      ```

   c. Verifique se o `stream-manager-sdk.zip` arquivo contém a `aws-greengrass-stream-manager-sdk` pasta e seu conteúdo. Execute o comando a seguir para listar o conteúdo do arquivo ZIP.

Linux or Unix

```
unzip -l stream-manager-sdk.zip
```

Windows Command Prompt (CMD)

```
tar -tf stream-manager-sdk.zip
```

A saída deve ser semelhante à seguinte.

```
Archive:  stream-manager-sdk.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
        0  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/
      369  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/package.json
     1017  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/util.js
     8374  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/utilInternal.js
     1937  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/exceptions.js
        0  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/data/
   353343  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/data/index.js
    22599  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/client.js
      216  02-24-2021 22:36   aws-greengrass-stream-manager-sdk/index.js
---------                     -------
   387855                     9 files
```

3. Copie o artefato do SDK do Stream Manager para a pasta de artefatos do seu componente. Substitua *~/greengrass-components pelo* caminho para a pasta que você usa para desenvolvimento local.

   Linux or Unix

   ```
   cp stream-manager-sdk.zip ~/greengrass-components/artifacts/
   com.example.StreamManagerS3JS/1.0.0/
   ```

   Windows Command Prompt (CMD)

   ```
   robocopy . %USERPROFILE%\greengrass-components\artifacts
   \com.example.StreamManagerS3JS\1.0.0 stream-manager-sdk.zip
   ```

PowerShell

```
cp .\stream-manager-sdk.zip ~\greengrass-components\artifacts
\com.example.StreamManagerS3JS\1.0.0\
```

4. Crie sua receita de componentes. Na receita, faça o seguinte:

   a. Defina `stream-manager-sdk.zip` como um artefato.

   b. Defina seu JavaScript aplicativo como um artefato.

   c. No ciclo de vida da instalação, instale o SDK do Stream Manager a partir do artefato. `stream-manager-sdk.zip` Esse `npm install` comando cria uma `node_modules` pasta que contém o SDK do Stream Manager e suas dependências.

   d. No ciclo de vida de execução, anexe a `node_modules` pasta e execute seu `NODE_PATH` aplicativo. JavaScript

   Sua receita de componente pode ser parecida com o exemplo a seguir. Esse componente executa o exemplo do StreamManagerS3.

   JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.example.StreamManagerS3JS",
  "ComponentVersion": "1.0.0",
  "ComponentDescription": "Uses stream manager to upload a file to an S3
 bucket.",
  "ComponentPublisher": "Amazon",
  "ComponentDependencies": {
    "aws.greengrass.StreamManager": {
      "VersionRequirement": "^2.0.0"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "npm install {artifacts:decompressedPath}/stream-manager-sdk/
aws-greengrass-stream-manager-sdk",
```

```
        "run": "export NODE_PATH=$NODE_PATH:{work:path}/node_modules; node
 {artifacts:path}/index.js"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/stream-manager-sdk.zip",
          "Unarchive": "ZIP"
        },
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/index.js"
        }
      ]
    },
    {
      "Platform": {
        "os": "windows"
      },
      "Lifecycle": {
        "install": "npm install {artifacts:decompressedPath}/stream-manager-sdk/
aws-greengrass-stream-manager-sdk",
        "run": "set \"NODE_PATH=%NODE_PATH%;{work:path}/node_modules\" & node
 {artifacts:path}/index.js"
      },
      "Artifacts": [
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/stream-manager-sdk.zip",
          "Unarchive": "ZIP"
        },
        {
          "URI": "s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/index.js"
        }
      ]
    }
  ]
}
```

YAML

```
---
```

```
RecipeFormatVersion: '2020-01-25'
ComponentName: com.example.StreamManagerS3JS
ComponentVersion: 1.0.0
ComponentDescription: Uses stream manager to upload a file to an S3 bucket.
ComponentPublisher: Amazon
ComponentDependencies:
  aws.greengrass.StreamManager:
    VersionRequirement: "^2.0.0"
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: npm install {artifacts:decompressedPath}/stream-manager-sdk/aws-
greengrass-stream-manager-sdk
      run: |
        export NODE_PATH=$NODE_PATH:{work:path}/node_modules
        node {artifacts:path}/index.js
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/stream-manager-sdk.zip
        Unarchive: ZIP
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/index.js
  - Platform:
      os: windows
    Lifecycle:
      install: npm install {artifacts:decompressedPath}/stream-manager-sdk/aws-
greengrass-stream-manager-sdk
      run: |
        set "NODE_PATH=%NODE_PATH%;{work:path}/node_modules"
        node {artifacts:path}/index.js
    Artifacts:
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/stream-manager-sdk.zip
        Unarchive: ZIP
      - URI: s3://DOC-EXAMPLE-BUCKET/artifacts/
com.example.StreamManagerS3JS/1.0.0/index.js
```

Para obter mais informações sobre como desenvolver e testar componentes, consulteCrie AWS IoT Greengrass componentes.

Conecte-se ao gerenciador de streaming no código do aplicativo

Para se conectar ao gerenciador de streams em seu aplicativo, crie uma instância `StreamManagerClient` do SDK do Stream Manager. Esse cliente se conecta ao componente do gerenciador de fluxo em sua porta padrão 8088 ou na porta especificada por você. Para obter mais informações sobre como usar `StreamManagerClient` depois de criar uma instância, consulteUse StreamManagerClient para trabalhar com streams.

Example Exemplo: Conecte-se ao gerenciador de streams com a porta padrão

Java

```
import com.amazonaws.greengrass.streammanager.client.StreamManagerClient;

public class MyStreamManagerComponent {

    void connectToStreamManagerWithDefaultPort() {
        StreamManagerClient client = StreamManagerClientFactory.standard().build();

        // Use the client.
    }
}
```

Python

```
from stream_manager import (
    StreamManagerClient
)

def connect_to_stream_manager_with_default_port():
    client = StreamManagerClient()

    # Use the client.
```

JavaScript

```
const {
    StreamManagerClient
} = require('aws-greengrass-stream-manager-sdk');

function connectToStreamManagerWithDefaultPort() {
    const client = new StreamManagerClient();
```

```
    // Use the client.
}
```

Example Exemplo: Conecte-se ao gerenciador de streaming com porta não padrão

Se você configurar o stream manager com uma porta diferente da padrão, deverá usar a comunicação entre processos para recuperar a porta da configuração do componente.

> **Note**
>
> O parâmetro `port` de configuração contém o valor que você especifica `STREAM_MANAGER_SERVER_PORT` ao implantar o gerenciador de fluxo.

Java

```
void connectToStreamManagerWithCustomPort() {
    EventStreamRPCConnection eventStreamRpcConnection =
 IPCUtils.getEventStreamRpcConnection();
    GreengrassCoreIPCClient greengrassCoreIPCClient = new
 GreengrassCoreIPCClient(eventStreamRpcConnection);
    List<String> keyPath = new ArrayList<>();
    keyPath.add("port");

    GetConfigurationRequest request = new GetConfigurationRequest();
    request.setComponentName("aws.greengrass.StreamManager");
    request.setKeyPath(keyPath);
    GetConfigurationResponse response =
            greengrassCoreIPCClient.getConfiguration(request,
 Optional.empty()).getResponse().get();
    String port = response.getValue().get("port").toString();
    System.out.print("Stream Manager is running on port: " + port);

    final StreamManagerClientConfig config = StreamManagerClientConfig.builder()

 .serverInfo(StreamManagerServerInfo.builder().port(Integer.parseInt(port)).build()).build()

    StreamManagerClient client =
 StreamManagerClientFactory.standard().withClientConfig(config).build();
```

```
    // Use the client.
}
```

Python

```
import awsiot.greengrasscoreipc
from awsiot.greengrasscoreipc.model import (
    GetConfigurationRequest
)
from stream_manager import (
    StreamManagerClient
)

TIMEOUT = 10

def connect_to_stream_manager_with_custom_port():
    # Use IPC to get the port from the stream manager component configuration.
    ipc_client = awsiot.greengrasscoreipc.connect()
    request = GetConfigurationRequest()
    request.component_name = "aws.greengrass.StreamManager"
    request.key_path = ["port"]
    operation = ipc_client.new_get_configuration()
    operation.activate(request)
    future_response = operation.get_response()
    response = future_response.result(TIMEOUT)
    stream_manager_port = str(response.value["port"])

    # Use port to create a stream manager client.
    stream_client = StreamManagerClient(port=stream_manager_port)

    # Use the client.
```

Use StreamManagerClient para trabalhar com streams

Os componentes do Greengrass definidos pelo usuário que são executados no dispositivo principal do Greengrass podem usar `StreamManagerClient` o objeto no SDK do Stream Manager para criar fluxos no gerenciador de fluxos e depois interagir com os fluxos. Quando um componente cria um fluxo, ele define os Nuvem AWS destinos, a priorização e outras políticas de exportação e retenção de dados para o fluxo. Para enviar dados ao gerenciador de fluxo, os componentes anexam

os dados ao fluxo. Se um destino de exportação for definido para o fluxo, o gerenciador de fluxo exportará o fluxo automaticamente.

> **Note**
>
> Normalmente, os clientes do stream manager são componentes do Greengrass definidos pelo usuário. Se seu caso de negócios exigir isso, você também pode permitir que processos sem componentes executados no núcleo do Greengrass (por exemplo, um contêiner Docker) interajam com o gerenciador de stream. Para ter mais informações, consulte the section called “Autenticação de cliente”.

Os snippets neste tópico mostram como os clientes chamam o `StreamManagerClient` para trabalhar com fluxos. Para obter detalhes de implementação sobre os métodos e seus argumentos, use os links para a referência do SDK listada após cada snippet.

Se você usa o gerenciador de stream em uma função do Lambda, sua função do Lambda deve ser instanciada `StreamManagerClient` fora do manipulador da função. Se instanciado no manipulador, a função cria um `client` e uma conexão para o gerenciador de fluxo sempre que for invocado.

> **Note**
>
> Se você instanciar `StreamManagerClient` no manipulador, você deve chamar explicitamente o método `close()` quando o `client` concluir seu trabalho. Caso contrário, o `client` mantém a conexão aberta e outro thread em execução até que o script seja encerrado.

`StreamManagerClient` comporta as operações a seguir:

- the section called “Criar stream de mensagens”
- the section called “Anexar mensagem”
- the section called “Ler Mensagens”
- the section called “Listar streams”
- the section called “Descrever stream de mensagens”
- the section called “Atualize o fluxo de mensagens”

- the section called “Excluir stream de mensagens”

Criar stream de mensagens

Para criar um stream, um componente do Greengrass definido pelo usuário chama o método create e passa um objeto. `MessageStreamDefinition` Esse objeto especifica o nome exclusivo do fluxo e define como o gerenciador de fluxo deve lidar com novos dados quando o tamanho máximo do fluxo for atingido. Você pode usar `MessageStreamDefinition` e os tipos de dados (como `ExportDefinition`, `StrategyOnFull` e `Persistence`) para definir outras propriedades de fluxo. Isso inclui:

- O destino AWS IoT Analytics, o Kinesis Data Streams, o AWS IoT SiteWise e os destinos do Amazon S3, para exportações automáticas. Para ter mais informações, consulte the section called “Configurações de exportação para destinos de nuvem compatíveis”.
- Prioridade da exportação. O gerenciador de fluxo exporta fluxos de prioridade mais alta antes de fluxos de prioridade mais baixa.
- Tamanho máximo do lote e intervalo de lote para AWS IoT Analytics Kinesis Data Streams e destinos do AWS IoT SiteWise. O gerenciador de fluxo exporta mensagens quando qualquer condição é atendida.
- T ime-to-live (TTL). O tempo necessário para garantir que os dados de fluxo estejam disponíveis para processamento. Você deve certificar-se de que os dados podem ser consumidos nesse período de tempo. Esta não é uma política de exclusão. Os dados podem não ser excluídos imediatamente após o período de TTL.
- Persistência do fluxo. Selecione salvar fluxos no sistema de arquivos para persistir os dados nas reinicializações do núcleo ou salve os fluxos na memória.
- Número de sequência inicial. Especifique o número de sequência da mensagem a ser usada como mensagem inicial na exportação.

Para obter mais informações sobre `MessageStreamDefinition`, consulte a referência do SDK para a sua linguagem de destino:

- MessageStreamDefinitionno Java SDK
- MessageStreamDefinitionno SDK do Node.js
- MessageStreamDefinitionno SDK do Python

> **Note**
>
> O `StreamManagerClient` também fornece um destino alvo que você pode usar para exportar fluxos para um servidor HTTP. Este destino deve ser usado apenas para fins de teste. Ele não é estável e nem compatível para uso em ambientes de produção.

Depois que um stream é criado, seus componentes do Greengrass podem anexar mensagens ao stream para enviar dados para exportação e ler mensagens do stream para processamento local. O número de fluxos criados depende dos seus recursos de hardware e caso de negócios. Uma estratégia é criar um fluxo para cada canal de destino no AWS IoT Analytics ou no fluxo de dados do Kinesis, embora você possa definir vários destinos para um fluxo. Um fluxo tem longa duração.

Requisitos

Essa operação tem os seguintes requisitos:

- Versão mínima do SDK do Stream Manager: Python: 1.1.0 | Java: 1.1.0 | Node.js: 1.1.0

Exemplos

O snippet a seguir cria um fluxo chamado `StreamName`. Ele define as propriedades de fluxo em `MessageStreamDefinition` e nos tipos de dados subordinados.

Python

```
client = StreamManagerClient()

try:
    client.create_message_stream(MessageStreamDefinition(
        name="StreamName",    # Required.
        max_size=268435456,    # Default is 256 MB.
        stream_segment_size=16777216,    # Default is 16 MB.
        time_to_live_millis=None,    # By default, no TTL is enabled.
        strategy_on_full=StrategyOnFull.OverwriteOldestData,    # Required.
        persistence=Persistence.File,    # Default is File.
        flush_on_write=False,    # Default is false.
        export_definition=ExportDefinition(    # Optional. Choose where/how the
 stream is exported to the Nuvem AWS.
            kinesis=None,
```

```
            iot_analytics=None,
            iot_sitewise=None,
            s3_task_executor=None
        )
    ))
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Referência do SDK do Python: create_message_stream | MessageStreamDefinition

Java

```
try (final StreamManagerClient client =
 StreamManagerClientFactory.standard().build()) {
    client.createMessageStream(
            new MessageStreamDefinition()
                    .withName("StreamName") // Required.
                    .withMaxSize(268435456L)    // Default is 256 MB.
                    .withStreamSegmentSize(16777216L)    // Default is 16 MB.
                    .withTimeToLiveMillis(null)    // By default, no TTL is enabled.
                    .withStrategyOnFull(StrategyOnFull.OverwriteOldestData)    //
 Required.
                    .withPersistence(Persistence.File)    // Default is File.
                    .withFlushOnWrite(false)    // Default is false.
                    .withExportDefinition(    // Optional. Choose where/how the
 stream is exported to the Nuvem AWS.
                            new ExportDefinition()
                                    .withKinesis(null)
                                    .withIotAnalytics(null)
                                    .withIotSitewise(null)
                                    .withS3(null)
                    )

    );
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Referência do SDK Java: | createMessageStreamMessageStreamDefinition

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
  try {
    await client.createMessageStream(
      new MessageStreamDefinition()
        .withName("StreamName") // Required.
        .withMaxSize(268435456)  // Default is 256 MB.
        .withStreamSegmentSize(16777216)  // Default is 16 MB.
        .withTimeToLiveMillis(null)  // By default, no TTL is enabled.
        .withStrategyOnFull(StrategyOnFull.OverwriteOldestData)  // Required.
        .withPersistence(Persistence.File)  // Default is File.
        .withFlushOnWrite(false)  // Default is false.
        .withExportDefinition(  // Optional. Choose where/how the stream is exported
 to the Nuvem AWS.
          new ExportDefinition()
            .withKinesis(null)
            .withIotAnalytics(null)
            .withIotSiteWise(null)
            .withS3(null)
        )
    );
  } catch (e) {
    // Properly handle errors.
  }
});
client.onError((err) => {
  // Properly handle connection errors.
  // This is called only when the connection to the StreamManager server fails.
});
```

Referência do SDK do Node.js: | createMessageStreamMessageStreamDefinition

Para obter mais informações sobre como configurar destinos de exportação, consulte the section called “Configurações de exportação para destinos de nuvem compatíveis”.

Anexar mensagem

Para enviar dados ao stream manager para exportação, seus componentes do Greengrass anexam os dados ao stream de destino. O destino da exportação determina o tipo de dados a ser passado para esse método.

Requisitos

Essa operação tem os seguintes requisitos:

- Versão mínima do SDK do Stream Manager: Python: 1.1.0 | Java: 1.1.0 | Node.js: 1.1.0

Exemplos

Destinos de exportação do AWS IoT Analytics ou do Kinesis Data Streams

O snippet a seguir anexa uma mensagem ao fluxo chamado `StreamName`. Para AWS IoT Analytics nossos destinos do Kinesis Data Streams, seus componentes do Greengrass acrescentam um blob de dados.

Esse snippet tem os seguintes requisitos:

- Versão mínima do SDK do Stream Manager: Python: 1.1.0 | Java: 1.1.0 | Node.js: 1.1.0

Python

```
client = StreamManagerClient()

try:
    sequence_number = client.append_message(stream_name="StreamName",
 data=b'Arbitrary bytes data')
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Referência do SDK em Python: append_message

Java

```
try (final StreamManagerClient client =
 StreamManagerClientFactory.standard().build()) {
    long sequenceNumber = client.appendMessage("StreamName", "Arbitrary byte
 array".getBytes());
} catch (StreamManagerException e) {
```

```
    // Properly handle exception.
}
```

Referência do SDK em Java: appendMessage

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        const sequenceNumber = await client.appendMessage("StreamName",
 Buffer.from("Arbitrary byte array"));
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Referência do SDK em Node.js: appendMessage

Destinos de exportação do AWS IoT SiteWise

O snippet a seguir anexa uma mensagem ao fluxo chamado `StreamName`. Para AWS IoT SiteWise destinos, seus componentes do Greengrass acrescentam um objeto serializado. `PutAssetPropertyValueEntry` Para ter mais informações, consulte the section called "Exportando para o AWS IoT SiteWise".

> **Note**
>
> Ao enviar dados para o AWS IoT SiteWise, os dados devem atender aos requisitos da ação `BatchPutAssetPropertyValue`. Para obter mais informações, consulte BatchPutAssetPropertyValue na Referência da API do AWS IoT SiteWise.

Esse snippet tem os seguintes requisitos:

- Versão mínima do SDK do Stream Manager: Python: 1.1.0 | Java: 1.1.0 | Node.js: 1.1.0

Python

```
client = StreamManagerClient()

try:
    # SiteWise requires unique timestamps in all messages and also needs timestamps
 not earlier
    # than 10 minutes in the past. Add some randomness to time and offset.

    # Note: To create a new asset property data, you should use the classes defined
 in the
    # greengrasssdk.stream_manager module.

    time_in_nanos = TimeInNanos(
        time_in_seconds=calendar.timegm(time.gmtime()) - random.randint(0, 60),
 offset_in_nanos=random.randint(0, 10000)
    )
    variant = Variant(double_value=random.random())
    asset = [AssetPropertyValue(value=variant, quality=Quality.GOOD,
 timestamp=time_in_nanos)]
    putAssetPropertyValueEntry =
 PutAssetPropertyValueEntry(entry_id=str(uuid.uuid4()),
 property_alias="PropertyAlias", property_values=asset)
    sequence_number = client.append_message(stream_name="StreamName",
 Util.validate_and_serialize_to_json_bytes(putAssetPropertyValueEntry))
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Referência do SDK do Python: append_message | PutAssetPropertyValueEntry

Java

```
try (final StreamManagerClient client =
 GreengrassClientBuilder.streamManagerClient().build()) {
    Random rand = new Random();
    // Note: To create a new asset property data, you should use the classes defined
 in the
    // com.amazonaws.greengrass.streammanager.model.sitewise package.
    List<AssetPropertyValue> entries = new ArrayList<>() ;
```

```
    // IoTSiteWise requires unique timestamps in all messages and also needs
 timestamps not earlier
    // than 10 minutes in the past. Add some randomness to time and offset.
    final int maxTimeRandomness = 60;
    final int maxOffsetRandomness = 10000;
    double randomValue = rand.nextDouble();
    TimeInNanos timestamp = new TimeInNanos()
            .withTimeInSeconds(Instant.now().getEpochSecond() -
 rand.nextInt(maxTimeRandomness))
            .withOffsetInNanos((long) (rand.nextInt(maxOffsetRandomness)));
    AssetPropertyValue entry = new AssetPropertyValue()
            .withValue(new Variant().withDoubleValue(randomValue))
            .withQuality(Quality.GOOD)
            .withTimestamp(timestamp);
    entries.add(entry);

    PutAssetPropertyValueEntry putAssetPropertyValueEntry = new
 PutAssetPropertyValueEntry()
            .withEntryId(UUID.randomUUID().toString())
            .withPropertyAlias("PropertyAlias")
            .withPropertyValues(entries);
    long sequenceNumber = client.appendMessage("StreamName",
 ValidateAndSerialize.validateAndSerializeToJsonBytes(putAssetPropertyValueEntry));
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Referência do SDK Java: appendMessage | PutAssetPropertyValueEntry

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        const maxTimeRandomness = 60;
        const maxOffsetRandomness = 10000;
        const randomValue = Math.random();
        // Note: To create a new asset property data, you should use the classes
 defined in the
        // aws-greengrass-core-sdk StreamManager module.
        const timestamp = new TimeInNanos()
            .withTimeInSeconds(Math.round(Date.now() / 1000) -
 Math.floor(Math.random() - maxTimeRandomness))
            .withOffsetInNanos(Math.floor(Math.random() * maxOffsetRandomness));
```

```
        const entry = new AssetPropertyValue()
            .withValue(new Variant().withDoubleValue(randomValue))
            .withQuality(Quality.GOOD)
            .withTimestamp(timestamp);

        const putAssetPropertyValueEntry =    new PutAssetPropertyValueEntry()
            .withEntryId(`${ENTRY_ID_PREFIX}${i}`)
            .withPropertyAlias("PropertyAlias")
            .withPropertyValues([entry]);
        const sequenceNumber = await client.appendMessage("StreamName",
 util.validateAndSerializeToJsonBytes(putAssetPropertyValueEntry));
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Referência do SDK do Node.js: appendMessage | PutAssetPropertyValueEntry

Destinos de exportação do Amazon S3

O snippet a seguir anexa uma tarefa de exportação ao fluxo chamada `StreamName`. Para destinos do Amazon S3, seus componentes do Greengrass acrescentam um objeto serializado que contém informações sobre o arquivo de entrada de origem e o `S3ExportTaskDefinition` objeto Amazon S3 de destino. Se o objeto especificado não existir, o gerenciador de fluxo criará o objeto para você. Para ter mais informações, consulte the section called “Exportar para o Amazon S3”.

Esse snippet tem os seguintes requisitos:

- Versão mínima do SDK do Stream Manager: Python: 1.1.0 | Java: 1.1.0 | Node.js: 1.1.0

Python

```
client = StreamManagerClient()

try:
    # Append an Amazon S3 Task definition and print the sequence number.
```

```
    s3_export_task_definition = S3ExportTaskDefinition(input_url="URLToFile",
 bucket="BucketName", key="KeyName")
    sequence_number = client.append_message(stream_name="StreamName",
 Util.validate_and_serialize_to_json_bytes(s3_export_task_definition))
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Referência do SDK do Python: append_message | S3 ExportTaskDefinition

Java

```
try (final StreamManagerClient client =
 GreengrassClientBuilder.streamManagerClient().build()) {
    // Append an Amazon S3 export task definition and print the sequence number.
    S3ExportTaskDefinition s3ExportTaskDefinition = new S3ExportTaskDefinition()
        .withBucket("BucketName")
        .withKey("KeyName")
        .withInputUrl("URLToFile");
    long sequenceNumber = client.appendMessage("StreamName",
 ValidateAndSerialize.validateAndSerializeToJsonBytes(s3ExportTaskDefinition));
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Referência do SDK Java: appendMessage | S3 ExportTaskDefinition

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
     // Append an Amazon S3 export task definition and print the sequence number.
     const taskDefinition = new S3ExportTaskDefinition()
        .withBucket("BucketName")
        .withKey("KeyName")
        .withInputUrl("URLToFile");
        const sequenceNumber = await client.appendMessage("StreamName",
 util.validateAndSerializeToJsonBytes(taskDefinition)));
    } catch (e) {
        // Properly handle errors.
```

```
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Referência do SDK do Node.js: appendMessage | S3 ExportTaskDefinition

Ler Mensagens

Ler mensagens de um fluxo.

Requisitos

Essa operação tem os seguintes requisitos:

- Versão mínima do SDK do Stream Manager: Python: 1.1.0 | Java: 1.1.0 | Node.js: 1.1.0

Exemplos

O snippet a seguir lê mensagens do fluxo chamado `StreamName`. O método de leitura usa um objeto `ReadMessagesOptions` opcional que especifica o número de sequência a partir do qual começar a ler, os números mínimo e máximo a ler e um tempo limite para ler mensagens.

Python

```
client = StreamManagerClient()

try:
    message_list = client.read_messages(
        stream_name="StreamName",
        # By default, if no options are specified, it tries to read one message from
 the beginning of the stream.
        options=ReadMessagesOptions(
            desired_start_sequence_number=100,
            # Try to read from sequence number 100 or greater. By default, this is
 0.
            min_message_count=10,
            # Try to read 10 messages. If 10 messages are not available, then
 NotEnoughMessagesException is raised. By default, this is 1.
```

```
            max_message_count=100,    # Accept up to 100 messages. By default this
 is 1.
            read_timeout_millis=5000
            # Try to wait at most 5 seconds for the min_messsage_count to be
 fulfilled. By default, this is 0, which immediately returns the messages or an
 exception.
        )
    )
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Referência do SDK do Python: read_messages | ReadMessagesOptions

Java

```
try (final StreamManagerClient client =
 StreamManagerClientFactory.standard().build()) {
    List<Message> messages = client.readMessages("StreamName",
            // By default, if no options are specified, it tries to read one message
 from the beginning of the stream.
            new ReadMessagesOptions()
                    // Try to read from sequence number 100 or greater. By default
 this is 0.
                    .withDesiredStartSequenceNumber(100L)
                    // Try to read 10 messages. If 10 messages are not available,
 then NotEnoughMessagesException is raised. By default, this is 1.
                    .withMinMessageCount(10L)
                    // Accept up to 100 messages. By default this is 1.
                    .withMaxMessageCount(100L)
                    // Try to wait at most 5 seconds for the min_messsage_count to
 be fulfilled. By default, this is 0, which immediately returns the messages or an
 exception.
                    .withReadTimeoutMillis(Duration.ofSeconds(5L).toMillis())
    );
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Referência do SDK Java: readMessages | ReadMessagesOptions

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        const messages = await client.readMessages("StreamName",
            // By default, if no options are specified, it tries to read one message
 from the beginning of the stream.
            new ReadMessagesOptions()
                // Try to read from sequence number 100 or greater. By default this
 is 0.
                .withDesiredStartSequenceNumber(100)
                // Try to read 10 messages. If 10 messages are not available, then
 NotEnoughMessagesException is thrown. By default, this is 1.
                .withMinMessageCount(10)
                // Accept up to 100 messages. By default this is 1.
                .withMaxMessageCount(100)
                // Try to wait at most 5 seconds for the minMessageCount to be
 fulfilled. By default, this is 0, which immediately returns the messages or an
 exception.
                .withReadTimeoutMillis(5 * 1000)
        );
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Referência do SDK do Node.js: readMessages | ReadMessagesOptions

Listar streams

Obtenha a lista de fluxos no gerenciador de fluxos.

Requisitos

Essa operação tem os seguintes requisitos:

- Versão mínima do SDK do Stream Manager: Python: 1.1.0 | Java: 1.1.0 | Node.js: 1.1.0

Exemplos

O snippet a seguir obtém uma lista dos fluxos (por nome) no gerenciador de fluxo.

Python

```
client = StreamManagerClient()

try:
    stream_names = client.list_streams()
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Referência do SDK em Python: list_streams

Java

```
try (final StreamManagerClient client =
 StreamManagerClientFactory.standard().build()) {
    List<String> streamNames = client.listStreams();
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Referência do SDK em Java: ListStreams

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        const streams = await client.listStreams();
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
```

```
});
```

Referência do SDK em Node.js: listStreams

Descrever stream de mensagens

Obtenha metadados sobre um fluxo, incluindo a definição, o tamanho e o status de exportação do fluxo.

Requisitos

Essa operação tem os seguintes requisitos:

- Versão mínima do SDK do Stream Manager: Python: 1.1.0 | Java: 1.1.0 | Node.js: 1.1.0

Exemplos

O snippet a seguir obtém metadados sobre o fluxo chamado `StreamName`, incluindo a definição, o tamanho e o status do exportador do fluxo.

Python

```
client = StreamManagerClient()

try:
    stream_description = client.describe_message_stream(stream_name="StreamName")
    if stream_description.export_statuses[0].error_message:
        # The last export of export destination 0 failed with some error
        # Here is the last sequence number that was successfully exported
        stream_description.export_statuses[0].last_exported_sequence_number

    if (stream_description.storage_status.newest_sequence_number >
            stream_description.export_statuses[0].last_exported_sequence_number):
        pass
        # The end of the stream is ahead of the last exported sequence number
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Referência do SDK em Python: describe_message_stream

Java

```
try (final StreamManagerClient client =
 StreamManagerClientFactory.standard().build()) {
    MessageStreamInfo description = client.describeMessageStream("StreamName");
    String lastErrorMessage =
 description.getExportStatuses().get(0).getErrorMessage();
    if (lastErrorMessage != null && !lastErrorMessage.equals("")) {
        // The last export of export destination 0 failed with some error.
        // Here is the last sequence number that was successfully exported.
        description.getExportStatuses().get(0).getLastExportedSequenceNumber();
    }

    if (description.getStorageStatus().getNewestSequenceNumber() >
            description.getExportStatuses().get(0).getLastExportedSequenceNumber())
 {
        // The end of the stream is ahead of the last exported sequence number.
    }
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Referência do Java SDK: describeMessageStream

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        const description = await client.describeMessageStream("StreamName");
        const lastErrorMessage = description.exportStatuses[0].errorMessage;
        if (lastErrorMessage) {
            // The last export of export destination 0 failed with some error.
            // Here is the last sequence number that was successfully exported.
            description.exportStatuses[0].lastExportedSequenceNumber;
        }

        if (description.storageStatus.newestSequenceNumber >
            description.exportStatuses[0].lastExportedSequenceNumber) {
            // The end of the stream is ahead of the last exported sequence number.
        }
    } catch (e) {
```

```
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Referência do SDK do Node.js: describeMessageStream

Atualize o fluxo de mensagens

Atualize as propriedades de um fluxo existente. Talvez você queira atualizar um fluxo se seus requisitos mudarem após a criação do fluxo. Por exemplo: .

- Adicione uma nova configuração de exportação para um destino Nuvem AWS.
- Aumente o tamanho máximo de um fluxo para alterar a forma como os dados são exportados ou retidos. Por exemplo, o tamanho do fluxo em combinação com sua estratégia em configurações completas pode resultar na exclusão ou rejeição dos dados antes que o gerenciador de fluxo possa processá-los.
- Pause e retome as exportações; por exemplo, se as tarefas de exportação forem demoradas e você quiser racionar seus dados de upload.

Seus componentes do Greengrass seguem esse processo de alto nível para atualizar um stream:

1. Obter a descrição do fluxo.
2. Atualizar as propriedades de destino nos objetos correspondentes `MessageStreamDefinition` e subordinados.
3. Passar o atualizado `MessageStreamDefinition`. Certifique-se de incluir as definições completas do objeto para o fluxo atualizado. As propriedades indefinidas revertem para os valores padrão.

   Você pode especificar o número de sequência da mensagem a ser usada como mensagem inicial na exportação.

Requisitos

Essa operação tem os seguintes requisitos:

- Versão mínima do SDK do Stream Manager: Python: 1.1.0 | Java: 1.1.0 | Node.js: 1.1.0

Exemplos

O snippet a seguir atualiza o fluxo chamado `StreamName`. Ele atualiza várias propriedades de um fluxo que é exportado para o Kinesis Data Streams.

Python

```
client = StreamManagerClient()

try:
    message_stream_info = client.describe_message_stream(STREAM_NAME)
    message_stream_info.definition.max_size=536870912
    message_stream_info.definition.stream_segment_size=33554432
    message_stream_info.definition.time_to_live_millis=3600000
    message_stream_info.definition.strategy_on_full=StrategyOnFull.RejectNewData
    message_stream_info.definition.persistence=Persistence.Memory
    message_stream_info.definition.flush_on_write=False
    message_stream_info.definition.export_definition.kinesis=
        [KinesisConfig(
            # Updating Export definition to add a Kinesis Stream configuration.
            identifier=str(uuid.uuid4()), kinesis_stream_name=str(uuid.uuid4()))]
    client.update_message_stream(message_stream_info.definition)
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Referência do SDK do Python: | updateMessageStreamMessageStreamDefinition

Java

```
try (final StreamManagerClient client =
 GreengrassClientBuilder.streamManagerClient().build()) {
    MessageStreamInfo messageStreamInfo = client.describeMessageStream(STREAM_NAME);
    // Update the message stream with new values.
```

```
    client.updateMessageStream(
        messageStreamInfo.getDefinition()
            .withStrategyOnFull(StrategyOnFull.RejectNewData) // Required. Updating
 Strategy on full to reject new data.
           // Max Size update should be greater than initial Max Size defined in
 Create Message Stream request
            .withMaxSize(536870912L) // Update Max Size to 512 MB.
            .withStreamSegmentSize(33554432L) // Update Segment Size to 32 MB.
            .withFlushOnWrite(true) // Update flush on write to true.
            .withPersistence(Persistence.Memory) // Update the persistence to
 Memory.
            .withTimeToLiveMillis(3600000L)    // Update TTL to 1 hour.
            .withExportDefinition(
                // Optional. Choose where/how the stream is exported to the Nuvem
 AWS.
                messageStreamInfo.getDefinition().getExportDefinition().
                    // Updating Export definition to add a Kinesis Stream
 configuration.
                    .withKinesis(new ArrayList<KinesisConfig>() {{
                        add(new KinesisConfig()
                            .withIdentifier(EXPORT_IDENTIFIER)
                            .withKinesisStreamName("test"));
                        }})
            );
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Referência do SDK Java: update_message_stream | MessageStreamDefinition

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        const messageStreamInfo = await c.describeMessageStream(STREAM_NAME);
        await client.updateMessageStream(
            messageStreamInfo.definition
                // Max Size update should be greater than initial Max Size defined
 in Create Message Stream request
                .withMaxSize(536870912)    // Default is 256 MB. Updating Max Size
 to 512 MB.
                .withStreamSegmentSize(33554432)    // Default is 16 MB. Updating
 Segment Size to 32 MB.
```

```
                .withTimeToLiveMillis(3600000)    // By default, no TTL is enabled.
 Update TTL to 1 hour.
                .withStrategyOnFull(StrategyOnFull.RejectNewData)    // Required.
 Updating Strategy on full to reject new data.
                .withPersistence(Persistence.Memory)    // Default is File. Update
 the persistence to Memory
                .withFlushOnWrite(true)    // Default is false. Updating to true.
                .withExportDefinition(
                    // Optional. Choose where/how the stream is exported to the
 Nuvem AWS.
                    messageStreamInfo.definition.exportDefinition
                        // Updating Export definition to add a Kinesis Stream
 configuration.
                        .withKinesis([new
 KinesisConfig().withIdentifier(uuidv4()).withKinesisStreamName(uuidv4())])
                )
        );
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Referência do SDK do Node.js: | updateMessageStreamMessageStreamDefinition

Restrições para a atualização de fluxos

As restrições a seguir se aplicam ao atualizar fluxos. A menos que indicado na lista a seguir, as atualizações entrarão em vigor imediatamente.

- Não é possível atualizar a persistência de um fluxo. Para alterar esse comportamento, exclua o fluxo e crie um fluxo que defina a nova política de persistência.
- Você só pode atualizar o tamanho máximo de um fluxo sob as seguintes condições:
  - O tamanho máximo deve ser maior que o tamanho atual do fluxo. Para encontrar essas informações, descreva o fluxo e, em seguida, verifique o status de armazenamento do objeto `MessageStreamInfo` retornado.
  - O tamanho máximo deve ser maior ou igual ao tamanho do segmento do fluxo.

- Você pode atualizar o tamanho do segmento do fluxo para um valor menor que o tamanho máximo do fluxo. A configuração atualizada se aplica aos novos segmentos.
- As atualizações da propriedade tempo de vida (TTL) se aplicam às novas operações de anexação. Se você diminuir esse valor, o gerenciador de fluxo também poderá excluir segmentos existentes que excedam o TTL.
- As atualizações da estratégia em toda a propriedade se aplicam às novas operações de anexação. Se você definir a estratégia para substituir os dados mais antigos, o gerenciador de fluxo também poderá substituir os segmentos existentes com base na nova configuração.
- As atualizações na propriedade “descartar após gravação” se aplicam às novas mensagens.
- As atualizações nas configurações de exportação se aplicam às novas exportações. A solicitação de atualização deve incluir todas as configurações de exportação às quais você deseja oferecer suporte. Caso contrário, o gerenciador de fluxo as excluirá.
  - Ao atualizar uma configuração de exportação, especifique o identificador da configuração de exportação de destino.
  - Para adicionar uma configuração de exportação, especifique um identificador exclusivo para a nova configuração de exportação.
  - Para excluir uma configuração de exportação, omita a configuração de exportação.
- Para atualizar o número da sequência inicial de uma configuração de exportação em um fluxo, você deve especificar um valor menor que o número de sequência mais recente. Para encontrar essas informações, descreva o fluxo e, em seguida, verifique o status de armazenamento do objeto `MessageStreamInfo` retornado.

Excluir stream de mensagens

Exclui um fluxo. Quando você exclui um fluxo, todos os dados armazenados para o fluxo são excluídos do disco.

Requisitos

Essa operação tem os seguintes requisitos:

- Versão mínima do SDK do Stream Manager: Python: 1.1.0 | Java: 1.1.0 | Node.js: 1.1.0

Exemplos

O snippet a seguir exclui o fluxo chamado `StreamName`.

Python

```
client = StreamManagerClient()

try:
    client.delete_message_stream(stream_name="StreamName")
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

Referência do SDK do Python: deleteMessageStream

Java

```
try (final StreamManagerClient client =
 StreamManagerClientFactory.standard().build()) {
    client.deleteMessageStream("StreamName");
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Referência do SDK em Java: delete_message_stream

Node.js

```
const client = new StreamManagerClient();
client.onConnected(async () => {
    try {
        await client.deleteMessageStream("StreamName");
    } catch (e) {
        // Properly handle errors.
    }
});
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

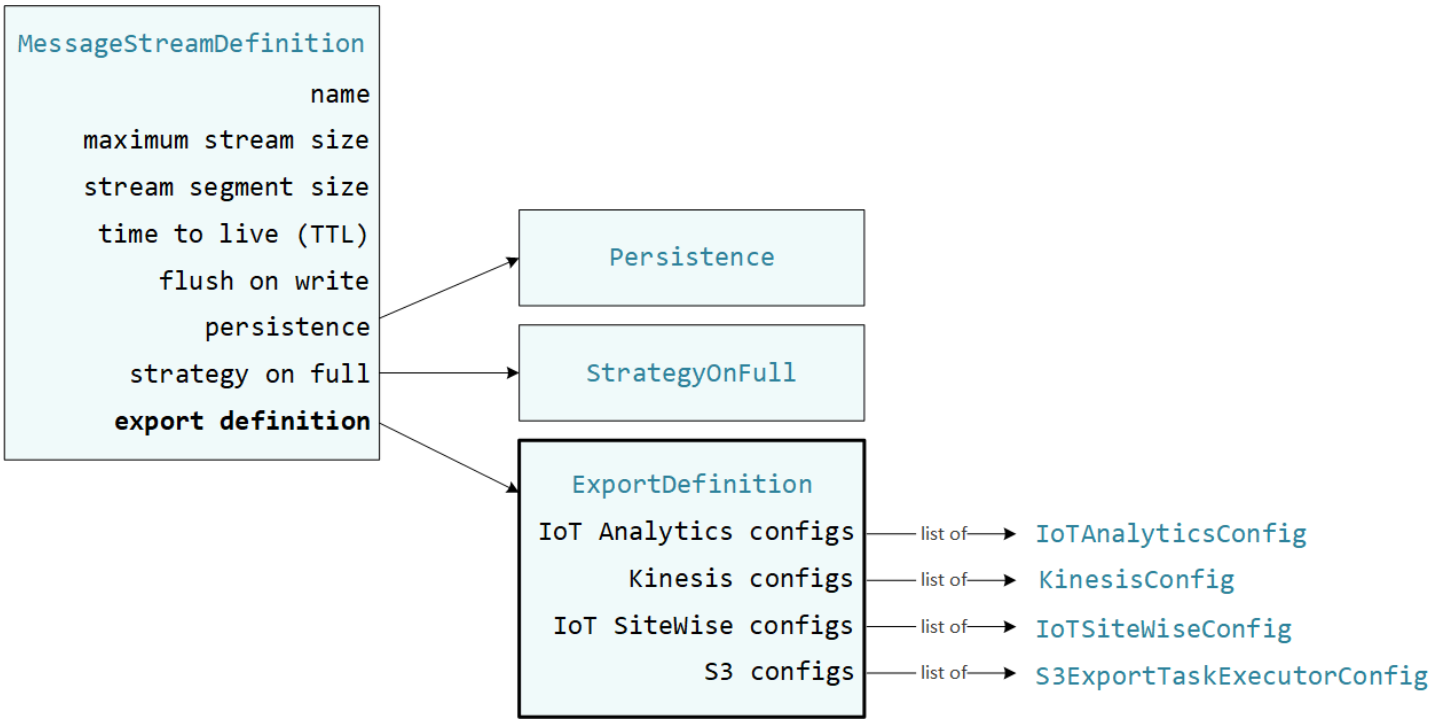
Referência do SDK do Node.js: deleteMessageStream

Consulte também

- Gerencie fluxos de dados nos dispositivos principais do Greengrass
- Configurar o gerenciador de fluxo do AWS IoT Greengrass
- Configurações de exportação para destinos compatíveis do Nuvem AWS
- `StreamManagerClient`na referência do SDK do Stream Manager:
  - Python
  - Java
  - Node.js

Configurações de exportação para destinos compatíveis do Nuvem AWS

Os componentes do Greengrass definidos pelo usuário são `StreamManagerClient` usados no SDK do Stream Manager para interagir com o stream manager. Quando um componente cria um fluxo ou atualiza um fluxo, ele passa um `MessageStreamDefinition` objeto que representa as propriedades do fluxo, incluindo a definição de exportação. O objeto `ExportDefinition` contém as configurações de exportação definidas para o fluxo. O gerenciador de fluxo usa essas configurações de exportação para determinar onde e como exportar o fluxo.



Você pode definir zero ou mais configurações de exportação em um fluxo, incluindo várias configurações de exportação para um único tipo de destino. Por exemplo, você pode exportar um fluxo para dois canais do AWS IoT Analytics e um fluxo de dados do Kinesis.

Para tentativas de exportação malsucedidas, o gerenciador de fluxo tenta continuamente exportar dados para a Nuvem AWS em intervalos de até cinco minutos. Não há um limite máximo para o número de novas tentativas.

> **Note**
>
> O `StreamManagerClient` também fornece um destino alvo que você pode usar para exportar fluxos para um servidor HTTP. Este destino deve ser usado apenas para fins de teste. Ele não é estável e nem compatível para uso em ambientes de produção.

Destinos Nuvem AWS compatíveis

- Canais do AWS IoT Analytics
- Amazon Kinesis Data Streams
- Propriedades do ativo AWS IoT SiteWise
- Objetos do Amazon S3

Você é responsável pela manutenção desses recursos da Nuvem AWS.

Canais do AWS IoT Analytics

O gerenciador de fluxo fornece suporte a exportações automáticas para o AWS IoT Analytics. O AWS IoT Analytics permite realizar análises avançadas em seus dados para ajudar a tomar decisões de negócios e aprimorar os modelos de machine learning. Para obter mais informações, consulte O que é o AWS IoT Analytics? no AWS IoT Analytics Guia do usuário do .

No SDK do Stream Manager, seus componentes do Greengrass usam `IoTAnalyticsConfig` o para definir a configuração de exportação para esse tipo de destino. Para mais informações, consulte a referência do SDK para seu idioma de destino:

- IoT AnalyticsConfig no SDK do Python
- IoT AnalyticsConfig no Java SDK

- IoT AnalyticsConfig no SDK do Node.js

Requisitos

Esse destino de exportação tem os seguintes requisitos:

- Os canais de destino AWS IoT Analytics devem estar no mesmo dispositivo principal Conta da AWS e Região da AWS no dispositivo principal do Greengrass.
- O Autorize os dispositivos principais a interagir com os serviços AWS deve conceder a permissão `iotanalytics:BatchPutMessage` para os canais de destino. Por exemplo: .

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iotanalytics:BatchPutMessage"
      ],
      "Resource": [
        "arn:aws:iotanalytics:region:account-id:channel/channel_1_name",
        "arn:aws:iotanalytics:region:account-id:channel/channel_2_name"
      ]
    }
  ]
}
```

Você pode conceder acesso granular ou condicional aos recursos, por exemplo, usando um esquema de nomeação * curinga. Para obter mais informações, consulte Adicionando e removendo políticas do IAM no Guia do usuário do IAM.

Exportando para o AWS IoT Analytics

Para criar um fluxo que exporta paraAWS IoT Analytics, seus componentes do Greengrass criam um fluxo com uma definição de exportação que inclui um ou mais `IoTAnalyticsConfig` objetos. Esse objeto define as configurações de exportação, como canal de destino, tamanho do lote, intervalo do lote e prioridade.

Quando seus componentes do Greengrass recebem dados de dispositivos, eles acrescentam mensagens que contêm uma bolha de dados ao stream de destino.

Em seguida, o gerenciador de fluxo exporta os dados com base nas configurações de lote e na prioridade definidas nas configurações de exportação do fluxo.

Amazon Kinesis Data Streams

O gerenciador de fluxos é compatível com exportações automáticas para o Amazon Kinesis Data Streams. O Kinesis Data Streams é comumente usado para agregar dados de alto volume e carregá-los em um data warehouse ou cluster. MapReduce Para obter mais informações, consulte O que é o Amazon Kinesis Data Streams? no Guia do desenvolvedor do Amazon Kinesis.

No SDK do Stream Manager, seus componentes do Greengrass usam `KinesisConfig` o para definir a configuração de exportação para esse tipo de destino. Para mais informações, consulte a referência do SDK para seu idioma de destino:

- KinesisConfigno SDK do Python
- KinesisConfigno Java SDK
- KinesisConfigno SDK do Node.js

Requisitos

Esse destino de exportação tem os seguintes requisitos:

- Os streams de destino no Kinesis Data Streams devem estar no mesmo dispositivo principal do Conta da AWS Região da AWS Greengrass.
- O Autorize os dispositivos principais a interagir com os serviços AWS deve conceder a permissão `kinesis:PutRecords` para os fluxos de dados de destino. Por exemplo: .

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kinesis:PutRecords"
      ],
      "Resource": [
        "arn:aws:kinesis:region:account-id:stream/stream_1_name",
        "arn:aws:kinesis:region:account-id:stream/stream_2_name"
      ]
    }
```

```
  ]
}
```

Você pode conceder acesso granular ou condicional aos recursos, por exemplo, usando um esquema de nomeação * curinga. Para obter mais informações, consulte Adicionando e removendo políticas do IAM no Guia do usuário do IAM.

Exportação do Kinesis Data Streams

Para criar um stream que exporte para o Kinesis Data Streams, seus componentes do Greengrass criam um stream com uma definição de exportação que inclui um ou mais objetos. `KinesisConfig` Esse objeto define as configurações de exportação, como fluxo de dados, tamanho do lote, intervalo do lote e prioridade.

Quando seus componentes do Greengrass recebem dados de dispositivos, eles acrescentam mensagens que contêm uma bolha de dados ao stream de destino. Em seguida, o gerenciador de fluxo exporta os dados com base nas configurações de lote e na prioridade definidas nas configurações de exportação do fluxo.

O gerenciador de fluxo gera uma UUID exclusiva e aleatória como chave de partição para cada registro carregado no Amazon Kinesis.

Propriedades do ativo AWS IoT SiteWise

O gerenciador de fluxo fornece suporte a exportações automáticas para o AWS IoT SiteWise. O AWS IoT SiteWise permite coletar, organizar e analisar dados de equipamentos industriais em escala. Para mais informações, consulte O que é o AWS IoT SiteWise? no AWS IoT SiteWise Guia do usuário.

No SDK do Stream Manager, seus componentes do Greengrass usam `IoTSiteWiseConfig` o para definir a configuração de exportação para esse tipo de destino. Para mais informações, consulte a referência do SDK para seu idioma de destino:

- IoT SiteWiseConfig no SDK do Python
- IoT SiteWiseConfig no Java SDK
- IoT SiteWiseConfig no SDK do Node.js

> **Note**
>
> AWStambém fornece AWS IoT SiteWise componentes, que oferecem uma solução pré-construída que você pode usar para transmitir dados de fontes OPC-UA. Para ter mais informações, consulte Coletor IoT OPC-UA SiteWise .

Requisitos

Esse destino de exportação tem os seguintes requisitos:

- As propriedades do ativo de destino AWS IoT SiteWise devem estar no mesmo Conta da AWS Região da AWS dispositivo central do Greengrass.

  > **Note**
  >
  > Para ver a lista de Região da AWS s que AWS IoT SiteWise oferecem suporte, consulte AWS IoT SiteWiseendpoints e cotas na Referência AWSgeral.

- O Autorize os dispositivos principais a interagir com os serviços AWS deve conceder a permissão `iotsitewise:BatchPutAssetPropertyValue` para as propriedades do ativo do destino. O exemplo de política a seguir usa a chave de condição `iotsitewise:assetHierarchyPath` para conceder acesso a um ativo raiz de destino e seus ativos secundários. É possível remover o `Condition` da política para conceder acesso a todos os seus ativos AWS IoT SiteWise, ou especificar ARNs para determinados ativos.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
       "Effect": "Allow",
       "Action": "iotsitewise:BatchPutAssetPropertyValue",
       "Resource": "*",
       "Condition": {
         "StringLike": {
           "iotsitewise:assetHierarchyPath": [
             "/root node asset ID",
             "/root node asset ID/*"
           ]
         }
```

```
        }
      }
    ]
}
```

Você pode conceder acesso granular ou condicional aos recursos, por exemplo, usando um esquema de nomeação * curinga. Para obter mais informações, consulte Adicionando e removendo políticas do IAM no Guia do usuário do IAM.

Para obter informações de segurança importantes, consulte a BatchPutAssetPropertyValue autorização no Guia AWS IoT SiteWise do usuário.

Exportando para o AWS IoT SiteWise

Para criar um fluxo que exporta paraAWS IoT SiteWise, seus componentes do Greengrass criam um fluxo com uma definição de exportação que inclui um ou mais `IoTSiteWiseConfig` objetos. Esse objeto define as configurações de exportação, como tamanho do lote, intervalo do lote e prioridade.

Quando seus componentes do Greengrass recebem dados de propriedades de ativos dos dispositivos, eles acrescentam mensagens que contêm os dados ao stream de destino. As mensagens são objetos `PutAssetPropertyValueEntry` serializados em JSON que contêm valores de propriedade para uma ou mais propriedades de ativos. Para obter mais informações, consulte Anexar mensagem para destinos de exportação do AWS IoT SiteWise.

> **Note**
>
> Ao enviar dados para o AWS IoT SiteWise, os dados devem atender aos requisitos da ação `BatchPutAssetPropertyValue`. Para obter mais informações, consulte BatchPutAssetPropertyValue na Referência da API do AWS IoT SiteWise.

Em seguida, o gerenciador de fluxo exporta os dados com base nas configurações de lote e na prioridade definidas nas configurações de exportação do fluxo.

Você pode ajustar as configurações do gerenciador de stream e a lógica dos componentes do Greengrass para criar sua estratégia de exportação. Por exemplo: .

- Para exportações quase em tempo real, defina configurações baixas de tamanho de lote e intervalo e anexe os dados ao fluxo quando forem recebidos.

- Para otimizar o agrupamento em lotes, mitigar as restrições de largura de banda ou minimizar os custos, seus componentes do Greengrass podem agrupar os pontos de dados timestamp-quality-value (TQV) recebidos para uma única propriedade do ativo antes de anexar os dados ao stream. Uma estratégia é agrupar entradas para até 10 (dez) combinações diferentes de propriedade e ativo, ou aliases de propriedade, em uma mensagem, em vez de enviar mais de uma entrada para a mesma propriedade. Isso ajuda o gerenciador de fluxo a permanecer dentro das cotas do AWS IoT SiteWise.

Objetos do Amazon S3

O gerenciador de fluxo é compatível com exportações automáticas para o Amazon S3. Você pode utilizar o Amazon S3 para armazenar e recuperar grandes volumes de dados. Para obter mais informações, consulte O que é o Amazon S3? no Guia do desenvolvedor do Amazon Simple Storage Service.

No SDK do Stream Manager, seus componentes do Greengrass usam `S3ExportTaskExecutorConfig` o para definir a configuração de exportação para esse tipo de destino. Para mais informações, consulte a referência do SDK para seu idioma de destino:

- S3 ExportTaskExecutorConfig no SDK do Python
- S3 ExportTaskExecutorConfig no Java SDK
- S3 ExportTaskExecutorConfig no SDK do Node.js

Requisitos

Esse destino de exportação tem os seguintes requisitos:

- Os buckets do Amazon S3 de destino devem estar no mesmo dispositivo principal Conta da AWS do Greengrass.
- Se uma função Lambda executada no modo de contêiner do Greengrass grava arquivos de entrada em um diretório de arquivos de entrada, você deve montar o diretório como um volume no contêiner com permissões de gravação. Isso garante que os arquivos sejam gravados no sistema de arquivos raiz e visíveis para o componente do gerenciador de fluxo, que é executado fora do contêiner.
- Se um componente de contêiner do Docker gravar arquivos de entrada em um diretório de arquivos de entrada, você deverá montar o diretório como um volume no contêiner com

permissões de gravação. Isso garante que os arquivos sejam gravados no sistema de arquivos raiz e visíveis para o componente do gerenciador de fluxo, que é executado fora do contêiner.

- O Autorize os dispositivos principais a interagir com os serviços AWS deve conceder as permissões a seguir para os buckets de destino. Por exemplo: .

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:AbortMultipartUpload",
        "s3:ListMultipartUploadParts"
      ],
      "Resource": [
        "arn:aws:s3:::bucket-1-name/*",
        "arn:aws:s3:::bucket-2-name/*"
      ]
    }
  ]
}
```

Você pode conceder acesso granular ou condicional aos recursos, por exemplo, usando um esquema de nomeação * curinga. Para obter mais informações, consulte Adicionando e removendo políticas do IAM no Guia do usuário do IAM.

Exportar para o Amazon S3

Para criar um stream que exporta para o Amazon S3, seus componentes do Greengrass usam o `S3ExportTaskExecutorConfig` objeto para configurar a política de exportação. A política define as configurações de exportação, como o limite e a prioridade de upload em várias partes. Para exportações do Amazon S3, o gerenciador de fluxo carrega dados que ele lê de arquivos locais no dispositivo principal. Para iniciar um upload, seus componentes do Greengrass anexam uma tarefa de exportação ao stream de destino. A tarefa de exportação contém informações sobre o arquivo de entrada e o objeto de destino do Amazon S3. O gerenciador de fluxo executa tarefas na sequência em que elas são anexadas ao fluxo.

> **Note**
>
> O bucket de destino já deve existir na sua Conta da AWS. Se um objeto para a chave especificada não existir, o gerenciador de fluxo criará o objeto para você.

O gerenciador de fluxo usa a propriedade de limite de upload de várias partes, a configuração do tamanho mínimo das partes e o tamanho do arquivo de entrada para determinar como fazer upload dos dados. O limite de upload de várias partes deve ser maior que o tamanho mínimo das partes. Se você quiser fazer upload de dados em paralelo, pode criar vários fluxos.

As chaves que especificam seus objetos de destino do Amazon S3 podem incluir DateTimeFormatter cadeias de caracteres Java válidas em espaços reservados. `!{timestamp:value}` Você pode usar esses espaços reservados de data e hora para particionar dados no Amazon S3 com base na hora em que os dados do arquivo de entrada foram carregados. Por exemplo, o nome da chave a seguir é resolvido para um valor como `my-key/2020/12/31/data.txt`.

```
my-key/!{timestamp:YYYY}/!{timestamp:MM}/!{timestamp:dd}/data.txt
```

> **Note**
>
> Se você quiser monitorar o status de exportação de um fluxo, primeiro crie um fluxo de status e, em seguida, configure o fluxo de exportação para usá-lo. Para ter mais informações, consulte the section called “Monitorar tarefas de exportação”.

Gerenciar dados de entrada

Você pode criar códigos que os aplicativos de IoT usam para gerenciar o ciclo de vida dos dados de entrada. O exemplo de fluxo de trabalho a seguir mostra como você pode usar os componentes do Greengrass para gerenciar esses dados.

1. Um processo local recebe dados de dispositivos ou periféricos e, em seguida, grava os dados em arquivos em um diretório no dispositivo principal. Esses são os arquivos de entrada para o gerenciador de fluxo.
2. Um componente do Greengrass escaneia o diretório e anexa uma tarefa de exportação ao stream de destino quando um novo arquivo é criado. A tarefa é um objeto `S3ExportTaskDefinition`

serializado em JSON que especifica a URL do arquivo de entrada, o bucket e a chave do Amazon S3 de destino, além dos metadados opcionais do usuário.

3. O gerenciador de fluxo lê o arquivo de entrada e exporta os dados para o Amazon S3 na ordem das tarefas anexadas. O bucket de destino já deve existir na sua Conta da AWS. Se um objeto para a chave especificada não existir, o gerenciador de fluxo criará o objeto para você.
4. O componente Greengrass lê mensagens de um fluxo de status para monitorar o status da exportação. Depois que as tarefas de exportação forem concluídas, o componente Greengrass poderá excluir os arquivos de entrada correspondentes. Para ter mais informações, consulte the section called “Monitorar tarefas de exportação”.

Monitorar tarefas de exportação

Você pode criar códigos que os aplicativos de IoT usam para monitorar o status das suas exportações do Amazon S3. Seus componentes do Greengrass devem criar um fluxo de status e depois configurar o fluxo de exportação para gravar atualizações de status no fluxo de status. Um único fluxo de status pode receber atualizações de status de vários fluxos que são exportados para o Amazon S3.

Primeiro, crie um fluxo para usar como fluxo de status. Você pode configurar as políticas de tamanho e retenção do fluxo para controlar a vida útil das mensagens de status. Por exemplo: .

- Defina `Persistence` como `Memory` se você não quiser armazenar as mensagens de status.
- Defina `StrategyOnFull` como `OverwriteOldestData` para que as novas mensagens de status não sejam perdidas.

Em seguida, crie ou atualize o fluxo de exportação para usar o fluxo de status. Especificamente, defina a propriedade de configuração de status da configuração de exportação `S3ExportTaskExecutorConfig` do fluxo. Essa configuração instrui o gerenciador de fluxo a escrever mensagens de status sobre as tarefas de exportação para o fluxo de status. No objeto `StatusConfig`, especifique o nome do fluxo de status e o nível de detalhe. Os valores suportados a seguir variam do menos detalhado (`ERROR`) ao mais detalhado (`TRACE`). O padrão é `INFO`.

- `ERROR`
- `WARN`
- `INFO`
- `DEBUG`

- `TRACE`

O exemplo de fluxo de trabalho a seguir mostra como os componentes do Greengrass podem usar um fluxo de status para monitorar o status de exportação.

1. Conforme descrito no fluxo de trabalho anterior, um componente do Greengrass anexa uma tarefa de exportação a um fluxo configurado para gravar mensagens de status sobre tarefas de exportação em um fluxo de status. A operação de append retorna um número de sequência que representa a ID da tarefa.
2. Um componente do Greengrass lê mensagens sequencialmente do fluxo de status e, em seguida, filtra as mensagens com base no nome do fluxo e no ID da tarefa ou com base em uma propriedade da tarefa de exportação do contexto da mensagem. Por exemplo, o componente Greengrass pode filtrar pela URL do arquivo de entrada da tarefa de exportação, que é representada pelo `S3ExportTaskDefinition` objeto no contexto da mensagem.

   Os códigos de status a seguir indicam que uma tarefa de exportação atingiu um estado concluído:

   - `Success`. O upload foi concluído com êxito.
   - `Failure`. O gerenciador de fluxo encontrou um erro, por exemplo, o bucket especificado não existe. Depois de resolver o problema, você pode reanexar a tarefa de exportação ao fluxo.
   - `Canceled`. A tarefa foi interrompida porque a definição de fluxo ou exportação foi excluída ou o período time-to-live (TTL) da tarefa expirou.

   > **Note**
   >
   > A tarefa também pode ter um status de `InProgress` ou `Warning`. O gerenciador de fluxo emite avisos quando um evento retorna um erro que não afeta a execução da tarefa. Por exemplo, uma falha na limpeza de um upload parcial retorna um aviso.

3. Depois que as tarefas de exportação forem concluídas, o componente Greengrass poderá excluir os arquivos de entrada correspondentes.

O exemplo a seguir mostra como um componente do Greengrass pode ler e processar mensagens de status.

Python

```
import time
```

```
from stream_manager import (
    ReadMessagesOptions,
    Status,
    StatusConfig,
    StatusLevel,
    StatusMessage,
    StreamManagerClient,
)
from stream_manager.util import Util

client = StreamManagerClient()

try:
    # Read the statuses from the export status stream
    is_file_uploaded_to_s3 = False
    while not is_file_uploaded_to_s3:
        try:
            messages_list = client.read_messages(
                "StatusStreamName", ReadMessagesOptions(min_message_count=1,
 read_timeout_millis=1000)
            )
            for message in messages_list:
                # Deserialize the status message first.
                status_message = Util.deserialize_json_bytes_to_obj(message.payload,
 StatusMessage)

                # Check the status of the status message. If the status is
 "Success",
                # the file was successfully uploaded to S3.
                # If the status was either "Failure" or "Cancelled", the server was
 unable to upload the file to S3.
                # We will print the message for why the upload to S3 failed from the
 status message.
                # If the status was "InProgress", the status indicates that the
 server has started uploading
                # the S3 task.
                if status_message.status == Status.Success:
                    logger.info("Successfully uploaded file at path " + file_url + "
 to S3.")
                    is_file_uploaded_to_s3 = True
                elif status_message.status == Status.Failure or
 status_message.status == Status.Canceled:
                    logger.info(
```

```
                        "Unable to upload file at path " + file_url + " to S3.
 Message: " + status_message.message
                    )
                    is_file_uploaded_to_s3 = True
            time.sleep(5)
        except StreamManagerException:
            logger.exception("Exception while running")
except StreamManagerException:
    pass
    # Properly handle errors.
except ConnectionError or asyncio.TimeoutError:
    pass
    # Properly handle errors.
```

[Referência do SDK do Python: read_messages | StatusMessage]

Java

```
import com.amazonaws.greengrass.streammanager.client.StreamManagerClient;
import com.amazonaws.greengrass.streammanager.client.StreamManagerClientFactory;
import com.amazonaws.greengrass.streammanager.client.utils.ValidateAndSerialize;
import com.amazonaws.greengrass.streammanager.model.ReadMessagesOptions;
import com.amazonaws.greengrass.streammanager.model.Status;
import com.amazonaws.greengrass.streammanager.model.StatusConfig;
import com.amazonaws.greengrass.streammanager.model.StatusLevel;
import com.amazonaws.greengrass.streammanager.model.StatusMessage;

 try (final StreamManagerClient client =
 StreamManagerClientFactory.standard().build()) {
    try {
        boolean isS3UploadComplete = false;
        while (!isS3UploadComplete) {
            try {
                // Read the statuses from the export status stream
                List<Message> messages = client.readMessages("StatusStreamName",
                    new
 ReadMessagesOptions().withMinMessageCount(1L).withReadTimeoutMillis(1000L));
                for (Message message : messages) {
                    // Deserialize the status message first.
                    StatusMessage statusMessage =
 ValidateAndSerialize.deserializeJsonBytesToObj(message.getPayload(),
 StatusMessage.class);
                    // Check the status of the status message. If the status is
 "Success", the file was successfully uploaded to S3.
```

```
                    // If the status was either "Failure" or "Canceled", the server
 was unable to upload the file to S3.
                    // We will print the message for why the upload to S3 failed
 from the status message.
                    // If the status was "InProgress", the status indicates that the
 server has started uploading the S3 task.
                    if (Status.Success.equals(statusMessage.getStatus())) {
                        System.out.println("Successfully uploaded file at path " +
 FILE_URL + " to S3.");
                        isS3UploadComplete = true;
                     } else if (Status.Failure.equals(statusMessage.getStatus()) ||
 Status.Canceled.equals(statusMessage.getStatus())) {
                        System.out.println(String.format("Unable to upload file at
 path %s to S3. Message %s",

 statusMessage.getStatusContext().getS3ExportTaskDefinition().getInputUrl(),
                            statusMessage.getMessage()));
                        sS3UploadComplete = true;
                    }
                }
            } catch (StreamManagerException ignored) {
            } finally {
                // Sleep for sometime for the S3 upload task to complete before
 trying to read the status message.
                Thread.sleep(5000);
            }
        } catch (e) {
        // Properly handle errors.
    }
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Referência do SDK Java: readMessages | StatusMessage

Node.js

```
const {
    StreamManagerClient, ReadMessagesOptions,
    Status, StatusConfig, StatusLevel, StatusMessage,
    util,
} = require(*'aws-greengrass-stream-manager-sdk'*);

const client = new StreamManagerClient();
```

```
client.onConnected(async () => {
    try {
        let isS3UploadComplete = false;
        while (!isS3UploadComplete) {
            try {
                // Read the statuses from the export status stream
                const messages = await c.readMessages("StatusStreamName",
                    new ReadMessagesOptions()
                        .withMinMessageCount(1)
                        .withReadTimeoutMillis(1000));

                messages.forEach((message) => {
                    // Deserialize the status message first.
                    const statusMessage =
 util.deserializeJsonBytesToObj(message.payload, StatusMessage);
                    // Check the status of the status message. If the status is
 'Success', the file was successfully uploaded to S3.
                    // If the status was either 'Failure' or 'Cancelled', the server
 was unable to upload the file to S3.
                    // We will print the message for why the upload to S3 failed
 from the status message.
                    // If the status was "InProgress", the status indicates that the
 server has started uploading the S3 task.
                    if (statusMessage.status === Status.Success) {
                        console.log(`Successfully uploaded file at path ${FILE_URL}
 to S3.`);
                        isS3UploadComplete = true;
                    } else if (statusMessage.status === Status.Failure ||
 statusMessage.status === Status.Canceled) {
                        console.log(`Unable to upload file at path ${FILE_URL} to
 S3. Message: ${statusMessage.message}`);
                        isS3UploadComplete = true;
                    }
                });
                // Sleep for sometime for the S3 upload task to complete before
 trying to read the status message.
                await new Promise((r) => setTimeout(r, 5000));
            } catch (e) {
                // Ignored
            }
    } catch (e) {
        // Properly handle errors.
    }
});
```

```
client.onError((err) => {
    // Properly handle connection errors.
    // This is called only when the connection to the StreamManager server fails.
});
```

Referência do SDK do Node.js: readMessages | StatusMessage

Configurar o gerenciador de fluxo do AWS IoT Greengrass

Nos dispositivos principais do Greengrass, o gerenciador de streams pode armazenar, processar e exportar dados de dispositivos de IoT. O gerenciador de fluxo fornece parâmetros que você usa para definir as configurações de tempo de execução. Essas configurações se aplicam a todos os streams no dispositivo principal do Greengrass. Você pode usar o AWS IoT Greengrass console ou a API para definir as configurações do gerenciador de stream ao implantar o componente. As alterações entram em vigor após a conclusão da implantação.

Parâmetros do gerenciador de fluxo

O Stream Manager fornece os seguintes parâmetros que você pode configurar ao implantar o componente em seus dispositivos principais. Todos os parâmetros são opcionais.

Diretório de armazenamento

Nome do parâmetro: `STREAM_MANAGER_STORE_ROOT_DIR`

O caminho absoluto da pasta local usada para armazenar fluxos. Esse valor deve começar com uma barra (por exemplo, `/data`).

Você deve especificar uma pasta existente, e o usuário do sistema que executa o componente do gerenciador de stream deve ter permissões para ler e gravar nessa pasta. Por exemplo, você pode executar os comandos a seguir para criar e configurar uma pasta`/var/greengrass/streams`, que você especifica como a pasta raiz do gerenciador de streams. Esses comandos permitem que o usuário padrão do sistema`ggc_user`,, leia e grave nessa pasta.

```
sudo mkdir /var/greengrass/streams
sudo chown ggc_user /var/greengrass/streams
sudo chmod 700 /var/greengrass/streams
```

Para obter informações sobre como proteger dados de fluxo, consulte the section called “Segurança de dados locais”.

Padrão: `/greengrass/v2/work/aws.greengrass.StreamManager`

Porta do servidor

Nome do parâmetro: `STREAM_MANAGER_SERVER_PORT`

O número da porta local usado para se comunicar com o gerenciador de fluxo. O padrão é `8088`.

Você pode especificar `0` o uso de uma porta disponível aleatória.

Autenticar cliente

Nome do parâmetro: `STREAM_MANAGER_AUTHENTICATE_CLIENT`

Indica se os clientes devem ser autenticados de modo a interagir com o gerenciador de fluxo. Toda interação entre clientes e o gerenciador de streams é controlada pelo SDK do Stream Manager. Esse parâmetro determina quais clientes podem chamar o SDK do Stream Manager para trabalhar com streams. Para ter mais informações, consulte the section called “Autenticação de cliente”.

Os valores válidos são `true` ou `false`. O padrão é `true` (recomendado).

- `true`. Permite somente componentes do Greengrass como clientes. Os componentes usam protocolos AWS IoT Greengrass principais internos para se autenticar com o SDK do Stream Manager.
- `false`. Permite que qualquer processo executado no AWS IoT Greengrass Core seja um cliente. Não defina o valor como, a `false` menos que seu caso de negócios exija isso. Por exemplo, use `false` somente se processos não componentes no dispositivo principal precisarem se comunicar diretamente com o gerenciador de fluxo.

Largura máxima de banda

Nome do parâmetro: `STREAM_MANAGER_EXPORTER_MAX_BANDWIDTH`

A média da largura máxima de banda (em kilobits por segundo) que pode ser usada para exportar dados. O padrão permite o uso ilimitado da largura de banda disponível.

Tamanho do grupo de threads

Nome do parâmetro: `STREAM_MANAGER_EXPORTER_THREAD_POOL_SIZE`

O número máximo de threads ativos que podem ser usados para exportar dados. O padrão é `5`.

O tamanho ideal depende do hardware, do volume do fluxo e do número planejado de fluxos de exportação. Se a velocidade de exportação for lenta, você poderá ajustar essa configuração

para encontrar o tamanho ideal para seu hardware e caso de negócios. A CPU e a memória do hardware do dispositivo de núcleo são fatores limitantes. Para iniciar, você pode tentar definir esse valor igual ao número de núcleos do processador no dispositivo.

Tenha cuidado para não definir um tamanho superior ao que o seu hardware pode suportar. Cada fluxo consome recursos de hardware, então tente limitar o número de fluxos de exportação em dispositivos restritos.

Argumentos JVM

Nome do parâmetro: `JVM_ARGS`

Argumentos personalizados da Java Virtual Machine para passar para o gerenciador de fluxo na startup. Se houver vários argumentos, separe-os por espaços.

Só use esse parâmetro quando precisar substituir as configurações padrão usadas pela JVM. Por exemplo, talvez seja necessário aumentar o tamanho do heap padrão caso você planeje exportar um grande número de fluxos.

Nível de registro

Nome do parâmetro: `LOG_LEVEL`

O nível de registro do componente. Escolha entre os seguintes níveis de registro, listados aqui em ordem de nível:

- `TRACE`
- `DEBUG`
- `INFO`
- `WARN`
- `ERROR`

Padrão: `INFO`

Tamanho mínimo para upload de várias partes

Nome do parâmetro: `STREAM_MANAGER_EXPORTER_S3_DESTINATION_MULTIPART_UPLOAD_MIN_PART_SIZE_BYTES`

O tamanho mínimo (em bytes) de uma parte em um upload multiparte para o Amazon S3. O gerenciador de fluxo usa essa configuração e o tamanho do arquivo de entrada para determinar

como agrupar dados em lote em uma solicitação PUT de várias partes. O valor mínimo e padrão é de `5242880` bytes (5 MB).

> **Note**
>
> O gerenciador de fluxo usa a propriedade `sizeThresholdForMultipartUploadBytes` do fluxo para determinar se deve exportar para o Amazon S3 como um upload de uma ou várias partes. Os componentes do Greengrass definidos pelo usuário definem esse limite quando criam um stream que exporta para o Amazon S3. O limite padrão é 5 MB.

Consulte também

- Gerencie fluxos de dados nos dispositivos principais do Greengrass
- Use StreamManagerClient para trabalhar com streams
- Configurações de exportação para destinos compatíveis do Nuvem AWS

Executar a inferência de machine learning

ComAWS IoT Greengrass, você pode realizar inferência de aprendizado de máquina (ML) em seus dispositivos de ponta em dados gerados localmente usando modelos treinados na nuvem. Você se beneficia da baixa latência e da redução de custos na execução da inferência local, e ainda aproveita a capacidade de computação em nuvem para modelos de treinamento e processamento complexo.

AWS IoT Greengrasstorna as etapas necessárias para realizar a inferência mais eficientes. Você pode treinar seus modelos de inferência em qualquer lugar e implantá-los localmente como componentes de aprendizado de máquina. Por exemplo, você pode criar e treinar modelos de aprendizado profundo na Amazon SageMaker ou modelos de visão computacional no Amazon Lookout for Vision. Em seguida, você pode armazenar esses modelos em um bucket do Amazon S3, para poder usá-los como artefatos em seus componentes para realizar inferências em seus dispositivos principais.

Tópicos

Como a inferência de ML do AWS IoT Greengrass funciona

AWSfornece componentes de aprendizado de máquina que você pode usar para criar implantações em uma etapa para realizar inferências de aprendizado de máquina em seu dispositivo. Você também pode usar esses componentes como modelos para criar componentes personalizados para atender às suas necessidades específicas.

AWSfornece as seguintes categorias de componentes de aprendizado de máquina:

- Componente de modelo — contém modelos de aprendizado de máquina como artefatos do Greengrass.
- Componente de tempo de execução — contém o script que instala a estrutura de aprendizado de máquina e suas dependências no dispositivo principal do Greengrass.
- Componente de inferência — contém o código de inferência e inclui dependências de componentes para instalar a estrutura de aprendizado de máquina e baixar modelos de aprendizado de máquina pré-treinados.

Cada implantação que você cria para realizar inferência de aprendizado de máquina consiste em pelo menos um componente que executa seu aplicativo de inferência, instala a estrutura de aprendizado de máquina e baixa seus modelos de aprendizado de máquina. Para realizar inferência de amostra com os componentes AWS fornecidos, você implanta um componente de inferência em seu dispositivo principal, que inclui automaticamente o modelo correspondente e os componentes de tempo de execução como dependências. Para personalizar suas implantações, você pode conectar ou trocar os componentes do modelo de amostra por componentes do modelo personalizado ou pode usar as receitas de componentes dos componentes AWS fornecidos como modelos para criar seus próprios componentes personalizados de inferência, modelo e tempo de execução.

Para realizar inferências de aprendizado de máquina usando componentes personalizados:

1. Crie um componente de modelo. Esse componente contém os modelos de aprendizado de máquina que você deseja usar para realizar inferências. AWSfornece exemplos de modelos DLR e TensorFlow Lite pré-treinados. Para usar um modelo personalizado, crie seu próprio componente de modelo.
2. Crie um componente de tempo de execução. Esse componente contém os scripts necessários para instalar o tempo de execução do aprendizado de máquina para seus modelos. AWSfornece exemplos de componentes de tempo de execução para Deep Learning Runtime (DLR) e TensorFlow Lite. Para usar outros tempos de execução com seus modelos personalizados e código de inferência, crie seus próprios componentes de tempo de execução.
3. Crie um componente de inferência. Esse componente contém seu código de inferência e inclui seus componentes de modelo e tempo de execução como dependências. AWSfornece componentes de inferência de amostra para classificação de imagens e detecção de objetos usando DLR e TensorFlow Lite. Para realizar outros tipos de inferência ou usar modelos e tempos de execução personalizados, crie seu próprio componente de inferência.

4. Implante o componente de inferência. Quando você implanta esse componente, AWS IoT Greengrass também implanta automaticamente as dependências do modelo e do componente de tempo de execução.

Para começar a usar os componentes AWS fornecidos, consultethe section called “Execute inferência de classificação de imagens de amostra”.

Para obter informações sobre a criação de componentes personalizados de aprendizado de máquina, consultePersonalize seus componentes de aprendizado de máquina.

O que há de diferente na AWS IoT Greengrass versão 2?

AWS IoT Greengrassconsolida unidades funcionais para aprendizado de máquina, como modelos, tempos de execução e código de inferência, em componentes que permitem que você use um processo de uma etapa para instalar o tempo de execução do aprendizado de máquina, baixar seus modelos treinados e realizar inferências em seu dispositivo.

Ao usar os componentes AWS de aprendizado de máquina fornecidos, você tem a flexibilidade de começar a realizar inferências de aprendizado de máquina com exemplos de código de inferência e modelos pré-treinados. Você pode conectar componentes de modelo personalizados para usar seus próprios modelos personalizados com os componentes de inferência e tempo de execução fornecidos. AWS Para uma solução de aprendizado de máquina totalmente personalizada, você pode usar os componentes públicos como modelos para criar componentes personalizados e usar qualquer tempo de execução, modelo ou tipo de inferência que desejar.

Requisitos

Para criar e usar componentes de aprendizado de máquina, você deve ter o seguinte:

- Um dispositivo principal do Greengrass. Se você não tiver uma, consulte Tutorial: Conceitos básicos do AWS IoT Greengrass V2.
- Espaço mínimo de armazenamento local de 500 MB para usar AWS — exemplos de componentes de aprendizado de máquina fornecidos.

Fontes de modelo compatíveis

AWS IoT Greengrasssuporta o uso de modelos de aprendizado de máquina personalizados que são armazenados no Amazon S3. Você também pode usar os trabalhos de empacotamento de SageMaker borda da Amazon para criar diretamente componentes de modelo para seus modelos SageMaker compilados pela NEO. Para obter informações sobre como usar o SageMaker Edge Manager comAWS IoT Greengrass, consulteUse o Amazon SageMaker Edge Manager nos dispositivos principais do Greengrass. Você também pode usar trabalhos de empacotamento de modelos do Amazon Lookout for Vision para criar componentes de modelo para seus modelos do Lookout for Vision. Para obter mais informações sobre como usar o Lookout for Vision AWS IoT Greengrass com, Amazon Loout Lookout for Vision consulte.

Os buckets S3 que contêm seus modelos devem atender aos seguintes requisitos:

- Eles não devem ser criptografados usando SSE-C. Para buckets que usam criptografia do lado do servidor, a inferência de aprendizado de AWS IoT Greengrass máquina atualmente oferece suporte somente às opções de criptografia SSE-S3 ou SSE-KMS. Para obter mais informações sobre as opções criptografia no lado do servidor, consulte Protegendo dados usando criptografia no lado do servidor no Guia do usuário do Amazon Simple Storage Service.
- Seus nomes não devem incluir pontos (`.`). Para obter mais informações, consulte a regra sobre como usar buckets hospedados virtualmente com SSL em Regras para nomenclatura de buckets no Guia do usuário do Amazon Simple Storage Service.
- Os buckets do S3 que armazenam suas fontes de modelo devem estar nos mesmos componentes de aprendizado de máquina Conta da AWS e nos Região da AWS mesmos.
- AWS IoT Greengrassdeve ter `read` permissão para acessar a fonte do modelo. Para permitir o acesso AWS IoT Greengrass aos buckets do S3, a função do dispositivo Greengrass deve permitir a ação. `s3:GetObject` Para obter mais informações sobre a função do dispositivo, consulteAutorize os dispositivos principais a interagir com os serviços AWS.

Tempos de execução de aprendizado de máquina compatíveis

AWS IoT Greengrasspermite que você crie componentes personalizados para usar qualquer tempo de execução de aprendizado de máquina de sua escolha para realizar inferências de aprendizado de máquina com seus modelos treinados de forma personalizada. Para obter informações sobre a criação de componentes personalizados de aprendizado de máquina, consultePersonalize seus componentes de aprendizado de máquina.

Para tornar o processo de introdução ao aprendizado de máquina mais eficiente, AWS IoT Greengrass fornece exemplos de componentes de inferência, modelo e tempo de execução que usam os seguintes tempos de execução de aprendizado de máquina:

- Tempo de execução de aprendizado profundo (DLR) v1.6.0 e v1.3.0
- TensorFlow Lite v2.5.0

AWS-componentes de aprendizado de máquina fornecidos

A tabela a seguir lista os componentes AWS fornecidos usados para aprendizado de máquina.

> **Note**
>
> Vários componentes AWS fornecidos dependem de versões secundárias específicas do núcleo do Greengrass. Por causa dessa dependência, você precisa atualizar esses componentes ao atualizar o núcleo do Greengrass para uma nova versão secundária. Para obter informações sobre as versões específicas do núcleo das quais cada componente depende, consulte o tópico do componente correspondente. Para obter mais informações sobre a atualização do núcleo, consulteAtualize o software AWS IoT Greengrass principal (OTA).

Componente	Descrição	Tipo de componente	SO com suporte	Código aberto
Lookout for Vision Edge Agent	Implanta o tempo de execução do Amazon Lookout for Vision no dispositivo principal do Greengrass, para que você possa	Genérico	Linux	Não

Componente	Descrição	Tipo de componente	SO com suporte	Código aberto
	usar a visão computacional para encontrar defeitos em produtos industriais.			
SageMaker Gerente de borda	Implanta o agente Amazon SageMaker Edge Manager no dispositivo principal do Greengrass.	Genérico	Linux, Windows	Não

Componente	Descrição	Tipo de componente	SO com suporte	Código aberto
Classificação de imagens DLR	Componente de inferência que usa o repositório de modelos de classificação de imagem DLR e o componente de tempo de execução do DLR como dependências para instalar o DLR, baixar exemplos de modelos de classificação de imagens e realizar inferência de classificação de imagens em dispositivos compatíveis.	Genérico	Linux, Windows	Não

Componente	Descrição	Tipo de componente	SO com suporte	Código aberto
Detecção de objetos DLR	Componente de inferência que usa o repositório de modelos de detecção de objetos DLR e o componente de tempo de execução do DLR como dependências para instalar o DLR, baixar modelos de detecção de objetos de amostra e realizar inferência de detecção de objetos em dispositivos compatíveis.	Genérico	Linux, Windows	Não

Componente	Descrição	Tipo de componente	SO com suporte	Código aberto
Armazenamento de modelos de classificação de imagens DLR	Componente de modelo que contém amostras de ResNet -50 modelos de classificação de imagens como artefatos do Greengrass.	Genérico	Linux, Windows	Não
Armazenamento de modelos de detecção de objetos DLR	Componente de modelo que contém exemplos de modelos de detecção de objetos YOLOv3 como artefatos do Greengrass.	Genérico	Linux, Windows	Não

Componente	Descrição	Tipo de componente	SO com suporte	Código aberto
Tempo de execução do DLR	Componente de tempo de execução que contém um script de instalação usado para instalar o DLR e suas dependências no dispositivo principal do Greengrass.	Genérico	Linux, Windows	Não

Componente	Descrição	Tipo de componente	SO com suporte	Código aberto
TensorFlow Classificação de imagens Lite	Componente de inferência que usa o TensorFlow repositório de modelos de classificação de imagem TensorFlow Lite e o componente de tempo de execução Lite como dependências para instalar o TensorFlow Lite, baixar exemplos de modelos de classificação de imagens e realizar inferência de classificação de imagens em dispositivos compatíveis.	Genérico	Linux, Windows	Não

Componente	Descrição	Tipo de componente	SO com suporte	Código aberto
TensorFlow Detecção leve de objetos	Componente de inferência que usa o TensorFlow repositório de modelos de detecção de objetos TensorFlow Lite e o componente de tempo de execução Lite como dependências para instalar o TensorFlow Lite, baixar modelos de detecção de objetos de amostra e realizar inferência de detecção de objetos em dispositivos compatíveis.	Genérico	Linux, Windows	Não

Componente	Descrição	Tipo de componente	SO com suporte	Código aberto
TensorFlow Loja de modelos de classificação de imagens Lite	Componente de modelo que contém um modelo MobileNet v1 de amostra como artefato do Greengrass.	Genérico	Linux, Windows	Não
TensorFlow Loja de modelos de detecção de objetos Lite	Componente de modelo que contém um MobileNet modelo de amostra de detecção de disparo único (SSD) como um artefato do Greengrass.	Genérico	Linux, Windows	Não

Componente	Descrição	Tipo de componente	SO com suporte	Código aberto
TensorFlow Tempo de execução leve	Componente de tempo de execução que contém um script de instalação usado para instalar o TensorFlow Lite e suas dependências no dispositivo principal do Greengrass.	Genérico	Linux, Windows	Não

Use o Amazon SageMaker Edge Manager nos dispositivos principais do Greengrass

> **Important**
>
> SageMaker O Edge Manager será descontinuado em 26 de abril de 2024. Para obter mais informações sobre como continuar implantando seus modelos em dispositivos de ponta, consulte Fim da vida útil do SageMaker Edge Manager.

O Amazon SageMaker Edge Manager é um agente de software executado em dispositivos periféricos. SageMaker O Edge Manager fornece gerenciamento de modelos para dispositivos de borda para que você possa empacotar e usar modelos SageMaker compilados pelo Amazon Neo diretamente nos dispositivos principais do Greengrass. Ao usar o SageMaker Edge Manager, você também pode amostrar dados de entrada e saída do modelo de seus dispositivos principais e enviar esses dados Nuvem AWS para monitoramento e análise. Como o SageMaker Edge Manager

usa SageMaker o Neo para otimizar seus modelos para o hardware de destino, você não precisa instalar o tempo de execução do DLR diretamente no seu dispositivo. Nos dispositivos Greengrass, o SageMaker Edge Manager não carrega AWS IoT certificados locais nem liga diretamente para o endpoint do provedor de AWS IoT credenciais. Em vez disso, o SageMaker Edge Manager usa o serviço de troca de tokens para buscar credenciais temporárias de um endpoint TES.

Esta seção descreve como o SageMaker Edge Manager funciona nos dispositivos principais do Greengrass.

Como o SageMaker Edge Manager funciona nos dispositivos Greengrass

Para implantar o agente do SageMaker Edge Manager em seus dispositivos principais, crie uma implantação que inclua o `aws.greengrass.SageMakerEdgeManager` componente. AWS IoT Greengrassgerencia a instalação e o ciclo de vida do agente Edge Manager em seus dispositivos. Quando uma nova versão do binário do agente estiver disponível, implante a versão atualizada do `aws.greengrass.SageMakerEdgeManager` componente para atualizar a versão do agente que está instalada no seu dispositivo.

Quando você usa o SageMaker Edge Manager comAWS IoT Greengrass, seu fluxo de trabalho inclui as seguintes etapas de alto nível:

1. Compile modelos com o SageMaker Neo.
2. Empacote seus modelos SageMaker compilados pela NEO usando trabalhos de empacotamento de SageMaker ponta. Ao executar um trabalho de empacotamento de borda para seu modelo, você pode optar por criar um componente de modelo com o modelo empacotado como um artefato que pode ser implantado em seu dispositivo principal do Greengrass.
3. Crie um componente de inferência personalizado. Você usa esse componente de inferência para interagir com o agente do Edge Manager para realizar inferência no dispositivo principal. Essas operações incluem carregar modelos, invocar solicitações de previsão para executar inferências e descarregar modelos quando o componente é desligado.
4. Implante o componente SageMaker Edge Manager, o componente do modelo empacotado e o componente de inferência para executar seu modelo no mecanismo de SageMaker inferência (agente do Edge Manager) em seu dispositivo.

Para obter mais informações sobre a criação de trabalhos de empacotamento de borda e componentes de inferência que funcionam com o SageMaker Edge Manager, consulte Deploy Model Package and Edge Manager Agent AWS IoT Greengrass no Amazon SageMaker Developer Guide.

O Tutorial: Comece a usar o SageMaker Edge Manager tutorial mostra como configurar e usar o agente do SageMaker Edge Manager em um dispositivo principal existente do Greengrass, usando um código AWS de exemplo fornecido que você pode usar para criar exemplos de inferência e componentes de modelo.

Ao usar o SageMaker Edge Manager nos dispositivos principais do Greengrass, você também pode usar o recurso de captura de dados para carregar dados de amostra para o. Nuvem AWS Capturar dados é um SageMaker recurso que você usa para carregar entradas de inferência, resultados de inferência e dados de inferência adicionais em um bucket do S3 ou em um diretório local para análise futura. Para obter mais informações sobre o uso de dados de captura com o SageMaker Edge Manager, consulte Gerenciar modelo no Amazon SageMaker Developer Guide.

Requisitos

Você deve atender aos seguintes requisitos para usar o agente do SageMaker Edge Manager nos dispositivos principais do Greengrass.

- Um dispositivo principal do Greengrass executado no Amazon Linux 2, uma plataforma Linux baseada em Debian (x86_64 ou Armv8) ou Windows (x86_64). Se você não tiver uma, consulte Tutorial: Conceitos básicos do AWS IoT Greengrass V2.
- Python 3.6 ou posterior, inclusive `pip` para sua versão do Python, instalada em seu dispositivo principal.
- A função do dispositivo Greengrass foi configurada com o seguinte:
  - Uma relação de confiança que permite `credentials.iot.amazonaws.com` e assume `sagemaker.amazonaws.com` a função, conforme mostrado no exemplo de política do IAM a seguir.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.iot.amazonaws.com"
       },
      "Action": "sts:AssumeRole"
    },
    {
      "Effect": "Allow",
```

```
      "Principal": {
        "Service": "sagemaker.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

- A política gerenciada AmazonSageMakerEdgeDeviceFleetPolicydo IAM.
- A `s3:PutObject` ação, conforme mostrado no exemplo de política do IAM a seguir.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:PutObject"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

- Um bucket do Amazon S3 criado no mesmo dispositivo central do Greengrass Conta da AWS e no Região da AWS mesmo dispositivo. SageMaker O Edge Manager requer um bucket S3 para criar uma frota de dispositivos de ponta e armazenar dados de amostra da execução de inferência em seu dispositivo. Para obter informações sobre a criação de buckets do S3, consulte Introdução ao Amazon S3.
- Uma frota de dispositivos de SageMaker ponta que usa o mesmo alias de AWS IoT função do seu dispositivo principal do Greengrass. Para ter mais informações, consulte Crie uma frota de dispositivos de ponta.
- Seu dispositivo principal do Greengrass foi registrado como um dispositivo de ponta em sua frota de dispositivos SageMaker Edge. O nome do dispositivo de borda deve corresponder ao AWS IoT nome do dispositivo principal. Para ter mais informações, consulte Registre seu dispositivo principal do Greengrass.

Comece a usar o SageMaker Edge Manager

Você pode concluir um tutorial para começar a usar o SageMaker Edge Manager. O tutorial mostra como começar a usar o SageMaker Edge Manager com componentes AWS de amostra fornecidos em um dispositivo principal existente. Esses componentes de amostra usam o componente SageMaker Edge Manager como uma dependência para implantar o agente do Edge Manager e realizar inferência usando modelos pré-treinados que foram compilados usando o Neo. SageMaker Para ter mais informações, consulte Tutorial: Comece a usar o SageMaker Edge Manager.

Amazon Loout Lookout for Vision

> **Note**
>
> AWS IoT Greengrassatualmente não oferece suporte a esse recurso nos dispositivos principais do Windows.

O Amazon Lookout for Vision é umAWS service (Serviço da AWS) que você pode usar para encontrar defeitos visuais em produtos industriais. Ele usa a visão computacional para identificar componentes ausentes em um produto industrial, danos a veículos ou estruturas, irregularidades nas linhas de produção, falta de capacitores em placas de circuito impresso e defeitos em pastilhas de silício ou qualquer outro item físico em que a qualidade é importante. Para obter mais informações, consulte O que é Lookout for Vision no Guia do desenvolvedor do Amazon Lookout for Vision.

Você pode criar aplicativos Greengrass que usam a inferência do Lookout for Vision para encontrar defeitos visuais nos dispositivos principais do Greengrass. Depois de implantar um fluxo de trabalho do Lookout for Vision em um dispositivo central do Greengrass, você pode realizar a visão computacional sem uma conexão com o serviço Lookout for Vision noNuvem AWS. Para criar um aplicativo Greengrass que usa o Lookout for Vision, você configura e implanta os seguintes componentes do Greengrass:

- Componentes do modelo Lookout for Vision — Contém os modelos de aprendizado de máquina do Lookout for Vision como artefatos do Greengrass. Você pode usar o console e a API do Lookout for Vision para gerar componentes de modelo que empacotam seus modelos de aprendizado de máquina pré-treinados. Esses componentes são componentes privados do Greengrass em seuConta da AWS. Para obter mais informações, consulte Criação de um modelo do Lookout for Vision e Empacotar um modelo do Lookout for Vision no Guia do desenvolvedor do Amazon Lookout for Vision.

- Componente Lookout for Vision Edge Agent — Fornece um servidor de tempo de execução local do Lookout for Vision que usa visão computacional para detectar anomalias usando os modelos de aprendizado de máquina que você fornece. Esse componente éAWS um componente fornecido. Para obter mais informações, consulte o componente Lookout for Vision Edge Agent.
- Componente do aplicativo cliente Lookout for Vision — interage com o componente Lookout for Vision Edge Agent para processar imagens em busca de anomalias. Você pode desenvolver componentes personalizados de aplicativos clientes que enviam imagens e streams de vídeo para o agente local do Lookout for Vision Edge e relatam quaisquer anomalias detectadas pelos modelos de aprendizado de máquina. Para obter mais informações, consulte Como escrever um componente de aplicativo cliente e a referência da API Lookout for Vision Edge Agent no Guia do desenvolvedor do Amazon Lookout for Vision.

Para obter mais informações sobre como criar, configurar e usar esses componentes, consulte Como usar um modelo do Lookout for Vision em um dispositivo de ponta no Guia do desenvolvedor do Amazon Lookout for Vision.

Personalize seus componentes de aprendizado de máquina

EmAWS IoT Greengrass, você pode configurar exemplos de componentes de aprendizado de máquina para personalizar a forma como você executa a inferência de aprendizado de máquina em seus dispositivos com os componentes de inferência, modelo e tempo de execução como blocos de construção. AWS IoT Greengrasstambém oferece a flexibilidade de usar os componentes de amostra como modelos e criar seus próprios componentes personalizados conforme necessário. Você pode misturar e combinar essa abordagem modular para personalizar seus componentes de inferência de aprendizado de máquina das seguintes maneiras:

Usando componentes de inferência de amostra

- Modifique a configuração dos componentes de inferência ao implantá-los.
- Use um modelo personalizado com o componente de inferência de amostra substituindo o componente de armazenamento de modelos de amostra por um componente de modelo personalizado. Seu modelo personalizado deve ser treinado usando o mesmo tempo de execução do modelo de amostra.

Usando componentes de inferência personalizados

- Use o código de inferência personalizado com os modelos de amostra e os tempos de execução adicionando componentes de modelo público e componentes de tempo de execução como dependências de componentes de inferência personalizados.

- Crie e adicione componentes de modelo personalizados ou componentes de tempo de execução como dependências de componentes de inferência personalizados. Você deve usar componentes personalizados se quiser usar um código de inferência personalizado ou um tempo de execução para o qual AWS IoT Greengrass não forneça um componente de amostra.

Tópicos

Modificar a configuração de um componente de inferência pública

No AWS IoT Greengrassconsole, a página do componente exibe a configuração padrão desse componente. Por exemplo, a configuração padrão do componente de classificação de imagem TensorFlow Lite tem a seguinte aparência:

```
{
  "accessControl": {
    "aws.greengrass.ipc.mqttproxy": {
      "aws.greengrass.TensorFlowLiteImageClassification:mqttproxy:1": {
        "policyDescription": "Allows access to publish via topic ml/tflite/image-
classification.",
        "operations": [
          "aws.greengrass#PublishToIoTCore"
        ],
        "resources": [
          "ml/tflite/image-classification"
        ]
      }
    }
  },
  "PublishResultsOnTopic": "ml/tflite/image-classification",
  "ImageName": "cat.jpeg",
  "InferenceInterval": 3600,
  "ModelResourceKey": {
    "model": "TensorFlowLite-Mobilenet"
  }
```

```
}
```

Ao implantar um componente de inferência pública, você pode modificar a configuração padrão para personalizar sua implantação. Para obter informações sobre os parâmetros de configuração disponíveis para cada componente de inferência pública, consulte o tópico do componente emAWS-componentes de aprendizado de máquina fornecidos.

Esta seção descreve como implantar um componente modificado a partir do AWS IoT Greengrass console. Para obter informações sobre a implantação de componentes usando oAWS CLI, consulteCriar implantações.

Para implantar um componente de inferência pública modificado (console)

1. Faça login no AWS IoT Greengrassconsole.
2. No menu de navegação, escolha Componentes.
3. Na página Componentes, na guia Componentes públicos, escolha o componente que você deseja implantar.
4. Na página do componente, escolha Implantar.
5. Em Adicionar à implantação, escolha uma das seguintes opções:
   a. Para mesclar esse componente a uma implantação existente em seu dispositivo de destino, escolha Adicionar à implantação existente e selecione a implantação que você deseja revisar.
   b. Para criar uma nova implantação em seu dispositivo de destino, escolha Criar nova implantação. Se você tiver uma implantação existente em seu dispositivo, escolher essa etapa substituirá a implantação existente.
6. Na página Especificar destino, faça o seguinte:
   a. Em Informações de implantação, insira ou modifique o nome amigável para sua implantação.
   b. Em Destinos de implantação, selecione um alvo para sua implantação e escolha Avançar. Você não pode alterar o destino de implantação se estiver revisando uma implantação existente.
7. Na página Selecionar componentes, em Componentes públicos, verifique se o componente de inferência com sua configuração modificada está selecionado e escolha Avançar.
8. Na página Configurar componentes, faça o seguinte:

a. Selecione o componente de inferência e escolha Configurar componente.

b. Em Atualização de configuração, insira os valores de configuração que você deseja atualizar. Por exemplo, insira a seguinte atualização de configuração na caixa Configuração a ser mesclada para alterar o intervalo de inferência para 15 segundos e instrua o componente a procurar a imagem nomeada `custom.jpg` na pasta. `/custom-ml-inference/images/`

```
{
  "InferenceInterval": "15",
  "ImageName": "custom.jpg",
  "ImageDirectory": "/custom-ml-inference/images/"
}
```

Para redefinir toda a configuração de um componente para seus valores padrão, especifique uma única string vazia `""` na caixa Redefinir caminhos.

c. Escolha Confirmar e, em seguida, Avançar.

9. Na página Definir configuração avançada, mantenha as configurações padrão e escolha Avançar.
10. Na página Revisar, escolha Implantar

Use um modelo personalizado com o componente de inferência de amostra

Se você quiser usar o componente de inferência de amostra com seus próprios modelos de aprendizado de máquina para um tempo de execução que AWS IoT Greengrass forneça um componente de tempo de execução de amostra, substitua os componentes do modelo público por componentes que usam esses modelos como artefatos. Em um alto nível, você conclui as etapas a seguir para usar um modelo personalizado com o componente de inferência de amostra:

1. Crie um componente de modelo que use um modelo personalizado em um bucket do S3 como artefato. Seu modelo personalizado deve ser treinado usando o mesmo tempo de execução do modelo que você deseja substituir.
2. Modifique o parâmetro de `ModelResourceKey` configuração no componente de inferência para usar o modelo personalizado. Para obter informações sobre como atualizar a configuração do componente de inferência, consulte Modificar a configuração de um componente de inferência pública

Quando você implanta o componente de inferência, AWS IoT Greengrass procura a versão mais recente de suas dependências de componentes. Ele substitui o componente do modelo público dependente se uma versão personalizada posterior do componente existir no mesmo e. Conta da AWS Região da AWS

Crie um componente de modelo personalizado (console)

1. Faça upload do seu modelo em um bucket do S3. Para obter informações sobre o upload de seus modelos em um bucket do S3, consulte Como trabalhar com os buckets do Amazon S3 no Guia do usuário do Amazon Simple Storage Service.

   > **Note**
   >
   > Você deve armazenar seus artefatos em buckets do S3 que estejam nos Região da AWS mesmos Conta da AWS componentes. Para permitir o AWS IoT Greengrass acesso a esses artefatos, a função do dispositivo Greengrass deve permitir `s3:GetObject` a ação. Para obter mais informações sobre a função do dispositivo, consulteAutorize os dispositivos principais a interagir com os serviços AWS.

2. No menu de navegação AWS IoT Greengrassdo console, escolha Componentes.
3. Recupere a receita do componente público do repositório de modelos.

   a. Na página Componentes, na guia Componentes públicos, procure e escolha o componente do modelo público para o qual você deseja criar uma nova versão. Por exemplo, `variant.DLR.ImageClassification.ModelStore`.

   b. Na página do componente, escolha Exibir receita e copie a receita JSON exibida.
4. Na página Componentes, na guia Meus componentes, escolha Criar componente.
5. Na página Criar componente, em Informações do componente, selecione Inserir receita como JSON como fonte do componente.
6. Na caixa Receita, cole a receita do componente que você copiou anteriormente.
7. Na receita, atualize os seguintes valores:

   - `ComponentVersion`: incremente a versão secundária do componente.

     Ao criar um componente personalizado para substituir um componente de modelo público, você deve atualizar somente a versão secundária da versão existente do componente. Por

exemplo, se a versão do componente público for`2.1.0`, você poderá criar um componente personalizado com a versão`2.1.1`.

- `Manifests.Artifacts.Uri`: atualize cada valor de URI para o URI do Amazon S3 do modelo que você deseja usar.

> **Note**
>
> Não altere o nome do componente.

8. Escolha Criar componente.

Crie um componente de modelo personalizado (AWS CLI)

1. Faça upload do seu modelo em um bucket do S3. Para obter informações sobre o upload de seus modelos em um bucket do S3, consulte Como trabalhar com os buckets do Amazon S3 no Guia do usuário do Amazon Simple Storage Service.

> **Note**
>
> Você deve armazenar seus artefatos em buckets do S3 que estejam nos Região da AWS mesmos Conta da AWS componentes. Para permitir o AWS IoT Greengrass acesso a esses artefatos, a função do dispositivo Greengrass deve permitir `s3:GetObject` a ação. Para obter mais informações sobre a função do dispositivo, consulteAutorize os dispositivos principais a interagir com os serviços AWS.

2. Execute o comando a seguir para recuperar a receita do componente público. Esse comando grava a receita do componente no arquivo de saída que você fornece em seu comando. Converta a string recuperada codificada em base64 em JSON ou YAML, conforme necessário.

   Linux, macOS, or Unix

```
aws greengrassv2 get-component \
    --arn <arn> \
    --recipe-output-format <recipe-format> \
    --query recipe \
    --output text | base64 --decode > <recipe-file>
```

Windows Command Prompt (CMD)

```
aws greengrassv2 get-component ^
    --arn <arn> ^
    --recipe-output-format <recipe-format> ^
    --query recipe ^
    --output text > <recipe-file>.base64

certutil -decode <recipe-file>.base64 <recipe-file>
```

PowerShell

```
aws greengrassv2 get-component `
    --arn <arn> `
    --recipe-output-format <recipe-format> `
    --query recipe `
    --output text > <recipe-file>.base64

certutil -decode <recipe-file>.base64 <recipe-file>
```

3. Atualize o nome do arquivo de receita para*<component-name>*-*<component-version>*, onde a versão do componente é a versão de destino do novo componente. Por exemplo, `variant.DLR.ImageClassification.ModelStore-2.1.1.yaml`.
4. Na receita, atualize os seguintes valores:

   - `ComponentVersion`: incremente a versão secundária do componente.

     Ao criar um componente personalizado para substituir um componente de modelo público, você deve atualizar somente a versão secundária da versão existente do componente. Por exemplo, se a versão do componente público for`2.1.0`, você poderá criar um componente personalizado com a versão`2.1.1`.

   - `Manifests.Artifacts.Uri`: atualize cada valor de URI para o URI do Amazon S3 do modelo que você deseja usar.

   > **Note**
   >
   > Não altere o nome do componente.

5. Execute o comando a seguir para criar um novo componente usando a receita que você recuperou e modificou.

```
aws greengrassv2 create-component-version \
    --inline-recipe fileb://path/to/component/recipe
```

> **Note**
>
> Essa etapa cria o componente no AWS IoT Greengrass serviço noNuvem AWS. Você pode usar a CLI do Greengrass para desenvolver, testar e implantar seu componente localmente antes de carregá-lo na nuvem. Para ter mais informações, consulte Desenvolva AWS IoT Greengrass componentes.

Para obter mais informações sobre a criação de componentes, consulteDesenvolva AWS IoT Greengrass componentes.

Crie componentes personalizados de aprendizado de máquina

Você deve criar componentes personalizados se quiser usar um código de inferência personalizado ou um tempo de execução para o qual AWS IoT Greengrass não forneça um componente de amostra. Você pode usar seu código de inferência personalizado com os exemplos AWS de modelos e tempos de execução de aprendizado de máquina fornecidos, ou pode desenvolver uma solução de inferência de aprendizado de máquina totalmente personalizada com seus próprios modelos e tempo de execução. Se seus modelos usam um tempo de execução que AWS IoT Greengrass fornece um componente de tempo de execução de amostra, você pode usar esse componente de tempo de execução e precisa criar componentes personalizados somente para seu código de inferência e os modelos que deseja usar.

Tópicos

- Recupere a receita de um componente público
- Recupere amostras de artefatos de componentes
- Carregar artefatos de componentes em um bucket do S3
- Crie componentes personalizados

Recupere a receita de um componente público

Você pode usar a receita de um componente público existente de aprendizado de máquina como modelo para criar um componente personalizado. Para ver a receita do componente para a versão mais recente de um componente público, use o console ou o AWS CLI seguinte:

- Como usar o console

  1. Na página Componentes, na guia Componentes públicos, procure e escolha o componente público.
  2. Na página do componente, escolha Exibir receita.

- Como usar o AWS CLI

  Execute o comando a seguir para recuperar a receita do componente da variante pública. Esse comando grava a receita do componente no arquivo de receita JSON ou YAML que você fornece em seu comando.

  Linux, macOS, or Unix

  ```
  aws greengrassv2 get-component \
      --arn <arn> \
      --recipe-output-format <recipe-format> \
      --query recipe \
      --output text | base64 --decode > <recipe-file>
  ```

  Windows Command Prompt (CMD)

  ```
  aws greengrassv2 get-component ^
      --arn <arn> ^
      --recipe-output-format <recipe-format> ^
      --query recipe ^
      --output text > <recipe-file>.base64

  certutil -decode <recipe-file>.base64 <recipe-file>
  ```

  PowerShell

  ```
  aws greengrassv2 get-component `
      --arn <arn> `
      --recipe-output-format <recipe-format> `
      --query recipe `
  ```

```
    --output text > <recipe-file>.base64

certutil -decode <recipe-file>.base64 <recipe-file>
```

Substitua os valores em seu comando da seguinte forma:

- *<arn>*. O Amazon Resource Name (ARN) do componente público.
- *<recipe-format>*. O formato no qual você deseja criar o arquivo de receita. Os valores compatíveis são `JSON` e `YAML`.
- *<recipe-file>*. O nome da receita no formato*<component-name>*-*<component-version>*.

Recupere amostras de artefatos de componentes

Você pode usar os artefatos usados pelos componentes públicos de aprendizado de máquina como modelos para criar seus artefatos de componentes personalizados, como código de inferência ou scripts de instalação em tempo de execução.

Para visualizar os artefatos de amostra que estão incluídos nos componentes públicos de aprendizado de máquina, implante o componente de inferência pública e, em seguida, visualize os artefatos em seu dispositivo na pasta. */greengrass/v2*`/packages/artifacts-unarchived/`*component-name*`/`*component-version*`/`

Carregar artefatos de componentes em um bucket do S3

Antes de criar um componente personalizado, você deve carregar os artefatos do componente em um bucket do S3 e usar os URIs do S3 na receita do componente. Por exemplo, para usar um código de inferência personalizado em seu componente de inferência, faça o upload do código em um bucket do S3. Em seguida, você pode usar o URI do Amazon S3 do seu código de inferência como um artefato em seu componente.

Para obter informações sobre o upload de conteúdo para um bucket do S3, consulte Como trabalhar com os buckets do Amazon S3 no Guia do usuário do Amazon Simple Storage Service.

> **Note**
>
> Você deve armazenar seus artefatos em buckets do S3 que estejam nos Região da AWS mesmos Conta da AWS componentes. Para permitir o AWS IoT Greengrass acesso a esses artefatos, a função do dispositivo Greengrass deve permitir `s3:GetObject` a ação. Para

obter mais informações sobre a função do dispositivo, consulteAutorize os dispositivos principais a interagir com os serviços AWS.

Crie componentes personalizados

Você pode usar os artefatos e receitas recuperados para criar seus componentes personalizados de aprendizado de máquina. Para ver um exemplo, consulte Crie um componente de inferência personalizado.

Para obter informações detalhadas sobre a criação e implantação de componentes em dispositivos Greengrass, Desenvolva AWS IoT Greengrass componentes consulte e. Implemente AWS IoT Greengrass componentes em dispositivos

Crie um componente de inferência personalizado

Esta seção mostra como criar um componente de inferência personalizado usando o componente de classificação de imagem DLR como modelo.

Tópicos

- Faça upload do seu código de inferência em um bucket do Amazon S3
- Crie uma receita para seu componente de inferência
- Crie o componente de inferência

Faça upload do seu código de inferência em um bucket do Amazon S3

Crie seu código de inferência e, em seguida, carregue-o em um bucket do S3. Para obter informações sobre o upload de conteúdo para um bucket do S3, consulte Como trabalhar com os buckets do Amazon S3 no Guia do usuário do Amazon Simple Storage Service.

> **Note**
>
> Você deve armazenar seus artefatos em buckets do S3 que estejam nos Região da AWS mesmos Conta da AWS componentes. Para permitir o AWS IoT Greengrass acesso a esses artefatos, a função do dispositivo Greengrass deve permitir `s3:GetObject` a ação. Para obter mais informações sobre a função do dispositivo, consulteAutorize os dispositivos principais a interagir com os serviços AWS.

Crie uma receita para seu componente de inferência

1. Execute o comando a seguir para recuperar a receita do componente de classificação de imagem DLR. Esse comando grava a receita do componente no arquivo de receita JSON ou YAML que você fornece em seu comando.

   Linux, macOS, or Unix

   ```
   aws greengrassv2 get-component \
       --arn
    arn:aws:greengrass:region:aws:components:aws.greengrass.DLRImageClassification:versions
    \
       --recipe-output-format JSON | YAML \
       --query recipe \
       --output text | base64 --decode > <recipe-file>
   ```

   Windows Command Prompt (CMD)

   ```
   aws greengrassv2 get-component ^
       --arn
    arn:aws:greengrass:region:aws:components:aws.greengrass.DLRImageClassification:versions
    ^
       --recipe-output-format JSON | YAML ^
       --query recipe ^
       --output text > <recipe-file>.base64

   certutil -decode <recipe-file>.base64 <recipe-file>
   ```

   PowerShell

   ```
   aws greengrassv2 get-component `
       --arn
    arn:aws:greengrass:region:aws:components:aws.greengrass.DLRImageClassification:versions
    `
       --recipe-output-format JSON | YAML `
       --query recipe `
       --output text > <recipe-file>.base64

   certutil -decode <recipe-file>.base64 <recipe-file>
   ```

<recipe-file>Substitua pelo nome da receita no formato*<component-name>*-*<component-version>*.

2. No `ComponentDependencies` objeto em sua receita, faça um ou mais dos seguintes, dependendo do modelo e dos componentes de tempo de execução que você deseja usar:

   - Mantenha a dependência do componente DLR se quiser usar modelos compilados pelo DLR. Você também pode substituí-lo por uma dependência em um componente de tempo de execução personalizado, conforme mostrado no exemplo a seguir.

     Componente Runtime

     JSON

     ```
     {
         "<runtime-component>": {
             "VersionRequirement": "<version>",
             "DependencyType": "HARD"
         }
     }
     ```

     YAML

     ```
     <runtime-component>:
         VersionRequirement: "<version>"
         DependencyType: HARD
     ```

   - Mantenha a dependência do armazenamento do modelo de classificação de imagem DLR para usar os modelos ResNet -50 pré-treinados que AWS fornecem, ou modifique-o para usar um componente de modelo personalizado. Quando você inclui uma dependência para um componente de modelo público, se uma versão personalizada posterior do componente existir no mesmo Conta da AWSRegião da AWS, o componente de inferência usará esse componente personalizado. Especifique a dependência do componente do modelo conforme mostrado nos exemplos a seguir.

     Componente de modelo público

     JSON

     ```
     {
         "variant.DLR.ImageClassification.ModelStore": {
     ```

```
        "VersionRequirement": "<version>",
        "DependencyType": "HARD"
    }
}
```

YAML

```
variant.DLR.ImageClassification.ModelStore:
    VersionRequirement: "<version>"
    DependencyType: HARD
```

Componente de modelo personalizado

JSON

```
{
    "<custom-model-component>": {
        "VersionRequirement": "<version>",
        "DependencyType": "HARD"
    }
}
```

YAML

```
<custom-model-component>:
    VersionRequirement: "<version>"
    DependencyType: HARD
```

3. No `ComponentConfiguration` objeto, adicione a configuração padrão para esse componente. Posteriormente, você poderá modificar essa configuração ao implantar o componente. O trecho a seguir mostra a configuração do componente de classificação de imagem DLR.

   Por exemplo, se você usar um componente de modelo personalizado como uma dependência para seu componente de inferência personalizado, modifique `ModelResourceKey` para fornecer os nomes dos modelos que você está usando.

   JSON

```
{
  "accessControl": {
    "aws.greengrass.ipc.mqttproxy": {
```

```
      "aws.greengrass.ImageClassification:mqttproxy:1": {
        "policyDescription": "Allows access to publish via topic ml/dlr/image-
classification.",
        "operations": [
          "aws.greengrass#PublishToIoTCore"
        ],
        "resources": [
          "ml/dlr/image-classification"
        ]
      }
    }
  },
  "PublishResultsOnTopic": "ml/dlr/image-classification",
  "ImageName": "cat.jpeg",
  "InferenceInterval": 3600,
  "ModelResourceKey": {
    "armv7l": "DLR-resnet50-armv7l-cpu-ImageClassification",
    "x86_64": "DLR-resnet50-x86_64-cpu-ImageClassification",
    "aarch64": "DLR-resnet50-aarch64-cpu-ImageClassification"
  }
}
```

YAML

```
accessControl:
    aws.greengrass.ipc.mqttproxy:
        'aws.greengrass.ImageClassification:mqttproxy:1':
            policyDescription: 'Allows access to publish via topic ml/dlr/image-
classification.'
            operations:
                - 'aws.greengrass#PublishToIoTCore'
            resources:
                - ml/dlr/image-classification
PublishResultsOnTopic: ml/dlr/image-classification
ImageName: cat.jpeg
InferenceInterval: 3600
ModelResourceKey:
    armv7l: "DLR-resnet50-armv7l-cpu-ImageClassification"
    x86_64: "DLR-resnet50-x86_64-cpu-ImageClassification"
    aarch64: "DLR-resnet50-aarch64-cpu-ImageClassification"
```

4. No `Manifests` objeto, forneça informações sobre os artefatos e a configuração desse componente que são usados quando o componente é implantado em plataformas diferentes

e qualquer outra informação necessária para executar o componente com êxito. O trecho a seguir mostra a configuração do `Manifests` objeto para a plataforma Linux no componente de classificação de imagem DLR.

JSON

```
{
  "Manifests": [
    {
      "Platform": {
        "os": "linux",
        "architecture": "arm"
      },
      "Name": "32-bit armv7l - Linux (raspberry pi)",
      "Artifacts": [
        {
          "URI": "s3://SAMPLE-BUCKET/sample-artifacts-directory/
image_classification.zip",
          "Unarchive": "ZIP"
        }
      ],
      "Lifecycle": {
        "Setenv": {
          "DLR_IC_MODEL_DIR":
 "{variant.DLR.ImageClassification.ModelStore:artifacts:decompressedPath}/
{configuration:/ModelResourceKey/armv7l}",
          "DEFAULT_DLR_IC_IMAGE_DIR": "{artifacts:decompressedPath}/
image_classification/sample_images/"
        },
        "run": {
          "RequiresPrivilege": true,
          "script": ". {variant.DLR:configuration:/MLRootPath}/
greengrass_ml_dlr_venv/bin/activate\npython3 {artifacts:decompressedPath}/
image_classification/inference.py"
        }
      }
    }
  ]
}
```

YAML

```
Manifests:
  - Platform:
      os: linux
      architecture: arm
    Name: 32-bit armv7l - Linux (raspberry pi)
    Artifacts:
      - URI: s3://SAMPLE-BUCKET/sample-artifacts-directory/
image_classification.zip
        Unarchive: ZIP
    Lifecycle:
      Setenv:
        DLR_IC_MODEL_DIR:
 "{variant.DLR.ImageClassification.ModelStore:artifacts:decompressedPath}/
{configuration:/ModelResourceKey/armv7l}"
        DEFAULT_DLR_IC_IMAGE_DIR: "{artifacts:decompressedPath}/
image_classification/sample_images/"
      run:
        RequiresPrivilege: true
        script: |-
          . {variant.DLR:configuration:/MLRootPath}/greengrass_ml_dlr_venv/bin/
activate
          python3 {artifacts:decompressedPath}/image_classification/inference.py
```

Para obter informações detalhadas sobre a criação de receitas de componentes, consulteAWS IoT Greengrass referência da receita do componente.

Crie o componente de inferência

Use o AWS IoT Greengrass console ou o AWS CLI para criar um componente usando a receita que você acabou de definir. Depois de criar o componente, você pode implantá-lo para realizar inferências em seu dispositivo. Para obter um exemplo de como implantar um componente de inferência, consulteTutorial: Execute inferência de classificação de imagens de amostra usando o Lite TensorFlow .

Crie um componente de inferência personalizado (console)

1. Faça login no AWS IoT Greengrassconsole.
2. No menu de navegação, escolha Componentes.

3. Na página Componentes, na guia Meus componentes, escolha Criar componente.
4. Na página Criar componente, em Informações do componente, selecione Inserir receita como JSON ou Inserir receita como YAML como fonte do componente.
5. Na caixa Receita, insira a receita personalizada que você criou.
6. Clique em Criar componente.

Crie um componente de inferência personalizado () AWS CLI

Execute o comando a seguir para criar um novo componente personalizado usando a receita que você criou.

```
aws greengrassv2 create-component-version \
    --inline-recipe fileb://path/to/recipe/file
```

> **Note**
>
> Essa etapa cria o componente no AWS IoT Greengrass serviço noNuvem AWS. Você pode usar a CLI do Greengrass para desenvolver, testar e implantar seu componente localmente antes de carregá-lo na nuvem. Para ter mais informações, consulte Desenvolva AWS IoT Greengrass componentes.

Solução de problemas de inferência de aprendizado de máquina

Use as informações e soluções de solução de problemas nesta seção para ajudar a resolver problemas com seus componentes de aprendizado de máquina. Para os componentes públicos de inferência de aprendizado de máquina, consulte as mensagens de erro nos seguintes registros de componentes:

Linux or Unix

- */greengrass/v2*/logs/aws.greengrass.DLRImageClassification.log
- */greengrass/v2*/logs/aws.greengrass.DLRObjectDetection.log
- */greengrass/v2*/logs/aws.greengrass.TensorFlowLiteImageClassification.log
- */greengrass/v2*/logs/aws.greengrass.TensorFlowLiteObjectDetection.log

Windows

- *C:\greengrass\v2*\logs\aws.greengrass.DLRImageClassification.log
- *C:\greengrass\v2*\logs\aws.greengrass.DLRObjectDetection.log
- *C:\greengrass\v2*\logs\aws.greengrass.TensorFlowLiteImageClassification.log
- *C:\greengrass\v2*\logs\aws.greengrass.TensorFlowLiteObjectDetection.log

Se um componente for instalado corretamente, o registro do componente conterá a localização da biblioteca que ele usa para inferência.

Problemas

- Falha ao buscar a biblioteca
- Cannot open shared object file
- Error: ModuleNotFoundError: No module named '<library>'
- Nenhum dispositivo compatível com CUDA foi detectado
- Esse arquivo ou diretório não existe
- RuntimeError: module compiled against API version 0xf but this version of NumPy is <version>
- picamera.exc.PiCameraError: Camera is not enabled
- Erros de memória
- Erros de espaço em disco
- Erros de tempo limite

Falha ao buscar a biblioteca

O erro a seguir ocorre quando o script do instalador não consegue baixar uma biblioteca necessária durante a implantação em um dispositivo Raspberry Pi.

```
Err:2 http://raspbian.raspberrypi.org/raspbian buster/main armhf python3.7-dev armhf
 3.7.3-2+deb10u1
404 Not Found [IP: 93.93.128.193 80]
E: Failed to fetch http://raspbian.raspberrypi.org/raspbian/pool/main/p/python3.7/
libpython3.7-dev_3.7.3-2+deb10u1_armhf.deb 404 Not Found [IP: 93.93.128.193 80]
```

Execute `sudo apt-get update` e implante seu componente novamente.

Cannot open shared object file

Você pode ver erros semelhantes aos seguintes quando o script do instalador não consegue baixar uma dependência necessária `opencv-python` durante a implantação em um dispositivo Raspberry Pi.

```
ImportError: libopenjp2.so.7: cannot open shared object file: No such file or directory
```

Execute o comando a seguir para instalar manualmente as dependências para`opencv-python`:

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
 libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

Error: ModuleNotFoundError: No module named '<library>'

Talvez você veja esse erro nos registros do componente de tempo de execução de ML (`variant.DLR.log`ou`variant.TensorFlowLite.log`) quando a biblioteca de tempo de execução de ML ou suas dependências não estiverem instaladas corretamente. Esse erro pode ocorrer nos seguintes casos:

- Se você usar a `UseInstaller` opção, que é ativada por padrão, esse erro indica que o componente de tempo de execução de ML falhou ao instalar o tempo de execução ou suas dependências. Faça o seguinte:
  1. Configure o componente de tempo de execução de ML para desativar a `UseInstaller` opção.
  2. Instale o tempo de execução do ML e suas dependências e disponibilize-os para o usuário do sistema que executa os componentes do ML. Para ver mais informações, consulte:
     - Opção de tempo de execução UseInstaller do DLR
     - TensorFlow UseInstaller Opção Lite Runtime
- Se você não usar a `UseInstaller` opção, esse erro indica que o tempo de execução do ML ou suas dependências não estão instalados para o usuário do sistema que executa os componentes do ML. Faça o seguinte:
  1. Verifique se a biblioteca está instalada para o usuário do sistema que executa os componentes de ML. Substitua *`ggc_user`* *`pelo`* nome do usuário do sistema e substitua *`tflite_runtime`* pelo nome da biblioteca a ser verificada.

Linux or Unix

```
sudo -H -u ggc_user bash -c "python3 -c 'import tflite_runtime'"
```

Windows

```
runas /user:ggc_user "py -3 -c \"import tflite_runtime\""
```

2. Se a biblioteca não estiver instalada, instale-a para esse usuário. Substitua *ggc_user pelo* nome do usuário do sistema e substitua *tflite_runtime* pelo nome da biblioteca.

   Linux or Unix

   ```
   sudo -H -u ggc_user bash -c "python3 -m pip install --user tflite_runtime"
   ```

   Windows

   ```
   runas /user:ggc_user "py -3 -m pip install --user tflite_runtime"
   ```

   Para obter mais informações sobre as dependências de cada tempo de execução de ML, consulte o seguinte:

   - Opção de tempo de execução UseInstaller do DLR
   - TensorFlow UseInstaller Opção Lite Runtime

3. Se o problema persistir, instale a biblioteca para outro usuário para confirmar se esse dispositivo pode instalar a biblioteca. O usuário pode ser, por exemplo, seu usuário, o usuário root ou um usuário administrador. Se você não conseguir instalar a biblioteca com êxito para nenhum usuário, talvez seu dispositivo não seja compatível com a biblioteca. Consulte a documentação da biblioteca para analisar os requisitos e solucionar problemas de instalação.

Nenhum dispositivo compatível com CUDA foi detectado

Você pode ver o erro a seguir ao usar a aceleração de GPU. Execute o comando a seguir para habilitar o acesso à GPU para o usuário do Greengrass.

```
sudo usermod -a -G video ggc_user
```

Esse arquivo ou diretório não existe

Os erros a seguir indicam que o componente de tempo de execução não conseguiu configurar o ambiente virtual corretamente:

- *MLRootPath*`/greengrass_ml_dlr_conda/bin/conda: No such file or directory`
- *MLRootPath*`/greengrass_ml_dlr_venv/bin/activate: No such file or directory`
- *MLRootPath*`/greengrass_ml_tflite_conda/bin/conda: No such file or directory`
- *MLRootPath*`/greengrass_ml_tflite_venv/bin/activate: No such file or directory`

Verifique os registros para se certificar de que todas as dependências de tempo de execução foram instaladas corretamente. Para obter mais informações sobre as bibliotecas instaladas pelo script do instalador, consulte os tópicos a seguir:

- Tempo de execução do DLR
- TensorFlow Tempo de execução leve

Por padrão, o *ML RootPath* está definido como*/greengrass/v2*`/work/`*component-name*`/greengrass_ml`. Para alterar esse local, inclua o componente Tempo de execução do DLR ou TensorFlow Tempo de execução leve runtime diretamente em sua implantação e especifique um valor modificado para o `MLRootPath` parâmetro em uma atualização de mesclagem de configuração. Para obter mais informações sobre a configuração do componente, consulteAtualizar configurações de componentes.

> **Note**
>
> Para o componente DLR v1.3.x, você define o `MLRootPath` parâmetro na configuração do componente de inferência e o valor padrão é. `$HOME/greengrass_ml`

RuntimeError: module compiled against API version 0xf but this version of NumPy is <version>

Você pode ver os seguintes erros ao executar a inferência de aprendizado de máquina em um Raspberry Pi executando o Raspberry Pi OS Bullseye.

```
RuntimeError: module compiled against API version 0xf but this version of numpy is 0xd
ImportError: numpy.core.multiarray failed to import
```

Esse erro ocorre porque o Raspberry Pi OS Bullseye inclui uma versão anterior à versão exigida pelo NumPy OpenCV. Para corrigir esse problema, execute o comando a seguir para atualizar NumPy para a versão mais recente.

```
pip3 install --upgrade numpy
```

picamera.exc.PiCameraError: Camera is not enabled

Você pode ver o seguinte erro ao executar a inferência de aprendizado de máquina em um Raspberry Pi executando o Raspberry Pi OS Bullseye.

```
picamera.exc.PiCameraError: Camera is not enabled. Try running 'sudo raspi-config' and
 ensure that the camera has been enabled.
```

Esse erro ocorre porque o Raspberry Pi OS Bullseye inclui uma nova pilha de câmeras que não é compatível com os componentes de ML. Para corrigir esse problema, ative a pilha de câmeras antiga.

Para habilitar a pilha de câmeras antiga

1. Execute o comando a seguir para abrir a ferramenta de configuração do Raspberry Pi.

   ```
   sudo raspi-config
   ```

2. Selecione Opções de interface.
3. Selecione Câmera antiga para ativar a pilha de câmeras antigas.
4. Reinicie o Raspberry Pi.

Erros de memória

Os erros a seguir geralmente ocorrem quando o dispositivo não tem memória suficiente e o processo do componente é interrompido.

- `stderr. Killed.`
- `exitCode=137`

Recomendamos um mínimo de 500 MB de memória para implantar um componente público de inferência de aprendizado de máquina.

Erros de espaço em disco

O `no space left on device` erro geralmente ocorre quando um dispositivo não tem armazenamento suficiente. Verifique se há espaço em disco suficiente disponível em seu dispositivo antes de implantar o componente novamente. Recomendamos um mínimo de 500 MB de espaço livre em disco para implantar um componente público de inferência de aprendizado de máquina.

Erros de tempo limite

Os componentes públicos de aprendizado de máquina baixam grandes arquivos de modelo de aprendizado de máquina com mais de 200 MB. Se o download expirar durante a implantação, verifique a velocidade da sua conexão com a Internet e tente implantar novamente.

Gerencie os principais dispositivos do Greengrass com AWS Systems Manager

> **Note**
>
> AWS IoT Greengrassatualmente não oferece suporte a esse recurso nos dispositivos principais do Windows.

O Systems Manager é um AWS serviço que você pode usar para visualizar e controlar sua infraestruturaAWS, incluindo instâncias do Amazon EC2, servidores e máquinas virtuais (VMs) locais e dispositivos periféricos. O Systems Manager permite que você visualize dados operacionais, automatize tarefas operacionais e mantenha a segurança e a conformidade. Quando você registra uma máquina no Systems Manager, ela é chamada de nó gerenciado. Para obter mais informações, consulte O que é o AWS Systems Manager? no Guia do usuário do AWS Systems Manager.

O AWS Systems Manager Agente (Systems Manager Agent) é um software que você pode instalar em dispositivos para permitir que o Systems Manager os atualize, gerencie e configure. Para instalar o Systems Manager Agent nos dispositivos principais do Greengrass, implante o componente Systems Manager Agent. Quando você implanta o Systems Manager Agent pela primeira vez, ele registra o dispositivo principal como um nó gerenciado do Systems Manager. O Systems Manager Agent é executado no dispositivo para permitir a comunicação com o serviço Systems Manager noNuvem AWS. Para obter mais informações sobre como instalar e configurar o componente Systems Manager Agent, consulteInstalar o agente do AWS Systems Manager.

As ferramentas e os recursos do Systems Manager são chamados de capacidades. Os dispositivos principais do Greengrass oferecem suporte a todos os recursos do Systems Manager. Para obter mais informações sobre esses recursos e como usar o Systems Manager para gerenciar dispositivos principais, consulte os recursos do Systems Manager no Guia AWS Systems Manager do Usuário.

AWS Systems Manageroferece uma camada de instâncias padrão e uma camada de instâncias avançadas para nós gerenciados do Systems Manager. Se você estiver usando o Systems Manager pela primeira vez, comece no nível de instâncias padrão. No nível de instâncias padrão, você pode registrar até 1.000 nós gerenciados por cadaRegião da AWS. Conta da AWS Se você precisar registrar mais de 1.000 nós gerenciados em uma única conta e região, ou se precisar usar o recurso Gerenciador de Sessões, use o nível de instâncias avançadas. Para obter mais informações, consulte Como configurar níveis de instância no Guia do AWS Systems Managerusuário.

Tópicos

Instalar o agente do AWS Systems Manager

O AWS Systems Manager Agent (Systems Manager Agent) é um software da Amazon que você instala para permitir que o Systems Manager atualize, gerencie e configure os principais dispositivos do Greengrass, instâncias do Amazon EC2 e outros recursos. O agente processa e executa solicitações do serviço Systems Manager noNuvem AWS. Em seguida, o agente envia as informações de status e tempo de execução para o serviço Systems Manager. Para obter mais informações, consulte Sobre o Systems Manager Agent no Guia AWS Systems Manager do Usuário.

AWSfornece o Systems Manager Agent como um componente do Greengrass que você pode implantar em seus dispositivos principais do Greengrass para gerenciá-los com o Systems Manager. O componente Systems Manager Agent instala o software Systems Manager Agent e registra o dispositivo principal como um nó gerenciado no Systems Manager. Siga as etapas desta página para concluir os pré-requisitos e implantar o componente Systems Manager Agent em um dispositivo principal ou grupo de dispositivos principais.

Tópicos

Etapa 1: Concluir as etapas gerais de configuração do Systems Manager

Se você ainda não tiver feito isso, conclua as etapas gerais de configuração doAWS Systems Manager. Para obter mais informações, consulte as etapas gerais completas de configuração do Systems Manager no Guia AWS Systems Manager do usuário.

Etapa 2: criar uma função de serviço do IAM para Systems Manager

O Systems Manager Agent usa uma função de serviço AWS Identity and Access Management (IAM) para se comunicar comAWS Systems Manager. O Systems Manager assume essa função para habilitar os recursos do Systems Manager em cada dispositivo principal. O componente Systems Manager Agent também usa essa função para registrar o dispositivo principal como um nó gerenciado do Systems Manager quando você implanta o componente. Se você ainda não tiver feito isso, crie uma função de serviço do Systems Manager para usar o componente Systems Manager Agent. Para obter mais informações, consulte Criar uma função de serviço do IAM para dispositivos periféricos no Guia AWS Systems Manager do usuário.

Etapa 3: adicionar permissões à função de troca de tokens

Os dispositivos principais do Greengrass usam uma função de serviço do IAM, chamada função de troca de tokens, para interagir com AWS os serviços. Cada dispositivo principal tem uma função de troca de tokens que você cria ao instalar o software AWS IoT Greengrass Core. Muitos componentes do Greengrass, como o Systems Manager Agent, exigem permissões adicionais nessa função. O componente de agente do Systems Manager requer as seguintes permissões, que incluem permissão para usar a função que você criou emEtapa 2: criar uma função de serviço do IAM para Systems Manager.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "iam:PassRole"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:iam::account-id:role/SSMServiceRole"
      ]
    },
    {
      "Action": [
        "ssm:AddTagsToResource",
        "ssm:RegisterManagedInstance"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
```

```
  ]
}
```

Se você ainda não tiver feito isso, adicione essas permissões à função de troca de tokens do dispositivo principal para permitir que o Systems Manager Agent opere. Você pode adicionar uma nova política à função de troca de tokens para conceder essa permissão.

Para adicionar permissões à função de troca de tokens (console)

1. No menu de navegação do console do IAM, escolha Roles.
2. Escolha a função do IAM que você configurou como função de troca de tokens ao instalar o software AWS IoT Greengrass Core. Se você não especificou um nome para a função de troca de tokens ao instalar o software AWS IoT Greengrass Core, ele criou uma função chamada`GreengrassV2TokenExchangeRole`.
3. Em Permissões, escolha Adicionar permissões e, em seguida, escolha Anexar políticas.
4. Escolha Criar política. A página Criar política é aberta em uma nova guia do navegador.
5. Na página Create policy (Criar política) faça o seguinte:

   a. Escolha JSON para abrir o editor JSON.

   b. Cole a política a seguir no editor de JSON. Substitua *`SSM ServiceRole`* pelo nome da função de serviço que você criou emEtapa 2: criar uma função de serviço do IAM para Systems Manager.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "iam:PassRole"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:iam::account-id:role/SSMServiceRole"
      ]
    },
    {
      "Action": [
        "ssm:AddTagsToResource",
        "ssm:RegisterManagedInstance"
      ],
```

```
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

c. Escolha Próximo: etiquetas.

d. Escolha Próximo: revisar.

e. Insira um Name (Nome) para a política, como **`GreengrassSSMAgentComponentPolicy`**.

f. Escolha Criar política.

g. Alterne para a guia anterior do navegador, onde você tem a função de troca de tokens aberta.

6. Na página Adicionar permissões, escolha o botão Atualizar e, em seguida, selecione a política de agente do Greengrass Systems Manager que você criou na etapa anterior.

7. Escolha Anexar políticas.

   Os dispositivos principais que usam essa função de troca de tokens agora têm permissão para interagir com o serviço Systems Manager.

Para adicionar permissões à função de troca de tokens (AWS CLI)

Para adicionar uma política que conceda permissão para usar o Systems Manager

1. Crie um arquivo chamado `ssm-agent-component-policy.json` e copie o seguinte JSON para o arquivo. Substitua *`SSM ServiceRole`* pelo nome da função de serviço que você criou emEtapa 2: criar uma função de serviço do IAM para Systems Manager.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "iam:PassRole"
      ],
      "Effect": "Allow",
      "Resource": [
        "arn:aws:iam::account-id:role/SSMServiceRole"
      ]
    },
```

```
    {
      "Action": [
        "ssm:AddTagsToResource",
        "ssm:RegisterManagedInstance"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

2. Execute o comando a seguir para criar a política a partir do documento de política em `ssm-agent-component-policy.json`.

   Linux or Unix

   ```
   aws iam create-policy \
     --policy-name GreengrassSSMAgentComponentPolicy \
     --policy-document file://ssm-agent-component-policy.json
   ```

   Windows Command Prompt (CMD)

   ```
   aws iam create-policy ^
     --policy-name GreengrassSSMAgentComponentPolicy ^
     --policy-document file://ssm-agent-component-policy.json
   ```

   PowerShell

   ```
   aws iam create-policy `
     --policy-name GreengrassSSMAgentComponentPolicy `
     --policy-document file://ssm-agent-component-policy.json
   ```

   Copie a política Amazon Resource Name (ARN) dos metadados da política na saída. Você usa esse ARN para anexar essa política à função do dispositivo principal na próxima etapa.

3. Execute o comando a seguir para anexar a política à função de troca de tokens.

   - Substitua *GreengrassV2 TokenExchangeRole* pelo nome da função de troca de tokens que você especificou ao instalar o software Core. AWS IoT Greengrass Se você não especificou um nome para a função de troca de tokens ao instalar o software AWS IoT Greengrass Core, ele criou uma função chamada`GreengrassV2TokenExchangeRole`.

- Substitua o ARN da política pelo ARN da etapa anterior.

Linux or Unix

```
aws iam attach-role-policy \
  --role-name GreengrassV2TokenExchangeRole \
  --policy-arn
 arn:aws:iam::123456789012:policy/GreengrassSSMAgentComponentPolicy
```

Windows Command Prompt (CMD)

```
aws iam attach-role-policy ^
  --role-name GreengrassV2TokenExchangeRole ^
  --policy-arn
 arn:aws:iam::123456789012:policy/GreengrassSSMAgentComponentPolicy
```

PowerShell

```
aws iam attach-role-policy `
  --role-name GreengrassV2TokenExchangeRole `
  --policy-arn
 arn:aws:iam::123456789012:policy/GreengrassSSMAgentComponentPolicy
```

Se o comando não tiver saída, ele foi bem-sucedido. Os dispositivos principais que usam essa função de troca de tokens agora têm permissão para interagir com o serviço Systems Manager.

Etapa 4: Implantar o componente Systems Manager Agent

Conclua as etapas a seguir para implantar e configurar o componente Systems Manager Agent. Você pode implantar o componente em um único dispositivo central ou em um grupo de dispositivos principais.

Para implantar o componente Systems Manager Agent (console)

1. No menu de navegação AWS IoT Greengrassdo console, escolha Componentes.
2. Na página Componentes, escolha a guia Componentes públicos e, em seguida, escolha aws.greengrass.SystemsManagerAgent.

3. Na página aws.greengrass.SystemsManagerAgent, escolha Implantar.
4. Em Adicionar à implantação, escolha uma implantação existente para revisar ou opte por criar uma nova implantação e, em seguida, escolha Avançar.
5. Se você optar por criar uma nova implantação, escolha o dispositivo principal ou grupo de itens de destino para a implantação. Na página Especificar destino, em Destino de implantação, escolha um dispositivo principal ou grupo de itens e, em seguida, escolha Avançar.
6. Na página Selecionar componentes, verifique se o aws.greengrass.SystemsManagerAgentcomponente está selecionado e escolha Avançar.
7. Na página Configurar componentes aws.greengrass.SystemsManagerAgent, selecione e faça o seguinte:

   a. Escolha Configurar componente.

   b. No aws.greengrass.SystemsManagerAgent modal Configurar, em Atualização de configuração, em Configuração para mesclar, insira a seguinte atualização de configuração. Substitua *`SSM ServiceRole`* pelo nome da função de serviço que você criou em[Etapa 2: criar uma função de serviço do IAM para Systems Manager](#).

   ```
   {
     "SSMRegistrationRole": "SSMServiceRole",
     "SSMOverrideExistingRegistration": false
   }
   ```

   > **Note**
   >
   > Se o dispositivo principal já executa o Systems Manager Agent registrado com uma ativação híbrida, mude `SSMOverrideExistingRegistration` para`true`. Esse parâmetro especifica se o componente Systems Manager Agent registra o dispositivo principal quando o Systems Manager Agent já está sendo executado no dispositivo com uma ativação híbrida.
   > Você também pode especificar tags (`SSMResourceTags`) para adicionar ao nó gerenciado do Systems Manager que o componente Systems Manager Agent cria para o dispositivo principal. Para obter mais informações, consulte [Configuração do componente do Systems Manager Agent](#).

   c. Escolha Confirmar para fechar o modal e, em seguida, escolha Avançar.

8. Na página Definir configurações avançadas, mantenha as configurações padrão e escolha Avançar.
9. Na página Review, escolha Deploy.

   A implantação pode levar até um minuto para ser concluída.

Para implantar o componente Systems Manager Agent (AWS CLI)

Para implantar o componente Systems Manager Agent, crie um documento de implantação que inclua `aws.greengrass.SystemsManagerAgent` no `components` objeto e especifique a atualização de configuração para o componente. Siga as instruções Criar implantações para criar uma nova implantação ou revisar uma implantação existente.

O exemplo de documento de implantação parcial a seguir especifica o uso de uma função de serviço chamada`SSMServiceRole`. Substitua *`SSM ServiceRole`* pelo nome da função de serviço que você criou emEtapa 2: criar uma função de serviço do IAM para Systems Manager.

```
{
  ...,
  "components": {
    ...,
    "aws.greengrass.SystemsManagerAgent": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "merge": "{\"SSMRegistrationRole\":\"SSMServiceRole\",
\"SSMOverrideExistingRegistration\":false}"
      }
    }
  }
}
```

> **Note**
>
> Se o dispositivo principal já executa o Systems Manager Agent registrado com uma ativação híbrida, mude `SSMOverrideExistingRegistration` para`true`. Esse parâmetro especifica se o componente Systems Manager Agent registra o dispositivo principal quando o Systems Manager Agent já está sendo executado no dispositivo com uma ativação híbrida. Você também pode especificar tags (`SSMResourceTags`) para adicionar ao nó gerenciado do Systems Manager que o componente Systems Manager Agent cria para o dispositivo

principal. Para obter mais informações, consulte Configuração do componente do Systems Manager Agent.

A implantação pode levar vários minutos para ser concluída. Você pode usar o AWS IoT Greengrass serviço para verificar o status da implantação e verificar os registros do software AWS IoT Greengrass principal e os registros de componentes do Systems Manager Agent para verificar se o Systems Manager Agent é executado com êxito. Para ver mais informações, consulte:

- Verificar status da implantação
- Monitore AWS IoT Greengrass os registros
- Visualizando registros do Systems Manager Agent no Guia AWS Systems Manager do Usuário

Se a implantação falhar ou o Systems Manager Agent não for executado, você poderá solucionar o problema da implantação em cada dispositivo principal. Para ver mais informações, consulte:

- Solução de problemas AWS IoT Greengrass V2
- Solução de problemas do Systems Manager Agent no Guia AWS Systems Manager do Usuário

Etapa 5: Verificar o registro do dispositivo principal com o Systems Manager

Quando o componente Systems Manager Agent é executado, ele registra o dispositivo principal como um nó gerenciado no Systems Manager. Você pode usar o AWS IoT Greengrass console, o console do Systems Manager e a API do Systems Manager para verificar se um dispositivo principal está registrado como um nó gerenciado. Os nós gerenciados também são chamados de instâncias em partes do AWS console e da API.

Para verificar o registro do dispositivo principal (AWS IoT Greengrassconsole)

1. No menu de navegação AWS IoT Greengrassdo console, escolha Dispositivos principais.
2. Escolha o dispositivo principal para verificar.
3. Na página de detalhes do dispositivo principal, encontre a propriedade da AWS Systems Managerinstância. Se essa propriedade estiver presente e exibir um link para o console do Systems Manager, o dispositivo principal será registrado como um nó gerenciado.

Você também pode encontrar a propriedade AWS Systems Managerping status para verificar o status do Systems Manager Agent no dispositivo principal. Quando o status é Online, você pode gerenciar o dispositivo principal com o Systems Manager.

Para verificar o registro do dispositivo principal (console do Systems Manager)

1. No menu de navegação do console do Systems Manager, escolha Fleet Manager.
2. Em Nós gerenciados, faça o seguinte:
   a. Adicione um filtro onde está o tipo de fonte AWS::IoT::Thing.
   b. Adicione um filtro em que Source ID seja o nome do dispositivo principal a ser verificado.
3. Encontre o dispositivo principal na tabela de nós gerenciados. Se o dispositivo principal estiver na tabela, ele será registrado como um nó gerenciado.

   Você também pode encontrar a propriedade ping status do Systems Manager Agent para verificar o status do Systems Manager Agent no dispositivo principal. Quando o status é Online, você pode gerenciar o dispositivo principal com o Systems Manager.

Para verificar o registro do dispositivo principal (AWS CLI)

- Use a DescribeInstanceInformationoperação para obter a lista de nós gerenciados que correspondem a um filtro especificado por você. Execute o comando a seguir para verificar se um dispositivo principal está registrado como um nó gerenciado. *MyGreengrassCore*Substitua pelo nome do dispositivo principal para verificar.

  ```
  aws ssm describe-instance-information --filter
   Key=SourceIds,Values=MyGreengrassCore Key=SourceTypes,Values=AWS::IoT::Thing
  ```

  A resposta contém a lista de nós gerenciados que correspondem ao filtro. Se a lista contiver um nó gerenciado, o dispositivo principal será registrado como um nó gerenciado. Você também pode encontrar outras informações sobre o nó gerenciado do dispositivo principal na resposta. Se a `PingStatus` propriedade for`Online`, você poderá gerenciar o dispositivo principal com o Systems Manager.

Depois de verificar se um dispositivo principal está registrado como um nó gerenciado no Systems Manager, você pode usar o console e a API do Systems Manager para gerenciar esse dispositivo

principal. Para obter mais informações sobre os recursos do Systems Manager que você pode usar para gerenciar os principais dispositivos do Greengrass, consulte os recursos do Systems Manager no Guia do AWS Systems ManagerUsuário.

Desinstalar o agente do AWS Systems Manager

Se você não quiser mais gerenciar um dispositivo principal do GreengrassAWS Systems Manager, você pode cancelar o registro do dispositivo principal no Systems Manager e desinstalar oAWS Systems Manager Agente (Agente do Systems Manager) do dispositivo.

Você pode registrar novamente um dispositivo principal a qualquer momento. Para fazer isso, implante novamente o componente Agente do Systems Manager, que registra o dispositivo principal no Systems Manager quando ele é instalado. O Systems Manager armazena o histórico de comandos para um dispositivo central com registro cancelado por 30 dias.

Tópicos

Etapa 1: cancele o registro do dispositivo principal do Systems Manager

É possível usar o console do Systems Manager ou a API para cancelar o registro do dispositivo principal. Para obter mais informações, consulte Cancelar o registro de nós gerenciados no GuiaAWS Systems Manager do usuário.

Etapa 2: Desinstalar o componente Agente do Systems Manager

Depois de cancelar o registro do dispositivo principal, desinstale o componente Systems Manager Agent do dispositivo. Para remover um componente de um dispositivo central do Greengrass, revise a implantação que instalou o componente e remova o componente da implantação. O softwareAWS IoT Greengrass Core desinstala um componente quando nenhuma das implantações de um dispositivo principal especifica esse componente. Para obter mais informações, consulte Implemente AWS IoT Greengrass componentes em dispositivos.

Para desinstalar o componente Systems Manager Agent (console)

1. No menu de navegação AWS IoT Greengrassdo console, escolha Dispositivos principais.

2. Escolha o dispositivo principal em que você deseja desinstalar o componente Agente do Systems Manager.
3. Na página de detalhes do dispositivo principal, escolha a guia Implantações.
4. Escolha a implantação que implanta o componente Systems Manager Agent no dispositivo principal.
5. Na página de detalhes da implantação, escolha Revise.
6. No modal Revise deployment, escolha Revise deployment.
7. Na Etapa 1: Especifique o alvo, escolha Avançar.
8. Na Etapa 2: Selecione componentes, desmarque a seleção do aws.greengrass.SystemsManagerAgentcomponente e escolha Avançar.
9. Na Etapa 3: Configurar componentes, escolha Avançar.
10. Na Etapa 4: Definir configurações avançadas, escolha Avançar.
11. Na Etapa 5: Revisão, escolha Implantar.

Para desinstalar o componente Systems Manager Agent (CLI)

Para desinstalar o componente Systems Manager Agent, revise a implantação que o implanta e remova-o da implantação. Para obter mais informações, consulte Revise as implantações.

Pode demorar vários minutos para que a implantação seja concluída. É possível usar oAWS IoT Greengrass serviço para verificar o status da implantação. Para obter mais informações, consulte Verificar status da implantação.

Etapa 3: desinstalar o software Systems Manager Agent

O software Systems Manager Agent continua sendo executado no dispositivo principal depois que você remove o componente Systems Manager Agent. Para remover o software Systems Manager Agent, você pode executar comandos no dispositivo principal. Para obter mais informações, consulte Desinstalar o Agente do Systems Manager de instâncias Linux no GuiaAWS Systems Manager do Usuário.

Segurança no AWS IoT Greengrass

A segurança na nuvem na AWS é a nossa maior prioridade. Como cliente da AWS, você contará com um datacenter e uma arquitetura de rede criados para atender aos requisitos das organizações com as maiores exigências de segurança.

A segurança é uma responsabilidade compartilhada entre a AWS e você. O modelo de responsabilidade compartilhadadescreve isso como a segurança da nuvem e segurança na nuvem:

- Segurança da nuvem: AWS é responsável pela proteção da infraestrutura que executaAWS produtos da Nuvem AWS na AWS. A também fornece serviços que podem ser usados com segurança. Auditores de terceiros testam e verificam regularmente a eficácia da nossa segurança como parte dos AWSProgramas de conformidade. Para saber mais sobre os programas de conformidade que se aplicam ao AWS IoT Greengrass, consulte AWSServiços da em escopo por programa de conformidade.
- Segurança na nuvem: sua responsabilidade é determinada pelo serviço da AWS que você usa. Você também é responsável por outros fatores, incluindo a confidencialidade dos dados, os requisitos da empresa e as leis e regulamentos aplicáveis.

Quando você usa AWS IoT Greengrass, você também é responsável por proteger seus dispositivos, conexão de rede local e chaves privadas.

Esta documentação ajuda a entender como aplicar o modelo de responsabilidade compartilhada ao usar o .AWS IoT Greengrass Os tópicos a seguir mostram como configurar o AWS IoT Greengrass para atender aos seus objetivos de segurança e conformidade. Saiba também como usar outros produtos da AWS que ajudam a monitorar e proteger os recursos do AWS IoT Greengrass.

Tópicos

- Análise de vulnerabilidade e configuração no AWS IoT Greengrass
- Integridade do código emAWS IoT Greengrass V2
- AWS IoT Greengrass e endpoint da VPC de interface (AWS PrivateLink)
- Melhores práticas de segurança do AWS IoT Greengrass

Proteção de dados no AWS IoT Greengrass

O modelo de responsabilidade compartilhada da AWS se aplica à proteção de dados no AWS IoT Greengrass. Conforme descrito nesse modelo, a AWS é responsável por proteger a infraestrutura global que executa toda a Nuvem AWS. Você é responsável por manter o controle sobre seu conteúdo hospedado nessa infraestrutura. Você também é responsável pelas tarefas de configuração e gerenciamento de segurança dos Serviços da AWS que usa. Para obter mais informações sobre a privacidade de dados, consulte as Perguntas frequentes sobre privacidade de dados. Para mais informações sobre a proteção de dados na Europa, consulte o artigo AWS Shared Responsibility Model and GDPR no Blog de segurança da AWS.

Para fins de proteção de dados, recomendamos que você proteja asConta da AWS credenciais da e configure as contas de usuário individuais com o AWS IAM Identity Center ou o AWS Identity and Access Management (IAM). Dessa maneira, cada usuário receberá apenas as permissões necessárias para cumprir suas obrigações de trabalho. Recomendamos também que você proteja seus dados das seguintes formas:

- Use uma autenticação multifator (MFA [multi-factor authentication]) com cada conta.
- Use SSL/TLS para se comunicar com os atributos da AWS. Exigimos TLS 1.2 e recomendamos TLS 1.3.
- Configure o registro em log das atividades da API e do usuário com o .AWS CloudTrail
- Use AWS as soluções de criptografia da , juntamente com todos os controles de segurança padrão dos Serviços da AWS.
- Use serviços gerenciados de segurança avançada, como o Amazon Macie, que ajuda a localizar e proteger dados sigilosos armazenados no Amazon S3.
- Se você precisar de módulos criptográficos validados pelo FIPS 140-2 ao acessar a AWS por meio de uma interface de linha de comandos ou uma API, use um endpoint do FIPS. Para ter mais informações sobre endpoints do FIPS, consulte Federal Information Processing Standard (FIPS) 140-2.

É altamente recomendável que nunca sejam colocadas informações de identificação confidenciais, como endereços de e-mail dos seus clientes, em marcações ou campos de formato livre, como um campo Nome. Isso inclui trabalhar com a AWS IoT Greengrass ou outros Serviços da AWS usando o console, a API, a AWS CLI ou os AWS SDKs. Quaisquer dados inseridos em tags ou campos de texto de formato livre usados para nomes podem ser usados para logs de faturamento ou de diagnóstico. Se você fornecer um URL para um servidor externo, recomendemos fortemente que não sejam incluídas informações de credenciais no URL para validar a solicitação a esse servidor.

Para obter mais informações sobre como proteger informações confidenciais em AWS IoT Greengrass, consulte the section called “Não registrar em log informações confidenciais”.

Para obter mais informações sobre proteção de dados, consulte a publicação Modelo de responsabilidade compartilhada da AWS e do RGPD no Blog de segurança da AWS.

Tópicos

- Criptografia de dados
- Integração de segurança de hardware

Criptografia de dados

O AWS IoT Greengrass usa criptografia para proteger dados durante o trânsito (pela Internet ou rede local) e em repouso (armazenados na Nuvem AWS).

Os dispositivos em um ambiente AWS IoT Greengrass geralmente coletam dados enviados para serviços da AWS para processamento posterior. Para obter mais informações sobre criptografia de dados em outros serviços da AWS, consulte a documentação de segurança desse serviço.

Tópicos

- Criptografia em trânsito
- Criptografia em repouso
- Gerenciamento de chaves para o dispositivo de núcleo do Greengrass

Criptografia em trânsito

AWS IoT GreengrassO tem dois modos de comunicação em que os dados estão em trânsito:

- the section called “Dados em trânsito pela Internet”. Comunicação entre um núcleo do Greengrass eAWS IoT GreengrassO através da internet é criptografado.

- the section called “Dados no dispositivo de núcleo”. A comunicação entre componentes no dispositivo de núcleo do Greengrass não é criptografada.

Dados em trânsito pela Internet

O AWS IoT Greengrass usa Transport Layer Security (TLS) para criptografar todas as comunicações por meio da Internet. Todos os dados enviados aoNuvem AWSÉ enviado por meio de uma conexão TLS usando protocolos MQTT ou HTTPS, portanto, é seguro por padrão.AWS IoT GreengrassO usa oAWS IoTModelo de segurança de transporte do. Para obter mais informações, consulte Segurança de transporte no Guia do desenvolvedor do AWS IoT Core.

Dados no dispositivo de núcleo

O AWS IoT Greengrass não criptografa dados trocados localmente no dispositivo de núcleo do Greengrass porque os dados não saem do dispositivo. Isso inclui comunicação entre componentes definidos pelo usuário, oAWS IoTSDK do dispositivo e componentes públicos, como gerenciador de fluxo.

Criptografia em repouso

O AWS IoT Greengrass armazena seus dados:

- the section called “Dados em repouso noNuvem AWS”. Esses dados são criptografados.
- the section called “Dados em repouso sobre o núcleo do Greengrass”. Esses dados não são criptografados (exceto cópias locais de seus segredos).

Dados em repouso noNuvem AWS

AWS IoT GreengrassO criptografa os dados do cliente armazenados noNuvem AWS. Esses dados são protegidos usando chaves AWS KMS gerenciadas pelo AWS IoT Greengrass.

Dados em repouso sobre o núcleo do Greengrass

O AWS IoT Greengrass depende de permissões de arquivo Unix e da criptografia do disco inteiro (se habilitada) para proteger dados em repouso no núcleo. É sua responsabilidade proteger o sistema de arquivos e o dispositivo.

No entanto, o AWS IoT Greengrass não criptografa cópias locais de seus segredos recuperados do AWS Secrets Manager. Para obter mais informações, consulte o .Secret managerComponente.

Gerenciamento de chaves para o dispositivo de núcleo do Greengrass

É responsabilidade do cliente garantir o armazenamento seguro de chaves criptográficas (públicas e privadas) no dispositivo de núcleo do Greengrass.AWS IoT Greengrassusa chaves públicas e privadas para o seguinte cenário:

- A chave de cliente da IoT é usada com o certificado IoT para autenticar o handshake do Transport Layer Security (TLS) quando um núcleo do Greengrass se conecta ao AWS IoT Core. Para obter mais informações, consultethe section called “Autorização e autenticação do dispositivo”

> **Note**
>
> A chave e o certificado também são conhecidos como a chave privada do núcleo e o certificado do dispositivo do núcleo.

Um dispositivo de núcleo do Greengrass oferece suporte para o armazenamento de chaves privadas usando permissões do sistema de arquivos ou ummódulo de segurança de hardware. Se você usar chaves privadas baseadas no sistema de arquivos, você será responsável por seu armazenamento seguro no dispositivo de núcleo.

Integração de segurança de hardware

> **Note**
>
> Esse recurso está disponível para a versão 2.5.3 e posterior do componente núcleo do Greengrass. AWS IoT Greengrassatualmente não oferece suporte a esse recurso nos dispositivos principais do Windows.

Você pode configurar o software AWS IoT Greengrass Core para usar um módulo de segurança de hardware (HSM) por meio da interface PKCS #11. Esse recurso permite que você armazene com segurança a chave privada e o certificado do dispositivo para que eles não sejam expostos ou duplicados no software. Você pode armazenar a chave privada e o certificado em um módulo de hardware, como um HSM ou um Trusted Platform Module (TPM).

O software AWS IoT Greengrass Core usa uma chave privada e um certificado X.509 para autenticar conexões com os serviços e. AWS IoT AWS IoT Greengrass O componente gerenciador de

segredos usa essa chave privada para criptografar e descriptografar com segurança os segredos que você implanta em um dispositivo principal do Greengrass. Quando você configura um dispositivo principal para usar um HSM, esses componentes usam a chave privada e o certificado que você armazena no HSM.

O componente de agente Moquette MQTT também armazena uma chave privada para seu certificado de servidor MQTT local. Esse componente armazena a chave privada no sistema de arquivos do dispositivo na pasta de trabalho do componente. Atualmente, AWS IoT Greengrass não é compatível com o armazenamento dessa chave privada ou certificado em um HSM.

> **Tip**
>
> Pesquise dispositivos compatíveis com esse recurso no Catálogo de dispositivos do AWS parceiro.

Tópicos

Requisitos

Você deve atender aos seguintes requisitos para usar um HSM em um dispositivo principal do Greengrass:

- Greengrass nucleus v2.5.3 ou posterior instalado no dispositivo principal. Você pode escolher uma versão compatível ao instalar o software AWS IoT Greengrass Core em um dispositivo principal.
- O componente do provedor PKCS #11 instalado no dispositivo principal. Você pode baixar e instalar esse componente ao instalar o software AWS IoT Greengrass Core em um dispositivo principal.
- Um módulo de segurança de hardware que suporta o esquema de assinatura PKCS #1 v1.5 e chaves RSA com tamanho de chave RSA-2048 (ou maior) ou chaves ECC.

> **Note**
>
> Para usar um módulo de segurança de hardware com chaves ECC, você deve usar o Greengrass nucleus v2.5.6 ou posterior.
> Para usar um módulo de segurança de hardware e um gerenciador secreto, você deve usar um módulo de segurança de hardware com chaves RSA.

- Uma biblioteca do provedor PKCS #11 que o software AWS IoT Greengrass Core pode carregar em tempo de execução (usando libdl) para invocar as funções do PKCS #11. A biblioteca do provedor PKCS #11 deve implementar as seguintes operações da API PKCS #11:
  - `C_Initialize`
  - `C_Finalize`
  - `C_GetSlotList`
  - `C_GetSlotInfo`
  - `C_GetTokenInfo`
  - `C_OpenSession`
  - `C_GetSessionInfo`
  - `C_CloseSession`
  - `C_Login`
  - `C_Logout`
  - `C_GetAttributeValue`
  - `C_FindObjectsInit`
  - `C_FindObjects`
  - `C_FindObjectsFinal`
  - `C_DecryptInit`
  - `C_Decrypt`
  - `C_DecryptUpdate`
  - `C_DecryptFinal`
  - `C_SignInit`
  - `C_Sign`
  - `C_SignUpdate`

  - `C_SignFinal`
  - `C_GetMechanismList`
  - `C_GetMechanismInfo`
  - `C_GetInfo`
  - `C_GetFunctionList`
- O módulo de hardware deve ser solucionado pelo rótulo do slot, conforme definido na especificação PKCS#11.
- Você deve armazenar a chave privada e o certificado no HSM no mesmo slot, e eles devem usar o mesmo rótulo de objeto e ID de objeto, se o HSM suportar IDs de objeto.
- O certificado e a chave privada devem ser resolvidos por rótulos de objetos.
- A chave privada deve ter as seguintes permissões:
  - `sign`
  - `decrypt`
- (Opcional) Para usar o componente gerenciador de segredos, você deve usar a versão 2.1.0 ou posterior, e a chave privada deve ter as seguintes permissões:
  - `unwrap`
  - `wrap`

Práticas recomendadas de segurança de hardware

Considere as seguintes melhores práticas ao configurar a segurança de hardware nos dispositivos principais do Greengrass.

- Gerar chaves privadas diretamente no HSM usando o hardware interno gerador de número aleatório. Essa abordagem é mais segura do que importar uma chave privada que você gera em outro lugar, porque a chave privada permanece dentro do HSM.
- Configure as chaves privadas para serem imutáveis e proibir a exportação.
- Use a ferramenta de provisionamento recomendada pelo fornecedor de hardware do HSM para gerar uma solicitação de assinatura de certificado (CSR) usando a chave privada protegida por hardware e, em seguida, use o console ou a API para gerar um certificado de cliente. AWS IoT

> **Note**
>
> A melhor prática de segurança para alternar chaves não se aplica quando você gera chaves privadas em um HSM.

Instale o software AWS IoT Greengrass Core com segurança de hardware

Ao instalar o software AWS IoT Greengrass Core, você pode configurá-lo para usar uma chave privada gerada em um HSM. Essa abordagem segue as melhores práticas de segurança para gerar a chave privada no HSM, de forma que a chave privada permaneça dentro do HSM.

Para instalar o software AWS IoT Greengrass Core com segurança de hardware, faça o seguinte:

1. Gere uma chave privada no HSM.
2. Crie uma solicitação de assinatura de certificado (CSR) a partir da chave privada.
3. Crie um certificado do CSR. Você pode criar um certificado assinado por AWS IoT ou por outra autoridade de certificação (CA) raiz. Para obter mais informações sobre como usar outra CA raiz, consulte Criar seus próprios certificados de cliente no Guia do AWS IoT Core desenvolvedor.
4. Faça o download do AWS IoT certificado e importe-o para o HSM.
5. Instale o software AWS IoT Greengrass Core a partir de um arquivo de configuração que especifica o uso do componente provedor PKCS #11 e da chave privada e do certificado no HSM.

Você pode escolher uma das seguintes opções de instalação para instalar o software AWS IoT Greengrass Core com segurança de hardware:

- Instalação manual

  Escolha essa opção para criar manualmente os AWS recursos necessários e configurar a segurança do hardware. Para ter mais informações, consulte Instale o software AWS IoT Greengrass principal com provisionamento manual de recursos.
- Instalação com provisionamento personalizado

  Escolha essa opção para desenvolver um aplicativo Java personalizado que cria automaticamente os AWS recursos necessários e configura a segurança do hardware. Para ter mais informações, consulte Instale o software AWS IoT Greengrass principal com provisionamento personalizado de recursos.

Atualmente, AWS IoT Greengrass não oferece suporte à instalação do software AWS IoT Greengrass Core com segurança de hardware quando você instala com provisionamento automático de recursos ou provisionamento de AWS IoT frota.

Configurar a segurança do hardware em um dispositivo principal existente

Você pode importar a chave privada e o certificado de um dispositivo principal para um HSM para configurar a segurança do hardware.

> **Considerações**
>
> - Você deve ter acesso root ao sistema de arquivos do dispositivo principal.
> - Neste procedimento, você desliga o software AWS IoT Greengrass Core para que o dispositivo principal fique off-line e indisponível enquanto você configura a segurança do hardware.

Para configurar a segurança do hardware em um dispositivo principal existente, faça o seguinte:

1. Inicialize o HSM.
2. Implante o componente do provedor PKCS #11 no dispositivo principal.
3. Pare o software AWS IoT Greengrass principal.
4. Importe a chave privada e o certificado do dispositivo principal para o HSM.
5. Atualize o arquivo de configuração do software AWS IoT Greengrass principal para usar a chave privada e o certificado no HSM.
6. Inicie o software AWS IoT Greengrass principal.

Etapa 1: Inicializar o módulo de segurança de hardware

Conclua a etapa a seguir para inicializar o HSM em seu dispositivo principal.

Para inicializar o módulo de segurança de hardware

- Inicialize um token PKCS #11 no HSM e salve o ID do slot e o PIN do usuário para o token. Consulte a documentação do seu HSM para saber como inicializar um token. Você usa o ID do slot e o PIN do usuário posteriormente ao implantar e configurar o componente provedor PKCS #11.

Etapa 2: Implantar o componente do provedor PKCS #11

Conclua as etapas a seguir para implantar e configurar o componente provedor PKCS #11. Você pode implantar o componente em um ou mais dispositivos principais.

Para implantar o componente do provedor PKCS #11 (console)

1. No menu de navegação AWS IoT Greengrassdo console, escolha Componentes.
2. Na página Componentes, escolha a guia Componentes públicos e, em seguida, escolha aws.greengrass.crypto.Pkcs11Provider.
3. Na página aws.greengrass.crypto.Pkcs11Provider, escolha Implantar.
4. Em Adicionar à implantação, escolha uma implantação existente para revisar ou escolha criar uma nova implantação e, em seguida, escolha Avançar.
5. Se você optar por criar uma nova implantação, escolha o dispositivo principal ou grupo de itens de destino para a implantação. Na página Especificar destino, em Destino de implantação, escolha um dispositivo principal ou grupo de itens e, em seguida, escolha Avançar.
6. Na página Selecionar componentes, em Componentes públicos, selecione e aws.greengrass.crypto.Pkcs11Provider, em seguida, escolha Avançar.
7. Na página Configurar componentes aws.greengrass.crypto.Pkcs11Provider, selecione e faça o seguinte:

   a. Escolha Configurar componente.

   b. No aws.greengrass.crypto.Pkcs11Provider modal Configurar, em Atualização de configuração, em Configuração para mesclar, insira a seguinte atualização de configuração. Atualize os seguintes parâmetros de configuração com valores para os dispositivos principais de destino. Especifique o ID do slot e o PIN do usuário em que você inicializou o token PKCS #11 anteriormente. Posteriormente, você importa a chave privada e o certificado para esse slot no HSM.

      `name`

      Um nome para a configuração PKCS #11.

      `library`

      O caminho absoluto do arquivo para a biblioteca da implementação do PKCS #11 que o software AWS IoT Greengrass Core pode carregar com libdl.

`slot`

O ID do slot que contém a chave privada e o certificado do dispositivo. Esse valor é diferente do índice do slot ou do rótulo do slot.

`userPin`

O PIN do usuário a ser usado para acessar o slot.

```
{
  "name": "softhsm_pkcs11",
  "library": "/usr/lib/softhsm/libsofthsm2.so",
  "slot": 1,
  "userPin": "1234"
}
```

c. Escolha Confirmar para fechar o modal e, em seguida, escolha Avançar.

8. Na página Definir configurações avançadas, mantenha as configurações padrão e escolha Avançar.
9. Na página Review, escolha Deploy.

   A implantação pode levar até um minuto para ser concluída.

Para implantar o componente do provedor PKCS #11 () AWS CLI

Para implantar o componente do provedor PKCS #11, crie um documento de implantação que inclua `aws.greengrass.crypto.Pkcs11Provider` no `components` objeto e especifique a atualização de configuração do componente. Siga as instruções Criar implantações para criar uma nova implantação ou revisar uma implantação existente.

O exemplo de documento de implantação parcial a seguir especifica a implantação e a configuração do componente provedor PKCS #11. Atualize os seguintes parâmetros de configuração com valores para os dispositivos principais de destino. Salve o ID do slot e o PIN do usuário para usar posteriormente ao importar a chave privada e o certificado para o HSM.

`name`

Um nome para a configuração PKCS #11.

`library`

O caminho absoluto do arquivo para a biblioteca da implementação do PKCS #11 que o software AWS IoT Greengrass Core pode carregar com libdl.

`slot`

O ID do slot que contém a chave privada e o certificado do dispositivo. Esse valor é diferente do índice do slot ou do rótulo do slot.

`userPin`

O PIN do usuário a ser usado para acessar o slot.

```
{
  "name": "softhsm_pkcs11",
  "library": "/usr/lib/softhsm/libsofthsm2.so",
  "slot": 1,
  "userPin": "1234"
}
```

```
{
  ...,
  "components": {
    ...,
    "aws.greengrass.crypto.Pkcs11Provider": {
      "componentVersion": "2.0.0",
      "configurationUpdate": {
        "merge": "{\"name\":\"softhsm_pkcs11\",\"library\":\"/usr/lib/softhsm/
libsofthsm2.so\",\"slot\":1,\"userPin\":\"1234\"}"
      }
    }
  }
}
```

A implantação pode levar vários minutos para ser concluída. Você pode usar o AWS IoT Greengrass serviço para verificar o status da implantação. Você pode verificar os registros do software AWS IoT Greengrass principal para verificar se o componente do provedor PKCS #11 foi implantado com êxito. Para ver mais informações, consulte:

- Verificar status da implantação
- Monitore AWS IoT Greengrass os registros

Se a implantação falhar, você poderá solucionar o problema da implantação em cada dispositivo principal. Para ter mais informações, consulte Solução de problemas AWS IoT Greengrass V2.

Etapa 3: atualizar a configuração no dispositivo principal

O software AWS IoT Greengrass Core usa um arquivo de configuração que especifica como o dispositivo opera. Esse arquivo de configuração inclui onde encontrar a chave privada e o certificado que o dispositivo usa para se conectar aoNuvem AWS. Conclua as etapas a seguir para importar a chave privada e o certificado do dispositivo principal para o HSM e atualizar o arquivo de configuração para usar o HSM.

Para atualizar a configuração no dispositivo principal para usar a segurança de hardware

1. Pare o software AWS IoT Greengrass principal. Se você configurou o software AWS IoT Greengrass Core como um serviço do sistema com o systemd, você pode executar o comando a seguir para interromper o software.

   ```
   sudo systemctl stop greengrass.service
   ```

2. Encontre a chave privada e os arquivos de certificado do dispositivo principal.

   - Se você instalou o software AWS IoT Greengrass Core com provisionamento automático ou provisionamento de frota, a chave privada existe em*/greengrass/v2*/privKey.key, e o certificado existe em. */greengrass/v2*/thingCert.crt
   - Se você instalou o software AWS IoT Greengrass Core com provisionamento manual, a chave privada existe */greengrass/v2*/private.pem.key por padrão e o certificado existe */greengrass/v2*/device.pem.crt por padrão.

   Você também pode verificar as system.certificateFilePath propriedades system.privateKeyPath e */greengrass/v2*/config/effectiveConfig.yaml para encontrar a localização desses arquivos.

3. Importe a chave privada e o certificado para o HSM. Consulte a documentação do seu HSM para saber como importar chaves privadas e certificados para ele. Importe a chave privada e o certificado usando o ID do slot e o PIN do usuário em que você inicializou o token PKCS #11 anteriormente. Você deve usar o mesmo rótulo de objeto e ID de objeto para a chave privada e o certificado. Salve o rótulo do objeto que você especifica ao importar cada arquivo. Você usa esse rótulo posteriormente ao atualizar a configuração do software AWS IoT Greengrass principal para usar a chave privada e o certificado no HSM.

4. Atualize a configuração AWS IoT Greengrass principal para usar a chave privada e o certificado no HSM. Para atualizar a configuração, você modifica o arquivo de configuração AWS IoT Greengrass principal e executa o software AWS IoT Greengrass principal com o arquivo de configuração atualizado para aplicar a nova configuração.

   Faça o seguinte:

   a. Crie um backup do arquivo de configuração AWS IoT Greengrass principal. Você pode usar esse backup para restaurar o dispositivo principal se tiver problemas ao configurar a segurança do hardware.

   ```
   sudo cp /greengrass/v2/config/effectiveConfig.yaml ~/ggc-config-backup.yaml
   ```

   b. Abra o arquivo de configuração AWS IoT Greengrass principal em um editor de texto. Por exemplo, você pode executar o seguinte comando para usar o GNU nano para editar o arquivo. */greengrass/v2*Substitua pelo caminho para a pasta raiz do Greengrass.

   ```
   sudo nano /greengrass/v2/config/effectiveConfig.yaml
   ```

   c. Substitua o valor do pelo `system.privateKeyPath` URI PKCS #11 para a chave privada no HSM. Substitua *iotdevicekey* pelo rótulo do objeto em que você importou a chave privada e o certificado anteriormente.

   ```
   pkcs11:object=iotdevicekey;type=private
   ```

   d. Substitua o valor do pelo `system.certificateFilePath` URI PKCS #11 para o certificado no HSM. Substitua *iotdevicekey* pelo rótulo do objeto em que você importou a chave privada e o certificado anteriormente.

   ```
   pkcs11:object=iotdevicekey;type=cert
   ```

   Depois de concluir essas etapas, a `system` propriedade no arquivo de configuração AWS IoT Greengrass principal deve ser semelhante ao exemplo a seguir.

   ```
   system:
     certificateFilePath: "pkcs11:object=iotdevicekey;type=cert"
     privateKeyPath: "pkcs11:object=iotdevicekey;type=private"
     rootCaPath: "/greengrass/v2/rootCA.pem"
     rootpath: "/greengrass/v2"
   ```

```
  thingName: "MyGreengrassCore"
```

5. Aplique a configuração no `effectiveConfig.yaml` arquivo atualizado. Execute `Greengrass.jar` com o `--init-config` parâmetro no qual aplicar a configuração`effectiveConfig.yaml`. */greengrass/v2*Substitua pelo caminho para a pasta raiz do Greengrass.

```
sudo java -Droot="/greengrass/v2" \
  -jar /greengrass/v2/alts/current/distro/lib/Greengrass.jar \
  --start false \
  --init-config /greengrass/v2/config/effectiveConfig.yaml
```

6. Inicie o software AWS IoT Greengrass principal. Se você configurou o software AWS IoT Greengrass Core como um serviço do sistema com o systemd, você pode executar o comando a seguir para iniciar o software.

```
sudo systemctl start greengrass.service
```

   Para ter mais informações, consulte Execute o software AWS IoT Greengrass Core.

7. Verifique os registros do software AWS IoT Greengrass principal para verificar se o software é iniciado e se conecta aoNuvem AWS. O software AWS IoT Greengrass Core usa a chave privada e o certificado para se conectar aos AWS IoT Greengrass serviços AWS IoT e.

```
sudo tail -f /greengrass/v2/logs/greengrass.log
```

   As seguintes mensagens de registro em nível de informação indicam que o software AWS IoT Greengrass Core se conecta com êxito aos serviços AWS IoT e. AWS IoT Greengrass

```
2021-12-06T22:47:53.702Z [INFO] (Thread-3)
 com.aws.greengrass.mqttclient.AwsIotMqttClient: Successfully connected to AWS IoT
 Core. {clientId=MyGreengrassCore5, sessionPresent=false}
```

8. (Opcional) Depois de verificar se o software AWS IoT Greengrass Core funciona com a chave privada e o certificado no HSM, exclua a chave privada e os arquivos do certificado do sistema de arquivos do dispositivo. Execute o comando a seguir e substitua os caminhos dos arquivos pelos caminhos para a chave privada e os arquivos de certificado.

```
sudo rm /greengrass/v2/privKey.key
```

```
sudo rm /greengrass/v2/thingCert.crt
```

Use hardware sem suporte ao PKCS #11

Em geral, a biblioteca PKCS#11 é fornecida pelo fornecedor de hardware ou é de código aberto. Por exemplo, com hardware compatível com os padrões (como TPM1.2), pode ser possível usar software de código aberto existente. No entanto, se o seu hardware não tiver uma implementação correspondente da biblioteca PKCS #11 ou se você quiser criar um provedor PKCS #11 personalizado, entre em contato com seu representante do Amazon Web Services Enterprise Support com perguntas relacionadas à integração.

Consulte também

- Guia de uso da interface de token criptográfico PKCS #11, versão 2.4.0
- RFC 7512
- PKCS #1: RSA Encryption versão 1.5

Autorização e autenticação do dispositivo para o AWS IoT Greengrass

Os dispositivos em ambientes AWS IoT Greengrass usam certificados X.509 para autenticação e políticas de AWS IoT para autorização. Certificados e políticas permitem que os dispositivos se conectem com segurança entre si, AWS IoT Core, e AWS IoT Greengrass.

Os certificados X.509 são certificados digitais que usam a infraestrutura de chave pública X.509 padrão para associar uma chave pública a uma identidade contida em um certificado. Os certificados X.509 são emitidos por uma entidade confiável chamada de autoridade de certificação (CA). A CA mantém um ou mais certificados especiais chamados certificados CA que são usados para emitir certificados X.509. Somente a autoridade de certificação tem acesso aos certificados CA.

Políticas AWS IoT definem o conjunto de operações permitidas para dispositivos de AWS IoT. Especificamente, elas permitem e negam acesso às operações de plano de dados do AWS IoT Core e AWS IoT Greengrass, como a publicação de mensagens MQTT e recuperação de sombras de dispositivo.

Todos os dispositivos exigem uma entrada no registro do AWS IoT Core e um certificado X.509 ativado com uma política de AWS IoT anexada. Os dispositivos se enquadram em duas categorias:

- Dispositivos principais do Greengrass

  Os principais dispositivos do Greengrass usam certificados e AWS IoT políticas para se conectar a e. AWS IoT Core AWS IoT Greengrass Os certificados e políticas também permitem AWS IoT Greengrass implantar componentes e configurações nos dispositivos principais.

- Dispositivos do cliente

  Os dispositivos clientes MQTT usam certificados e políticas para se conectar ao AWS IoT Core AWS IoT Greengrass serviço. Isso permite que os dispositivos do cliente AWS IoT Greengrass usem a descoberta na nuvem para encontrar e se conectar a um dispositivo principal do Greengrass. Um dispositivo cliente usa o mesmo certificado para se conectar ao serviço de AWS IoT Core nuvem e aos dispositivos principais. Os dispositivos cliente também usam informações de descoberta para autenticação mútua com o dispositivo de núcleo. Para ter mais informações, consulte Interaja com dispositivos IoT locais.

Certificados X.509

A comunicação entre dispositivos principais e dispositivos clientes e entre dispositivos AWS IoT Core e/ou AWS IoT Greengrass deve ser autenticada. Esta autenticação mútua é baseada nos certificados e nas chaves criptográficas do dispositivo X.509 registrado.

Em um ambiente AWS IoT Greengrass, os dispositivos usam certificados com chaves públicas e privadas para as seguintes conexões Transport Layer Security (TLS):

- O componente AWS IoT cliente no dispositivo principal do Greengrass que se conecta à Internet AWS IoT Core e AWS IoT Greengrass pela Internet.
- Dispositivos clientes que se conectam pela AWS IoT Greengrass Internet para descobrir os dispositivos principais.
- O componente intermediário MQTT no núcleo do Greengrass que se conecta aos dispositivos Greengrass no grupo pela rede local.

AWS IoT Greengrassos dispositivos principais armazenam certificados na pasta raiz do Greengrass.

Certificados da autoridade de certificação (CA)

Os dispositivos principais e clientes do Greengrass baixam um certificado CA raiz usado para autenticação com os serviços AWS IoT Core e. AWS IoT Greengrass Recomendamos que você use

um certificado CA raiz do Amazon Trust Services (ATS), como o Amazon Root CA 1. Para obter mais informações, consulte Certificados CA para a autenticação do servidor no Guia do desenvolvedor do AWS IoT Core.

Os dispositivos cliente também baixam um certificado CA do dispositivo principal do Greengrass. Eles usam esse certificado para validar o certificado do servidor MQTT no dispositivo principal durante a autenticação mútua.

Rotação de certificados no corretor MQTT local

Quando você ativa o suporte ao dispositivo cliente, os dispositivos principais do Greengrass geram um certificado de servidor MQTT local que os dispositivos cliente usam para autenticação mútua. Esse certificado é assinado pelo certificado CA do dispositivo principal, que o dispositivo principal armazena na AWS IoT Greengrass nuvem. Os dispositivos cliente recuperam o certificado CA do dispositivo principal quando descobrem o dispositivo principal. Eles usam o certificado CA do dispositivo principal para verificar o certificado do servidor MQTT do dispositivo principal quando se conectam ao dispositivo principal. O certificado CA do dispositivo principal expira após 5 anos.

O certificado do servidor MQTT expira a cada 7 dias por padrão, e você pode configurar essa duração entre 2 e 10 dias. Esse período limitado é baseado nas melhores práticas de segurança. Essa rotação ajuda a mitigar a ameaça de um invasor roubar o certificado e a chave privada do servidor MQTT para se passar pelo dispositivo principal do Greengrass.

O dispositivo principal do Greengrass gira o certificado do servidor MQTT 24 horas antes de expirar. O dispositivo principal do Greengrass gera um novo certificado e reinicia o agente MQTT local. Quando isso acontece, todos os dispositivos clientes conectados ao dispositivo principal do Greengrass são desconectados. Os dispositivos cliente podem se reconectar ao dispositivo principal do Greengrass após um curto período de tempo.

Políticas do AWS IoT para operações de plano de dados

Use AWS IoT políticas para autorizar o acesso aos planos de AWS IoT Greengrass dados AWS IoT Core e. O plano de AWS IoT Core dados fornece operações para dispositivos, usuários e aplicativos. Essas operações incluem a capacidade de se conectar AWS IoT Core e se inscrever em tópicos. O plano AWS IoT Greengrass de dados fornece operações para dispositivos Greengrass. Para ter mais informações, consulte Ações de políticas do AWS IoT Greengrass V2. Essas operações incluem a capacidade de resolver dependências de componentes e baixar artefatos de componentes públicos.

Uma política de AWS IoT é um documento JSON semelhante a uma política do IAM. Ela contém uma ou mais declarações de política que especificam as seguintes propriedades:

- `Effect`. O modo de acesso, que pode ser `Allow` ou `Deny`.
- `Action`. A lista de ações permitidas ou negadas pela política.
- `Resource`. A lista de recursos em que a ação é permitida ou negada.

As políticas AWS IoT oferecem suporte a * como um caractere curinga e tratam os caracteres curinga (+ e #) do MQTT como sequências literais. Para obter mais informações sobre o caractere curinga *, consulte Usando o caractere curinga em ARNs de recursos no Guia do usuário do AWS Identity and Access Management.

Para obter mais informações, consulte Políticas do AWS IoT e Ações de políticas do AWS IoT no Guia do desenvolvedor do AWS IoT Core.

Important

As variáveis de política Thing (`iot:Connection.Thing.*`) não são suportadas em AWS IoT políticas para dispositivos principais ou operações de plano de dados do Greengrass. Em vez disso, você pode usar um caractere curinga que corresponda a vários dispositivos com nomes semelhantes. Por exemplo, você pode especificar `MyGreengrassDevice*` para corresponder `MyGreengrassDevice1MyGreengrassDevice2`, e assim por diante.

Note

AWS IoT Core permite que você anexe políticas AWS IoT a grupos de coisas para definir permissões para grupos de dispositivos. As políticas de grupos de coisas não permitem acesso às operações do plano de dados AWS IoT Greengrass. Para permitir que uma coisa acesse uma operação de plano de dados AWS IoT Greengrass, adicione a permissão a uma política AWS IoT que você anexa ao certificado da coisa.

Ações de políticas do AWS IoT Greengrass V2

AWS IoT Greengrass V2define as seguintes ações de política que os dispositivos principais e clientes do Greengrass podem usar nas AWS IoT políticas. Para especificar um recurso para uma ação política, você usa o Amazon Resource Name (ARN) do recurso.

Principais ações do dispositivo

`greengrass:GetComponentVersionArtifact`

Concede permissão para obter uma URL pré-assinada para baixar um artefato de componente público ou um artefato de componente Lambda.

Essa permissão é avaliada quando um dispositivo principal recebe uma implantação que especifica um componente público ou um Lambda com artefatos. Se o dispositivo principal já tiver o artefato, ele não baixará o artefato novamente.

Tipo de recurso: `componentVersion`

Formato ARN do recurso: `arn:aws:greengrass:`*`region`*`:`*`account-id`*`:components:`*`component-name`*`:versions:`*`component-version`*

`greengrass:ResolveComponentCandidates`

Concede permissão para identificar uma lista de componentes que atendem aos requisitos de componente, versão e plataforma para uma implantação. Se os requisitos entrarem em conflito ou se não existirem componentes que atendam aos requisitos, essa operação retornará um erro e a implantação falhará no dispositivo.

Essa permissão é avaliada quando um dispositivo principal recebe uma implantação que especifica componentes.

Tipo de recurso: Nenhum

Formato ARN do recurso: *

`greengrass:GetDeploymentConfiguration`

Concede permissão para obter uma URL pré-assinada para baixar um grande documento de implantação.

Essa permissão é avaliada quando um dispositivo principal recebe uma implantação que especifica um documento de implantação maior que 7 KB (se a implantação for direcionada a uma coisa) ou 31 KB (se a implantação for direcionada a um grupo de coisas). O documento de implantação inclui configurações de componentes, políticas de implantação e metadados de implantação. Para ter mais informações, consulte Implemente AWS IoT Greengrass componentes em dispositivos.

Esse recurso está disponível para a versão 2.3.0 e posterior do componente de núcleo do Greengrass.

Tipo de recurso: Nenhum

Formato ARN do recurso: *

`greengrass:ListThingGroupsForCoreDevice`

Concede permissão para obter a hierarquia de grupos de coisas de um dispositivo principal.

Essa permissão é verificada quando um dispositivo principal recebe uma implantação doAWS IoT Greengrass. O dispositivo principal usa essa ação para identificar se ele foi removido de um grupo de coisas desde a última implantação. Se o dispositivo principal foi removido de um grupo de coisas e esse grupo de coisas é o alvo de uma implantação no dispositivo principal, o dispositivo principal remove os componentes instalados por essa implantação.

Esse recurso é usado pela versão 2.5.0 e posterior do componente de núcleo do Greengrass.

Tipo de recurso: `thing` (dispositivo principal)

Formato ARN do recurso: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`core-device-thing-name`*

`greengrass:VerifyClientDeviceIdentity`

Concede permissão para verificar a identidade de um dispositivo cliente que se conecta a um dispositivo principal.

Essa permissão é avaliada quando um dispositivo principal executa o componente de autenticação do dispositivo cliente e recebe uma conexão MQTT de um dispositivo cliente. O dispositivo cliente apresenta seu certificado de AWS IoT dispositivo. Em seguida, o dispositivo principal envia o certificado do dispositivo ao serviço de AWS IoT Greengrass nuvem para verificar a identidade do dispositivo cliente. Para ter mais informações, consulte Interaja com dispositivos IoT locais.

Tipo de recurso: Nenhum

Formato ARN do recurso: *

`greengrass:VerifyClientDeviceIoTCertificateAssociation`

Concede permissão para verificar se um dispositivo cliente está associado a um AWS IoT certificado.

Essa permissão é avaliada quando um dispositivo principal executa o componente de autenticação do dispositivo cliente e autoriza um dispositivo cliente a se conectar pelo MQTT. Para ter mais informações, consulte Interaja com dispositivos IoT locais.

> **Note**
>
> Para que um dispositivo principal use essa operação, a função de serviço do Greengrass deve estar associada à sua Conta da AWS e permitir a `iot:DescribeCertificate` permissão.

Tipo de recurso: `thing` (dispositivo cliente)

Formato ARN do recurso: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`client-device-thing-name`*

`greengrass:PutCertificateAuthorities`

Concede permissão para carregar certificados de autoridade de certificação (CA) que os dispositivos cliente podem baixar para verificar o dispositivo principal.

Essa permissão é avaliada quando um dispositivo principal instala e executa o componente de autenticação do dispositivo cliente. Esse componente cria uma autoridade de certificação local e usa essa operação para carregar seus certificados CA. Os dispositivos clientes baixam esses certificados CA quando usam a operação Discover para encontrar dispositivos principais aos quais podem se conectar. Quando dispositivos clientes se conectam a um agente MQTT em um dispositivo principal, eles usam esses certificados CA para verificar a identidade do dispositivo principal. Para ter mais informações, consulte Interaja com dispositivos IoT locais.

Tipo de recurso: Nenhum

Formato ARN: *

`greengrass:GetConnectivityInfo`

Concede permissão para obter informações de conectividade para um dispositivo principal. Essas informações descrevem como os dispositivos cliente podem se conectar ao dispositivo principal.

Essa permissão é avaliada quando um dispositivo principal instala e executa o componente de autenticação do dispositivo cliente. Esse componente usa as informações de conectividade para gerar certificados CA válidos para serem carregados no serviço de AWS IoT Greengrass nuvem

com a PutCertificateAuthoritiesoperação. Os dispositivos cliente usam esses certificados CA para verificar a identidade do dispositivo principal. Para ter mais informações, consulte Interaja com dispositivos IoT locais.

Você também pode usar essa operação no plano de AWS IoT Greengrass controle para visualizar as informações de conectividade de um dispositivo principal. Para obter mais informações, consulte GetConnectivityInfo na Referência da API do AWS IoT Greengrass V1.

Tipo de recurso: `thing` (dispositivo principal)

Formato ARN do recurso: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`core-device-thing-name`*

`greengrass:UpdateConnectivityInfo`

Concede permissão para atualizar as informações de conectividade de um dispositivo principal. Essas informações descrevem como os dispositivos cliente podem se conectar ao dispositivo principal.

Essa permissão é avaliada quando um dispositivo principal executa o componente detector de IP. Esse componente identifica as informações que os dispositivos cliente precisam para se conectar ao dispositivo principal na rede local. Em seguida, esse componente usa essa operação para carregar as informações de conectividade no serviço de AWS IoT Greengrass nuvem, para que os dispositivos clientes possam recuperar essas informações com a operação Discover. Para ter mais informações, consulte Interaja com dispositivos IoT locais.

Você também pode usar essa operação no plano de AWS IoT Greengrass controle para atualizar manualmente as informações de conectividade de um dispositivo principal. Para obter mais informações, consulte UpdateConnectivityInfo na Referência da API do AWS IoT Greengrass V1.

Tipo de recurso: `thing` (dispositivo principal)

Formato ARN do recurso: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`core-device-thing-name`*

Ações do dispositivo cliente

`greengrass:Discover`

Concede permissão para descobrir informações de conectividade para dispositivos principais aos quais um dispositivo cliente pode se conectar. Essas informações descrevem como

o dispositivo cliente pode se conectar aos dispositivos principais. Um dispositivo cliente pode descobrir somente os dispositivos principais aos quais você o associou usando a BatchAssociateClientDeviceWithCoreDeviceoperação. Para ter mais informações, consulte Interaja com dispositivos IoT locais.

Tipo de recurso: `thing` (dispositivo cliente)

Formato ARN do recurso: `arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`client-device-thing-name`*

Atualizar a AWS IoT política de um dispositivo principal

Você pode usar os AWS IoT consoles AWS IoT Greengrass e ou a AWS IoT API para visualizar e atualizar a AWS IoT política de um dispositivo principal.

> **Note**
>
> Se você usou o instalador de software AWS IoT Greengrass Core para provisionar recursos, seu dispositivo principal tem uma AWS IoT política que permite acesso a todas as AWS IoT Greengrass ações (`greengrass:*`). Você pode seguir essas etapas para restringir o acesso somente às ações que um dispositivo principal usa.

Revise e atualize a AWS IoT política de um dispositivo principal (console)

1. No menu de navegação AWS IoT Greengrassdo console, escolha Dispositivos principais.
2. Na página Dispositivos principais, escolha o dispositivo principal a ser atualizado.
3. Na página de detalhes do dispositivo principal, escolha o link para o item do dispositivo principal. Esse link abre a página de detalhes do item no AWS IoT console.
4. Na página de detalhes do item, escolha Certificados.
5. Na guia Certificados, escolha o certificado ativo do item.
6. Na página de detalhes do certificado, escolha Políticas.
7. Na guia Políticas, escolha a AWS IoT política a ser revisada e atualizada. Você pode adicionar as permissões necessárias a qualquer política anexada ao certificado ativo do dispositivo principal.

> **Note**
>
> Se você usou o instalador de software AWS IoT Greengrass Core para provisionar recursos, você tem duas AWS IoT políticas. Recomendamos que você escolha a política nomeada GreengrassV2IoTThingPolicy, se ela existir. Os dispositivos principais que você cria com o instalador rápido usam esse nome de política por padrão. Se você adicionar permissões a essa política, também estará concedendo essas permissões a outros dispositivos principais que usam essa política.

8. Na visão geral da política, escolha Editar versão ativa.
9. Revise a política e adicione, remova ou edite as permissões, conforme necessário.
10. Para definir uma nova versão da política como a versão ativa, em Status da versão da política, selecione Definir a versão editada como a versão ativa desta política.
11. Selecione Salvar como nova versão.

Revise e atualize a AWS IoT política de um dispositivo principal (AWS CLI)

1. Liste os princípios básicos do AWS IoT dispositivo principal. Os principais podem ser certificados de dispositivos X.509 ou outros identificadores. Execute o comando a seguir e *MyGreengrassCore*substitua pelo nome do dispositivo principal.

   ```
   aws iot list-thing-principals --thing-name MyGreengrassCore
   ```

   A operação retorna uma resposta que lista os princípios básicos do dispositivo principal.

   ```
   {
       "principals": [
           "arn:aws:iot:us-west-2:123456789012:cert/certificateId"
       ]
   }
   ```

2. Identifique o certificado ativo do dispositivo principal. *Execute o comando a seguir e substitua CertificateID pela ID de cada certificado da etapa anterior até encontrar o certificado ativo.* O ID do certificado é a string hexadecimal no final do ARN do certificado. O `--query` argumento especifica a saída somente do status do certificado.

```
aws iot describe-certificate --certificate-id certificateId --query
 'certificateDescription.status'
```

A operação retorna o status do certificado como uma string. Por exemplo, se o certificado estiver ativo, essa operação será exibida. `"ACTIVE"`

3. Liste as AWS IoT políticas anexadas ao certificado. Execute o comando a seguir e substitua o ARN do certificado pelo ARN do certificado.

```
aws iot list-principal-policies --principal arn:aws:iot:us-
west-2:123456789012:cert/certificateId
```

A operação retorna uma resposta que lista as AWS IoT políticas anexadas ao certificado.

```
{
    "policies": [
        {
            "policyName":
 "GreengrassTESCertificatePolicyMyGreengrassCoreTokenExchangeRoleAlias",
            "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassTESCertificatePolicyMyGreengrassCoreTokenExchangeRoleAlias"
        },
        {
            "policyName": "GreengrassV2IoTThingPolicy",
            "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassV2IoTThingPolicy"
        }
    ]
}
```

4. Escolha a política a ser visualizada e atualizada.

> **Note**
>
> Se você usou o instalador de software AWS IoT Greengrass Core para provisionar recursos, você tem duas AWS IoT políticas. Recomendamos que você escolha a política nomeada GreengrassV2IoTThingPolicy, se ela existir. Os dispositivos principais que você cria com o instalador rápido usam esse nome de política por padrão. Se você

adicionar permissões a essa política, também estará concedendo essas permissões a outros dispositivos principais que usam essa política.

5. Obtenha o documento da política. Execute o comando a seguir e substitua *GreengrassV2IoTThingPolicy* pelo nome da política.

```
aws iot get-policy --policy-name GreengrassV2IoTThingPolicy
```

A operação retorna uma resposta que contém o documento da política e outras informações sobre a política. O documento de política é um objeto JSON serializado como uma string.

```
{
    "policyName": "GreengrassV2IoTThingPolicy",
    "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
GreengrassV2IoTThingPolicy",
    "policyDocument": "{\
  \\"Version\\": \\"2012-10-17\\",\
  \\"Statement\\": [\
    {\
      \\"Effect\\": \\"Allow\\",\
      \\"Action\\": [\
                \\"iot:Connect\\",\
                \\"iot:Publish\\",\
                \\"iot:Subscribe\\",\
                \\"iot:Receive\\",\
                \\"greengrass:*\\"\
],\
      \\"Resource\\": \\"*\\"\
    }\
  ]\
}",
    "defaultVersionId": "1",
    "creationDate": "2021-02-05T16:03:14.098000-08:00",
    "lastModifiedDate": "2021-02-05T16:03:14.098000-08:00",
    "generationId":
 "f19144b798534f52c619d44f771a354f1b957dfa2b850625d9f1d0fde530e75f"
}
```

6. Use um conversor on-line ou outra ferramenta para converter a string do documento de política em um objeto JSON e, em seguida, salve-a em um arquivo chamado`iot-policy.json`.

Por exemplo, se você tiver a ferramenta jq instalada, poderá executar o comando a seguir para obter o documento de política, convertê-lo em um objeto JSON e salvá-lo como um objeto JSON.

```
aws iot get-policy --policy-name GreengrassV2IoTThingPolicy --query
 'policyDocument' | jq fromjson >> iot-policy.json
```

7. Revise o documento de política e adicione, remova ou edite as permissões conforme necessário.

   Por exemplo, em um sistema baseado em Linux, você pode executar o seguinte comando para usar o GNU nano para abrir o arquivo.

   ```
   nano iot-policy.json
   ```

   Quando você terminar, o documento de política pode parecer semelhante à AWS IoTpolítica mínima para dispositivos principais.

8. Salve as alterações como uma nova versão da política. Execute o comando a seguir e substitua *GreengrassV2IoT ThingPolicy* pelo nome da política.

   ```
   aws iot create-policy-version --policy-name GreengrassV2IoTThingPolicy --policy-
   document file://iot-policy.json --set-as-default
   ```

   Se for bem-sucedida, a operação retornará uma resposta semelhante à do exemplo a seguir.

   ```
   {
       "policyArn": "arn:aws:iot:us-west-2:123456789012:policy/
   GreengrassV2IoTThingPolicy",
       "policyDocument": "{\
     \\"Version\\": \\"2012-10-17\\",\
     \\"Statement\\": [\
       {\
         \\"Effect\\": \\"Allow\\",\
         \\"Action\\": [\
   \\t\\t\\"iot:Connect\\",\
   \\t\\t\\"iot:Publish\\",\
   \\t\\t\\"iot:Subscribe\\",\
   \\t\\t\\"iot:Receive\\",\
   \\t\\t\\"greengrass:*\\"\
   ```

```
      ],\
      \\"Resource\\": \\"*\\"\
    }\
  ]\
}",
    "policyVersionId": "2",
    "isDefaultVersion": true
}
```

AWS IoTPolítica mínima para dispositivos AWS IoT Greengrass V2 principais

> **Important**
>
> Versões posteriores do componente nucleus do Greengrass exigem permissões adicionais sobre a política mínima. AWS IoT Talvez seja necessário atualizar as AWS IoT políticas dos seus dispositivos principais para conceder permissões adicionais.
>
> - Os dispositivos principais que executam o Greengrass nucleus v2.5.0 e versões posteriores usam a `greengrass:ListThingGroupsForCoreDevice` permissão para desinstalar componentes quando você remove um dispositivo principal de um grupo de coisas.
> - Os dispositivos principais que executam o Greengrass nucleus v2.3.0 e versões posteriores usam a `greengrass:GetDeploymentConfiguration` permissão para oferecer suporte a grandes documentos de configuração de implantação.

A política de exemplo a seguir inclui o conjunto mínimo de ações necessárias para oferecer suporte à funcionalidade básica do Greengrass para seu dispositivo de núcleo.

- A `Connect` política inclui o `*` caractere curinga após o nome do dispositivo principal (por exemplo,*`core-device-thing-name`**). O dispositivo principal usa o mesmo certificado de dispositivo para fazer várias assinaturas simultâneasAWS IoT Core, mas o ID do cliente em uma conexão pode não corresponder exatamente ao nome do dispositivo principal. Após as primeiras 50 assinaturas, o dispositivo principal é usado *`core-device-thing-name`*#*`number`* como ID do cliente, onde *`number`* incrementa para cada 50 assinaturas adicionais. Por exemplo, quando

um dispositivo principal chamado `MyCoreDevice` cria 150 assinaturas simultâneas, ele usa as seguintes IDs de cliente:

- Assinaturas 1 a 50: `MyCoreDevice`
- Assinaturas 51 a 100: `MyCoreDevice#2`
- Assinaturas 101 a 150: `MyCoreDevice#3`

O caractere curinga permite que o dispositivo principal se conecte quando usa esses IDs de cliente que têm um sufixo.

- A política lista os tópicos MQTT e filtros de tópicos nos quais o dispositivo de núcleo pode publicar mensagens, assinar e receber mensagens, incluindo tópicos usados para o estado de shadow. Para oferecer suporte à troca de mensagens entre AWS IoT Core componentes do Greengrass e dispositivos cliente, especifique os tópicos e os filtros de tópicos que você deseja permitir. Para obter mais informações, consulte Exemplos de políticas de publicação/assinatura no Guia do desenvolvedor do AWS IoT Core.
- A política concede permissão para publicar dados de telemetria no tópico a seguir.

  ```
  $aws/things/core-device-thing-name/greengrass/health/json
  ```

  Você pode remover essa permissão para dispositivos principais nos quais você desativa a telemetria. Para ter mais informações, consulte Colete dados de telemetria de integridade do sistema a partir dos dispositivos principais AWS IoT Greengrass.
- A política concede permissão para assumir uma função do IAM por meio de um alias de AWS IoT função. O dispositivo principal usa essa função, chamada de função de troca de tokens, para adquirir AWS credenciais que ele pode usar para autenticar solicitaçõesAWS. Para ter mais informações, consulte Autorize os dispositivos principais a interagir com os serviços AWS.

  Ao instalar o software AWS IoT Greengrass Core, você cria e anexa uma segunda AWS IoT política que inclui somente essa permissão. Se você incluir essa permissão na AWS IoT política primária do seu dispositivo principal, poderá desanexar e excluir a outra AWS IoT política.

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

```
                "iot:Connect"
            ],
            "Resource": "arn:aws:iot:region:account-id:client/core-device-thing-name*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive",
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:topic/$aws/things/core-device-thing-
name/greengrass/health/json",
                "arn:aws:iot:region:account-id:topic/$aws/things/core-device-thing-
name/greengrassv2/health/json",
                "arn:aws:iot:region:account-id:topic/$aws/things/core-device-thing-
name/jobs/*",
                "arn:aws:iot:region:account-id:topic/$aws/things/core-device-thing-
name/shadow/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:topicfilter/$aws/things/core-device-
thing-name/jobs/*",
                "arn:aws:iot:region:account-id:topicfilter/$aws/things/core-device-
thing-name/shadow/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": "iot:AssumeRoleWithCertificate",
            "Resource": "arn:aws:iot:region:account-id:rolealias/token-exchange-role-
alias-name"
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:GetComponentVersionArtifact",
                "greengrass:ResolveComponentCandidates",
```

```
                "greengrass:GetDeploymentConfiguration",
                "greengrass:ListThingGroupsForCoreDevice"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS IoTPolítica mínima para oferecer suporte aos dispositivos do cliente

O exemplo de política a seguir inclui o conjunto mínimo de ações necessárias para oferecer suporte à interação com dispositivos clientes em um dispositivo principal. Para oferecer suporte a dispositivos cliente, um dispositivo principal deve ter as permissões dessa AWS IoT política, além da AWS IoTpolítica Mínima para operação básica.

- A política permite que o dispositivo principal atualize suas próprias informações de conectividade. Essa permissão (`greengrass:UpdateConnectivityInfo`) é necessária somente se você implantar o componente detector de IP no dispositivo principal.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:topic/$aws/things/core-device-thing-name-gci/shadow/get"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:topicfilter/$aws/things/core-device-thing-name-gci/shadow/update/delta",
```

```
                "arn:aws:iot:region:account-id:topicfilter/$aws/things/core-device-
thing-name-gci/shadow/get/accepted"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:topic/$aws/things/core-device-thing-
name-gci/shadow/update/delta",
                "arn:aws:iot:region:account-id:topic/$aws/things/core-device-thing-
name-gci/shadow/get/accepted"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:PutCertificateAuthorities",
                "greengrass:VerifyClientDeviceIdentity"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:VerifyClientDeviceIoTCertificateAssociation"
            ],
            "Resource": "arn:aws:iot:region:account-id:thing/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:GetConnectivityInfo",
                "greengrass:UpdateConnectivityInfo"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:thing/core-device-thing-name"
            ]
        }
    ]
}
```

AWS IoTPolítica mínima para dispositivos clientes

O exemplo de política a seguir inclui o conjunto mínimo de ações necessárias para que um dispositivo cliente descubra os dispositivos principais nos quais eles se conectam e se comunicam pelo MQTT. A AWS IoT política do dispositivo cliente deve incluir a `greengrass:Discover` ação para permitir que o dispositivo descubra as informações de conectividade dos dispositivos principais do Greengrass associados. Na `Resource` seção, especifique o Amazon Resource Name (ARN) do dispositivo cliente, não o ARN do dispositivo principal do Greengrass.

- A política permite a comunicação sobre todos os tópicos do MQTT. Para seguir as melhores práticas de segurança, restrinja o `iot:Publishiot:Subscribe`,, e `iot:Receive` as permissões ao conjunto mínimo de tópicos que um dispositivo cliente exige para seu caso de uso.
- A política permite que a coisa descubra os principais dispositivos para todas as AWS IoT coisas. Para seguir as melhores práticas de segurança, restrinja a `greengrass:Discover` permissão ao AWS IoT item do dispositivo cliente ou a um caractere curinga que corresponda a um conjunto de AWS IoT itens.

  > **Important**
  >
  > As variáveis de política Thing (`iot:Connection.Thing.*`) não são suportadas em AWS IoT políticas para dispositivos principais ou operações de plano de dados do Greengrass. Em vez disso, você pode usar um caractere curinga que corresponda a vários dispositivos com nomes semelhantes. Por exemplo, você pode especificar `MyGreengrassDevice*` para corresponder `MyGreengrassDevice1MyGreengrassDevice2`, e assim por diante.

- A AWS IoT política de um dispositivo cliente normalmente não exige permissões ou `iot:DeleteThingShadow` ações `iot:GetThingShadowiot:UpdateThingShadow`, porque o dispositivo principal do Greengrass gerencia as operações de sincronização paralela para dispositivos cliente. Para permitir que o dispositivo principal manipule as sombras do dispositivo cliente, verifique se a AWS IoT política do dispositivo principal permite essas ações e se a `Resource` seção inclui os ARNs dos dispositivos cliente.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
```

```
            "Action": [
                "iot:Connect"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:topic/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:topicfilter/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:topic/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:Discover"
            ],
            "Resource": [
                "arn:aws:iot:region:account-id:thing/*"
            ]
        }
    ]
}
```

Gerenciamento de identidade e acesso para o AWS IoT Greengrass

O AWS Identity and Access Management (IAM) é um serviço da AWS service (Serviço da AWS) que ajuda a controlar o acesso aos recursos da AWS de forma segura. Os administradores do IAM controlam quem pode ser autenticado (conectado) e autorizado (ter permissões) a usar os recursos do AWS IoT Greengrass. O IAM é um AWS service (Serviço da AWS) que pode ser usado sem custo adicional.

> **Note**
>
> Este tópico descreve conceitos e atributos do IAM. Para obter informações sobre atributos do IAM compatíveis com AWS IoT Greengrass, consulte the section called “Como o AWS IoT Greengrass funciona com o IAM”.

Público

O uso do AWS Identity and Access Management (IAM) varia dependendo do trabalho que for realizado no AWS IoT Greengrass.

Usuário do serviço – Se você usar o serviço AWS IoT Greengrass para fazer o trabalho, o administrador fornecerá as credenciais e as permissões necessárias. À medida que usar mais recursos do AWS IoT Greengrass para fazer seu trabalho, você poderá precisar de permissões adicionais. Entender como o acesso é gerenciado pode ajudá-lo a solicitar as permissões corretas ao seu administrador. Se não for possível acessar um recurso no AWS IoT Greengrass, consulte Solução de problemas de identidade e acesso do AWS IoT Greengrass.

Administrador do serviço – Se você for o responsável pelos recursos do AWS IoT Greengrass na empresa, provavelmente terá acesso total ao AWS IoT Greengrass. Cabe a você determinar quais funcionalidades e recursos do AWS IoT Greengrass os usuários do serviço devem acessar. Assim, você deve enviar solicitações ao administrador do IAM para alterar as permissões dos usuários de seu serviço. Analise as informações nesta página para entender os conceitos básicos do IAM. Para saber mais sobre como a empresa pode usar o IAM com o AWS IoT Greengrass, consulte Como o AWS IoT Greengrass funciona com o IAM.

Administrador do IAM – Se você for um administrador do IAM, talvez queira saber detalhes sobre como pode gravar políticas para gerenciar o acesso ao AWS IoT Greengrass. Para visualizar

exemplos AWS IoT Greengrass de políticas baseadas em identidade do que podem ser usadas no IAM, consulte Exemplos de políticas baseadas em identidade para o AWS IoT Greengrass.

Como autenticar com identidades

A autenticação é a forma como você faz login na AWS usando suas credenciais de identidade. É necessário ser autenticado (fazer login na AWS) como o usuário raiz da Usuário raiz da conta da AWS, como usuário do IAM ou assumindo um perfil do IAM.

Você pode fazer login na AWS como uma identidade federada usando credenciais fornecidas por uma fonte de identidades. Os usuários do AWS IAM Identity Center (IAM Identity Center), a autenticação única da empresa e as suas credenciais do Google ou do Facebook são exemplos de identidades federadas. Quando você faz login como uma identidade federada, o administrador já configurou anteriormente a federação de identidades utilizando perfis do IAM. Quando você acessa a AWS usando a federação, está indiretamente assumindo um perfil.

É possível fazer login no AWS Management Console ou no de acesso da AWS dependendo do tipo de usuário que você é. Para obter mais informações sobre como fazer login na AWS, consulte Como fazer login na conta da Conta da AWS no Guia do usuário do Início de Sessão da AWS.

Se você acessar a AWS programaticamente, a AWS fornecerá um kit de desenvolvimento de software (SDK) e uma interface da linha de comando (CLI) para você assinar criptograficamente as solicitações usando as suas credenciais. Se você não utilizar as ferramentas da AWS, deverá assinar as solicitações por conta própria. Para obter mais informações sobre como usar o método recomendado para assinar solicitações por conta própria, consulte Assinar solicitações de API da AWS no Guia do usuário do IAM.

Independentemente do método de autenticação usado, também pode ser exigido que você forneça mais informações de segurança. Por exemplo, a AWS recomenda o uso da autenticação multifator (MFA) para aumentar a segurança de sua conta. Para saber mais, consulte Autenticação multifator no Guia do usuário do AWS IAM Identity Center e Usar a autenticação multifator (MFA) naAWS no Guia do usuário do IAM.

Usuário raiz da Conta da AWS

Ao criar uma Conta da AWS, você começa com uma identidade de login que tem acesso completo a todos os atributos e Serviços da AWS na conta. Essa identidade, denominada usuário raiz da Conta da AWS, e é acessada por login com o endereço de e-mail e a senha que você usou para criar a conta. É altamente recomendável não utilizar o usuário raiz para tarefas diárias. Proteja as

credenciais do usuário raiz e use-as para executar as tarefas que somente ele puder executar. Para obter a lista completa das tarefas que exigem login como usuário raiz, consulte Tarefas que exigem credenciais de usuário raiz no Guia do usuário do IAM.

Usuários e grupos do IAM

Um usuário do IAM é uma identidade dentro da Conta da AWS que tem permissões específicas para uma única pessoa ou aplicação. Sempre que possível, recomendamos contar com credenciais temporárias em vez de criar usuários do IAM com credenciais de longo prazo, como senhas e chaves de acesso. No entanto, se você tiver casos de utilização específicos que exijam credenciais de longo prazo com usuários do IAM, recomendamos alternar as chaves de acesso. Para obter mais informações, consulte Alterne as chaves de acesso regularmente para casos de uso que exijam credenciais de longo prazo no Guia do usuário do IAM.

Um grupo do IAM é uma identidade que especifica uma coleção de usuários do IAM. Não é possível fazer login como um grupo. É possível utilizar grupos para especificar permissões para vários usuários de uma vez. Os grupos facilitam o gerenciamento de permissões para grandes conjuntos de usuários. Por exemplo, você pode ter um grupo chamado IAMAdmins e atribuir a esse grupo permissões para administrar atributos do IAM.

Usuários são diferentes de perfis. Um usuário é exclusivamente associado a uma pessoa ou a uma aplicação, mas um perfil pode ser assumido por qualquer pessoa que precisar dele. Os usuários têm credenciais permanentes de longo prazo, mas os perfis fornecem credenciais temporárias. Para saber mais, consulte Quando criar um usuário do IAM (em vez de uma função) no Guia do usuário do IAM.

Perfis do IAM

Um perfil do IAM é uma identidade dentro da Conta da AWS que tem permissões específicas. Ele é semelhante a um usuário do IAM, mas não está associado a uma pessoa específica. É possível assumir temporariamente um perfil do IAM no AWS Management Console alternando perfis. É possível assumir um perfil chamando uma operação de API da AWS CLI ou da AWS, ou usando um URL personalizado. Para obter mais informações sobre métodos para o uso de perfis, consulte Usar perfis do IAM no Guia do usuário do IAM.

Perfis do IAM com credenciais temporárias são úteis nas seguintes situações:

- Acesso de usuário federado: para atribuir permissões a identidades federadas, você pode criar um perfil e definir permissões para ele. Quando uma identidade federada é autenticada, essa identidade é associada ao perfil e recebe as permissões definidas pelo mesmo. Para obter

mais informações sobre perfis para federação, consulte Criar uma função para um provedor de identidade de terceiros no Guia do usuário do IAM. Se você usar o IAM Identity Center, deverá configurar um conjunto de permissões. Para controlar o que suas identidades podem acessar após a autenticação, o Centro de identidade do IAM correlaciona o conjunto de permissões a um perfil no IAM. Para obter informações sobre conjuntos de permissões, consulte Conjuntos de permissões no Guia do usuário do AWS IAM Identity Center.

- Permissões temporárias para usuários do IAM: um usuário ou um perfil do IAM pode assumir um perfil do IAM para obter temporariamente permissões diferentes para uma tarefa específica.
- Acesso entre contas: é possível usar um perfil do IAM para permitir que alguém (uma entidade principal confiável) em outra conta acesse recursos em sua conta. Os perfis são a principal forma de conceder acesso entre contas. No entanto, alguns Serviços da AWS permitem que você anexe uma política diretamente a um atributo (em vez de usar um perfil como proxy). Para saber a diferença entre perfis e políticas baseadas em atributo para acesso entre contas, consulte Como os perfis do IAM diferem das políticas baseadas em atributo no Guia do usuário do IAM.
- Acesso entre serviços: alguns Serviços da AWS usam atributos em outros Serviços da AWS. Por exemplo, quando você faz uma chamada em um serviço, é comum que esse serviço execute aplicações no Amazon EC2 ou armazene objetos no Amazon S3. Um serviço pode fazer isso usando as permissões do principal de chamada, usando um perfil de serviço ou uma função vinculada ao serviço.
  - Encaminhamento de sessões de acesso (FAS): qualquer pessoa que utilizar uma função ou usuário do IAM para realizar ações na AWS é considerada uma entidade principal. Ao usar alguns serviços, você pode executar uma ação que inicia outra ação em um serviço diferente. O recurso FAS utiliza as permissões da entidade principal que chama um AWS service (Serviço da AWS), combinadas às permissões do AWS service (Serviço da AWS) solicitante, para realizar solicitações para serviços downstream. As solicitações de FAS só são feitas quando um serviço recebe uma solicitação que exige interações com outros Serviços da AWS ou com recursos para serem concluídas. Nesse caso, você precisa ter permissões para executar ambas as ações. Para obter detalhes da política ao fazer solicitações de FAS, consulte Encaminhar sessões de acesso.
  - Perfil de serviço: um perfil de serviço é um perfil do IAM que um serviço assume para realizar ações em seu nome. Um administrador do IAM pode criar, modificar e excluir um perfil de serviço do IAM. Para obter mais informações, consulte Criar um perfil para delegar permissões a um AWS service (Serviço da AWS) no Guia do usuário do IAM.
  - Função vinculada ao serviço: uma função vinculada a serviço é um tipo de função de serviço vinculada a um AWS service (Serviço da AWS). O serviço pode assumir o perfil para executar

uma ação em seu nome. Os perfis vinculados ao serviço aparecem em sua Conta da AWS e são de propriedade do serviço. Um administrador do IAM pode exibir, mas não pode editar as permissões para perfis vinculados ao serviço.

- Aplicações em execução no Amazon EC2: é possível usar um perfil do IAM para gerenciar credenciais temporárias para aplicações em execução em uma instância do EC2 e fazer solicitações da AWS CLI ou da AWS API. É preferível fazer isso a armazenar chaves de acesso na instância do EC2. Para atribuir um perfil da AWS a uma instância do EC2 e disponibilizá-la para todas as suas aplicações, crie um perfil de instância que esteja anexado a ela. Um perfil de instância contém a perfil e permite que os programas em execução na instância do EC2 obtenham credenciais temporárias. Para mais informações, consulte Uso de uma função do IAM para conceder permissões a aplicações em execução em instâncias do Amazon EC2 no Guia do usuário do IAM.

Para saber se deseja usar os perfis do IAM, consulte Quando criar um perfil do IAM (em vez de um usuário) no Guia do usuário do IAM.

Como gerenciar acesso usando políticas

Você controla o acesso na AWS criando políticas e anexando-as a identidades ou atributos da AWS. Uma política é um objeto na AWS que, quando associado a uma identidade ou atributo, define suas permissões. A AWS avalia essas políticas quando uma entidade principal (usuário, usuário raiz ou sessão de perfil) faz uma solicitação. As permissões nas políticas determinam se a solicitação será permitida ou negada. A maioria das políticas são armazenadas na AWS como documentos JSON. Para obter mais informações sobre a estrutura e o conteúdo de documentos de políticas JSON, consulte Visão geral das políticas JSON no Guia do usuário do IAM.

Os administradores podem usar AWS as políticas JSON da para especificar quem tem acesso a quê. Ou seja, qual entidade principal pode executar ações em quais recursos e em que condições.

Por padrão, usuários e funções não têm permissões. Para conceder aos usuários permissão para executar ações nos recursos de que eles precisam, um administrador do IAM pode criar políticas do IAM. O administrador pode então adicionar as políticas do IAM a perfis, e os usuários podem assumir os perfis.

As políticas do IAM definem permissões para uma ação independente do método usado para executar a operação. Por exemplo, suponha que você tenha uma política que permite a ação `iam:GetRole`. Um usuário com essa política pode obter informações de perfil do AWS Management Console, da AWS CLI ou da AWS API.

Políticas baseadas em identidade

As políticas baseadas em identidade são documentos de políticas de permissões JSON que você pode anexar a uma identidade, como usuário, grupo de usuários ou perfil do IAM. Essas políticas controlam quais ações os usuários e perfis podem realizar, em quais recursos e em que condições. Para saber como criar uma política baseada em identidade, consulte Criar políticas do IAM no Guia do usuário do IAM.

As políticas baseadas em identidade podem ser categorizadas ainda mais como políticas embutidas ou políticas gerenciadas. As políticas embutidas são anexadas diretamente a um único usuário, grupo ou função. As políticas gerenciadas são políticas independentes que podem ser anexadas a vários usuários, grupos e perfis na Conta da AWS. As políticas gerenciadas incluem políticas gerenciadas pela AWS e políticas gerenciadas pelo cliente. Para saber como escolher entre uma política gerenciada ou uma política em linha, consulte Escolher entre políticas gerenciadas e políticas em linha no Guia do usuário do IAM.

Políticas baseadas em recurso

Políticas baseadas em atributos são documentos de políticas JSON que você anexa a um atributo. São exemplos de políticas baseadas em recurso as políticas de confiança de função do IAM e as políticas de bucket do Amazon S3. Em serviços compatíveis com políticas baseadas em recurso, os administradores de serviço podem usá-las para controlar o acesso a um recurso específico. Para o recurso ao qual a política está anexada, a política define quais ações uma entidade principal especificado pode executar nesse recurso e em que condições. Você precisa especificar uma entidade principal em uma política baseada em recurso. As entidades principais podem incluir contas, usuários, perfis, usuários federados ou Serviços da AWS.

Políticas baseadas em atributos são políticas em linha que estão localizadas nesse serviço. Não é possível usar as políticas gerenciadas da AWS do IAM em uma política baseada em atributos.

Listas de controle de acesso (ACLs)

As listas de controle de acesso (ACLs) controlam quais entidades principais (membros, usuários ou funções da conta) têm permissões para acessar um recurso. As ACLs são semelhantes às políticas baseadas em recurso, embora não usem o formato de documento de política JSON.

Amazon S3, AWS WAF e Amazon VPC são exemplos de serviços compatíveis com ACLs. Para saber mais sobre ACLs, consulte Visão geral da lista de controle de acesso (ACL) no Guia do desenvolvedor do Amazon Simple Storage Service.

Outros tipos de política

A AWS aceita tipos de política menos comuns. Esses tipos de política podem definir o máximo de permissões concedidas a você pelos tipos de política mais comuns.

- Limites de permissões: um limite de permissões é um atributo avançado no qual você define o máximo de permissões que uma política baseada em identidade pode conceder a uma entidade do IAM (usuário ou perfil do IAM). É possível definir um limite de permissões para uma entidade. As permissões resultantes são a interseção das políticas baseadas em identidade de uma entidade com seus limites de permissões. As políticas baseadas em recurso que especificam o usuário ou perfil no campo `Principal` não são limitadas pelo limite de permissões. Uma negação explícita em qualquer uma dessas políticas substitui a permissão. Para obter mais informações sobre limites de permissões, consulte Limites de permissões para identidades do IAM no Guia do usuário do IAM.
- Políticas de controle de serviço (SCPs): SCPs são políticas JSON que especificam as permissões máximas para uma organização ou unidade organizacional (UO) no AWS Organizations. O AWS Organizations é um serviço para agrupar e gerenciar centralmente várias Contas da AWS pertencentes à sua empresa. Se você habilitar todos os atributos em uma organização, poderá aplicar políticas de controle de serviço (SCPs) a qualquer uma ou a todas as contas. O SCP limita as permissões para entidades em contas-membro, incluindo cada .Usuário raiz da conta da AWS Para obter mais informações sobre o Organizações e SCPs, consulte How SCPs work (Como os SCPs funcionam) noAWS Organizations Guia do usuário do .
- Políticas de sessão: são políticas avançadas que você transmite como um parâmetro quando cria de forma programática uma sessão temporária para um perfil ou um usuário federado. As permissões da sessão resultante são a interseção das políticas baseadas em identidade do usuário ou do perfil e das políticas de sessão. As permissões também podem ser provenientes de uma política baseada em atributo. Uma negação explícita em qualquer uma dessas políticas substitui a permissão. Para obter mais informações, consulte Políticas de sessão no Guia do usuário do IAM.

Vários tipos de política

Quando vários tipos de política são aplicáveis a uma solicitação, é mais complicado compreender as permissões resultantes. Para saber como a AWS determina se deve permitir uma solicitação quando há vários tipos de política envolvidos, consulte Lógica da avaliaçãode políticas no Guia do usuário do IAM.

Consulte também

- the section called “Como o AWS IoT Greengrass funciona com o IAM”
- the section called “Exemplos de políticas baseadas em identidade”
- the section called “Solução de problemas de identidade e acesso”

Como o AWS IoT Greengrass funciona com o IAM

Antes de usar o IAM para gerenciar o acesso aoAWS IoT Greengrass, você deve entender os recursos do IAM com os quais você pode usarAWS IoT Greengrass.

Recurso do IAM	Compatível com o Greengrass?
Políticas baseadas em identidade com permissões em nível de recurso	Sim
Políticas baseadas em recursos	Não
Listas de controle de acesso (ACLs)	Não
Autorização baseada em tags	Sim
Credenciais temporárias	Sim
Funções vinculadas ao serviço	Não
Funções de serviço	Sim

Para obter uma visão de alto nível de como outrosAWS serviços trabalham com o IAM, consulte AWSServiços compatíveis com o IAM no Guia do usuário do IAM.

Políticas baseadas em identidade para o AWS IoT Greengrass

Com as políticas baseadas em identidade do IAM, é possível especificar ações e recursos permitidos ou negados e as condições sob as quais as ações são permitidas ou negadas. O AWS IoT Greengrass oferece suporte a ações, recursos e chaves de condição específicos. Para saber mais sobre todos os elementos usados em uma política, consulte Referência de elementos de política JSON do IAM no Guia do usuário do IAM.

Ações

Os administradores podem usar AWS as políticas JSON da para especificar quem tem acesso a quê. Ou seja, qual principal pode executar ações em quais recursos, e em que condições.

O elemento `Action` de uma política JSON descreve as ações que você pode usar para permitir ou negar acesso em uma política. As ações de política geralmente têm o mesmo nome que a operação de API da AWS associada. Existem algumas exceções, como ações somente de permissão, que não têm uma operação de API correspondente. Há também algumas operações que exigem várias ações em uma política. Essas ações adicionais são chamadas de ações dependentes.

Inclua ações em uma política para conceder permissões para executar a operação associada.

As ações de política do AWS IoT Greengrass usam o prefixo `greengrass:` antes da ação. Por exemplo, para permitir que alguém use a operação da`ListCoreDevices` API para listarConta da AWS os`greengrass:ListCoreDevices` dispositivos principais do no. As declarações de política devem incluir um elemento `Action` ou `NotAction`. O AWS IoT Greengrass define seu próprio conjunto de ações que descrevem as tarefas que podem ser executadas com esse serviço.

Para especificar várias ações em uma única declaração, liste-as entre colchetes (`[]`) e separe-as com vírgulas, conforme o seguinte:

```
"Action": [
  "greengrass:action1",
  "greengrass:action2",
  "greengrass:action3"
]
```

Você pode usar curingas (*) para especificar várias ações. Por exemplo, para especificar todas as ações que começam com a palavra `List`, inclua a seguinte ação:

```
"Action": "greengrass:List*"
```

> **Note**
>
> Recomendamos que você evite o uso de curingas para especificar todas as ações disponíveis para um serviço. De acordo com as melhores práticas, você deve conceder permissões de privilégio mínimo e definir um escopo de permissões mais específico em

uma política. Para obter mais informações, consulte the section called “Conceder o mínimo possível de permissões”.

Para obter a lista completa deAWS IoT Greengrass ações, consulte Ações definidas porAWS IoT Greengrass no Guia do usuário do IAM.

Recursos

Os administradores podem usar AWS as políticas JSON da para especificar quem tem acesso a quê. Ou seja, qual principal pode executar ações em quais recursos, e em que condições.

O elemento `Resource` de política JSON especifica o objeto ou os objetos aos quais a ação se aplica. As instruções devem incluir um elemento `Resource` ou um elemento `NotResource`. Como prática recomendada, especifique um recurso usando seu Nome do recurso da Amazon (ARN). Isso pode ser feito para ações que oferecem suporte a um tipo de recurso específico, conhecido como permissões em nível de recurso.

Para ações que não oferecem suporte a permissões em nível de recurso, como operações de listagem, use um curinga (*) para indicar que a instrução se aplica a todos os recursos.

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

A tabela a seguir contém os ARNs de recurso do AWS IoT Greengrass que podem ser usados no elemento `Resource` de uma declaração de política. Para um mapeamento das permissões suportadas em nível de recurso paraAWS IoT Greengrass ações, consulte Ações definidas porAWS IoT Greengrass no Guia do usuário do IAM.

Algumas ações do AWS IoT Greengrass (por exemplo, algumas operações de lista), não podem ser executadas em um recurso específico. Nesses casos, você deve usar apenas o caractere curinga.

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

Para especificar vários ARNs de recursos em uma declaração, liste-os entre colchetes (`[]`) e separe-os com vírgulas, da seguinte forma:

```
"Resource": [
  "resource-arn1",
  "resource-arn2",
```

```
  "resource-arn3"
]
```

Para obter mais informações sobre formatos de ARN, consulte Nomes de recurso da Amazon (ARNs) e namespaces de serviços da AWS, no Referência geral da Amazon Web Services.

Chaves de condição

Os administradores podem usar AWS as políticas JSON da para especificar quem tem acesso a quê. Ou seja, qual entidade principal pode executar ações em quais recursos e em que condições.

O elemento `Condition` (ou bloco de `Condition`) permite que você especifique condições nas quais uma instrução está em vigor. O elemento `Condition` é opcional. É possível criar expressões condicionais que usam agentes de condição, como “igual a” ou “menor que”, para fazer a condição da política corresponder aos valores na solicitação.

Se você especificar vários elementos `Condition` em uma instrução ou várias chaves em um único elemento `Condition`, a AWS os avaliará usando uma operação lógica `AND`. Se você especificar vários valores para uma única chave de condição, a AWS avaliará a condição usando uma operação lógica `OR`. Todas as condições devem ser atendidas para que as permissões da instrução sejam concedidas.

Você também pode usar variáveis de espaço reservado ao especificar as condições. Por exemplo, é possível conceder a um usuário do IAM permissão para acessar um recurso somente se ele estiver marcado com seu nome de usuário do IAM. Para obter mais informações, consulte Elementos de política do IAM: variáveis e tags no Guia do usuário do IAM.

A AWS oferece suporte a chaves de condição globais e chaves de condição específicas do serviço. Para ver todas as chaves de condição globais da AWS, consulte Chaves de contexto de condição globais da AWS no Guia do usuário do IAM.

Exemplos

Para visualizar exemplos de políticas baseadas em identidade do AWS IoT Greengrass, consulte the section called “Exemplos de políticas baseadas em identidade”.

Políticas baseadas em recursos para o AWS IoT Greengrass

O AWS IoT Greengrass não oferece suporte a políticas baseadas em recurso.

Listas de controle de acesso (ACLs)

O AWS IoT Greengrass não oferece suporte a ACLs.

Autorização baseada em tags do AWS IoT Greengrass

É possível anexar tags a recursos do AWS IoT Greengrass compatíveis ou passar tags em uma solicitação ao AWS IoT Greengrass. Para controlar o acesso baseado em tags, forneça informações sobre as tags no elemento de condição de uma política usando as chaves de condição `aws:ResourceTag/${TagKey}`, `aws:RequestTag/${TagKey}` ou `aws:TagKeys`. Para obter mais informações, consulte Marcar com tag os recursos do .

Funções do IAM paraAWS IoT Greengrass

Uma função do IAM é uma entidade dentro da sua Conta da AWS que tem permissões específicas.

Usar credenciais temporárias com o AWS IoT Greengrass

As credenciais temporárias são usadas para fazer login com federação, assumir uma função do IAM ou assumir uma função entre contas. As credenciais de segurança temporárias são obtidas chamando operações deAWS STS API do, como AssumeRoleou GetFederationToken.

No núcleo do Greengrass, as credenciais temporárias para a função de dispositivo são disponibilizadas para os componentes do Greengrass. Se seus componentes usarem oAWS SDK, você não precisará adicionar lógica para obter as credenciais porque oAWS SDK faz isso por você.

Funções vinculadas ao serviço

O AWS IoT Greengrass não oferece suporte às funções vinculadas ao serviço.

Perfis de serviço

Esse recurso permite que um serviço assuma um perfil de serviço em seu nome. A função permite que o serviço acesse recursos em outros serviços para concluir uma ação em seu nome. As funções de serviço aparecem em sua conta do IAM e são de propriedade da conta. Isso significa que um administrador do IAM pode alterar as permissões para essa função. Porém, fazer isso pode alterar a funcionalidade do serviço.

AWS IoT Greengrassos dispositivos principais usam uma função de serviço para permitir que os componentes do Greengrass e as funções do Lambda acessem alguns de seusAWS recursos

em seu nome. Para obter mais informações, consulte the section called “Autorize os dispositivos principais a interagir com os serviços AWS”.

O AWS IoT Greengrass usa uma função de serviço para acessar alguns dos recursos da AWS em seu nome. Para obter mais informações, consulte Perfil de serviço do Greengrass.

Exemplos de políticas baseadas em identidade para o AWS IoT Greengrass

Por padrão, os usuários e as funções do IAM não têm permissão para criar ou modificar recursos do AWS IoT Greengrass. Eles também não podem executar tarefas usando o AWS Management Console, a AWS CLI ou uma API da AWS. Um administrador do IAM deve criar políticas do IAM que concedam aos usuários e perfis permissão para executarem operações de API específicas nos recursos especificados de que precisam. O administrador deve anexar essas políticas aos usuários ou grupos do IAM que exigem essas permissões.

Práticas recomendadas de políticas

As políticas baseadas em identidade determinam se alguém pode criar, acessar ou excluir recursos do AWS IoT Greengrass em sua conta. Essas ações podem incorrer em custos para a Conta da AWS. Ao criar ou editar políticas baseadas em identidade, siga estas diretrizes e recomendações:

- Comece com as políticas gerenciadas pela AWS e avance para as permissões de privilégio mínimo: para começar a conceder permissões a seus usuários e workloads, use as políticas gerenciadas pela AWS que concedem permissões para muitos casos de uso comuns. Eles estão disponíveis na sua Conta da AWS. Recomendamos que você reduza ainda mais as permissões definindo políticas gerenciadas pelo cliente da AWS específicas para seus casos de uso. Para obter mais informações, consulte Políticas gerenciadas pela AWS ou Políticas gerenciadas pela AWS para funções de trabalho no Guia do usuário do IAM.
- Aplique permissões de privilégio mínimo: ao definir permissões com as políticas do IAM, conceda apenas as permissões necessárias para executar uma tarefa. Você faz isso definindo as ações que podem ser executadas em recursos específicos sob condições específicas, também conhecidas como permissões de privilégio mínimo. Para obter mais informações sobre como usar o IAM para aplicar permissões, consulte Políticas e permissões no IAM no Guia do usuário do IAM.
- Use condições nas políticas do IAM para restringir ainda mais o acesso: você pode adicionar uma condição às políticas para limitar o acesso a ações e recursos. Por exemplo, você pode escrever uma condição de política para especificar que todas as solicitações devem ser enviadas usando SSL. Você também pode usar condições para conceder acesso a ações de serviço, se elas forem

usadas por meio de um AWS service (Serviço da AWS) específico, como o AWS CloudFormation. Para obter mais informações, consulte Elementos de política JSON do IAM: Condition no Manual do usuário do IAM.

- Use o IAM Access Analyzer para validar suas políticas do IAM a fim de garantir permissões seguras e funcionais: o IAM Access Analyzer valida as políticas novas e existentes para que elas sigam a linguagem de política do IAM (JSON) e as práticas recomendadas do IAM. O IAM Access Analyzer oferece mais de 100 verificações de política e recomendações acionáveis para ajudar você a criar políticas seguras e funcionais. Para obter mais informações, consulte Validação de políticas do IAM Access Analyzer no Guia do usuário do IAM.
- Require multi-factor authentication (MFA) (Exigir autenticação multifator (MFA)): se houver um cenário que exija usuários do IAM ou um usuário raiz em sua Conta da AWS, ative a MFA para obter segurança adicional. Para exigir a MFA quando as operações de API forem chamadas, adicione condições de MFA às suas políticas. Para obter mais informações, consulte Configuração de acesso à API protegido por MFA no Guia do usuário do IAM.

Para obter mais informações sobre as práticas recomendadas do IAM, consulte Práticas recomendadas de segurança no IAM no Guia do usuário do IAM.

Exemplos de políticas

As políticas de exemplo definidas pelo cliente a seguir concedem permissões para cenários comuns.

Exemplos

- Permitir que os usuários visualizem suas próprias permissões

Para saber como criar uma política baseada em identidade do IAM usando esses exemplos de documentos de política JSON, consulte Criar políticas na guia JSON no Guia do usuário do IAM.

Permitir que os usuários visualizem suas próprias permissões

Este exemplo mostra como você pode criar uma política que permite que os usuários do IAM visualizem as políticas gerenciadas e em linha anexadas a sua identidade de usuário. Essa política inclui permissões para concluir essa ação no console ou de forma programática usando a AWS CLI ou a API da AWS.

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Autorize os dispositivos principais a interagir com os serviços AWS

AWS IoT Greengrass os dispositivos principais usam o provedor de AWS IoT Core credenciais para autorizar chamadas para AWS serviços. O provedor de AWS IoT Core credenciais permite que os dispositivos usem seus certificados X.509 como a identidade exclusiva do dispositivo para autenticar solicitações. AWS Isso elimina a necessidade de armazenar uma ID de chave de AWS acesso e uma chave de acesso secreta em seus dispositivos AWS IoT Greengrass principais. Para obter mais informações, consulte Autorização de chamadas diretas para AWS serviços no Guia do AWS IoT Core desenvolvedor.

Ao executar o software AWS IoT Greengrass Core, você pode optar por provisionar os AWS recursos que o dispositivo principal exige. Isso inclui a função AWS Identity and Access Management

(IAM) que seu dispositivo principal assume por meio do provedor de AWS IoT Core credenciais. Use o `--provision true` argumento para configurar uma função e políticas que permitam que o dispositivo principal obtenha AWS credenciais temporárias. Esse argumento também configura um alias de AWS IoT função que aponta para essa função do IAM. Você pode especificar o nome da função do IAM e o alias da AWS IoT função a serem usados. Se você especificar `--provision true` sem esses outros parâmetros de nome, o dispositivo principal do Greengrass cria e usa os seguintes recursos padrão:

- Função do IAM: `GreengrassV2TokenExchangeRole`

  Essa função tem uma política nomeada `GreengrassV2TokenExchangeRoleAccess` e uma relação de confiança que `credentials.iot.amazonaws.com` permite assumir a função. A política inclui as permissões mínimas para o dispositivo principal.

  > **Important**
  >
  > Essa política não inclui acesso a arquivos em buckets do S3. Você deve adicionar permissões à função para permitir que os dispositivos principais recuperem artefatos de componentes dos buckets do S3. Para ter mais informações, consulte Permitir acesso aos buckets do S3 para artefatos de componentes.

- AWS IoT alias de função: `GreengrassV2TokenExchangeRoleAlias`

  Esse alias de função se refere à função do IAM.

Para ter mais informações, consulte Etapa 3: Instalar o software doAWS IoT Greengrass Performance.

Você também pode definir o alias de função para um dispositivo principal existente. Para fazer isso, configure o parâmetro de `iotRoleAlias` configuração do componente do núcleo do Greengrass.

Você pode adquirir AWS credenciais temporárias para essa função do IAM para realizar AWS operações em seus componentes personalizados. Para ter mais informações, consulte Interaja com AWS os serviços.

Tópicos

- Permissões de função de serviço para dispositivos principais
- Permitir acesso aos buckets do S3 para artefatos de componentes

Permissões de função de serviço para dispositivos principais

A função permite que o seguinte serviço assuma a função:

- `credentials.iot.amazonaws.com`

Se você usa o software AWS IoT Greengrass Core para criar essa função, ele usa a seguinte política de permissões para permitir que os dispositivos principais se conectem e enviem registros para AWS. O nome padrão da política é o nome da função do IAM que termina com. `Access` Por exemplo, se você usar o nome de função padrão do IAM, o nome dessa política será`GreengrassV2TokenExchangeRoleAccess`.

Greengrass nucleus v2.5.0 and later

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",
        "s3:GetBucketLocation"
      ],
      "Resource": "*"
    }
  ]
}
```

v2.4.x

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:DescribeCertificate",
        "logs:CreateLogGroup",
```

```
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",
        "s3:GetBucketLocation"
      ],
      "Resource": "*"
    }
  ]
}
```

Earlier than v2.4.0

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:DescribeCertificate",
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:DescribeLogStreams",
        "iot:Connect",
        "iot:Publish",
        "iot:Subscribe",
        "iot:Receive",
        "s3:GetBucketLocation"
      ],
      "Resource": "*"
    }
  ]
}
```

Permitir acesso aos buckets do S3 para artefatos de componentes

A função padrão do dispositivo principal não permite que os dispositivos principais acessem os buckets do S3. Para implantar componentes que tenham artefatos em buckets do S3, você deve adicionar a `s3:GetObject` permissão para permitir que os dispositivos principais baixem artefatos de componentes. Você pode adicionar uma nova política à função principal do dispositivo para conceder essa permissão.

Para adicionar uma política que permita o acesso a artefatos de componentes no Amazon S3

1. Crie um arquivo chamado `component-artifact-policy.json` e copie o seguinte JSON para o arquivo. Essa política permite acesso a todos os arquivos em um bucket do S3. Substitua DOC-EXAMPLE-BUCKET pelo nome do bucket S3 para permitir o acesso do dispositivo principal.

   ```
   {
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Action": [
           "s3:GetObject"
         ],
         "Resource": "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*"
       }
     ]
   }
   ```

2. Execute o comando a seguir para criar a política a partir do documento de política em`component-artifact-policy.json`.

   Linux or Unix

   ```
   aws iam create-policy \
     --policy-name MyGreengrassV2ComponentArtifactPolicy \
     --policy-document file://component-artifact-policy.json
   ```

   Windows Command Prompt (CMD)

   ```
   aws iam create-policy ^
     --policy-name MyGreengrassV2ComponentArtifactPolicy ^
     --policy-document file://component-artifact-policy.json
   ```

   PowerShell

   ```
   aws iam create-policy `
     --policy-name MyGreengrassV2ComponentArtifactPolicy `
     --policy-document file://component-artifact-policy.json
   ```

Copie a política Amazon Resource Name (ARN) dos metadados da política na saída. Você usa esse ARN para anexar essa política à função do dispositivo principal na próxima etapa.

3. Execute o comando a seguir para anexar a política à função do dispositivo principal. Substitua *GreengrassV2 TokenExchangeRole* pelo nome da função que você especificou ao executar o software Core. AWS IoT Greengrass Em seguida, substitua o ARN da política pelo ARN da etapa anterior.

   Linux or Unix

   ```
   aws iam attach-role-policy \
     --role-name GreengrassV2TokenExchangeRole \
     --policy-arn
    arn:aws:iam::123456789012:policy/MyGreengrassV2ComponentArtifactPolicy
   ```

   Windows Command Prompt (CMD)

   ```
   aws iam attach-role-policy ^
     --role-name GreengrassV2TokenExchangeRole ^
     --policy-arn
    arn:aws:iam::123456789012:policy/MyGreengrassV2ComponentArtifactPolicy
   ```

   PowerShell

   ```
   aws iam attach-role-policy `
     --role-name GreengrassV2TokenExchangeRole `
     --policy-arn
    arn:aws:iam::123456789012:policy/MyGreengrassV2ComponentArtifactPolicy
   ```

   Se o comando não tiver saída, ele foi bem-sucedido e seu dispositivo principal poderá acessar os artefatos que você carrega nesse bucket do S3.

Política mínima de IAM para o instalador provisionar recursos

Ao instalar o software AWS IoT Greengrass Core, você pode provisionar AWS os recursos necessários, como uma AWS IoT coisa e uma função do IAM para seu dispositivo. Você também pode implantar ferramentas de desenvolvimento local no dispositivo. O instalador requer AWS

credenciais para poder realizar essas ações no seu Conta da AWS. Para ter mais informações, consulte Instalar o software do AWS IoT Greengrass Core.

O exemplo de política a seguir inclui o conjunto mínimo de ações que o instalador exige para provisionar esses recursos. Essas permissões são necessárias se você especificar o `--provision` argumento para o instalador. Substitua *`account-id`* pelo seu Conta da AWS ID e substitua *`GreengrassV2`* pelo nome da função de troca de tokens que você especifica TokenExchangeRole com o argumento do instalador. `--tes-role-name`

> **Note**
>
> A declaração `DeployDevTools` de política é necessária somente se você especificar o `--deploy-dev-tools` argumento para o instalador.

Greengrass nucleus v2.5.0 and later

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CreateTokenExchangeRole",
            "Effect": "Allow",
            "Action": [
                "iam:AttachRolePolicy",
                "iam:CreatePolicy",
                "iam:CreateRole",
                "iam:GetPolicy",
                "iam:GetRole",
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::account-id:role/GreengrassV2TokenExchangeRole",
                "arn:aws:iam::account-
id:policy/GreengrassV2TokenExchangeRoleAccess",
                "arn:aws:iam::aws:policy/GreengrassV2TokenExchangeRoleAccess"
            ]
        },
        {
            "Sid": "CreateIoTResources",
            "Effect": "Allow",
            "Action": [
```

```
                "iot:AddThingToThingGroup",
                "iot:AttachPolicy",
                "iot:AttachThingPrincipal",
                "iot:CreateKeysAndCertificate",
                "iot:CreatePolicy",
                "iot:CreateRoleAlias",
                "iot:CreateThing",
                "iot:CreateThingGroup",
                "iot:DescribeEndpoint",
                "iot:DescribeRoleAlias",
                "iot:DescribeThingGroup",
                "iot:GetPolicy"
            ],
            "Resource": "*"
        },
        {
            "Sid": "DeployDevTools",
            "Effect": "Allow",
            "Action": [
                "greengrass:CreateDeployment",
                "iot:CancelJob",
                "iot:CreateJob",
                "iot:DeleteThingShadow",
                "iot:DescribeJob",
                "iot:DescribeThing",
                "iot:DescribeThingGroup",
                "iot:GetThingShadow",
                "iot:UpdateJob",
                "iot:UpdateThingShadow"
            ],
            "Resource": "*"
        }
    ]
}
```

Earlier than v2.5.0

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CreateTokenExchangeRole",
            "Effect": "Allow",
```

```
            "Action": [
                "iam:AttachRolePolicy",
                "iam:CreatePolicy",
                "iam:CreateRole",
                "iam:GetPolicy",
                "iam:GetRole",
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::account-id:role/GreengrassV2TokenExchangeRole",
                "arn:aws:iam::account-
id:policy/GreengrassV2TokenExchangeRoleAccess",
                "arn:aws:iam::aws:policy/GreengrassV2TokenExchangeRoleAccess"
            ]
        },
        {
            "Sid": "CreateIoTResources",
            "Effect": "Allow",
            "Action": [
                "iot:AddThingToThingGroup",
                "iot:AttachPolicy",
                "iot:AttachThingPrincipal",
                "iot:CreateKeysAndCertificate",
                "iot:CreatePolicy",
                "iot:CreateRoleAlias",
                "iot:CreateThing",
                "iot:CreateThingGroup",
                "iot:DescribeEndpoint",
                "iot:DescribeRoleAlias",
                "iot:DescribeThingGroup",
                "iot:GetPolicy"
            ],
            "Resource": "*"
        },
        {
            "Sid": "DeployDevTools",
            "Effect": "Allow",
            "Action": [
                "greengrass:CreateDeployment",
                "iot:CancelJob",
                "iot:CreateJob",
                "iot:DeleteThingShadow",
                "iot:DescribeJob",
                "iot:DescribeThing",
```

```
                "iot:DescribeThingGroup",
                "iot:GetThingShadow",
                "iot:UpdateJob",
                "iot:UpdateThingShadow"
            ],
            "Resource": "*"
        }
    ]
}
```

Perfil de serviço do Greengrass

perfil de serviçoO perfil de serviço do Greengrass é um perfil de serviço do (IAM) AWS Identity and Access Management que autoriza o AWS IoT Greengrass a acessar recursos de serviços da AWS em seu nome. Essa função possibilita verificar AWS IoT Greengrass a identidade dos dispositivos clientes e gerenciar as principais informações de conectividade do dispositivo.

> **Note**
>
> AWS IoT Greengrass V1também usa essa função para realizar tarefas essenciais. Para obter mais informações, consulte a função de serviço do Greengrass no Guia do AWS IoT Greengrass V1Desenvolvedor.

Para permitir que o AWS IoT Greengrass acesse seus recursos, o perfil de serviço do Greengrass deve estar associado à sua Conta da AWS e você deve especificar o AWS IoT Greengrass como uma entidade confiável. A função deve incluir a política AWSGreengrassResourceAccessRolePolicygerenciada ou uma política personalizada que defina permissões equivalentes para os AWS IoT Greengrass recursos que você usa. AWSmantém essa política, que define o conjunto de permissões que você AWS IoT Greengrass usa para acessar seus AWS recursos. Para ter mais informações, consulte Política gerenciada da AWS: AWSGreengrassResourceAccessRolePolicy.

Você pode reutilizar a mesma função de serviço do Greengrass Regiões da AWS em todas as partes, mas deve associá-la à sua conta em Região da AWS todos os lugares em que usa. AWS IoT Greengrass Se a função de serviço não estiver configurada na atualRegião da AWS, os dispositivos principais falharão em verificar os dispositivos cliente e não atualizarão as informações de conectividade.

As seções a seguir descrevem como criar e gerenciar a função de serviço do Greengrass com o AWS Management Console ou. AWS CLI

Tópicos

> **Note**
>
> Além da função de serviço que autoriza o acesso em nível de serviço, você atribui uma função de troca de tokens aos dispositivos principais do Greengrass. A função de troca de tokens é uma função separada do IAM que controla como os componentes do Greengrass e as funções do Lambda no dispositivo principal podem acessar os serviços. AWS Para ter mais informações, consulte Autorize os dispositivos principais a interagir com os serviços AWS.

Gerenciar a função de serviço do Greengrass (console)

O console do AWS IoT facilita o gerenciamento do perfil de serviço do Greengrass. Por exemplo, quando você configura a descoberta do dispositivo cliente para um dispositivo principal, o console verifica se você Conta da AWS está conectado a uma função de serviço do Greengrass no momento. Região da AWS Caso contrário, o console pode criar e configurar um perfil de serviço para você. Para ter mais informações, consulte the section called “Criar o perfil de serviço do Greengrass”.

É possível usar o console do para as seguintes tarefas de gerenciamento de função:

Tópicos

> **Note**
>
> O usuário que está conectado no console deve ter permissões para visualizar, criar ou alterar o perfil de serviço.

Encontrar o perfil de serviço do Greengrass (console)

Use as etapas a seguir para encontrar a função de serviço AWS IoT Greengrass usada na atualRegião da AWS.

1. Navegue até o console do AWS IoT.
2. No painel de navegação, selecione Configurações.
3. Role até a seção Perfil de serviço do Greengrass para ver o perfil de serviço e as políticas dela.

   Se você não vê uma função de serviço, o console pode criar ou configurar uma para você. Para ter mais informações, consulte Criar o perfil de serviço do Greengrass.

Criar o perfil de serviço do Greengrass (console)

O console pode criar e configurar um perfil de serviço padrão do Greengrass para você. Essa função tem as propriedades a seguir.

Propriedade	Valor
Nome	`Greengrass_ServiceRole`
Entidade confiável	`AWS service: greengrass`
Política	AWSGreengrassResourceAccessRolePolicy

> **Note**
>
> Se você criar essa função com o script de configuração do AWS IoT Greengrass V1 dispositivo, o nome da função será`GreengrassServiceRole_`*`random-string`*.

Quando você configura a descoberta do dispositivo cliente para um dispositivo principal, o console verifica se uma função de serviço do Greengrass está associada à sua Conta da AWS no momento. Região da AWS Caso contrário, o console solicita sua permissão para que o AWS IoT Greengrass faça leitura e gravação em serviços AWS em seu nome.

Se você conceder permissão, o console verifica se uma função chamada `Greengrass_ServiceRole` existe na Conta da AWS.

- Se a função existir, o console anexará o perfil de serviço à Conta da AWS na Região da AWS atual.
- Se a função não existir, o console criará um perfil de serviço padrão do Greengrass e a anexará à Conta da AWS na Região da AWS atual.

> **Note**
>
> Se quiser criar um perfil de serviço com políticas de função personalizadas, use o console do IAM para criar ou modificar a função. Para obter mais informações, consulte Criando uma função para delegar permissões a um serviço da AWS ou Modificando uma função no Manual do usuário do IAM. Verifique se a função concede permissões equivalentes à política gerenciada `AWSGreengrassResourceAccessRolePolicy` para os atributos e as características que você utiliza. Recomendamos que você também inclua as chaves de contexto de condição global `aws:SourceArn` e `aws:SourceAccount` em sua política de confiança para ajudar a evitar o problema de segurança confused deputy. As chaves de contexto de condição restringem o acesso para permitir somente as solicitações provenientes da conta especificada e do espaço de trabalho do Greengrass. Para obter mais informações sobre o problema confused deputy, consulte Prevenção do problema do substituto confuso entre serviços.
>
> Se você criar uma função de serviço, retorne ao AWS IoT console e anexe a função à suaConta da AWS. Você pode fazer isso na função de serviço do Greengrass na página Configurações.

Alterar o perfil de serviço do Greengrass (console)

Use o procedimento a seguir para escolher outro perfil de serviço do Greengrass para anexar à Conta da AWS na Região da AWS que está selecionada no console.

1. Navegue até o console do AWS IoT.

2. No painel de navegação, selecione Configurações.
3. Em Perfil de serviço do Greengrass, selecione Change role (Mudar perfil).

   A caixa de diálogo Atualizar perfil de serviço do Greengrass é aberta e mostra as funções do IAM em sua Conta da AWS que definem AWS IoT Greengrass como uma entidade confiável.
4. Selecione o perfil de serviço do Greengrass a ser anexado.
5. Selecione Anexar função.

Desanexar o perfil de serviço do Greengrass (console)

Use o procedimento a seguir para separar a função de serviço do Greengrass da AWS sua conta atual. Região da AWS Isso revoga as permissões para que o AWS IoT Greengrass acesse os serviços da AWS na Região da AWS atual.

> **Important**
>
> Desanexar o perfil de serviço pode interromper operações ativas.

1. Navegue até o console do AWS IoT.
2. No painel de navegação, selecione Configurações.
3. Em Perfil de serviço do Greengrass, selecione Detach role (Desanexar função).
4. Na caixa de diálogo de confirmação, selecione Detach (Desvincular).

> **Note**
>
> Se você não precisar mais da função, poderá excluí-la no console do IAM. Para obter mais informações sobre como excluir uma função, consulte Excluir funções ou perfis de instância no Manual do usuário do IAM.
> Outras funções podem permitir que o AWS IoT Greengrass acesse os recursos. Para encontrar todas as funções que permitem que o AWS IoT Greengrass assuma permissões em seu nome, no console do IAM, na página Funções, procure as funções que incluem AWS service: greengrass na coluna Entidades confiáveis.

Gerenciar a função de serviço (CLI) do Greengrass

Nos procedimentos a seguir, presumimos que o AWS Command Line Interface esteja instalado e configurado para usar seuConta da AWS. Para obter mais informações, consulte Instalação, atualização e desinstalação do AWS CLI e Configuração do AWS CLI no Guia do AWS Command Line Interface Usuário.

É possível usar a AWS CLI para as seguintes tarefas de gerenciamento de função:

Tópicos

Obter o perfil de serviço do Greengrass (CLI)

Use o procedimento a seguir para descobrir se um perfil de serviço do Greengrass está associado à Conta da AWS em uma Região da AWS.

- Obtenha o perfil de serviço. Substitua *região* por sua Região da AWS (por exemplo, `us-west-2`).

  ```
  aws greengrassv2 get-service-role-for-account --region region
  ```

  Se uma função de serviço do Greengrass já estiver associada à sua conta, a solicitação retornará os seguintes metadados da função.

  ```
  {
    "associatedAt": "timestamp",
    "roleArn": "arn:aws:iam::account-id:role/path/role-name"
  }
  ```

  Se a solicitação não retornar metadados da função, você deverá criar a função de serviço (se ela não existir) e associá-la à sua conta noRegião da AWS.

Criar o perfil de serviço do Greengrass (CLI)

Use as etapas a seguir para criar uma função e associá-la à sua Conta da AWS.

Como criar o perfil de serviço usando o IAM

1. Crie uma função com uma política de confiança que permita que o AWS IoT Greengrass assuma a função. Este exemplo cria uma função chamada `Greengrass_ServiceRole`, mas você pode usar um nome diferente. Recomendamos que você também inclua as chaves de contexto de condição global `aws:SourceArn` e `aws:SourceAccount` em sua política de confiança para ajudar a evitar o problema de segurança confused deputy. As chaves de contexto de condição restringem o acesso para permitir somente as solicitações provenientes da conta especificada e do espaço de trabalho do Greengrass. Para obter mais informações sobre o problema confused deputy, consulte Prevenção do problema do substituto confuso entre serviços.

   Linux or Unix

   ```
   aws iam create-role --role-name Greengrass_ServiceRole --assume-role-policy-
   document '{
     "Version": "2012-10-17",
     "Statement": [
       {
         "Effect": "Allow",
         "Principal": {
           "Service": "greengrass.amazonaws.com"
         },
         "Action": "sts:AssumeRole",
         "Condition": {
           "ArnLike": {
             "aws:SourceArn": "arn:aws:greengrass:region:account-id:*"
           },
           "StringEquals": {
             "aws:SourceAccount": "account-id"
           }
         }
       }
     ]
   }'
   ```

   Windows Command Prompt (CMD)

   ```
   aws iam create-role --role-name Greengrass_ServiceRole --assume-role-
   policy-document "{\"Version\":\"2012-10-17\",\"Statement\":[{\"Effect
   \":\"Allow\",\"Principal\":{\"Service\":\"greengrass.amazonaws.com\"},
   \"Action\":\"sts:AssumeRole\",\"Condition\":{\"ArnLike\":{\"aws:SourceArn
   ```

```
\":\"arn:aws:greengrass:region:account-id:*\"},\"StringEquals\":
{\"aws:SourceAccount\":\"account-id\"}}}]}"
```

PowerShell

```
aws iam create-role --role-name Greengrass_ServiceRole --assume-role-policy-
document '{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "greengrass.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:greengrass:region:account-id:*"
        },
        "StringEquals": {
          "aws:SourceAccount": "account-id"
        }
      }
    }
  ]
}'
```

2. Copie o ARN da função dos metadados da função na saída. Você usará o ARN para associar a função à sua conta.
3. Anexe a política do `AWSGreengrassResourceAccessRolePolicy` à função.

```
aws iam attach-role-policy --role-name Greengrass_ServiceRole --policy-arn
 arn:aws:iam::aws:policy/service-role/AWSGreengrassResourceAccessRolePolicy
```

Para associar o perfil de serviço à sua conta da Conta da AWS

- Associe a função à sua conta. Substitua *`role-arn`* pelo ARN do perfil de serviço e *`região`* por sua Região da AWS (por exemplo, `us-west-2`).

```
aws greengrassv2 associate-service-role-to-account --role-arn role-arn --
region region
```

Se for bem-sucedida, a solicitação retornará a seguinte resposta.

```
{
  "associatedAt": "timestamp"
}
```

Remover o perfil de serviço do Greengrass (CLI)

Use as etapas a seguir para desassociar o perfil de serviço do Greengrass de sua Conta da AWS.

- Desassocie a perfil de serviço da conta. Substitua *região* por sua Região da AWS (por exemplo, `us-west-2`).

```
aws greengrassv2 disassociate-service-role-from-account --region region
```

Se houver êxito, a resposta a seguir será retornada.

```
{
  "disassociatedAt": "timestamp"
}
```

> **Note**
>
> Você deverá excluir o perfil de serviço se não o estiver usando em nenhuma Região da AWS. Primeiro, use delete-role-policy para desanexar a política gerenciada `AWSGreengrassResourceAccessRolePolicy` da função e, depois, use delete-role para excluir a função. Para obter mais informações sobre como excluir uma função, consulte Excluir funções ou perfis de instância no Manual do usuário do IAM.

Consulte também

- Criar um perfil para delegar permissões a um serviço da AWS no Guia do usuário do IAM.

- Modificando uma função no Guia do usuário do IAM
- Excluir funções ou perfis de instância no Manual do usuário do IAM.
- Comandos do AWS IoT Greengrass disponíveis na Referência de comandos do AWS CLI
  - associate-service-role-to-conta
  - disassociate-service-role-from-conta
  - get-service-role-for-conta
- Comandos do IAM disponíveis na Referência de comandos do AWS CLI
  - attach-role-policy
  - create-role
  - delete-role
  - delete-role-policy

Políticas gerenciadas pela AWS para o AWS IoT Greengrass

Uma política gerenciada pela AWS é uma política independente criada e administrada pela AWS. As políticas gerenciadas pela AWS são criadas para fornecer permissões a vários casos de uso comuns a fim de que você possa começar a atribuir permissões a usuários, grupos e perfis.

Lembre-se de que as políticas gerenciadas pela AWS podem não conceder permissões de privilégio mínimo para seus casos de uso específicos porque estão disponíveis para todos os clientes da AWS usarem. Recomendamos que você reduza ainda mais as permissões definindo políticas gerenciadas pelo cliente específicas para seus casos de uso.

Você não pode alterar as permissões definidas em políticas gerenciadas pela AWS. Se a AWS atualiza as permissões definidas em um política gerenciada pela AWS, a atualização afeta todas as identidades de entidades principais (usuários, grupos e perfis) às quais a política está vinculada. É mais provável que a AWS atualize uma política gerenciada pela AWS quando um novo AWS service (Serviço da AWS) é lançado ou novas operações de API são disponibilizadas para os serviços existentes.

Para obter mais informações, consulte Políticas gerenciadas pela AWS no Manual do usuário do IAM.

Tópicos

- Política gerenciada da AWS: AWSGreengrassFullAccess
- Política gerenciada da AWS: AWSGreengrassReadOnlyAccess

- Política gerenciada da AWS: AWSGreengrassResourceAccessRolePolicy
- Atualizações do AWS IoT Greengrass para políticas gerenciadas pela AWS

Política gerenciada da AWS: AWSGreengrassFullAccess

É possível anexar a política `AWSGreengrassFullAccess` a suas identidades do IAM.

Essa política concede permissões administrativas que permitem ao principal acesso total a todosAWS IoT Greengrassações.

Detalhes da permissão

Esta política inclui as seguintes permissões:

- `greengrass`— Permite que os diretores tenham acesso total a todosAWS IoT Greengrassações.

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

Política gerenciada da AWS: AWSGreengrassReadOnlyAccess

É possível anexar a política `AWSGreengrassReadOnlyAccess` a suas identidades do IAM.

Essa política concede permissões somente de leitura que permitem que um diretor visualize, mas não modifique, informações emAWS IoT Greengrass. Por exemplo, diretores com essas permissões podem ver a lista de componentes implantados em um dispositivo principal do Greengrass, mas não podem criar uma implantação para alterar os componentes que são executados nesse dispositivo.

Detalhes da permissão

Esta política inclui as seguintes permissões:

- `greengrass`— Permite que os diretores realizem ações que retornam uma lista de itens ou detalhes sobre um item. Isso inclui operações de API que começam com`List`ou`Get`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:List*",
                "greengrass:Get*"
            ],
            "Resource": "*"
        }
    ]
}
```

Política gerenciada da AWS: AWSGreengrassResourceAccessRolePolicy

Você pode anexar o`AWSGreengrassResourceAccessRolePolicy`política para suas entidades do IAM.AWS IoT Greengrasstambém anexa essa política a uma função de serviço que permiteAWS IoT Greengrasspara realizar ações em seu nome. Para obter mais informações, consulte Perfil de serviço do Greengrass.

Essa política concede permissões administrativas que permitemAWS IoT Greengrasspara realizar tarefas essenciais, como recuperar suas funções do Lambda, gerenciarAWS IoTsombras de dispositivos e verificação de dispositivos clientes do Greengrass.

Detalhes da permissão

Esta política inclui as seguintes permissões.

- `greengrass`— Gerencie os recursos do Greengrass.
- `iot(*Shadow)` — GerenciarAWS IoTsombras que têm os seguintes identificadores especiais em seus nomes. Essas permissões são necessárias para queAWS IoT Greengrasspode se comunicar com dispositivos principais.
  - `*-gci`—AWS IoT Greengrassusa essa sombra para armazenar as principais informações de conectividade do dispositivo, para que os dispositivos clientes possam descobrir e se conectar aos dispositivos principais.

  - `*-gcm`—AWS IoT Greengrass V1usa essa sombra para notificar o dispositivo principal de que o certificado de autoridade de certificação (CA) do grupo Greengrass foi alternado.
  - `*-gda`—AWS IoT Greengrass V1usa essa sombra para notificar o dispositivo principal sobre uma implantação.
  - `GG_*`— Não utilizado.
- `iot(DescribeThingeDescribeCertificate)` — Recuperar informações sobreAWS IoTcoisas e certificados. Essas permissões são necessárias para queAWS IoT Greengrasspode verificar dispositivos clientes que se conectam a um dispositivo principal. Para obter mais informações, consulte Interaja com dispositivos IoT locais.
- `lambda`— Recuperar informações sobreAWS Lambdafunções. Essa permissão é necessária para queAWS IoT Greengrass V1pode implantar funções do Lambda nos núcleos do Greengrass. Para obter mais informações, consulteExecute a função Lambda noAWS IoT Greengrassabdômennaabdômen AWS IoT Greengrass V1Guia do desenvolvedor.
- `secretsmanager`— Recupere o valor deAWS Secrets Managersegredos cujos nomes começam com`greengrass-`. Essa permissão é necessária para queAWS IoT Greengrass V1pode implantar segredos do Secrets Manager nos núcleos do Greengrass. Para obter mais informações, consulteImplante segredos noAWS IoT Greengrassabdômennaabdômen AWS IoT Greengrass V1Guia do desenvolvedor.
- `s3`— Recupere arquivos, objetos de buckets do S3 cujos nomes contenham`greengrassousagemaker`. Essas permissões são necessárias para queAWS IoT Greengrass V1pode implantar recursos de aprendizado de máquina que você armazena em buckets do S3. Para obter mais informações, consulteRecursos de aprendizado de máquinanaAWS IoT Greengrass V1Guia do desenvolvedor.
- `sagemaker`— Recuperar informações sobre a AmazonSageMakermodelos de inferência de aprendizado de máquina. Essa permissão é necessária para queAWS IoT Greengrass V1pode implantar modelos de ML nos núcleos do Greengrass. Para obter mais informações, consulteExecute inferência de aprendizado de máquinanaAWS IoT Greengrass V1Guia do desenvolvedor.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowGreengrassAccessToShadows",
            "Action": [
```

```
                "iot:DeleteThingShadow",
                "iot:GetThingShadow",
                "iot:UpdateThingShadow"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:iot:*:*:thing/GG_*",
                "arn:aws:iot:*:*:thing/*-gcm",
                "arn:aws:iot:*:*:thing/*-gda",
                "arn:aws:iot:*:*:thing/*-gci"
            ]
        },
        {
            "Sid": "AllowGreengrassToDescribeThings",
            "Action": [
                "iot:DescribeThing"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:iot:*:*:thing/*"
        },
        {
            "Sid": "AllowGreengrassToDescribeCertificates",
            "Action": [
                "iot:DescribeCertificate"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:iot:*:*:cert/*"
        },
        {
            "Sid": "AllowGreengrassToCallGreengrassServices",
            "Action": [
                "greengrass:*"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Sid": "AllowGreengrassToGetLambdaFunctions",
            "Action": [
                "lambda:GetFunction",
                "lambda:GetFunctionConfiguration"
            ],
            "Effect": "Allow",
            "Resource": "*"
```

```
        },
        {
            "Sid": "AllowGreengrassToGetGreengrassSecrets",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:secretsmanager:*:*:secret:greengrass-*"
        },
        {
            "Sid": "AllowGreengrassAccessToS3Objects",
            "Action": [
                "s3:GetObject"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:s3:::*Greengrass*",
                "arn:aws:s3:::*GreenGrass*",
                "arn:aws:s3:::*greengrass*",
                "arn:aws:s3:::*Sagemaker*",
                "arn:aws:s3:::*SageMaker*",
                "arn:aws:s3:::*sagemaker*"
            ]
        },
        {
            "Sid": "AllowGreengrassAccessToS3BucketLocation",
            "Action": [
                "s3:GetBucketLocation"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Sid": "AllowGreengrassAccessToSageMakerTrainingJobs",
            "Action": [
                "sagemaker:DescribeTrainingJob"
            ],
            "Effect": "Allow",
            "Resource": [
                "arn:aws:sagemaker:*:*:training-job/*"
            ]
        }
    ]
```

```
}
```

Atualizações do AWS IoT Greengrass para políticas gerenciadas pela AWS

Você pode ver detalhes sobre as atualizações doAWSpolíticas gerenciadas paraAWS IoT Greengrassa partir do momento em que esse serviço começou a rastrear essas mudanças. Para receber alertas automáticos sobre alterações nesta página, assine o feed RSS noAWS IoT Greengrass V2página de histórico do documento.

Alteração	Descrição	Data
O AWS IoT Greengrass iniciou o rastreamento das alterações	O AWS IoT Greengrass começou a monitorar as alterações para as políticas gerenciadas da AWS.	2 de julho de 2021

Prevenção do problema do substituto confuso entre serviços

O problema de "confused deputy" é uma questão de segurança em que uma entidade que não tem permissão para executar uma ação pode coagir uma entidade mais privilegiada a executá-la. Em AWS, a personificação entre serviços pode resultar no problema do ‘confused deputy’. A personificação entre serviços pode ocorrer quando um serviço (o serviço de chamada) chama outro serviço (o serviço chamado). O serviço de chamada pode ser manipulado de modo a usar suas permissões para atuar nos recursos de outro cliente de uma forma na qual ele não deveria ter permissão para acessar. Para evitar isso, a AWS fornece ferramentas que ajudam você a proteger seus dados para todos os serviços com entidades principais de serviço que receberam acesso aos recursos em sua conta.

Recomendamos o uso das chaves de contexto de condição global `aws:SourceArn` e `aws:SourceAccount` em políticas de recursos para limitar as AWS IoT Greengrasspermissões que o concede a outro serviço no recurso para o recurso. Se você utilizar ambas as chaves de contexto de condição global, o valor `aws:SourceAccount` e a conta `aws:SourceArn` no valor deverão utilizar o mesmo ID de conta quando utilizados na mesma instrução de política.

O valor de`aws:SourceArn`deve ser o recurso do cliente Greengrass associado a`sts:AssumeRole`solicitação.

A maneira mais eficaz de se proteger do problema 'confused deputy' é usar a chave de contexto de condição global `aws:SourceArn` com o ARN completo do recurso. Se você não souber o ARN completo do recurso ou se estiver especificando vários recursos, use a chave de condição de contexto global `aws:SourceArn` com curingas (*) para as partes desconhecidas do ARN. Por exemplo, `arn:aws:greengrass::account-id:*`.

Para obter um exemplo de uma política que usa o`aws:SourceArn`e`aws:SourceAccount`chaves de contexto de condição globais, consulteCriar o perfil de serviço do Greengrass.

Solução de problemas de identidade e acesso do AWS IoT Greengrass

Use as seguintes informações para ajudar a diagnosticar e corrigir problemas comuns que podem ser encontrados ao trabalhar com o AWS IoT Greengrass e o IAM.

Problemas

- Não estou autorizado a realizar uma ação no AWS IoT Greengrass
- Não estou autorizado a executar iam:PassRole
- Sou administrador e desejo permitir que outras pessoas tenham acesso ao AWS IoT Greengrass
- Quero permitir que as pessoas fora da minha Conta da AWS acessem meus recursos do AWS IoT Greengrass

Para obter ajuda geral com a solução de problemas, consulte Solução de problemas.

Não estou autorizado a realizar uma ação no AWS IoT Greengrass

Se você receber uma mensagem de erro informando que você não está autorizado a executar a ação, entre em contato com o administrador para obter assistência. O administrador é a pessoa que forneceu a você o seu nome de usuário e senha.

O exemplo de erro a seguir ocorre quando o`mateojackson`O usuário do IAM tenta visualizar detalhes sobre um dispositivo principal, mas não tem`greengrass:GetCoreDevice`Permissões.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to
 perform: greengrass:GetCoreDevice on resource: arn:aws:greengrass:us-
west-2:123456789012:coreDevices/MyGreengrassCore
```

Neste caso, Mateo pede ao administrador para atualizar suas políticas para permitir a ele o acesso ao recurso `arn:aws:greengrass:us-west-2:123456789012:coreDevices/MyGreengrassCore` usando a ação `greengrass:GetCoreDevice`.

A seguir estão os problemas gerais do IAM que você venha a encontrar ao trabalhar com oAWS IoT Greengrass.

Não estou autorizado a executar iam:PassRole

Se você receber uma mensagem de erro informando que você não está autorizado a executar `oiam:PassRole`ação, suas políticas devem ser atualizadas para permitir que você transmita uma função paraAWS IoT Greengrass.

Alguns Serviços da AWS permitem que você transmita um perfil existente para o serviço, em vez de criar um perfil de serviço ou um perfil vinculado ao serviço. Para fazer isso, um usuário deve ter permissões para passar o perfil para o serviço.

O erro de exemplo a seguir ocorre quando uma usuária do IAM chamada `marymajor` tenta usar o console para executar uma ação no AWS IoT Greengrass. No entanto, a ação exige que o serviço tenha permissões concedidas por um perfil de serviço. Mary não tem permissões para passar a função para o serviço.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

Nesse caso, as políticas de Mary devem ser atualizadas para permitir que ela realize a ação `iam:PassRole`.

Se você precisar de ajuda, entre em contato com seu administrador da AWS. Seu administrador é a pessoa que forneceu a você suas credenciais de login.

Sou administrador e desejo permitir que outras pessoas tenham acesso ao AWS IoT Greengrass

Para permitir que outros usuários acessem o AWS IoT Greengrass, crie uma entidade do IAM (usuário ou função) para a pessoa ou a aplicação que precisa do acesso. Elas usarão as credenciais dessa entidade para acessar a AWS. Você deve anexar uma política à entidade que concede a eles as permissões corretas no AWS IoT Greengrass.

Para começar a usar imediatamente, consulte Criar os primeiros usuário e grupo delegados do IAM no Guia do usuário do IAM.

Quero permitir que as pessoas fora da minha Conta da AWS acessem meus recursos do AWS IoT Greengrass

Você pode criar uma função do IAM que os usuários de outras contas ou pessoas fora da sua organização possam usar para acessar oAWSRecursos. Você pode especificar quem é confiável para assumir a função. Para obter mais informações, consulteFornecimento de acesso a um usuário do IAM em outroConta da AWSque você possuieFornecimento de acesso aConta da AWSé de propriedade de terceirosnaManual do usuário do IAM.

O AWS IoT Greengrass não oferece suporte ao acesso entre contas com base em políticas baseadas em recursos ou listas de controle de acesso (ACLs).

Permitir o tráfego de dispositivos por meio de um proxy ou firewall

Os principais dispositivos e componentes do Greengrass realizam solicitações externas para AWS serviços e outros sites. Como medida de segurança, você pode limitar o tráfego de saída a uma pequena variedade de terminais e portas. Você pode usar as seguintes informações sobre endpoints e portas para limitar o tráfego de dispositivos por meio de um proxy, firewall ou grupo de segurança da Amazon VPC. Para obter mais informações sobre como configurar um dispositivo principal para usar um proxy, consulteConectar-se à porta 443 ou por meio de um proxy de rede.

Tópicos

- Endpoints para operação básica
- Endpoints para instalação com provisionamento automático
- Endpoints para componentes AWS fornecidos

Endpoints para operação básica

Os dispositivos principais do Greengrass usam os seguintes endpoints e portas para operação básica.

Recupere endpoints AWS IoT

Obtenha os AWS IoT endpoints para você Conta da AWS e salve-os para usar mais tarde. Seu dispositivo usa esses endpoints para se conectar a. AWS IoT Faça o seguinte:

1. Obtenha o endpoint de AWS IoT dados para vocêConta da AWS.

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

A resposta será semelhante ao exemplo a seguir, se a solicitação for bem-sucedida.

```
{
  "endpointAddress": "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
}
```

2. Obtenha o endpoint AWS IoT de credenciais para seu. Conta da AWS

```
aws iot describe-endpoint --endpoint-type iot:CredentialProvider
```

A resposta será semelhante ao exemplo a seguir, se a solicitação for bem-sucedida.

```
{
  "endpointAddress": "device-credentials-prefix.credentials.iot.us-
west-2.amazonaws.com"
}
```

Endpoint	Porta	Obrigatório	Descrição
greengrass-ats.iot. *region*.amazonaws.com	8443 ou 443	Sim	Usado para operações de plano de dados, como instalar implantações e trabalhar com dispositivos clientes.

Endpoint	Porta	Obrigatório	Descrição
device-data-prefix -ats.iot.*region*.amazonaws.com	MQTT: 8883 ou 443 HTTPS: 8443 ou 443	Sim	Usado para operações de plano de dados para gerenciamento de dispositivos, como comunicação MQTT e sincronização de sombra comAWS IoT Core.

Endpoint	Porta	Obrigatório	Descrição
device-credentials-prefix .credentials.iot. *region*.amazonaws.com	443	Sim	Usado para adquirir AWS credenciais, que o dispositivo principal usa para baixar artefatos de componentes do Amazon S3 e realizar outras operações. Para ter mais informações, consulte Autorize os dispositivos principais a interagir com os serviços AWS.

Endpoint	Porta	Obrigatório	Descrição

`*.s3.amazonaws.com`
 	
`*.s3.`*`region`*`.amazonaws.com`
	443	Sim	Usado para implantações. Esse formato inclui o * caractere, porque os prefixos de endpoint são controlados internamente e podem mudar a qualquer momento.

Endpoint	Porta	Obrigatório	Descrição
`data.iot.` *`region`*`.amazonaws.com`	443	Não	Obrigatório se o dispositivo principal executar uma versão do núcleo do Greengrass anterior à v2.4.0 e estiver configurado para usar um proxy de rede. O dispositivo principal usa esse endpoint para comunicação MQTT AWS IoT Core quando está atrás de um proxy. Para ter mais informações, consulte

Endpoint	Porta	Obrigatório	Descrição
			Configurar um proxy de rede.

Endpoints para instalação com provisionamento automático

Os dispositivos principais do Greengrass usam os seguintes endpoints e portas quando você instala o software AWS IoT Greengrass Core com provisionamento automático de recursos.

Endpoint	Porta	Obrigatório	Descrição
`iot.region.amazonaws.com`	443	Sim	Usado para criar AWS IoT recursos e recuperar informações sobre AWS IoT os recursos existentes.
`iam.amazonaws.com`	443	Sim	Usado para criar recursos do IAM e recuperar informações sobre os recursos existentes do IAM.

Endpoint	Porta	Obrigatório	Descrição
`sts.region.amazonaws.com`	443	Sim	Usado para obter a identificação do seuConta da AWS.
`greengrass. region.amazonaws.com`	443	Não	Obrigatório se você usar o `--deploy-dev-tools` argumento para implantar o componente CLI do Greengrass no dispositivo principal.

Endpoints para componentes AWS fornecidos

Os dispositivos principais do Greengrass usam endpoints adicionais, dependendo de quais componentes de software eles executam. Você pode encontrar os endpoints que cada componente AWS fornecido exige na seção Requisitos da página de cada componente neste guia do desenvolvedor. Para ter mais informações, consulte AWS-componentes fornecidos.

Validação de conformidade para AWS IoT Greengrass

Para saber se um AWS service (Serviço da AWS) está dentro do escopo de programas de conformidade específicos, consulte Serviços da AWS Escopo por Programa de Conformidade

Serviços da AWS e escolha o programa de conformidade em que você está interessado. Para obter informações gerais, consulte Programas de AWS conformidade Programas AWS de .

Você pode baixar relatórios de auditoria de terceiros usando AWS Artifact. Para obter mais informações, consulte Baixar relatórios em AWS Artifact .

Sua responsabilidade de conformidade ao usar Serviços da AWS é determinada pela confidencialidade de seus dados, pelos objetivos de conformidade de sua empresa e pelas leis e regulamentações aplicáveis. AWS fornece os seguintes recursos para ajudar na conformidade:

- Guias de início rápido sobre segurança e conformidade — Esses guias de implantação discutem considerações arquitetônicas e fornecem etapas para a implantação de ambientes básicos AWS focados em segurança e conformidade.
- Arquitetura para segurança e conformidade com a HIPAA na Amazon Web Services — Este whitepaper descreve como as empresas podem usar AWS para criar aplicativos qualificados para a HIPAA.

  > **Note**
  >
  > Nem todos Serviços da AWS são elegíveis para a HIPAA. Para obter mais informações, consulte Referência dos Serviços Qualificados pela HIPAA.

- AWS Recursos de https://aws.amazon.com/compliance/resources/ de conformidade — Essa coleção de pastas de trabalho e guias pode ser aplicada ao seu setor e local.
- AWS Guias de conformidade do cliente — Entenda o modelo de responsabilidade compartilhada sob a ótica da conformidade. Os guias resumem as melhores práticas de proteção Serviços da AWS e mapeiam as diretrizes para controles de segurança em várias estruturas (incluindo o Instituto Nacional de Padrões e Tecnologia (NIST), o Conselho de Padrões de Segurança do Setor de Cartões de Pagamento (PCI) e a Organização Internacional de Padronização (ISO)).
- Avaliação de recursos com regras no Guia do AWS Config desenvolvedor — O AWS Config serviço avalia o quão bem suas configurações de recursos estão em conformidade com as práticas internas, as diretrizes e os regulamentos do setor.
- AWS Security Hub— Isso AWS service (Serviço da AWS) fornece uma visão abrangente do seu estado de segurança interno AWS. O Security Hub usa controles de segurança para avaliar os atributos da AWS e verificar a conformidade com os padrões e as práticas recomendadas do setor de segurança. Para obter uma lista dos serviços com suporte e controles aceitos, consulte a Referência de controles do Security Hub.

- Amazon GuardDuty — Isso AWS service (Serviço da AWS) detecta possíveis ameaças às suas cargas de trabalho Contas da AWS, contêineres e dados monitorando seu ambiente em busca de atividades suspeitas e maliciosas. GuardDuty pode ajudá-lo a atender a vários requisitos de conformidade, como o PCI DSS, atendendo aos requisitos de detecção de intrusões exigidos por determinadas estruturas de conformidade.
- AWS Audit Manager— Isso AWS service (Serviço da AWS) ajuda você a auditar continuamente seu AWS uso para simplificar a forma como você gerencia o risco e a conformidade com as regulamentações e os padrões do setor.

Resiliência no AWS IoT Greengrass

OAWSA infraestrutura global da é criada com base em regiões da Amazon Web Services e zonas de disponibilidade. EACHRegião da AWSAs fornecem várias zonas de disponibilidade separadas e isoladas fisicamente, que são conectadas com baixa latência, altas taxas de transferência e redes altamente redundantes. Com as zonas de disponibilidade, é possível projetar e operar aplicações e bancos de dados que automaticamente executam o failover entre as zonas sem interrupção. As zonas de disponibilidade são mais altamente disponíveis, tolerantes a falhas e escaláveis que uma ou várias infraestruturas de data center tradicionais.

Para obter mais informações, consulte Infraestrutura global da AWS.

Além da infraestrutura global da AWS, o AWS IoT Greengrass oferece vários recursos para ajudar a oferecer suporte às suas necessidades de resiliência de dados e backup.

- Você pode configurar um dispositivo de núcleo do Greengrass para gravar logs no sistema de arquivos local e no CloudWatch Logs. Se o dispositivo de núcleo perder a conectividade, ele poderá continuar registrando mensagens no sistema de arquivos. Quando ele se reconecta, ele grava as mensagens de log em CloudWatch Logs. Para obter mais informações, consulteMonitore AWS IoT Greengrass os registros
- Se um dispositivo principal perder energia durante uma implantação, ele retomará a implantação após oAWS IoT GreengrassO software principal é iniciado novamente.
- Se um dispositivo de núcleo perder a conectividade com a Internet, os dispositivos do cliente do Greengrass poderão continuar a comunicação por meio da rede local.
- Você pode criar componentes do Greengrass que lêemGerenciador de streamOs fluxos e enviam os dados para destinos de armazenamento local.

Segurança da infraestrutura no AWS IoT Greengrass

Como um serviço gerenciado, o AWS IoT Greengrass é protegido pelos procedimentos de segurança de rede global da AWS que estão descritos no whitepaper Amazon Web Services: Overview of Security Processes.

Você usa chamadas de API publicadas pela AWS para acessar o AWS IoT Greengrass por meio da rede. Os clientes devem oferecer suporte a Transport Layer Security (TLS) 1.2 ou posterior. Recomendamos usar o TLS 1.3 ou posterior. Os clientes também devem oferecer suporte a pacotes de criptografia com Perfect Forward Secrecy (PFS — Sigilo de encaminhamento perfeito), como Ephemeral Diffie-Hellman (DHE) ou Ephemeral Elliptic Curve Diffie-Hellman (ECDHE). A maioria dos sistemas modernos, como Java 7 e versões posteriores, comporta esses modos.

As solicitações devem ser assinadas usando um ID da chave de acesso e uma chave de acesso secreta associada a uma entidade principal do IAM. Ou é possível usar o AWS Security Token Service (AWS STS) para gerar credenciais de segurança temporárias para assinar solicitações.

Em um AWS IoT Greengrass ambiente, os dispositivos usam certificados X.509 e chaves criptográficas para se conectar e se autenticar no. Nuvem AWS Para obter mais informações, consulte the section called “Autorização e autenticação do dispositivo”.

Análise de vulnerabilidade e configuração no AWS IoT Greengrass

Ambientes de IoT consistem em grandes quantidades de dispositivos com capacidades diversas, duradouros e geograficamente distribuídos. Essas características tornam a configuração do dispositivo complexa e propensa a erros. E como os dispositivos, quase sempre, têm restrições quanto à capacidade computacional e aos recursos de memória e armazenamento, isso limita o uso de criptografia e outras formas de segurança nos próprios dispositivos. Além disso, muitas vezes, os dispositivos usam software com vulnerabilidades conhecidas. Esses fatores tornam os dispositivos de IoT um alvo atrativo para hackers e tornam difícil protegê-los de forma contínua.

O AWS IoT Device Defender aborda esses desafios fornecendo ferramentas para identificar problemas de segurança e desvios das melhores práticas. Você pode usar o AWS IoT Device Defender para analisar, auditar e monitorar dispositivos conectados a fim de detectar comportamentos anormais e reduzir os riscos de segurança. O AWS IoT Device Defender pode auditar dispositivos para garantir a conformidade com as melhores práticas de segurança e detectar comportamentos anormais em dispositivos. Isso possibilita a aplicação de políticas de

segurança consistentes em todos os dispositivos e responder rapidamente quando eles estiverem comprometidos. Para obter mais informações, consulte os tópicos a seguir:

- OComponente Device Defender do
- AWS IoT Device Defender no Guia do desenvolvedor do AWS IoT Core.

Em ambientes AWS IoT Greengrass, você deve estar ciente das seguintes considerações:

- É sua responsabilidade proteger seus dispositivos físicos, o sistema de arquivos em seus dispositivos e a rede local.
- AWS IoT GreengrassO não impõe isolamento de rede para componentes do Greengrass do definidos pelo usuário, sejam eles executados ou não em um contêiner do Greengrass do. Portanto, é possível que os componentes do Greengrass se comuniquem com qualquer outro processo em execução no sistema ou fora da rede.

Integridade do código emAWS IoT Greengrass V2

AWS IoT Greengrassimplanta componentes de software a partir doNuvem AWSpara dispositivos que executam oAWS IoT GreengrassSoftware do núcleo do. Esses componentes de software incluemAWScomponentes fornecidosecomponentes personalizadosque você carrega para o seuConta da AWS. Cada componente é composto por uma receita. A receita define os metadados do componente e qualquer número de artefatos, que são binários de componentes, como código compilado e recursos estáticos. Os artefatos de componentes são armazenados no Amazon S3.

À medida que você desenvolve e implanta componentes do Greengrass, você segue estas etapas básicas que funcionam com artefatos de componentes em seuConta da AWSe em seus dispositivos:

1. Crie e carregue artefatos para buckets do S3.
2. Crie um componente a partir de uma receita e artefatos naAWS IoT Greengrassserviço, que calcula umhash de criptografiade cada artefato.
3. Implante um componente nos dispositivos principais do Greengrass, que baixam e verificam a integridade de cada artefato.

AWSé responsável por manter a integridade dos artefatos depois de carregar artefatos para buckets do S3, inclusive quando você implanta componentes em dispositivos principais do Greengrass. Você é responsável por proteger artefatos de software antes de carregar os artefatos para buckets do S3.

Você também é responsável por garantir o acesso aos recursos em seuConta da AWS, incluindo os buckets do S3 nos quais você carrega artefatos de componentes.

> **Note**
>
> O Amazon S3 fornece um recurso chamado S3 Object Lock que você pode usar para proteger contra alterações em artefatos de componentes em buckets do S3Conta da AWS. Você pode usar o bloqueio de objetos do S3 para evitar que artefatos de componentes sejam excluídos ou substituídos. Para obter mais informações, consulteUsar o S3 Object LocknoGuia do usuário do Amazon Simple Storage Service.

QuandoAWSpublica um componente público e, quando você carrega um componente personalizado,AWS IoT Greengrasscalcula um resumo criptográfico para cada artefato de componente.AWS IoT Greengrassatualiza a receita do componente para incluir o resumo de cada artefato e o algoritmo de hash usado para calcular esse resumo. Este resumo garante a integridade do artefato, porque se o artefato mudar noNuvem AWSou durante o download, seu resumo de arquivos não corresponderá ao resumo queAWS IoT Greengrasslojas na receita do componente. Para obter mais informações, consulteArtefatos na referência da receita do componente.

Quando você implanta um componente em um dispositivo principal, oAWS IoT GreengrassO software principal baixa a receita do componente e cada artefato de componente que a receita define. OAWS IoT GreengrassO software principal calcula o resumo de cada arquivo de artefato baixado e o compara com o resumo desse artefato na receita. Se os resumos não forem correspondentes, a implantação vai falhar e oAWS IoT GreengrassO software principal exclui os artefatos baixados do sistema de arquivos do dispositivo. Para obter mais informações sobre como conexões entre dispositivos principais eAWS IoT Greengrassestão protegidos, consulteCriptografia em trânsito.

Você é responsável por proteger arquivos de artefatos de componentes nos sistemas de arquivos de seus dispositivos principais. OAWS IoT GreengrassO software principal salva artefatos no`packages`pasta na pasta raiz Greengrass. Você pode usarAWS IoT Device Defenderpara analisar, auditar e monitorar dispositivos principais. Para obter mais informações, consulte Análise de vulnerabilidade e configuração no AWS IoT Greengrass.

AWS IoT Greengrass e endpoint da VPC de interface (AWS PrivateLink)

É possível estabelecer uma conexão privada entre a VPC e o ambiente de gerenciamento AWS IoT Greengrass criando um endpoint da VPC de interface. Você pode usar esse endpoint para gerenciar componentes, implantações e dispositivos principais no AWS IoT Greengrass serviço. Os endpoints de interface são habilitados por AWS PrivateLink, uma tecnologia que permite acessar as APIs do AWS IoT Greengrass de forma privada sem um gateway da Internet, um dispositivo NAT, uma conexão VPN ou uma conexão do AWS Direct Connect. As instâncias na VPC não precisam de endereços IP públicos para a comunicação com APIs do AWS IoT Greengrass. O tráfego de rede entre a VPC e o AWS IoT Greengrass não deixa a rede da Amazon.

Cada endpoint de interface é representado por uma ou mais Interfaces de Rede Elástica nas sub-redes.

Para obter mais informações, consulte Endpoints da VPC da interface (AWS PrivateLink) no Manual do Usuário do Amazon VPC.

Tópicos

- Considerações sobre endpoints da VPC do AWS IoT Greengrass
- Criar um endpoint da VPC de interface para operações AWS IoT Greengrass do ambiente de gerenciamento
- Criar uma política de endpoint da VPC para o AWS IoT Greengrass
- Opere um dispositivo AWS IoT Greengrass principal na VPC

Considerações sobre endpoints da VPC do AWS IoT Greengrass

Antes de configurar um endpoint da VPC de interface para o AWS IoT Greengrass, analise as Propriedades e limitações de endpoints de interface no Manual do usuário da Amazon VPC. Além disso, esteja ciente das seguintes considerações:

- O AWS IoT Greengrass oferece suporte a chamadas para todas as ações de API de ambiente de gerenciamento da VPC. O plano de controle inclui operações como CreateDeploymentListEffectiveDeploymentse. O plano de controle não inclui operações como ResolveComponentCandidatese Discover, que são operações do plano de dados.
- Não há suporte para endpoints da VPC para AWS IoT Greengrass nas Regiões da China AWS.

Criar um endpoint da VPC de interface para operações AWS IoT Greengrass do ambiente de gerenciamento

É possível criar um endpoint da VPC para o ambiente de gerenciamento AWS IoT Greengrass usando o console da Amazon VPC ou a AWS Command Line Interface (AWS CLI). Para obter mais informações, consulte Criar um endpoint de interface no Guia do Usuário do Amazon VPC.

Crie um endpoint da VPC para o AWS IoT Greengrass usando o seguinte nome de serviço:

- com.amazonaws.*region*.greengrass

Se você habilitar o DNS privado para o endpoint, poderá fazer solicitações de API para o AWS IoT Greengrass usando seu nome DNS padrão para a região, por exemplo, `greengrass.us-east-1.amazonaws.com`. O DNS privado é habilitado por padrão.

Para obter mais informações, consulte Acessar um serviço por um endpoint de interface no Manual do Usuário do Amazon VPC.

Criar uma política de endpoint da VPC para o AWS IoT Greengrass

É possível anexar uma política de endpoint ao endpoint da VPC que controla o acesso às operações de ambiente de gerenciamento AWS IoT Greengrass. Essa política especifica as seguintes informações:

- A entidade principal que pode executar ações.
- As ações que o principal pode executar.
- Os recursos nos quais a entidade principal pode executar ações.

Para obter mais informações, consulte Controlar o acesso a serviços com endpoint da VPC no Guia do usuário da Amazon VPC.

Example Exemplo: política de endpoint da VPC para ações do AWS IoT Greengrass

Veja a seguir um exemplo de uma política de endpoint para o AWS IoT Greengrass. Quando anexada a um endpoint, essa política concede acesso às ações indicadas do AWS IoT Greengrass para todos os principais em todos os recursos.

```
{
```

```
    "Statement": [
        {
            "Principal": "*",
            "Effect": "Allow",
            "Action": [
                "greengrass:CreateDeployment",
                "greengrass:ListEffectiveDeployments"
            ],
            "Resource": "*"
        }
    ]
}
```

Opere um dispositivo AWS IoT Greengrass principal na VPC

Você pode operar um dispositivo principal do Greengrass e realizar implantações em VPC sem acesso público à Internet. No mínimo, você deve configurar os seguintes VPC endpoints com os aliases de DNS correspondentes. Para obter mais informações sobre como criar e usar VPC endpoints, consulte Criar um VPC endpoint no Guia do usuário da Amazon VPC.

> **Note**
>
> O recurso VPC para criar automaticamente um registro DNS está desativado para AWS IoT data e Credentials. AWS IoT Para conectar esses endpoints, você deve criar manualmente um registro DNS privado. Para obter mais informações, consulte DNS privado para endpoints de interface. Para obter mais informações sobre as limitações da AWS IoT Core VPC, consulte Limitações dos endpoints da VPC.

Pré-requisitos

- Você deve instalar o software AWS IoT Greengrass Core usando as etapas de provisionamento manual. Para ter mais informações, consulte Instale o software AWS IoT Greengrass principal com provisionamento manual de recursos.

Limitações

- A operação de um dispositivo central do Greengrass em VPC não é suportada nas regiões da China e. AWS GovCloud (US) Regions

- Para obter mais informações sobre limitações AWS IoT data e fornecedores de AWS IoT credenciais VPC endpoints, consulte Limitações.

Configure seu dispositivo principal do Greengrass para operar em VPC

1. Obtenha os AWS IoT endpoints para você Conta da AWS e salve-os para usar mais tarde. Seu dispositivo usa esses endpoints para se conectar a. AWS IoT Faça o seguinte:

   a. Obtenha o endpoint de AWS IoT dados para vocêConta da AWS.

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

   A resposta será semelhante ao exemplo a seguir, se a solicitação for bem-sucedida.

   ```
   {
     "endpointAddress": "device-data-prefix-ats.iot.us-west-2.amazonaws.com"
   }
   ```

   b. Obtenha o endpoint AWS IoT de credenciais para seu. Conta da AWS

   ```
   aws iot describe-endpoint --endpoint-type iot:CredentialProvider
   ```

   A resposta será semelhante ao exemplo a seguir, se a solicitação for bem-sucedida.

   ```
   {
     "endpointAddress": "device-credentials-prefix.credentials.iot.us-
   west-2.amazonaws.com"
   }
   ```

2. Crie uma interface Amazon VPC para endpoints AWS IoT data e AWS IoT credencie:

   a. Navegue até o console VPC Endpoints, em Nuvem privada virtual no menu à esquerda, escolha Endpoints e, em seguida, Criar endpoint.

   b. Na página Criar endpoint, especifique as seguintes informações.

   - Escolha AWS service (Serviço da AWS)s para a Categoria de serviço.
   - Para Nome do serviço, pesquise inserindo a palavra-chave `iot`. Na lista de serviços do `iot` exibida, escolha o endpoint.

Se você criar um endpoint da VPC para o plano de dados do AWS IoT Core, escolha o endpoint da API do plano de dados do AWS IoT Core para sua região. O endpoint será do formato `com.amazonaws.region.iot.data`.

Se você criar um endpoint da VPC para o provedor de credenciais do AWS IoT Core, escolha o endpoint do provedor de credenciais do AWS IoT Core para sua região. O endpoint será do formato `com.amazonaws.region.iot.credentials`.

> **Note**
>
> O nome do serviço para o plano de dados do AWS IoT Core na região da China terá o formato `cn.com.amazonaws.region.iot.data`. A criação de endpoints da VPC para o provedor de credenciais do AWS IoT Core não é compatível na região da China.

- Para VPC e sub-redes, escolha a VPC em que deseja criar o endpoint e as zonas de disponibilidade (AZs) nas quais deseja criar a rede do endpoint.
- Em Ativar nome DNS, certifique-se de que a opção Ativar para este endpoint não esteja selecionada. Nem o plano de dados do AWS IoT Core nem o provedor de credenciais do AWS IoT Core são compatíveis com nomes DNS privados ainda.
- Em Grupo de segurança, selecione os grupos de segurança a serem associados às interfaces de rede do endpoint.
- Se quiser, adicione ou remova tags. As tags são pares de nome-valor usados para associar ao seu endpoint.

c. Para criar um endpoint da VPC, selecione Criar endpoint.

3. Depois de criar o endpoint AWS PrivateLink, na guia Detalhes do endpoint, você verá uma lista de nomes DNS. Você pode usar um desses nomes DNS criados nesta seção para configurar a zona hospedada privada.
4. Crie um endpoint do Amazon S3. Para obter mais informações, consulte Criar um VPC endpoint para o Amazon S3.
5. Se você estiver usando componentes AWSfornecidos pelo Greengrass, endpoints e configurações adicionais podem ser necessários. Para visualizar os requisitos dos endpoints, selecione o componente na lista AWS de componentes fornecidos e consulte a seção Requisitos. Por exemplo, os requisitos do componente do gerenciador de registros

recomendam que esse componente seja capaz de realizar solicitações de saída para o endpoint`logs.`*`region`*`.amazonaws.com`.

Se você estiver usando seu próprio componente, talvez seja necessário revisar as dependências e realizar testes adicionais para determinar se algum endpoint adicional é necessário.

6. Na configuração do núcleo do Greengrass, `greengrassDataPlaneEndpoint` deve ser definido como. **`iotdata`** Para obter mais informações, consulte Configuração do núcleo do Greengrass.
7. Se você estiver na `us-east-1` região, defina o parâmetro de configuração como `s3EndpointType` **`REGIONAL`** na configuração do núcleo do Greengrass. Esse recurso está disponível para as versões 2.11.3 ou posteriores do Greengrass nucleus.

Example Exemplo: configuração de componentes

```
{
"aws.greengrass.Nucleus": {
   "configuration": {
      "awsRegion": "us-east-1",
      "iotCredEndpoint": "xxxxxx.credentials.iot.region.amazonaws.com",
      "iotDataEndpoint": "xxxxxx-ats.iot.region.amazonaws.com",
      "greengrassDataPlaneEndpoint": "iotdata",
      "s3EndpointType": "REGIONAL"
      ...
     }
   }
}
```

A tabela a seguir fornece informações sobre os aliases DNS privados personalizados correspondentes.

Serviço	Nome do serviço de endpoint da VPC	Tipo de endpoint VPC	Alias de DNS privado personalizado	Observações
AWS IoT data	`com.amazonaws.`*`region`*`.iot.d`	Interface	*`prefix`*`-ats.iot.`*`region`*`.a` `s.com`	O registro DNS privado deve corresponder ao AWS IoT data endpoint da sua conta:. `aws iot describe-endpoint --endpoint-type iot:Data-ATS`

Credenciais da AWS IoT	`com.amazonaws.`*`region`*`.iot.c`
	`entials`
		Interface	*`prefix`*`.c`
			`als.iot.`
			`s.com`
				O registro DNS privado deve corresponder ao endpoint de AWS

Serviço	Nome do serviço de endpoint da VPC	Tipo de endpoint VPC	Alias de DNS privado personalizado	Observações
				IoT credenciais da sua conta:. `aws iot describe-endpoint --endpoint-type iot:CredentialProvider`
Amazon S3	`com.amazonaws.` *`region`*`.s3`	Interface		O registro DNS é criado automaticamente.

Melhores práticas de segurança do AWS IoT Greengrass

Este tópico contém as melhores práticas de segurança para o AWS IoT Greengrass.

Conceder o mínimo possível de permissões

Siga o princípio do menor privilégio para seus componentes executando-os como usuários sem privilégios. Os componentes não devem ser executados como root, a menos que seja absolutamente necessário.

Use o conjunto mínimo de permissões nas funções do IAM. Limite o uso do`*`curinga para o`Action`e`Resource`propriedades em suas políticas do IAM. Em vez disso, declare um conjunto finito de ações e recursos quando possível. Para obter mais informações sobre as melhores práticas de privilégio mínimo e outras de políticas, consulte the section called “Práticas recomendadas de políticas”.

A melhor prática de privilégios mínimos também se aplica aAWS IoTpolíticas que você anexa ao seu núcleo do Greengrass.

Não codifique credenciais nos componentes do Greengrass

Não codifique credenciais em seus componentes do Greengrass definidos pelo usuário. Como proteger melhor suas credenciais:

- Para interagir comAWSserviços, defina permissões para ações e recursos específicos noFunção de serviço de dispositivos principais do Greengrass.
- Use ocomponente de gerenciador secretopara armazenar suas credenciais. Ou, se a função usar oAWSSDK, use credenciais da cadeia de fornecedores de credenciais padrão.

Não registrar em log informações confidenciais

Você deve impedir o registro de credenciais e outras informações de identificação pessoal (PII). Recomendamos que você implemente as seguintes proteções, mesmo que o acesso aos registros locais em um dispositivo principal exija privilégios de root e acesso a CloudWatch Os registros exigem permissões do IAM.

- Não use informações confidenciais em caminhos de tópico MQTT.
- Não use informações confidenciais em nomes, tipos e atributos de dispositivo (coisa) no registro do AWS IoT Core.
- Não registre informações confidenciais em seus componentes do Greengrass definidos pelo usuário ou nas funções do Lambda.
- Não use informações confidenciais nos nomes e IDs dos recursos do Greengrass:
  - Dispositivos principais
  - Componentes
  - Implantações
  - Loggers

Manter o relógio do dispositivo sincronizado

É importante ter a hora exata no seu dispositivo. Os certificados X.509 têm data e hora de expiração. O relógio em seu dispositivo é usado para verificar se um certificado de servidor ainda é válido. Os relógios do dispositivo podem atrasar ao longo do tempo ou as baterias podem descarregar.

Para obter mais informações, consulte a melhor prática Manter o relógio do dispositivo sincronizado no Guia do desenvolvedor do AWS IoT Core.

Recomendações do Cipher Suite

O padrão do Greengrass seleciona os pacotes de criptografia TLS mais recentes disponíveis no dispositivo. Considere desativar o uso de pacotes de criptografia legados no dispositivo. Por exemplo, suítes de cifras CBC.

Para obter mais informações, consulte oConfiguração de criptografia Java.

Consulte também

- Melhores práticas de segurança emAWS IoT CorenoAWS IoTGuia do desenvolvedor
- Dez regras de ouro de segurança para soluções de IoT industrialnoInternet das coisas emAWSBlog oficial

Usando AWS IoT Device Tester para AWS IoT Greengrass V2

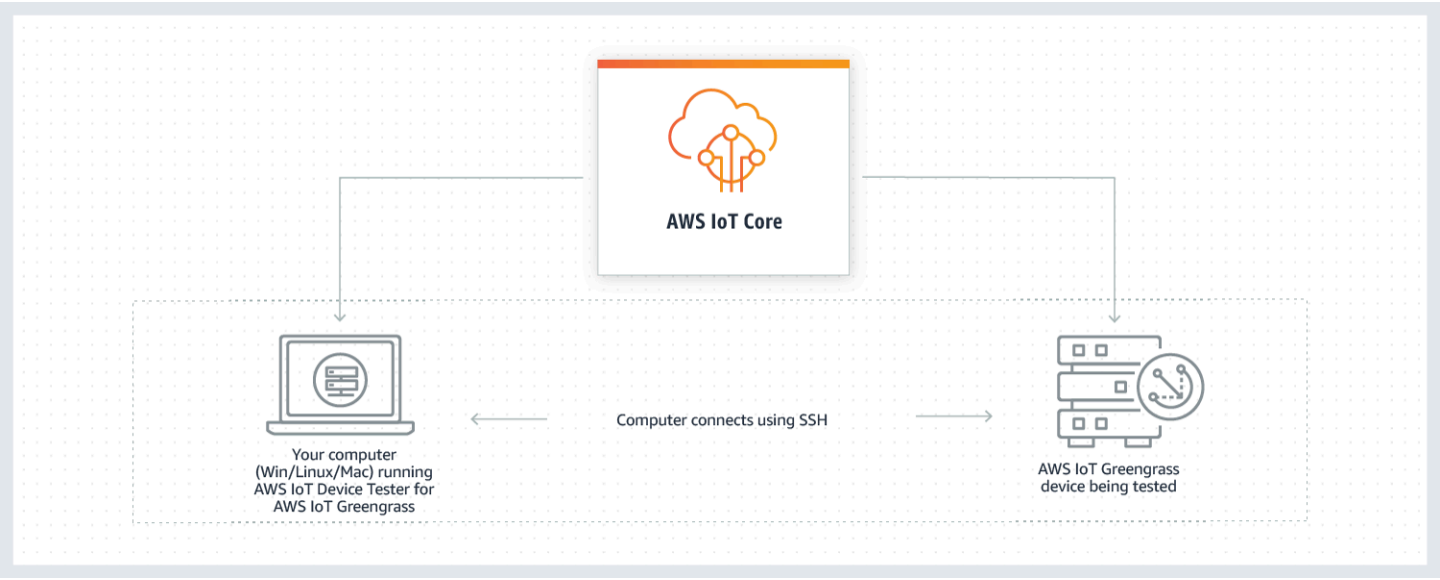
AWS IoT O Device Tester (IDT) é uma estrutura de teste disponível para download que permite validar dispositivos de IoT. Você pode usar o IDT AWS IoT Greengrass para executar o pacote de AWS IoT Greengrass qualificação e criar e executar conjuntos de testes personalizados para seus dispositivos.

O IDT for AWS IoT Greengrass executado em seu computador host (Windows, macOS ou Linux) conectado ao dispositivo a ser testado. Ele executa testes e agrega resultados. Ele também fornece uma interface de linha de comando para gerenciar o processo de teste.

AWS IoT Greengrass suíte de qualificação

Use AWS IoT Device Tester for AWS IoT Greengrass V2 para verificar se o software AWS IoT Greengrass Core é executado em seu hardware e pode se comunicar com o. Nuvem AWS Ele também realiza end-to-end testes com AWS IoT Core. Por exemplo, ele verifica se seu dispositivo pode implantar componentes e atualizá-los.

Se você quiser adicionar seu hardware ao Catálogo de AWS Partner dispositivos, execute o pacote de AWS IoT Greengrass qualificação para gerar relatórios de teste para os quais você possa enviar AWS IoT. Para obter mais informações, consulte Programa de Qualificação de Dispositivos da AWS.



O IDT for AWS IoT Greengrass V2 organiza testes usando os conceitos de suítes de testes e grupos de testes.

- Um conjunto de testes é o conjunto de grupos de teste usado para verificar se um dispositivo funciona com versões específicas do AWS IoT Greengrass.
- Um grupo de testes é o conjunto de testes individuais relacionados a um recurso específico, como implantações de componentes.

Para ter mais informações, consulte Use o IDT para executar o pacote de AWS IoT Greengrass qualificação.

Conjuntos de teste personalizados

A partir do IDT v4.0.1, o IDT for AWS IoT Greengrass V2 combina uma configuração padronizada e um formato de resultado com um ambiente de suíte de testes que permite desenvolver suítes de testes personalizadas para seus dispositivos e software de dispositivos. É possível adicionar testes personalizados para sua própria validação interna ou fornecê-los aos seus clientes para verificação de dispositivo.

A forma como um gravador de testes configura um pacote de testes personalizado determina as configurações necessárias para executar conjuntos de testes personalizados. Para ter mais informações, consulte Usar o IDT para desenvolver e executar os próprios pacotes de testes.

Versões compatíveis do AWS IoT Device Tester for AWS IoT Greengrass V2

Este tópico lista as versões suportadas do IDT para AWS IoT Greengrass V2. Como prática recomendada, recomendamos que você use a versão mais recente do IDT para AWS IoT Greengrass V2 que ofereça suporte à sua versão de destino da AWS IoT Greengrass V2. Novas versões do AWS IoT Greengrass podem exigir que você baixe uma nova versão do IDT para AWS IoT Greengrass V2. Você recebe uma notificação ao iniciar um teste se o IDT for AWS IoT Greengrass V2 não for compatível com a versão AWS IoT Greengrass que você está usando.

Ao baixar o software, você concorda com o Contrato AWS IoT Device Tester de Licença.

> **Note**
>
> O IDT não oferece suporte a execução por vários usuários em um local compartilhado, como um diretório NFS ou uma pasta compartilhada de rede do Windows. Recomendamos extrair o pacote do IDT para uma unidade local e executar o binário do IDT na estação de trabalho local.

Versão mais recente do IDT para AWS IoT Greengrass V2

Você pode usar essa versão do IDT para AWS IoT Greengrass V2 com a AWS IoT Greengrass versão listada aqui.

IDT v4.9.4 para AWS IoT Greengrass

AWS IoT Greengrass Versões suportadas:

- Núcleo Greengrass v2.12.0, v2.11.0, v2.10.0 e v2.9.5

Downloads de software IDT:

- IDT v4.9.4 com suíte de testes GGV2Q_2.5.4 para Linux
- IDT v4.9.4 com suíte de testes GGV2Q_2.5.4 para macOS
- IDT v4.9.4 com suíte de testes GGV2Q_2.5.4 para Windows

Notas de release:

- Permite a validação e qualificação de dispositivos que executam as versões 2.12.0, 2.11.0, 2.10.0 e 2.9.5 do software AWS IoT Greengrass Core.
- Remove o gerenciador de streams e os grupos de teste de aprendizado de máquina.

Notas adicionais:

- Se seu dispositivo usa um HSM e você está usando o nucleus 2.10.x, migre para o Greengrass nucleus versão 2.11.0 ou posterior.

Versão do conjunto de testes:

`GGV2Q_2.5.4`

- Lançado em 2024.05.03

Versões anteriores do IDT para AWS IoT Greengrass

As seguintes versões anteriores do IDT para AWS IoT Greengrass V2 também são suportadas.

IDT v4.9.3 para AWS IoT Greengrass

AWS IoT Greengrass Versões suportadas:

- Núcleo Greengrass v2.12.0, v2.11.0, v2.10.0 e v2.9.5

Downloads de software IDT:

- IDT v4.9.3 com suíte de testes GGV2Q_2.5.3 para Linux
- IDT v4.9.3 com suíte de testes GGV2Q_2.5.3 para macOS
- IDT v4.9.3 com suíte de testes GGV2Q_2.5.3 para Windows

Notas de release:

- Corrige um problema nos testes de componentes ao testar um dispositivo Linux a partir de um host Windows ou vice-versa.
- Remove o caso de `localcomponent` teste do grupo `component` de teste. Esse caso de teste não é mais necessário para a qualificação.

Notas adicionais:

- Se seu dispositivo usa um HSM e você está usando o nucleus 2.10.x, migre para o Greengrass nucleus versão 2.11.0 ou posterior.

Versão do conjunto de testes:

`GGV2Q_2.5.3`

- Lançado em 2024.04.05

Versões não suportadas do AWS IoT Device Tester for AWS IoT Greengrass V2

Este tópico lista as versões não suportadas do IDT para AWS IoT Greengrass V2. Versões não compatíveis não recebem correções de bugs ou atualizações. Para ter mais informações, consulte the section called “Política de suporte AWS IoT Device Tester para AWS IoT Greengrass”.

IDT v4.9.2 para AWS IoT Greengrass

Notas de release:

- Corrige um problema em que a suíte de testes Lambda falha devido à obsolescência do Java 8.

Versão do conjunto de testes:

`GGV2Q_2.5.2`

- Lançado em 2024.03.18

IDT v4.9.1 para AWS IoT Greengrass

Notas de release:

- Permite validar e qualificar dispositivos que executam as versões 2.12.0, 2.11.0, 2.10.0 e 2.9.5 do software AWS IoT Greengrass Core.
- Correções de erros secundárias.

Versão do conjunto de testes:

`GGV2Q_2.5.1`

- Lançado em 2023.10.05

IDT v4.7.0 para AWS IoT Greengrass

AWS IoT Greengrass Versões suportadas:

- Núcleo Greengrass v2.11.0, v2.10.0 e v2.9.5

Notas de release:

- Permite validar e qualificar dispositivos que executam as versões 2.11.0, 2.10.0 e 2.9.5 do software AWS IoT Greengrass Core.
- Adiciona suporte para armazenar valores de dados de usuário do IDT no AWS Systems Manager Parameter Store e buscá-los na configuração usando a sintaxe de espaço reservado.
- Correções de erros secundárias.

Versão do conjunto de testes:

`GGV2Q_2.5.0`

- Lançado em 13 de dezembro de 2021

IDT v4.5.11 para AWS IoT Greengrass

Notas de release:

- Permite validar e qualificar dispositivos que executam as versões 2.9.1, 2.9.0, 2.8.1, 2.8.0, 2.7.0 e 2.6.0 do software AWS IoT Greengrass Core.
- Adiciona suporte para testar o PreInstalled Greengrass em um dispositivo principal.
- Correções de erros secundárias.

Versão do conjunto de testes:

`GGV2Q_2.4.1`

- Lançado em 2022.10.13

IDT v4.5.8 para AWS IoT Greengrass

Notas de release:

- Permite validar e qualificar dispositivos que executam as versões 2.7.0, 2.6.0 e 2.5.6 do software AWS IoT Greengrass Core.
- Permite que você teste com o PreInstalled Greengrass em um dispositivo principal.
- Correções de erros secundárias.

Versão do conjunto de testes:

`GGV2Q_2.4.0`

- Lançado em 2022.08.12

IDT v4.5.3 para AWS IoT Greengrass

Notas de release:

- Permite validar e qualificar dispositivos que executam as versões 2.7.0, 2.6.0, 2.5.6, 2.5.5, 2.5.4 e 2.5.3 do software AWS IoT Greengrass Core.
- DockerApplicationManager Teste de atualizações para usar uma imagem docker baseada em ECR.
- Correções de erros secundárias.

Versão do conjunto de testes:

`GGV2Q_2.3.1`

- Lançado em 2022.04.15

IDT v4.5.1 para AWS IoT Greengrass

Notas de release:

- Permite validar e qualificar dispositivos que executam o software AWS IoT Greengrass Core v2.5.3.
- Adiciona suporte para validar e qualificar dispositivos baseados em Linux que usam um módulo de segurança de hardware (HSM) para armazenar a chave privada e o certificado usados pelo software Core. AWS IoT Greengrass
- Implementa o novo orquestrador de testes do IDT para configurar pacotes de testes personalizados. Para ter mais informações, consulte Configurar o orquestrador de teste IDT.

- Correções adicionais de pequenos bugs.

Versão do conjunto de testes:

`GGV2Q_2.3.0`

- Lançado em 2022.01.11

IDT v4.4.1 para AWS IoT Greengrass

Notas de release:

- Permite validar e qualificar dispositivos que executam o software AWS IoT Greengrass Core v2.5.2.
- Adiciona suporte ao uso de uma função do IAM definida pelo usuário como a função de troca de tokens que o dispositivo em teste presume interagir com AWS os recursos.

  Você pode especificar a função do IAM no `userdata.json`arquivo. Se você especificar uma função personalizada, o IDT usará essa função em vez de criar a função padrão de troca de tokens durante a execução do teste.
- Correções adicionais de pequenos bugs.

Versão do conjunto de testes:

`GGV2Q_2.2.1`

- Lançado em 2021.12.12

IDT v4.4.0 para AWS IoT Greengrass

Notas de release:

- Permite validar e qualificar dispositivos que executam o software AWS IoT Greengrass Core v2.5.0.
- Adiciona suporte para validar e qualificar dispositivos que executam o software AWS IoT Greengrass Core no Windows.
- Suporta o uso da validação de chave pública para conexões de dispositivos Secure Shell (SSH).
- Melhora a política do IAM de permissões de IDT com as melhores práticas de segurança.
- Correções adicionais de pequenos bugs.

Versão do conjunto de testes:

`GGV2Q_2.1.0`

- Lançado em 19/11/2021

IDT v4.2.0 para AWS IoT Greengrass

Notas de release:

- Inclui suporte para qualificação dos seguintes recursos em dispositivos que executam o software AWS IoT Greengrass Core v2.2.0 e versões posteriores:
  - Docker — Valida que os dispositivos podem baixar uma imagem de contêiner Docker do Amazon Elastic Container Registry (Amazon ECR).
  - Aprendizado de máquina — valida que os dispositivos podem realizar inferência de aprendizado de máquina (ML) usando as estruturas Deep Learning Runtime ou Lite ML. TensorFlow
  - Gerenciador de transmissão — valida se os dispositivos podem baixar, instalar e executar o gerenciador de transmissão. AWS IoT Greengrass
- Permite validar e qualificar dispositivos que executam o software AWS IoT Greengrass Core v2.4.0, v2.3.0, v2.2.0 e v2.1.0.
- Agrupa os registros de teste de cada caso de teste em uma pasta `< test-case-id >` separada dentro do `<device-tester-extract-location>/results/<execution-id>/logs/<test-group-id>` diretório.
- Correções adicionais de pequenos bugs.

Versão do conjunto de testes:

`GGV2Q_2.0.1`

- Lançado em 2021.08.31

IDT v4.1.0 para AWS IoT Greengrass

Notas de release:

- Permite validar e qualificar dispositivos que executam o software AWS IoT Greengrass Core v2.3.0, v2.2.0, v2.1.0 e v2.0.5.
- Melhora a `userdata.json` configuração removendo a necessidade de especificar as `GreengrassCLIVersion` propriedades `GreengrassNucleusVersion` e.
- Inclui suporte para qualificação de recursos Lambda e MQTT para o software AWS IoT Greengrass Core v2.1.0 e versões posteriores. Agora você pode usar o IDT for AWS IoT Greengrass V2 para validar se seu dispositivo principal pode executar funções do Lambda e se o dispositivo pode publicar e assinar tópicos do MQTT. AWS IoT Core
- Melhora os recursos de registro.
- Correções adicionais de pequenos bugs.

Versão do conjunto de testes:

`GGV2Q_1.1.1`

- Lançado em 2021.06.18

IDT v4.0.2 para AWS IoT Greengrass

Notas de release:

- Permite validar e qualificar dispositivos que executam o software AWS IoT Greengrass Core v2.1.0.
- Adiciona suporte para qualificação de recursos Lambda e MQTT para o software AWS IoT Greengrass Core v2.1.0 e versões posteriores. Agora você pode usar o IDT for AWS IoT Greengrass V2 para validar se seu dispositivo principal pode executar funções do Lambda e se o dispositivo pode publicar e assinar tópicos do MQTT. AWS IoT Core
- Melhora os recursos de registro.
- Correções adicionais de pequenos bugs.

Versão do conjunto de testes:

`GGV2Q_1.1.1`

- Lançado em 2021.05.05

IDT v4.0.1 para AWS IoT Greengrass

Notas de release:

- Permite validar e qualificar dispositivos que executam o software da AWS IoT Greengrass versão 2.
- Permite que você desenvolva e execute suas suítes de testes personalizadas usando AWS IoT Device Tester for AWS IoT Greengrass. Para ter mais informações, consulte Usar o IDT para desenvolver e executar os próprios pacotes de testes.
- Fornece aplicativos do IDT assinados por código para macOS e Windows. No macOS, talvez seja necessário conceder uma exceção de segurança para o IDT. Para ter mais informações, consulte Exceção de segurança no macOS.

Versão do conjunto de testes:

`GGV2Q_1.0.0`

- Lançado em 2020.12.22
- A suíte de testes executa somente os testes necessários para qualificação, a menos que você defina o correspondente `value` na `features` matriz comoyes.

Baixe o IDT para V2 AWS IoT Greengrass

Este tópico descreve as opções de download AWS IoT Device Tester para a AWS IoT Greengrass V2. Você pode usar um dos links de download de software a seguir ou seguir as instruções para baixar o IDT de forma programática.

Tópicos

Ao baixar o software, você concorda com o Contrato AWS IoT Device Tester de Licença.

> **Note**
>
> O IDT não oferece suporte a execução por vários usuários em um local compartilhado, como um diretório NFS ou uma pasta compartilhada de rede do Windows. Recomendamos extrair o pacote do IDT para uma unidade local e executar o binário do IDT na estação de trabalho local.

Baixar IDT manualmente

Este tópico lista as versões suportadas do IDT para AWS IoT Greengrass V2. Como prática recomendada, recomendamos que você use a versão mais recente do IDT para AWS IoT Greengrass V2 que ofereça suporte à sua versão de destino da AWS IoT Greengrass V2. Novas versões do AWS IoT Greengrass podem exigir que você baixe uma nova versão do IDT para AWS IoT Greengrass V2. Você recebe uma notificação ao iniciar um teste se o IDT for AWS IoT Greengrass V2 não for compatível com a versão AWS IoT Greengrass que você está usando.

IDT v4.9.4 para AWS IoT Greengrass

AWS IoT Greengrass Versões suportadas:

- Núcleo Greengrass v2.12.0, v2.11.0, v2.10.0 e v2.9.5

Downloads de software IDT:

- IDT v4.9.4 com suíte de testes GGV2Q_2.5.4 para Linux
- IDT v4.9.4 com suíte de testes GGV2Q_2.5.4 para macOS

- IDT v4.9.4 com suíte de testes GGV2Q_2.5.4 para Windows

Notas de release:

- Permite a validação e qualificação de dispositivos que executam as versões 2.12.0, 2.11.0, 2.10.0 e 2.9.5 do software AWS IoT Greengrass Core.
- Remove o gerenciador de streams e os grupos de teste de aprendizado de máquina.

Notas adicionais:

- Se seu dispositivo usa um HSM e você está usando o nucleus 2.10.x, migre para o Greengrass nucleus versão 2.11.0 ou posterior.

Versão do conjunto de testes:

`GGV2Q_2.5.4`

- Lançado em 2024.05.03

Baixar IDT de maneira programada

O IDT fornece uma operação de API que você pode usar para recuperar um URL na qual é possível baixar o IDT de maneira programada. Você também pode usar essa operação de API para verificar se você tem a versão mais recente do IDT. Essa operação da API tem o seguinte endpoint.

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

Para chamar essa operação de API, você deve ter a permissão para executar a ação **`iot-device-tester:LatestIdt`**. Inclua sua AWS assinatura e use `iot-device-tester` como nome do serviço.

Solicitações de API

HostOs — O sistema operacional da máquina host. Escolha uma das seguintes opções:

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

TestSuiteType — O tipo da suíte de testes. Escolha a seguinte opção:

`GGV2`— IDT para V2 AWS IoT Greengrass

ProductVersion

(Opcional) A versão do núcleo Greengrass. O serviço retorna a versão mais recente compatível do IDT para essa versão do núcleo Greengrass. Se você não especificar essa opção, o serviço retornará a versão mais recente do IDT.

Resposta da API

O resposta da API tem o seguinte formato. O arquivo `DownloadURL` inclui o arquivo zip.

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

Exemplos

Você pode consultar os exemplos a seguir para baixar o IDT de maneira programada. Esses exemplos usam credenciais que você armazena nas variáveis de ambiente `AWS_ACCESS_KEY_ID` e `AWS_SECRET_ACCESS_KEY`. Para seguir as práticas recomendadas de segurança, não armazene suas credenciais em seu código.

Example Exemplo: faça download usando cURL versão 7.75.0 ou posterior (Mac e Linux)

Se você tiver a versão 7.75.0 ou posterior do cURL, poderá usar a sinalização `aws-sigv4` para assinar a solicitação da API. Este exemplo usa jq para analisar o URL de download da resposta.

> **Warning**
>
> A `aws-sigv4` sinalização exige que os parâmetros de consulta da solicitação GET curl estejam na ordem de HostOs/ProductVersion/TestSuiteType ouHostOs/TestSuiteType. Pedidos que não estiverem em conformidade resultarão em um erro ao obter assinaturas incompatíveis para a string canônica do API Gateway.

Se o parâmetro opcional ProductVersion estiver incluído, você deverá usar uma versão de produto compatível, conforme documentado em Versões suportadas do AWS IoT Device Tester for AWS IoT Greengrass V2.

- Substitua *us-west-2* pelo seu. Região da AWS Para obter uma lista de códigos de região, consulte Endpoints regionais.
- Substitua *linux* pelo sistema operacional da sua máquina host.
- Substitua *2.5.3* pela sua versão do AWS IoT Greengrass nucleus.

```
url=$(curl --request GET "https://
download.devicetester.iotdevicesecosystem.amazonaws.com/latestidt?
HostOs=linux&ProductVersion=2.5.3&TestSuiteType=GGV2" \
--user $AWS_ACCESS_KEY_ID:$AWS_SECRET_ACCESS_KEY \
--aws-sigv4 "aws:amz:us-west-2:iot-device-tester" \
| jq -r '.LatestBk["DownloadURL"]')

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

Example Exemplo: faça download usando uma versão anterior do cURL (Mac e Linux)

Você pode usar o seguinte comando cURL com uma AWS assinatura que você assina e calcula. Para obter mais informações sobre como assinar e calcular uma AWS assinatura, consulte Assinatura de solicitações AWS da API.

- Substitua *linux* pelo sistema operacional da sua máquina host.
- Substitua *Timestamp* pela data e hora, como **20220210T004606Z**.
- Substitua a *date* pela data, como **20220210**.
- *AWSRegion*Substitua pelo seu Região da AWS. Para obter uma lista de códigos de região, consulte Endpoints regionais.
- *AWSSignature*Substitua pela AWS assinatura que você gera.

```
curl --location --request GET 'https://
download.devicetester.iotdevicesecosystem.amazonaws.com/latestidt?
HostOs=linux&TestSuiteType=GGV2' \
--header 'X-Amz-Date: Timestamp \
```

```
--header 'Authorization: AWS4-HMAC-SHA256 Credential=$AWS_ACCESS_KEY_ID/Date/AWSRegion/
iot-device-tester/aws4_request, SignedHeaders=host;x-amz-date, Signature=AWSSignature'
```

Example Exemplo: faça o download usando um script Python

Este exemplo usa a biblioteca de solicitações do Python. Esse exemplo foi adaptado do exemplo do Python para assinar uma solicitação de AWS API na Referência AWS geral.

- Substitua *us-west-2* pela sua região. Para obter uma lista de códigos de região, consulte Endpoints regionais.
- Substitua *linux* pelo sistema operacional da sua máquina host.

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

# See: http://docs.aws.amazon.com/general/latest/gr/sigv4_signing.html
# This version makes a GET request and passes the signature
# in the Authorization header.
import sys, os, base64, datetime, hashlib, hmac
import requests # pip install requests
# ************* REQUEST VALUES *************
method = 'GET'
service = 'iot-device-tester'
host = 'download.devicetester.iotdevicesecosystem.amazonaws.com'
region = 'us-west-2'
endpoint = 'https://download.devicetester.iotdevicesecosystem.amazonaws.com/latestidt'
request_parameters = 'HostOs=linux&TestSuiteType=GGV2'

# Key derivation functions. See:
# http://docs.aws.amazon.com/general/latest/gr/signature-v4-examples.html#signature-v4-
examples-python
def sign(key, msg):
```

```
    return hmac.new(key, msg.encode('utf-8'), hashlib.sha256).digest()

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

# Read AWS access key from env. variables or configuration file. Best practice is NOT
# to embed credentials in code.
access_key = os.environ.get('AWS_ACCESS_KEY_ID')
secret_key = os.environ.get('AWS_SECRET_ACCESS_KEY')
if access_key is None or secret_key is None:
    print('No access key is available.')
    sys.exit()

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

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

```
# Step 6: Create payload hash (hash of the request body content). For GET
# requests, the payload is an empty string ("").
payload_hash = hashlib.sha256(('').encode('utf-8')).hexdigest()
# Step 7: Combine elements to create canonical request
canonical_request = method + '\n' + canonical_uri + '\n' + canonical_querystring + '\n'
 + canonical_headers + '\n' + signed_headers + '\n' + payload_hash

# ************* TASK 2: CREATE THE STRING TO SIGN*************
# Match the algorithm to the hashing algorithm you use, either SHA-1 or
# SHA-256 (recommended)
algorithm = 'AWS4-HMAC-SHA256'
credential_scope = datestamp + '/' + region + '/' + service + '/' + 'aws4_request'
string_to_sign = algorithm + '\n' +  amzdate + '\n' +  credential_scope + '\n' +
 hashlib.sha256(canonical_request.encode('utf-8')).hexdigest()
# ************* TASK 3: CALCULATE THE SIGNATURE *************
# Create the signing key using the function defined above.
signing_key = getSignatureKey(secret_key, datestamp, region, service)
# Sign the string_to_sign using the signing_key
signature = hmac.new(signing_key, (string_to_sign).encode('utf-8'),
 hashlib.sha256).hexdigest()

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

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

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

Use o IDT para executar o pacote de AWS IoT Greengrass qualificação

Você pode usar o AWS IoT Device Tester for AWS IoT Greengrass V2 para verificar se o software AWS IoT Greengrass Core é executado em seu hardware e pode se comunicar com o. Nuvem AWS Ele também realiza end-to-end testes com AWS IoT Core. Por exemplo, ele verifica se seu dispositivo pode implantar componentes e atualizá-los.

Além de testar dispositivos, o IDT for AWS IoT Greengrass V2 cria recursos (por exemplo, AWS IoT coisas, grupos e assim por diante) Conta da AWS para facilitar o processo de qualificação.

Para criar esses recursos, o IDT for AWS IoT Greengrass V2 usa AWS as credenciais configuradas no `config.json` arquivo para fazer chamadas de API em seu nome. Esses recursos são provisionados em vários momentos durante o teste.

Quando você usa o IDT for AWS IoT Greengrass V2 para executar o pacote de AWS IoT Greengrass qualificação, ele executa as seguintes etapas:

1. Carrega e valida o dispositivo e configuração de credenciais.
2. Executa testes selecionados com os recursos locais e de nuvem necessários.
3. Remove recursos locais e de nuvem.
4. Gera relatórios de testes que indicam se a placa passou nos testes necessários para a qualificação.

Versões do pacote de testes

O IDT for AWS IoT Greengrass V2 organiza os testes em suítes de testes e grupos de teste.

- Um conjunto de testes é o conjunto de grupos de teste usado para verificar se um dispositivo funciona com versões específicas do AWS IoT Greengrass.
- Um grupo de testes é o conjunto de testes individuais relacionados a um recurso específico, como implantações de componentes.

As suítes de teste são versionadas usando um *major*.*minor*.*patch* formato, por exemplo. GGV2Q_1.0.0 Quando você baixa o IDT, o pacote inclui a versão mais recente do pacote de qualificação Greengrass.

> **Important**
>
> Os testes de versões do conjunto de testes não compatíveis não são válidos para qualificação do dispositivo. O IDT não imprime relatórios de qualificação para versões não compatíveis. Para ter mais informações, consulte the section called “Política de suporte AWS IoT Device Tester para AWS IoT Greengrass”.
> Você pode executar `list-supported-products` para listar as versões AWS IoT Greengrass e os conjuntos de testes compatíveis com sua versão atual do IDT.

Descrições dos grupos de testes

Grupos de teste necessários para a qualificação de núcleo

Esses grupos de teste são necessários para qualificar seu dispositivo AWS IoT Greengrass V2 para o Catálogo de AWS Partner dispositivos.

Dependências principais

Valida se o dispositivo atende a todos os requisitos de software e hardware do software AWS IoT Greengrass Core. Esse grupo de teste inclui o seguinte caso de teste:

Versão Java

Verifica se a versão necessária do Java está instalada no dispositivo em teste. AWS IoT Greengrass requer Java 8 ou posterior.

PreTest Validação

Verifica se o dispositivo atende aos requisitos de software para executar testes.

- Para dispositivos baseados em Linux, esse teste verifica se o dispositivo pode executar os seguintes comandos Linux:

  `chmod`, `cp`, `echo`, `grep`, `kill`, `ln`, `mkinfo`, `ps`, `rm`, `sh`, `uname`
- Para dispositivos baseados em Windows, esse teste verifica se o dispositivo tem o seguinte software da Microsoft instalado:

Powershell v5.1 ou posterior, .NET v4.6.1 ou posterior, Visual C++ 2017 ou posterior, utilitário PsExec

Verificador de versão

Verifica se a versão AWS IoT Greengrass fornecida é compatível com a versão do AWS IoT Device Tester que você está usando.

Componente

Valida que o dispositivo pode implantar componentes e atualizá-los. Esse grupo de teste inclui os seguintes testes:

Componente de nuvem

Valida a capacidade do dispositivo para componentes de nuvem.

Componente local

Valida a capacidade do dispositivo para componentes locais.

Lambda

Esse teste não é aplicável a dispositivos baseados em Windows.

Valida que o dispositivo pode implantar componentes da função Lambda que usam o Java Runtime e que as funções Lambda podem AWS IoT Core usar tópicos do MQTT como fontes de eventos para mensagens de trabalho.

MQTT

Valida que o dispositivo pode assinar e publicar tópicos do AWS IoT Core MQTT.

Grupos de testes opcionais

> **Note**
>
> Esses grupos de teste são opcionais e usados somente para qualificar os dispositivos principais do Greengrass baseados em Linux. Se você optar por se qualificar para testes opcionais, seu dispositivo será listado com recursos adicionais no Catálogo de AWS Partner dispositivos.

Dependências do Docker

Valida se o dispositivo atende a todas as dependências técnicas necessárias para usar o componente Docker application manager () AWS fornecido. `aws.greengrass.DockerApplicationManager`

Qualificação do Docker Application Manager

Valida que o dispositivo pode baixar uma imagem de contêiner Docker do Amazon ECR.

Dependências do Machine Learning

> **Note**
>
> O grupo de teste opcional de aprendizado de máquina é suportado somente no IDT v4.9.3.

Valida se o dispositivo atende a todas as dependências técnicas necessárias para usar os componentes de aprendizado AWS de máquina (ML) fornecidos.

Testes de inferência de Machine Learning

> **Note**
>
> O grupo de teste opcional de aprendizado de máquina é suportado somente no IDT v4.9.3.

Valida que o dispositivo pode realizar inferência de ML usando as estruturas Deep Learning Runtime e TensorFlow Lite ML.

Dependências do Stream Manager

> **Note**
>
> O grupo de teste opcional do gerenciador de stream é suportado somente no IDT v4.9.3.

Valida se o dispositivo pode baixar, instalar e executar o gerenciador de AWS IoT Greengrass streaming.

Integração de segurança de hardware (HSI)

> **Note**
>
> Esse teste está disponível no IDT v4.9.3 e versões posteriores somente para dispositivos baseados em Linux. AWS IoT Greengrass atualmente não oferece suporte à integração de segurança de hardware para dispositivos Windows.

Valida se o dispositivo pode autenticar conexões com os AWS IoT Greengrass serviços AWS IoT e usando uma chave privada e um certificado armazenados em um módulo de segurança de hardware (HSM). Esse teste também verifica se o componente provedor PKCS #11 AWS fornecido pode interagir com o HSM usando uma biblioteca PKCS #11 fornecida pelo fornecedor. Para ter mais informações, consulte Integração de segurança de hardware.

Pré-requisitos para executar o pacote de qualificação AWS IoT Greengrass

Esta seção descreve os pré-requisitos para usar AWS IoT Device Tester (IDT) para. AWS IoT Greengrass

Baixe a versão mais recente do AWS IoT Device Tester for AWS IoT Greengrass

Baixe a versão mais recente do IDT e extraia o software em um local (*< device-tester-extract-location >*) em seu sistema de arquivos onde você tenha permissões de leitura/gravação.

> **Note**
>
> O IDT não oferece suporte a execução por vários usuários em um local compartilhado, como um diretório NFS ou uma pasta compartilhada de rede do Windows. Recomendamos extrair o pacote do IDT para uma unidade local e executar o binário do IDT na estação de trabalho local.
>
> O Windows tem uma limitação de comprimento de caminho de 260 caracteres. Se você estiver usando o Windows, extraia o IDT para um diretório raiz como `C:\` ou `D:\` para manter os caminhos abaixo do limite de 260 caracteres.

Baixe o AWS IoT Greengrass software

O IDT for AWS IoT Greengrass V2 testa a compatibilidade do seu dispositivo com uma versão específica do. AWS IoT Greengrass Execute o comando a seguir para baixar o software AWS IoT Greengrass Core em um arquivo chamado`aws.greengrass.nucleus.zip`. Substitua a *`versão`* por uma versão de componente nuclear compatível para sua versão do IDT.

Linux or Unix

```
curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip >
 aws.greengrass.nucleus.zip
```

Windows Command Prompt (CMD)

```
curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip >
 aws.greengrass.nucleus.zip
```

PowerShell

```
iwr -Uri https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-version.zip -
OutFile aws.greengrass.nucleus.zip
```

Coloque o `aws.greengrass.nucleus.zip` arquivo baixado na *`<device-tester-extract-location>`*`/products/` pasta.

> **Note**
>
> Não coloque vários arquivos nesse diretório para o mesmo sistema operacional e arquitetura.

Crie e configure um Conta da AWS

Antes de poder usar AWS IoT Device Tester para a AWS IoT Greengrass V2, você deve executar as seguintes etapas:

1. Configure um Conta da AWS. Se você já tem um Conta da AWS, vá para a etapa 2.
2. Configure permissões para o IDT.

Essas permissões de conta permitem que a IDT acesse AWS serviços e crie AWS recursos, como AWS IoT itens e AWS IoT Greengrass componentes, em seu nome.

Para criar esses recursos, o IDT for AWS IoT Greengrass V2 usa AWS as credenciais configuradas no `config.json` arquivo para fazer chamadas de API em seu nome. Esses recursos são provisionados em vários momentos durante o teste.

> **Note**
>
> Embora a maioria dos testes se qualifique para o nível AWS gratuito, você deve fornecer um cartão de crédito ao se inscrever em um Conta da AWS. Para obter mais informações, consulte Por que preciso de uma forma de pagamento se minha conta está coberta pelo nível gratuito?.

Etapa 1: configurar um Conta da AWS

Nesta etapa, crie e configure uma Conta da AWS. Se você já tem uma Conta da AWS, pule para the section called “Etapa 2: Configurar permissões para o IDT”.

Se você não tiver um Conta da AWS, conclua as etapas a seguir para criar um.

Para se inscrever em um Conta da AWS

1. Abra https://portal.aws.amazon.com/billing/signup.
2. Siga as instruções on-line.

   Parte do procedimento de inscrição envolve receber uma chamada telefônica e digitar um código de verificação no teclado do telefone.

   Quando você se inscreve em um Conta da AWS, um Usuário raiz da conta da AWSé criado. O usuário-raiz tem acesso a todos os Serviços da AWS e recursos na conta. Como uma prática recomendada de segurança, atribua o acesso administrativo para um usuário e use somente o usuário-raiz para executar tarefas que requerem o acesso de usuário-raiz.

Para criar um usuário administrador, selecione uma das opções a seguir.

Selecionar uma forma de gerenciar o administrador	Para	Por	Você também pode
Centro de Identidade do IAM (Recomendado)	Use credenciais de curto prazo para acessar a AWS. Isso está de acordo com as práticas recomendadas de segurança. Para obter informações sobre as práticas recomendadas, consulte Práticas recomendadas de segurança no IAM no Guia do usuário do IAM.	Seguindo as instruções em Conceitos básicos no Guia do usuário do AWS IAM Identity Center .	Configure o acesso programático configurando o AWS CLI para uso AWS IAM Identity Center no Guia do AWS Command Line Interface usuário.
No IAM (Não recomendado)	Use credenciais de curto prazo para acessar a AWS.	Seguindo as instruções em Criar o seu primeiro usuário administrador e um grupo de usuários do IAM no Guia do usuário do IAM.	Para configurar o acesso programático, consulte Gerenciamento de chaves de acesso de usuários do IAM no Guia do usuário do IAM.

Etapa 2: Configurar permissões para o IDT

Nesta etapa, configure as permissões que o IDT for AWS IoT Greengrass V2 usa para executar testes e coletar dados de uso do IDT. Você pode usar o AWS Management Consoleor AWS Command Line Interface (AWS CLI) para criar uma política do IAM e um usuário de teste para o IDT

e, em seguida, anexar políticas ao usuário. Se você já criou um usuário de teste para o IDT, vá para. Configure seu dispositivo para executar testes de IDT

Como configurar permissões para o IDT (console)

1. Faça login no console do IAM.
2. Crie uma política gerenciada pelo cliente que conceda permissões para criar funções com permissões específicas.

   a. No painel de navegação, selecione Políticas e, em seguida, Criar política.

   b. Se você não estiver usando PreInstalled, na guia JSON, substitua o conteúdo do espaço reservado pela política a seguir. Se você estiver usando PreInstalled, vá para a etapa a seguir.

```
{
    "Version":"2012-10-17",
    "Statement":[
    {
      "Sid":"passRoleForResources",
      "Effect":"Allow",
      "Action":"iam:PassRole",
      "Resource":"arn:aws:iam::*:role/idt-*",
      "Condition":{
        "StringEquals":{
          "iam:PassedToService":[
            "iot.amazonaws.com",
            "lambda.amazonaws.com",
            "greengrass.amazonaws.com"
          ]
        }
      }
    },
    {
      "Sid":"lambdaResources",
      "Effect":"Allow",
      "Action":[
        "lambda:CreateFunction",
        "lambda:PublishVersion",
        "lambda:DeleteFunction",
        "lambda:GetFunction"
      ],
      "Resource":[
```

```
        "arn:aws:lambda:*:*:function:idt-*"
      ]
    },
    {
      "Sid":"iotResources",
      "Effect":"Allow",
      "Action":[
        "iot:CreateThing",
        "iot:DeleteThing",
        "iot:DescribeThing",
        "iot:CreateThingGroup",
        "iot:DeleteThingGroup",
        "iot:DescribeThingGroup",
        "iot:AddThingToThingGroup",
        "iot:RemoveThingFromThingGroup",
        "iot:AttachThingPrincipal",
        "iot:DetachThingPrincipal",
        "iot:UpdateCertificate",
        "iot:DeleteCertificate",
        "iot:CreatePolicy",
        "iot:AttachPolicy",
        "iot:DetachPolicy",
        "iot:DeletePolicy",
        "iot:GetPolicy",
        "iot:Publish",
        "iot:TagResource",
        "iot:ListThingPrincipals",
        "iot:ListAttachedPolicies",
        "iot:ListTargetsForPolicy",
        "iot:ListThingGroupsForThing",
        "iot:ListThingsInThingGroup",
        "iot:CreateJob",
        "iot:DescribeJob",
        "iot:DescribeJobExecution",
        "iot:CancelJob"
      ],
      "Resource":[
        "arn:aws:iot:*:*:thing/idt-*",
        "arn:aws:iot:*:*:thinggroup/idt-*",
        "arn:aws:iot:*:*:policy/idt-*",
        "arn:aws:iot:*:*:cert/*",
        "arn:aws:iot:*:*:topic/idt-*",
        "arn:aws:iot:*:*:job/*"
      ]
```

```
    },
    {
      "Sid":"s3Resources",
      "Effect":"Allow",
      "Action":[
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObjectVersion",
        "s3:DeleteObject",
        "s3:CreateBucket",
        "s3:ListBucket",
        "s3:ListBucketVersions",
        "s3:DeleteBucket",
        "s3:PutObjectTagging",
        "s3:PutBucketTagging"
      ],
      "Resource":"arn:aws:s3::*:idt-*"
    },
    {
      "Sid":"roleAliasResources",
      "Effect":"Allow",
      "Action":[
        "iot:CreateRoleAlias",
        "iot:DescribeRoleAlias",
        "iot:DeleteRoleAlias",
        "iot:TagResource",
        "iam:GetRole"
      ],
      "Resource":[
        "arn:aws:iot:*:*:rolealias/idt-*",
        "arn:aws:iam::*:role/idt-*"
      ]
    },
    {
      "Sid":"idtExecuteAndCollectMetrics",
      "Effect":"Allow",
      "Action":[
        "iot-device-tester:SendMetrics",
        "iot-device-tester:SupportedVersion",
        "iot-device-tester:LatestIdt",
        "iot-device-tester:CheckVersion",
        "iot-device-tester:DownloadTestSuite"
      ],
      "Resource":"*"
```

```
    },
    {
      "Sid":"genericResources",
      "Effect":"Allow",
      "Action":[
        "greengrass:*",
        "iot:GetThingShadow",
        "iot:UpdateThingShadow",
        "iot:ListThings",
        "iot:DescribeEndpoint",
        "iot:CreateKeysAndCertificate"
      ],
      "Resource":"*"
    },
    {
      "Sid":"iamResourcesUpdate",
      "Effect":"Allow",
      "Action":[
        "iam:CreateRole",
        "iam:DeleteRole",
        "iam:CreatePolicy",
        "iam:DeletePolicy",
        "iam:AttachRolePolicy",
        "iam:DetachRolePolicy",
        "iam:TagRole",
        "iam:TagPolicy",
        "iam:GetPolicy",
        "iam:ListAttachedRolePolicies",
        "iam:ListEntitiesForPolicy"
      ],
      "Resource":[
        "arn:aws:iam::*:role/idt-*",
        "arn:aws:iam::*:policy/idt-*"
      ]
    }
  ]
}
```

c. Se você estiver usando PreInstalled, na guia JSON, substitua o conteúdo do espaço reservado pela política a seguir. Certifique-se de que você:

- Substitua *ThingName* e *ThingGroup* na `iotResources` declaração pelo nome da coisa e pelo grupo da coisa que foram criados durante a instalação do Greengrass em seu dispositivo em teste (DUT) para adicionar permissões.
- Substitua o *PassRole e o RoleAlias* na instrução e na instrução `roleAliasResources` pelas funções que foram criadas durante `passRoleForResources` a instalação do Greengrass em seu DUT.

```
{
    "Version":"2012-10-17",
    "Statement":[
    {
      "Sid":"passRoleForResources",
      "Effect":"Allow",
      "Action":"iam:PassRole",
      "Resource":"arn:aws:iam::*:role/passRole",
      "Condition":{
        "StringEquals":{
          "iam:PassedToService":[
            "iot.amazonaws.com",
            "lambda.amazonaws.com",
            "greengrass.amazonaws.com"
          ]
        }
      }
    },
    {
      "Sid":"lambdaResources",
      "Effect":"Allow",
      "Action":[
        "lambda:CreateFunction",
        "lambda:PublishVersion",
        "lambda:DeleteFunction",
        "lambda:GetFunction"
      ],
      "Resource":[
        "arn:aws:lambda:*:*:function:idt-*"
      ]
    },
    {
      "Sid":"iotResources",
```

```
      "Effect":"Allow",
      "Action":[
        "iot:CreateThing",
        "iot:DeleteThing",
        "iot:DescribeThing",
        "iot:CreateThingGroup",
        "iot:DeleteThingGroup",
        "iot:DescribeThingGroup",
        "iot:AddThingToThingGroup",
        "iot:RemoveThingFromThingGroup",
        "iot:AttachThingPrincipal",
        "iot:DetachThingPrincipal",
        "iot:UpdateCertificate",
        "iot:DeleteCertificate",
        "iot:CreatePolicy",
        "iot:AttachPolicy",
        "iot:DetachPolicy",
        "iot:DeletePolicy",
        "iot:GetPolicy",
        "iot:Publish",
        "iot:TagResource",
        "iot:ListThingPrincipals",
        "iot:ListAttachedPolicies",
        "iot:ListTargetsForPolicy",
        "iot:ListThingGroupsForThing",
        "iot:ListThingsInThingGroup",
        "iot:CreateJob",
        "iot:DescribeJob",
        "iot:DescribeJobExecution",
        "iot:CancelJob"
      ],
      "Resource":[
        "arn:aws:iot:*:*:thing/thingName",
        "arn:aws:iot:*:*:thinggroup/thingGroup",
        "arn:aws:iot:*:*:policy/idt-*",
        "arn:aws:iot:*:*:cert/*",
        "arn:aws:iot:*:*:topic/idt-*",
        "arn:aws:iot:*:*:job/*"
      ]
    },
    {
      "Sid":"s3Resources",
      "Effect":"Allow",
      "Action":[
```

```
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObjectVersion",
        "s3:DeleteObject",
        "s3:CreateBucket",
        "s3:ListBucket",
        "s3:ListBucketVersions",
        "s3:DeleteBucket",
        "s3:PutObjectTagging",
        "s3:PutBucketTagging"
      ],
      "Resource":"arn:aws:s3::*:idt-*"
    },
    {
      "Sid":"roleAliasResources",
      "Effect":"Allow",
      "Action":[
        "iot:CreateRoleAlias",
        "iot:DescribeRoleAlias",
        "iot:DeleteRoleAlias",
        "iot:TagResource",
        "iam:GetRole"
      ],
      "Resource":[
        "arn:aws:iot:*:*:rolealias/roleAlias",
        "arn:aws:iam::*:role/idt-*"
      ]
    },
    {
      "Sid":"idtExecuteAndCollectMetrics",
      "Effect":"Allow",
      "Action":[
        "iot-device-tester:SendMetrics",
        "iot-device-tester:SupportedVersion",
        "iot-device-tester:LatestIdt",
        "iot-device-tester:CheckVersion",
        "iot-device-tester:DownloadTestSuite"
      ],
      "Resource":"*"
    },
    {
      "Sid":"genericResources",
      "Effect":"Allow",
      "Action":[
```

```
        "greengrass:*",
        "iot:GetThingShadow",
        "iot:UpdateThingShadow",
        "iot:ListThings",
        "iot:DescribeEndpoint",
        "iot:CreateKeysAndCertificate"
      ],
      "Resource":"*"
    },
    {
      "Sid":"iamResourcesUpdate",
      "Effect":"Allow",
      "Action":[
        "iam:CreateRole",
        "iam:DeleteRole",
        "iam:CreatePolicy",
        "iam:DeletePolicy",
        "iam:AttachRolePolicy",
        "iam:DetachRolePolicy",
        "iam:TagRole",
        "iam:TagPolicy",
        "iam:GetPolicy",
        "iam:ListAttachedRolePolicies",
        "iam:ListEntitiesForPolicy"
      ],
      "Resource":[
        "arn:aws:iam::*:role/idt-*",
        "arn:aws:iam::*:policy/idt-*"
      ]
    }
  ]
}
```

> **Note**
>
> Se você quiser usar uma função personalizada do IAM como função de troca de tokens para seu dispositivo em teste, certifique-se de atualizar a `roleAliasResources` declaração e a `passRoleForResources` declaração em sua política para permitir seu recurso de função personalizada do IAM.

d. Escolha Review policy (Revisar política).

e. Em Name (Nome), insira **`IDTGreengrassIAMPermissions`**. Em Summary (Resumo), revise as permissões concedidas pela política.

f. Selecione Create policy (Criar política).

3. Crie um usuário do IAM e anexe as permissões exigidas pelo IDT para o AWS IoT Greengrass.

   a. Criar um usuário do IAM. Siga as etapas de 1 a 5 em Criando usuários do IAM (console) no Guia do usuário do IAM.

   b. Anexe as permissões ao usuário do IAM:

      i. Na página Set permissions (Definir permissões), selecione Attach existing policies to user directly (Anexar políticas existentes diretamente ao usuário).

      ii. Procure a política IDTGreengrassIAMPermissions criada na etapa anterior. Marque a caixa de seleção.

   c. Selecione Next: Tags (Próximo: tags).

   d. Selecione Next: Review (Próximo: revisar) para exibir um resumo das suas escolhas.

   e. Selecione Criar usuário.

   f. Para exibir as chaves de acesso do usuário (IDs de chave de acesso e chaves de acesso secretas), selecione Show (Mostrar) ao lado da senha e da chave de acesso. Para salvar as chaves de acesso, selecione Download.csv (Fazer download do .csv) e salve o arquivo em um local seguro. Use essas informações posteriormente para configurar o arquivo de credenciais da AWS .

4. Próxima etapa: configure o dispositivo físico.

Como configurar permissões para o IDT (AWS CLI)

1. No seu computador, instale e configure o, AWS CLI se ainda não estiver instalado. Siga as etapas em Installing the (Instalando a) AWS CLI no Guia do usuário do AWS Command Line Interface .

   > **Note**
   >
   > AWS CLI É uma ferramenta de código aberto que você pode usar para interagir com AWS serviços do seu shell de linha de comando.

2. Crie uma política gerenciada pelo cliente que conceda permissões para gerenciar IDT e funções do AWS IoT Greengrass .

a. Se você não estiver usando PreInstalled, abra um editor de texto e salve o conteúdo da política a seguir em um arquivo JSON. Se você estiver usando PreInstalled, vá para a etapa a seguir.

```
{
    "Version":"2012-10-17",
    "Statement":[
    {
      "Sid":"passRoleForResources",
      "Effect":"Allow",
      "Action":"iam:PassRole",
      "Resource":"arn:aws:iam::*:role/idt-*",
      "Condition":{
        "StringEquals":{
          "iam:PassedToService":[
            "iot.amazonaws.com",
            "lambda.amazonaws.com",
            "greengrass.amazonaws.com"
          ]
        }
      }
    },
    {
      "Sid":"lambdaResources",
      "Effect":"Allow",
      "Action":[
        "lambda:CreateFunction",
        "lambda:PublishVersion",
        "lambda:DeleteFunction",
        "lambda:GetFunction"
      ],
      "Resource":[
        "arn:aws:lambda:*:*:function:idt-*"
      ]
    },
    {
      "Sid":"iotResources",
      "Effect":"Allow",
      "Action":[
        "iot:CreateThing",
        "iot:DeleteThing",
        "iot:DescribeThing",
```

```
        "iot:CreateThingGroup",
        "iot:DeleteThingGroup",
        "iot:DescribeThingGroup",
        "iot:AddThingToThingGroup",
        "iot:RemoveThingFromThingGroup",
        "iot:AttachThingPrincipal",
        "iot:DetachThingPrincipal",
        "iot:UpdateCertificate",
        "iot:DeleteCertificate",
        "iot:CreatePolicy",
        "iot:AttachPolicy",
        "iot:DetachPolicy",
        "iot:DeletePolicy",
        "iot:GetPolicy",
        "iot:Publish",
        "iot:TagResource",
        "iot:ListThingPrincipals",
        "iot:ListAttachedPolicies",
        "iot:ListTargetsForPolicy",
        "iot:ListThingGroupsForThing",
        "iot:ListThingsInThingGroup",
        "iot:CreateJob",
        "iot:DescribeJob",
        "iot:DescribeJobExecution",
        "iot:CancelJob"
      ],
      "Resource":[
        "arn:aws:iot:*:*:thing/idt-*",
        "arn:aws:iot:*:*:thinggroup/idt-*",
        "arn:aws:iot:*:*:policy/idt-*",
        "arn:aws:iot:*:*:cert/*",
        "arn:aws:iot:*:*:topic/idt-*",
        "arn:aws:iot:*:*:job/*"
      ]
    },
    {
      "Sid":"s3Resources",
      "Effect":"Allow",
      "Action":[
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObjectVersion",
        "s3:DeleteObject",
        "s3:CreateBucket",
```

```
        "s3:ListBucket",
        "s3:ListBucketVersions",
        "s3:DeleteBucket",
        "s3:PutObjectTagging",
        "s3:PutBucketTagging"
      ],
      "Resource":"arn:aws:s3::*:idt-*"
    },
    {
      "Sid":"roleAliasResources",
      "Effect":"Allow",
      "Action":[
        "iot:CreateRoleAlias",
        "iot:DescribeRoleAlias",
        "iot:DeleteRoleAlias",
        "iot:TagResource",
        "iam:GetRole"
      ],
      "Resource":[
        "arn:aws:iot:*:*:rolealias/idt-*",
        "arn:aws:iam::*:role/idt-*"
      ]
    },
    {
      "Sid":"idtExecuteAndCollectMetrics",
      "Effect":"Allow",
      "Action":[
        "iot-device-tester:SendMetrics",
        "iot-device-tester:SupportedVersion",
        "iot-device-tester:LatestIdt",
        "iot-device-tester:CheckVersion",
        "iot-device-tester:DownloadTestSuite"
      ],
      "Resource":"*"
    },
    {
      "Sid":"genericResources",
      "Effect":"Allow",
      "Action":[
        "greengrass:*",
        "iot:GetThingShadow",
        "iot:UpdateThingShadow",
        "iot:ListThings",
        "iot:DescribeEndpoint",
```

```
        "iot:CreateKeysAndCertificate"
      ],
      "Resource":"*"
    },
    {
      "Sid":"iamResourcesUpdate",
      "Effect":"Allow",
      "Action":[
        "iam:CreateRole",
        "iam:DeleteRole",
        "iam:CreatePolicy",
        "iam:DeletePolicy",
        "iam:AttachRolePolicy",
        "iam:DetachRolePolicy",
        "iam:TagRole",
        "iam:TagPolicy",
        "iam:GetPolicy",
        "iam:ListAttachedRolePolicies",
        "iam:ListEntitiesForPolicy"
      ],
      "Resource":[
        "arn:aws:iam::*:role/idt-*",
        "arn:aws:iam::*:policy/idt-*"
      ]
    }
  ]
}
```

b. Se você estiver usando PreInstalled, abra um editor de texto e salve o conteúdo da política a seguir em um arquivo JSON. Certifique-se de que você:

- Substitua *`ThingName`* e *`ThingGroup`* na `iotResources` declaração que foi criada durante a instalação do Greengrass em seu dispositivo em teste (DUT) para adicionar permissões.
- Substitua o *`PassRole e o RoleAlias`* na instrução e na instrução `roleAliasResources` pelas funções que foram criadas durante `passRoleForResources` a instalação do Greengrass em seu DUT.

```
{
    "Version":"2012-10-17",
    "Statement":[
```

```
    {
      "Sid":"passRoleForResources",
      "Effect":"Allow",
      "Action":"iam:PassRole",
      "Resource":"arn:aws:iam::*:role/passRole",
      "Condition":{
        "StringEquals":{
          "iam:PassedToService":[
            "iot.amazonaws.com",
            "lambda.amazonaws.com",
            "greengrass.amazonaws.com"
          ]
        }
      }
    },
    {
      "Sid":"lambdaResources",
      "Effect":"Allow",
      "Action":[
        "lambda:CreateFunction",
        "lambda:PublishVersion",
        "lambda:DeleteFunction",
        "lambda:GetFunction"
      ],
      "Resource":[
        "arn:aws:lambda:*:*:function:idt-*"
      ]
    },
    {
      "Sid":"iotResources",
      "Effect":"Allow",
      "Action":[
        "iot:CreateThing",
        "iot:DeleteThing",
        "iot:DescribeThing",
        "iot:CreateThingGroup",
        "iot:DeleteThingGroup",
        "iot:DescribeThingGroup",
        "iot:AddThingToThingGroup",
        "iot:RemoveThingFromThingGroup",
        "iot:AttachThingPrincipal",
        "iot:DetachThingPrincipal",
        "iot:UpdateCertificate",
        "iot:DeleteCertificate",
```

```
        "iot:CreatePolicy",
        "iot:AttachPolicy",
        "iot:DetachPolicy",
        "iot:DeletePolicy",
        "iot:GetPolicy",
        "iot:Publish",
        "iot:TagResource",
        "iot:ListThingPrincipals",
        "iot:ListAttachedPolicies",
        "iot:ListTargetsForPolicy",
        "iot:ListThingGroupsForThing",
        "iot:ListThingsInThingGroup",
        "iot:CreateJob",
        "iot:DescribeJob",
        "iot:DescribeJobExecution",
        "iot:CancelJob"
      ],
      "Resource":[
        "arn:aws:iot:*:*:thing/thingName",
        "arn:aws:iot:*:*:thinggroup/thingGroup",
        "arn:aws:iot:*:*:policy/idt-*",
        "arn:aws:iot:*:*:cert/*",
        "arn:aws:iot:*:*:topic/idt-*",
        "arn:aws:iot:*:*:job/*"
      ]
    },
    {
      "Sid":"s3Resources",
      "Effect":"Allow",
      "Action":[
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObjectVersion",
        "s3:DeleteObject",
        "s3:CreateBucket",
        "s3:ListBucket",
        "s3:ListBucketVersions",
        "s3:DeleteBucket",
        "s3:PutObjectTagging",
        "s3:PutBucketTagging"
      ],
      "Resource":"arn:aws:s3::*:idt-*"
    },
    {
```

```
      "Sid":"roleAliasResources",
      "Effect":"Allow",
      "Action":[
        "iot:CreateRoleAlias",
        "iot:DescribeRoleAlias",
        "iot:DeleteRoleAlias",
        "iot:TagResource",
        "iam:GetRole"
      ],
      "Resource":[
        "arn:aws:iot:*:*:rolealias/roleAlias",
        "arn:aws:iam::*:role/idt-*"
      ]
    },
    {
      "Sid":"idtExecuteAndCollectMetrics",
      "Effect":"Allow",
      "Action":[
        "iot-device-tester:SendMetrics",
        "iot-device-tester:SupportedVersion",
        "iot-device-tester:LatestIdt",
        "iot-device-tester:CheckVersion",
        "iot-device-tester:DownloadTestSuite"
      ],
      "Resource":"*"
    },
    {
      "Sid":"genericResources",
      "Effect":"Allow",
      "Action":[
        "greengrass:*",
        "iot:GetThingShadow",
        "iot:UpdateThingShadow",
        "iot:ListThings",
        "iot:DescribeEndpoint",
        "iot:CreateKeysAndCertificate"
      ],
      "Resource":"*"
    },
    {
      "Sid":"iamResourcesUpdate",
      "Effect":"Allow",
      "Action":[
        "iam:CreateRole",
```

```
        "iam:DeleteRole",
        "iam:CreatePolicy",
        "iam:DeletePolicy",
        "iam:AttachRolePolicy",
        "iam:DetachRolePolicy",
        "iam:TagRole",
        "iam:TagPolicy",
        "iam:GetPolicy",
        "iam:ListAttachedRolePolicies",
        "iam:ListEntitiesForPolicy"
      ],
      "Resource":[
        "arn:aws:iam::*:role/idt-*",
        "arn:aws:iam::*:policy/idt-*"
      ]
    }
  ]
}
```

> **Note**
>
> Se você quiser usar uma função personalizada do IAM como função de troca de tokens para seu dispositivo em teste, certifique-se de atualizar a `roleAliasResources` declaração e a `passRoleForResources` declaração em sua política para permitir seu recurso de função personalizada do IAM.

c. Execute o comando a seguir para criar uma política gerenciada pelo cliente chamada`IDTGreengrassIAMPermissions`. *`policy.json`*Substitua pelo caminho completo para o arquivo JSON que você criou na etapa anterior.

```
aws iam create-policy --policy-name IDTGreengrassIAMPermissions --policy-
document file://policy.json
```

3. Crie um usuário do IAM e anexe as permissões exigidas pelo IDT para o AWS IoT Greengrass.

a. Criar um usuário do IAM. Neste exemplo de configuração, o usuário é nomeado `IDTGreengrassUser`.

```
aws iam create-user --user-name IDTGreengrassUser
```

b. Anexe a política `IDTGreengrassIAMPermissions` criada na etapa 2 ao seu usuário do IAM. <account-id>Substitua o comando pelo ID do seu Conta da AWS.

```
aws iam attach-user-policy --user-name IDTGreengrassUser --policy-arn
 arn:aws:iam::<account-id>:policy/IDTGreengrassIAMPermissions
```

4. Crie uma chave de acesso secreta para o usuário.

```
aws iam create-access-key --user-name IDTGreengrassUser
```

Armazene a saída em um local seguro. Você usa essas informações posteriormente para configurar seu arquivo de AWS credenciais.

5. Próxima etapa: configure o dispositivo físico.

AWS IoT Device Tester permissões

As políticas a seguir descrevem AWS IoT Device Tester as permissões.

AWS IoT Device Tester requer essas permissões para verificação de versão e recursos de atualização automática.

- `iot-device-tester:SupportedVersion`

  Concede AWS IoT Device Tester permissão para buscar a lista de produtos compatíveis, suítes de teste e versões do IDT.
- `iot-device-tester:LatestIdt`

  Concede AWS IoT Device Tester permissão para obter a versão mais recente do IDT disponível para download.
- `iot-device-tester:CheckVersion`

  Concede AWS IoT Device Tester permissão para verificar a compatibilidade de versões do IDT, suítes de teste e produtos.
- `iot-device-tester:DownloadTestSuite`

  Concede AWS IoT Device Tester permissão para baixar atualizações de suítes de teste.

AWS IoT Device Tester também usa a seguinte permissão para relatórios de métricas opcionais:

- `iot-device-tester:SendMetrics`

  Concede permissão AWS para coletar métricas sobre o uso AWS IoT Device Tester interno. Se essa permissão for omitida, essas métricas não serão coletadas.

Configure seu dispositivo para executar testes de IDT

Para permitir que o IDT execute testes de qualificação de dispositivos, você deve configurar seu computador host para acessar seu dispositivo e configurar as permissões de usuário em seu dispositivo.

Instale o Java no computador host

A partir do IDT v4.2.0, os testes de qualificação opcionais AWS IoT Greengrass exigem que o Java seja executado.

Você pode usar o Java versão 8 ou superior. Recomendamos que você use as versões de suporte de longo prazo do Amazon Corretto ou do OpenJDK. É necessária a versão 8 ou superior.

Configurar o computador host para acessar o dispositivo em teste

O ITD é executado em seu computador host e deve ser capaz de usar o SSH para se conectar ao seu dispositivo. Há duas opções para permitir que o IDT obtenha acesso SSH aos dispositivos em teste:

1. Siga as instruções aqui para criar um par de chaves SSH e autorizar sua chave a fazer login no dispositivo em teste sem especificar uma senha.
2. Forneça um nome de usuário e uma senha para cada dispositivo no arquivo `device.json`. Para ter mais informações, consulte Configurar device.json.

Você pode usar qualquer implementação SSL para criar uma chave SSH. As instruções a seguir mostram como usar o SSH-KEYGEN ou PuTTYgen (para Windows). Se você estiver usando outra implementação de SSL, consulte a documentação para essa implementação.

O IDT usa chaves SSH para autenticar com o dispositivo em teste.

Para criar uma chave SSH com SSH-KEYGEN

1. Crie uma chave SSH.

Você pode usar o comando ssh-keygen Open SSH para criar um par de chaves SSH. Se você já tem um par de chaves SSH em seu computador host, é uma prática recomendada criar um par de chaves SSH especificamente para IDT. Dessa forma, depois de concluir o teste, o computador host não poderá mais se conectar ao dispositivo sem inserir uma senha. Ele também permite que você restrinja o acesso ao dispositivo remoto apenas para aqueles que precisam.

> **Note**
>
> O Windows não tem um cliente SSH instalado. Para obter informações sobre como instalar um cliente SSH no Windows, consulte Fazer download do software cliente SSH.

O comando ssh-keygen solicita que você informe um nome e caminho para armazenar o par de chaves. Por padrão, os arquivos de pares de chaves são nomeados como `id_rsa` (chave privada) e `id_rsa.pub` (chave pública). No macOS e no Linux, o local padrão desses arquivos é `~/.ssh/`. No Windows, o local padrão é `C:\Users\`*`<user-name>`*`\.ssh`.

Quando solicitado, insira uma frase-chave para proteger sua chave SSH. Para obter mais informações, consulte Gerar uma chave SSH.

2. Adição de chaves SSH autorizadas ao seu dispositivo em teste.

   O ITD deve usar sua chave privada SSH para fazer login no seu dispositivo em teste. Para autorizar sua chave privada SSH a fazer login no seu dispositivo em teste, use o comando ssh-copy-id do seu computador host. Esse comando adiciona sua chave pública ao arquivo `~/.ssh/authorized_keys` no seu dispositivo em teste. Por exemplo: .

   $ ssh-copy-id ***<remote-ssh-user>***@***<remote-device-ip>***

   Onde *`remote-ssh-user`*está o nome de usuário usado para fazer login no seu dispositivo em teste e *`remote-device-ip`*é o endereço IP do dispositivo em teste para executar testes. Por exemplo: .

   ssh-copy-id pi@192.168.1.5

   Quando solicitado, insira a senha para o nome de usuário especificado no comando ssh-copy-id.

ssh-copy-id supõe que a chave pública chama-se `id_rsa.pub` e está armazenada no local padrão (`~/.ssh/` no macOS e no Linux, e `C:\Users\<user-name>\.ssh` no Windows). Se você atribuiu à chave pública um nome diferente ou armazenou em um local diferente, você deve especificar o caminho para sua chave pública SSH usando a opção -i para ssh-copy-id (por exemplo, ssh-copy-id -i ~/my/path/myKey.pub). Para obter mais informações sobre a criação de chaves SSH e a cópia de chaves públicas, consulte SSH-COPY-ID.

Para criar uma chave SSH usando o PuTTYgen (somente Windows)

1. Verifique se o servidor e o cliente OpenSSH estão instalados no dispositivo em teste. Para obter mais informações, consulte OpenSSH.
2. Instale o PuTTYgen no dispositivo em teste.
3. Abra o PuTTYgen.
4. Selecione Generate (Gerar) e mova o cursor do mouse dentro da caixa para gerar uma chave privada.
5. No menu Conversions (Conversões), selecione Export OpenSSH key (Exportar chave OpenSSH) e salve a chave privada com uma extensão de arquivo `.pem`.
6. Adicione a chave pública ao arquivo `/home/<user>/.ssh/authorized_keys` no dispositivo em teste.

   a. Copie o texto da chave pública da janela PuTTYgen.

   b. Use o PuTTY para criar uma sessão no dispositivo em teste.

      i. Em um prompt de comando ou janela Windows Powershell, execute o seguinte comando:

         C:/**<path-to-putty>**/putty.exe -ssh **<user>**@**<dut-ip-address>**

      ii. Quando solicitado, insira a senha do dispositivo.

      iii. Use vi ou outro editor de texto para anexar a chave pública ao arquivo `/home/<user>/.ssh/authorized_keys` no dispositivo em teste.
7. Atualize o arquivo `device.json` com o nome de usuário, o endereço IP e o caminho para o arquivo da chave privada que você acabou de salvar no computador host para cada dispositivo em teste. Para ter mais informações, consulte the section called “Configurar device.json”. Você deve fornecer o caminho completo e o nome do arquivo para a chave privada e usar

barras ('/'). Por exemplo, para o caminho do Windows `C:\DT\privatekey.pem`, use `C:/DT/privatekey.pem` no arquivo `device.json`.

Configurar credenciais de usuário para dispositivos Windows

Para qualificar um dispositivo baseado em Windows, você deve configurar as credenciais do usuário na LocalSystem conta do dispositivo em teste para os seguintes usuários:

- O usuário padrão do Greengrass ()`ggc_user`.
- O usuário que você usa para se conectar ao dispositivo em teste. Você configura esse usuário no `device.json`arquivo.

Você deve criar cada usuário na LocalSystem conta no dispositivo em teste e, em seguida, armazenar o nome de usuário e a senha do usuário na instância do Credential Manager da LocalSystem conta.

Para configurar usuários em dispositivos Windows

1. Abra o prompt de comando do Windows (`cmd.exe`) como administrador.
2. Crie os usuários na LocalSystem conta no dispositivo Windows. Execute o comando a seguir para cada usuário que você deseja criar. *Para o usuário padrão do Greengrass, substitua nome de usuário por.* `ggc_user` *Substitua a senha* por uma senha segura.

   ```
   net user /add user-name password
   ```

3. Baixe e instale o PsExecutilitário da Microsoft no dispositivo.
4. Use o PsExec utilitário para armazenar o nome de usuário e a senha do usuário padrão na instância do Credential Manager da LocalSystem conta.

   Execute o comando a seguir para cada usuário que você deseja configurar no Credential Manager. *Para o usuário padrão do Greengrass, substitua nome de usuário por.* `ggc_user` Substitua a *senha* pela senha do usuário que você definiu anteriormente.

   ```
   psexec -s cmd /c cmdkey /generic:user-name /user:user-name /pass:password
   ```

Se for PsExec License Agreementaberto, escolha Acceptconcordar com a licença e execute o comando.

> **Note**
>
> Em dispositivos Windows, a LocalSystem conta executa o núcleo Greengrass, e você deve usar o PsExec utilitário para armazenar as informações do usuário na conta. LocalSystem O uso do aplicativo Credential Manager armazena essas informações na conta do Windows do usuário atualmente conectado, em vez da LocalSystem conta.

Configurar permissões de usuário no dispositivo

O ITD executa operações em vários diretórios e arquivos em um dispositivo em teste. Algumas dessas operações exigem permissões elevadas (usando sudo). Para automatizar essas operações, o IDT for AWS IoT Greengrass V2 deve ser capaz de executar comandos com sudo sem ser solicitado a fornecer uma senha.

Siga estas etapas no dispositivo em teste para permitir o acesso ao sudo sem receber uma solicitação de senha.

> **Note**
>
> `username` refere-se ao usuário SSH usado pelo IDT para acessar o dispositivo em teste.

Para adicionar o usuário ao grupo sudo

1. No dispositivo em teste, execute `sudo usermod -aG sudo` *`<username>`*.
2. Saia e faça login novamente para que as alterações entrem em vigor.
3. Para verificar se o nome de usuário foi adicionado com êxito, execute sudo echo test. Se você não receber uma solicitação de senha, o usuário foi configurado corretamente.
4. Abra o arquivo `/etc/sudoers` e adicione a linha a seguir ao final do arquivo:

   `<ssh-username>` `ALL=(ALL) NOPASSWD: ALL`

Configurar uma função personalizada de troca de tokens

Você pode optar por usar uma função personalizada do IAM como a função de troca de tokens que o dispositivo em teste presume interagir com os AWS recursos. Para aprender a criar perfis do IAM, consulte Como criar perfis do IAM no Guia do usuário do IAM.

Você deve atender aos seguintes requisitos para permitir que o IDT use sua função personalizada do IAM. É altamente recomendável que você adicione somente as ações políticas mínimas necessárias a essa função.

- O arquivo de configuração userdata.json deve ser atualizado para definir o parâmetro como. `GreengrassV2TokenExchangeRole` `true`
- A função personalizada do IAM deve ser configurada com a seguinte política de confiança mínima:

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Principal":{
            "Service":[
               "credentials.iot.amazonaws.com",
               "lambda.amazonaws.com",
               "sagemaker.amazonaws.com"
            ]
         },
         "Action":"sts:AssumeRole"
      }
   ]
}
```

- O papel personalizado do IAM deve ser configurado com a seguinte política de permissões mínimas:

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "iot:DescribeCertificate",
            "logs:CreateLogGroup",
```

```
            "logs:CreateLogStream",
            "logs:PutLogEvents",
            "logs:DescribeLogStreams",
            "iot:Connect",
            "iot:Publish",
            "iot:Subscribe",
            "iot:Receive",
            "iot:ListThingPrincipals",
            "iot:GetThingShadow",
            "iot:UpdateThingShadow",
            "s3:GetBucketLocation",
            "s3:GetObject",
            "s3:PutObject",
            "s3:AbortMultipartUpload",
            "s3:ListMultipartUploadParts"
         ],
         "Resource":"*"
      }
   ]
}
```

- O nome da função personalizada do IAM deve corresponder ao recurso de função do IAM que você especifica nas permissões do IAM para o usuário de teste. Por padrão, a política de usuário de teste permite o acesso às funções do IAM que têm o `idt-` prefixo em seus nomes de função. Se o nome da sua função do IAM não usar esse prefixo, adicione o `arn:aws:iam::*:role/`*`custom-iam-role-name`* recurso à `roleAliasResources` declaração e a `passRoleForResources` declaração na sua política de usuário de teste, conforme mostrado nos exemplos a seguir:

  Example Instrução **`passRoleForResources`**

```
{
   "Sid":"passRoleForResources",
   "Effect":"Allow",
   "Action":"iam:PassRole",
   "Resource":"arn:aws:iam::*:role/custom-iam-role-name",
   "Condition":{
      "StringEquals":{
         "iam:PassedToService":[
            "iot.amazonaws.com",
            "lambda.amazonaws.com",
            "greengrass.amazonaws.com"
```

```
            ]
        }
    }
}
```

Example Instrução **`roleAliasResources`**

```
{
    "Sid":"roleAliasResources",
    "Effect":"Allow",
    "Action":[
        "iot:CreateRoleAlias",
        "iot:DescribeRoleAlias",
        "iot:DeleteRoleAlias",
        "iot:TagResource",
        "iam:GetRole"
    ],
    "Resource":[
        "arn:aws:iot:*:*:rolealias/idt-*",
        "arn:aws:iam::*:role/custom-iam-role-name"
    ]
}
```

Configurar seu dispositivo para testar atributos opcionais

Esta seção descreve os requisitos do dispositivo para executar testes de IDT para recursos opcionais do Docker e do Machine Learning (ML). Os recursos de ML são compatíveis somente com o IDT v4.9.3. Você deve garantir que seu dispositivo atenda a esses requisitos somente se quiser testar esses recursos. Caso contrário, avance para the section called “Defina as configurações do IDT”.

Tópicos

Requisitos de qualificação do Docker

O IDT for AWS IoT Greengrass V2 fornece testes de qualificação do Docker para validar se seus dispositivos podem usar o componente gerenciador de aplicativos Docker AWS fornecido para baixar

imagens de contêiner do Docker que você pode executar usando componentes de contêiner Docker personalizados. Para obter informações sobre a criação de componentes personalizados do Docker, consulteExecute um contêiner Docker.

Para executar testes de qualificação do Docker, seus dispositivos em teste devem atender aos seguintes requisitos para implantar o componente Docker Application Manager.

- Docker Engine 1.9.1 ou posterior instalado no dispositivo principal do Greengrass. A versão 20.10 é a versão mais recente verificada para funcionar com o software AWS IoT Greengrass Core. Você deve instalar o Docker diretamente no dispositivo principal antes de implantar componentes que executam contêineres do Docker.
- O daemon do Docker foi iniciado e executado no dispositivo principal antes de você implantar esse componente.
- O usuário do sistema que executa um componente de contêiner do Docker deve ter permissões de raiz ou administrador, ou você deve configurar o Docker para executá-lo como usuário não raiz ou não administrador.
  - Em dispositivos Linux, você pode adicionar um usuário ao `docker` grupo para chamar `docker` comandos sem`sudo`.
  - Em dispositivos Windows, você pode adicionar um usuário ao `docker-users` grupo para chamar `docker` comandos sem privilégios de administrador.

  Linux or Unix

  Para adicionar `ggc_user` ou o usuário não root que você usa para executar componentes de contêiner do Docker ao `docker` grupo, execute o comando a seguir.

  ```
  sudo usermod -aG docker ggc_user
  ```

  Para obter mais informações, consulte Gerenciar o Docker como usuário não root.

  Windows Command Prompt (CMD)

  Para adicionar `ggc_user` ou o usuário que você usa para executar componentes de contêiner do Docker ao `docker-users` grupo, execute o comando a seguir como administrador.

  ```
  net localgroup docker-users ggc_user /add
  ```

Windows PowerShell

Para adicionar `ggc_user` ou o usuário que você usa para executar componentes de contêiner do Docker ao `docker-users` grupo, execute o comando a seguir como administrador.

```
Add-LocalGroupMember -Group docker-users -Member ggc_user
```

Requisitos de qualificação de ML

> **Note**
>
> O recurso de aprendizado de máquina é suportado somente no IDT v4.9.3.

O IDT for AWS IoT Greengrass V2 fornece testes de qualificação de ML para validar se seus dispositivos podem usar os componentes de aprendizado AWS de máquina fornecidos para realizar inferência de ML localmente usando as estruturas Deep Learning Runtime ou Lite ML. TensorFlow Para obter mais informações sobre como executar inferência de ML em dispositivos Greengrass, consulte. Executar a inferência de machine learning

Para executar testes de qualificação de ML, seus dispositivos em teste devem atender aos seguintes requisitos para implantar os componentes de aprendizado de máquina.

- Nos principais dispositivos do Greengrass que executam o Amazon Linux 2 ou o Ubuntu 18.04, a GNU C Library (glibc) versão 2.27 ou posterior está instalada no dispositivo.
- Em dispositivos ARMv7L, como o Raspberry Pi, dependências do OpenCV-Python instaladas no dispositivo. Execute o comando a seguir para instalar as dependências.

```
sudo apt-get install libopenjp2-7 libilmbase23 libopenexr-dev libavcodec-dev
 libavformat-dev libswscale-dev libv4l-dev libgtk-3-0 libwebp-dev
```

- Os dispositivos Raspberry Pi que executam o Raspberry Pi OS Bullseye devem atender aos seguintes requisitos:
  - NumPy 1.22.4 ou posterior instalado no dispositivo. O Raspberry Pi OS Bullseye inclui uma versão anterior do NumPy, então você pode executar o seguinte comando para atualizar NumPy o dispositivo.

```
pip3 install --upgrade numpy
```

- A pilha de câmeras antiga ativada no dispositivo. O Raspberry Pi OS Bullseye inclui uma nova pilha de câmeras que é ativada por padrão e não é compatível, portanto, você deve habilitar a pilha de câmeras antiga.

  Para habilitar a pilha de câmeras antiga

  1. Execute o comando a seguir para abrir a ferramenta de configuração do Raspberry Pi.

     ```
     sudo raspi-config
     ```

  2. Selecione Opções de interface.
  3. Selecione Câmera antiga para ativar a pilha de câmeras antigas.
  4. Reinicie o Raspberry Pi.

Requisitos de qualificação do HSM

AWS IoT Greengrass fornece o componente provedor PKCS #11 para integração com o Módulo de Segurança de Hardware (HSM) PKCS no dispositivo. A configuração do HSM depende do seu dispositivo e do módulo HSM que você escolheu. Desde que a configuração esperada do HSM, conforme documentada nas configurações do IDT, seja fornecida, o IDT terá as informações necessárias para executar esse teste opcional de qualificação de recursos.

Defina as configurações de IDT para executar o pacote de AWS IoT Greengrass qualificação

Antes de executar os testes, você deve definir as configurações de AWS credenciais e dispositivos no computador host.

Configurar AWS credenciais em config.json

Você deve configurar suas credenciais de usuário do IAM no arquivo *<device_tester_extract_location>*`/configs/config.json`. Use as credenciais do usuário IDT for AWS IoT Greengrass V2 criado em. the section called “Crie e configure um Conta da AWS” Você pode especificar suas credenciais de uma das seguintes formas:

- Em um arquivo de credenciais

- Como variáveis de ambiente

Configurar AWS credenciais com um arquivo de credenciais

O IDT usa o mesmo arquivo de credenciais que a AWS CLI. Para obter mais informações, consulte Arquivos de configuração e credenciais.

O local do arquivo de credenciais varia de acordo com o sistema operacional que você está usando:

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

Adicione suas AWS credenciais ao `credentials` arquivo no seguinte formato:

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

Para configurar o IDT para AWS IoT Greengrass V2 para usar AWS as credenciais do seu `credentials` arquivo, edite o arquivo da seguinte `config.json` forma:

```
{
  "awsRegion": "region",
  "auth": {
    "method": "file",
    "credentials": {
      "profile": "default"
    }
  }
}
```

> **Note**
>
> Se você não usar o `default` AWS perfil, não se esqueça de alterar o nome do perfil no seu `config.json` arquivo. Para obter mais informações, consulte Perfis nomeados.

Configurar AWS credenciais com variáveis de ambiente

As variáveis de ambiente são variáveis mantidas pelo sistema operacional e usadas pelos comandos do sistema. Elas não serão salvas se você fechar a sessão SSH. O IDT para AWS IoT Greengrass V2 pode usar as variáveis de `AWS_SECRET_ACCESS_KEY` ambiente `AWS_ACCESS_KEY_ID` e para armazenar suas AWS credenciais.

Para definir essas variáveis no Linux, macOS ou Unix, use :export

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

Para definir essas variáveis no Windows, use :set

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

Para configurar o IDT para usar as variáveis de ambiente, edite a seção `auth` no seu arquivo `config.json`. Exemplo:

```
{
  "awsRegion": "region",
  "auth": {
    "method": "environment"
  }
}
```

Configurar device.json

> **Note**
>
> O IDT v4.9.3 oferece suporte ao teste dos recursos`sml`, e. `docker` `streamManagement` O IDT v4.9.4 e versões posteriores oferecem suporte a testes. `docker` Se você não quiser testar esses recursos, defina o valor correspondente como`no`.

Além das AWS credenciais, o IDT for AWS IoT Greengrass V2 precisa de informações sobre os dispositivos nos quais os testes são executados. As informações de exemplo seriam endereço IP, informações de login, sistema operacional e arquitetura da CPU.

Você deve fornecer essas informações usando o modelo `device.json` localizado em *<device_tester_extract_location>*`/configs/device.json`:

IDT v4.9.3

```
[
  {
    "id": "<pool-id>",
    "sku": "<sku>",
    "features": [
      {
        "name": "arch",
        "value": "x86_64 | armv6l | armv7l | aarch64"
      },
      {
        "name": "ml",
        "value": "dlr | tensorflowlite | dlr,tensorflowlite | no"
      },
      {
        "name": "docker",
        "value": "yes | no"
      },
      {
        "name": "streamManagement",
        "value": "yes | no"
      },
      {
        "name": "hsi",
        "value": "hsm | no"
      }
    ],
    "devices": [
      {
        "id": "<device-id>",
        "operatingSystem": "Linux | Windows",
        "connectivity": {
          "protocol": "ssh",
          "ip": "<ip-address>",
          "port": 22,
          "publicKeyPath": "<public-key-path>",
          "auth": {
            "method": "pki | password",
            "credentials": {
```

```
            "user": "<user-name>",
            "privKeyPath": "/path/to/private/key",
            "password": "<password>"
          }
        }
      }
    }
  ]
}
]
```

> **Note**
>
> Especifique `privKeyPath` somente se `method` estiver definido como `pki`.
> Especifique `password` somente se `method` estiver definido como `password`.

Todas as propriedades que contêm valores são obrigatórias, conforme descrito aqui:

`id`

Um ID alfanumérico definido pelo usuário que identifica uma coleção de dispositivos chamada de grupo de dispositivos. Os dispositivos que pertencem a um grupo devem ter hardware idêntico. Quando um conjunto de testes é executado, os dispositivos do grupo são usados para paralelizar a carga de trabalho. Vários dispositivos são usados para executar testes diferentes.

`sku`

Um valor alfanumérico que identifica exclusivamente o dispositivo em teste. A SKU é usada para rastrear as placas qualificadas.

> **Note**
>
> Se você quiser listar seu dispositivo no Catálogo de AWS Partner dispositivos, o SKU especificado aqui deverá corresponder ao SKU que você usa no processo de listagem.

`features`

Uma matriz que contém atributos compatíveis com o dispositivo. Todos os atributos são obrigatórios.

`arch`

As arquiteturas de sistema operacional suportadas que a execução do teste valida. Os valores válidos são:

- `x86_64`
- `armv6l`
- `armv7l`
- `aarch64`

`ml`

Valida se o dispositivo atende a todas as dependências técnicas necessárias para usar os componentes de aprendizado AWS de máquina (ML) fornecidos.

A ativação desse recurso também valida que o dispositivo pode realizar inferência de ML usando as estruturas Deep Learning Runtime e TensorFlow Lite ML.

Os valores válidos são qualquer combinação de `dlr` `etensorflowlite`, ouno.

`docker`

Valida se o dispositivo atende a todas as dependências técnicas necessárias para usar o componente Docker application manager () AWS fornecido. `aws.greengrass.DockerApplicationManager`

A ativação desse recurso também valida que o dispositivo possa baixar uma imagem de contêiner Docker do Amazon ECR.

Os valores válidos são qualquer combinação de `yes` ouno.

`streamManagement`

Valida se o dispositivo pode baixar, instalar e executar o gerenciador de AWS IoT Greengrass streaming.

Os valores válidos são qualquer combinação de `yes` ouno.

`hsi`

Valida se o dispositivo pode autenticar conexões com os AWS IoT Greengrass serviços AWS IoT e usando uma chave privada e um certificado armazenados em um módulo de segurança de hardware (HSM). Esse teste também verifica se o componente provedor PKCS #11 AWS fornecido pode interagir com o HSM usando uma biblioteca PKCS #11 fornecida pelo fornecedor. Para ter mais informações, consulte Integração de segurança de hardware.

Os valores válidos são `hsm` ou `no`.

> **Note**
>
> O teste do `hsi` está disponível somente com o IDT v4.9.3 e versões posteriores.

`devices.id`

Um identificador exclusivo, definido pelo usuário, para o dispositivo que está sendo testado.

`devices.operatingSystem`

O sistema operacional do dispositivo. Os valores compatíveis são `Linux` e `Windows`.

`connectivity.protocol`

O protocolo de comunicação usado para se comunicar com esse dispositivo. Atualmente, o único valor suportado é `ssh` para dispositivos físicos.

`connectivity.ip`

O endereço IP do dispositivo que está sendo testado.

Essa propriedade será aplicada somente se `connectivity.protocol` estiver definido como `ssh`.

`connectivity.port`

Opcional. O número da porta usada nas conexões SSH.

O valor padrão é 22.

Essa propriedade será aplicada somente se `connectivity.protocol` estiver definido como `ssh`.

`connectivity.publicKeyPath`

Opcional. O caminho completo para a chave pública usada para autenticar conexões com o dispositivo em teste.

Ao especificar o `publicKeyPath`, o IDT valida a chave pública do dispositivo ao estabelecer uma conexão SSH com o dispositivo em teste. Se este valor não for especificado, o IDT cria uma conexão SSH, mas não valida a chave pública do dispositivo.

É altamente recomendável especificar o caminho para a chave pública e use um método seguro para buscar essa chave pública. Para clientes SSH padrão baseados em linha de comando, a chave pública é fornecida no arquivo `known_hosts`. Se especificar um arquivo de chave pública separado, esse arquivo deverá usar o mesmo formato do arquivo `known_hosts` , ou seja, *`ip-address key-type public-key`*. Se houver várias entradas com o mesmo endereço IP, a entrada para o tipo de chave usado pelo IDT deverá estar antes das outras entradas no arquivo.

`connectivity.auth`

Informações de autenticação da conexão.

Essa propriedade será aplicada somente se `connectivity.protocol` estiver definido como `ssh`.

`connectivity.auth.method`

O método de autenticação usado para acessar um dispositivo pelo protocolo de conectividade indicado.

Os valores compatíveis são:

- `pki`
- `password`

`connectivity.auth.credentials`

As credenciais usadas para autenticação.

`connectivity.auth.credentials.password`

A senha usada para fazer login no dispositivo que está sendo testado.

Esse valor se aplica somente se `connectivity.auth.method` estiver definido como `password`.

`connectivity.auth.credentials.privKeyPath`

O caminho completo para a chave privada usada para fazer login no dispositivo que está sendo testado.

Esse valor se aplica somente se `connectivity.auth.method` estiver definido como `pki`.

`connectivity.auth.credentials.user`

O nome de usuário para fazer login no dispositivo que está sendo testado.

IDT v4.9.4

```
[
  {
    "id": "<pool-id>",
    "sku": "<sku>",
    "features": [
      {
        "name": "arch",
        "value": "x86_64 | armv6l | armv7l | aarch64"
      },
      {
        "name": "docker",
        "value": "yes | no"
      },
      {
        "name": "hsi",
        "value": "hsm | no"
      }
    ],
    "devices": [
      {
        "id": "<device-id>",
        "operatingSystem": "Linux | Windows",
        "connectivity": {
          "protocol": "ssh",
          "ip": "<ip-address>",
          "port": 22,
          "publicKeyPath": "<public-key-path>",
          "auth": {
            "method": "pki | password",
```

```
            "credentials": {
              "user": "<user-name>",
              "privKeyPath": "/path/to/private/key",
              "password": "<password>"
            }
          }
        }
      }
    ]
  }
]
```

> **Note**
>
> Especifique `privKeyPath` somente se `method` estiver definido como `pki`.
> Especifique `password` somente se `method` estiver definido como `password`.

Todas as propriedades que contêm valores são obrigatórias, conforme descrito aqui:

`id`

Um ID alfanumérico definido pelo usuário que identifica uma coleção de dispositivos chamada de grupo de dispositivos. Os dispositivos que pertencem a um grupo devem ter hardware idêntico. Quando um conjunto de testes é executado, os dispositivos do grupo são usados para paralelizar a carga de trabalho. Vários dispositivos são usados para executar testes diferentes.

`sku`

Um valor alfanumérico que identifica exclusivamente o dispositivo em teste. A SKU é usada para rastrear as placas qualificadas.

> **Note**
>
> Se você quiser listar seu dispositivo no Catálogo de AWS Partner dispositivos, o SKU especificado aqui deverá corresponder ao SKU que você usa no processo de listagem.

`features`

Uma matriz que contém atributos compatíveis com o dispositivo. Todos os atributos são obrigatórios.

`arch`

As arquiteturas de sistema operacional suportadas que a execução do teste valida. Os valores válidos são:

- `x86_64`
- `armv6l`
- `armv7l`
- `aarch64`

`docker`

Valida se o dispositivo atende a todas as dependências técnicas necessárias para usar o componente Docker application manager () AWS fornecido. `aws.greengrass.DockerApplicationManager`

A ativação desse recurso também valida que o dispositivo possa baixar uma imagem de contêiner Docker do Amazon ECR.

Os valores válidos são qualquer combinação de `yes` ou`no`.

`hsi`

Valida se o dispositivo pode autenticar conexões com os AWS IoT Greengrass serviços AWS IoT e usando uma chave privada e um certificado armazenados em um módulo de segurança de hardware (HSM). Esse teste também verifica se o componente provedor PKCS #11 AWS fornecido pode interagir com o HSM usando uma biblioteca PKCS #11 fornecida pelo fornecedor. Para ter mais informações, consulte Integração de segurança de hardware.

Os valores válidos são `hsm` ou `no`.

> **Note**
>
> O teste do `hsi` está disponível somente com o IDT v4.9.3 e versões posteriores.

`devices.id`

Um identificador exclusivo, definido pelo usuário, para o dispositivo que está sendo testado.

`devices.operatingSystem`

O sistema operacional do dispositivo. Os valores compatíveis são `Linux` e `Windows`.

`connectivity.protocol`

O protocolo de comunicação usado para se comunicar com esse dispositivo. Atualmente, o único valor suportado é `ssh` para dispositivos físicos.

`connectivity.ip`

O endereço IP do dispositivo que está sendo testado.

Essa propriedade será aplicada somente se `connectivity.protocol` estiver definido como `ssh`.

`connectivity.port`

Opcional. O número da porta usada nas conexões SSH.

O valor padrão é 22.

Essa propriedade será aplicada somente se `connectivity.protocol` estiver definido como `ssh`.

`connectivity.publicKeyPath`

Opcional. O caminho completo para a chave pública usada para autenticar conexões com o dispositivo em teste.

Ao especificar o `publicKeyPath`, o IDT valida a chave pública do dispositivo ao estabelecer uma conexão SSH com o dispositivo em teste. Se este valor não for especificado, o IDT cria uma conexão SSH, mas não valida a chave pública do dispositivo.

É altamente recomendável especificar o caminho para a chave pública e use um método seguro para buscar essa chave pública. Para clientes SSH padrão baseados em linha de comando, a chave pública é fornecida no arquivo `known_hosts`. Se especificar um arquivo de chave pública separado, esse arquivo deverá usar o mesmo formato do arquivo `known_hosts` , ou seja, *`ip-address key-type public-key`*. Se houver várias entradas com o mesmo endereço IP, a entrada para o tipo de chave usado pelo IDT deverá estar antes das outras entradas no arquivo.

`connectivity.auth`

Informações de autenticação da conexão.

Essa propriedade será aplicada somente se `connectivity.protocol` estiver definido como `ssh`.

`connectivity.auth.method`

O método de autenticação usado para acessar um dispositivo pelo protocolo de conectividade indicado.

Os valores compatíveis são:

- `pki`
- `password`

`connectivity.auth.credentials`

As credenciais usadas para autenticação.

`connectivity.auth.credentials.password`

A senha usada para fazer login no dispositivo que está sendo testado.

Esse valor se aplica somente se `connectivity.auth.method` estiver definido como `password`.

`connectivity.auth.credentials.privKeyPath`

O caminho completo para a chave privada usada para fazer login no dispositivo que está sendo testado.

Esse valor se aplica somente se `connectivity.auth.method` estiver definido como `pki`.

`connectivity.auth.credentials.user`

O nome de usuário para fazer login no dispositivo que está sendo testado.

Configurar userdata.json

O IDT for AWS IoT Greengrass V2 também precisa de informações adicionais sobre a localização dos artefatos e do software de teste. AWS IoT Greengrass

Você deve fornecer essas informações usando o modelo `userdata.json` localizado em *<device_tester_extract_location>*`/configs/userdata.json`:

```
{
    "TempResourcesDirOnDevice": "/path/to/temp/folder",
    "InstallationDirRootOnDevice": "/path/to/installation/folder",
    "GreengrassNucleusZip": "/path/to/aws.greengrass.nucleus.zip",
    "PreInstalled": "yes/no",
    "GreengrassV2TokenExchangeRole": "custom-iam-role-name",
 "hsm": {
        "greengrassPkcsPluginJar": "/path/to/aws.greengrass.crypto.Pkcs11Provider-
latest.jar",
        "pkcs11ProviderLibrary": "/path/to/pkcs11-vendor-library",
        "slotId": "slot-id",
        "slotLabel": "slot-label",
        "slotUserPin": "slot-pin",
        "keyLabel": "key-label",
        "preloadedCertificateArn": "certificate-arn"
        "rootCA": "path/to/root-ca"
    }
}
```

Todas as propriedades que contêm valores são obrigatórias conforme descrito aqui:

`TempResourcesDirOnDevice`

O caminho completo para uma pasta temporária no dispositivo em teste na qual armazenar artefatos de teste. Certifique-se de que as permissões sudo não sejam necessárias para gravar nesse diretório.

> **Note**
>
> O IDT exclui o conteúdo dessa pasta quando termina de executar um teste.

`InstallationDirRootOnDevice`

O caminho completo para uma pasta no dispositivo no qual instalar AWS IoT Greengrass. Para o PreInstalled Greengrass, esse é o caminho para o diretório de instalação do Greengrass.

Você deve definir as permissões de arquivo necessárias para essa pasta. Execute o comando a seguir para cada pasta no caminho de instalação.

```
sudo chmod 755 folder-name
```

`GreengrassNucleusZip`

O caminho completo para o arquivo ZIP (`greengrass-nucleus-latest.zip`) do Greengrass nucleus em seu computador host. Esse campo não é obrigatório para testes com o PreInstalled Greengrass.

> **Note**
>
> Para obter informações sobre as versões suportadas do núcleo Greengrass para IDT for, consulte. AWS IoT GreengrassVersão mais recente do IDT para AWS IoT Greengrass V2 Para baixar o software Greengrass mais recente, consulte Baixar o AWS IoT Greengrass software.

`PreInstalled`

Esse recurso está disponível somente para o IDT v4.5.8 e versões posteriores em dispositivos Linux.

(Opcional) Quando o valor for *`sim`*, o IDT assumirá que o caminho mencionado em `InstallationDirRootOnDevice` é o diretório em que o Greengrass está instalado.

Para obter mais informações sobre como instalar o Greengrass em seu dispositivo, consulte. Instale o software AWS IoT Greengrass principal com provisionamento automático de recursos Se estiver instalando com provisionamento manual, inclua a etapa “Adicionar a AWS IoT coisa a um grupo de coisas novo ou existente” ao criar uma AWS IoT coisa manualmente. O IDT pressupõe que a coisa e o grupo de coisas sejam criados durante a configuração da instalação. Certifique-se de que esses valores sejam refletidos no `effectiveConfig.yaml` arquivo. O IDT verifica o arquivo `effectiveConfig.yaml` abaixo`<InstallationDirRootOnDevice>/config/effectiveConfig.yaml`.

Para executar testes com o HSM, certifique-se de que o `aws.greengrass.crypto.Pkcs11Provider` campo esteja atualizado em`effectiveConfig.yaml`.

`GreengrassV2TokenExchangeRole`

(Opcional) A função personalizada do IAM que você deseja usar como a função de troca de tokens que o dispositivo em teste presume interagir com os AWS recursos.

> **Note**
>
> O IDT usa essa função personalizada do IAM em vez de criar a função padrão de troca de tokens durante a execução do teste. Se você usar um papel personalizado, poderá atualizar as permissões do IAM para o usuário de teste para excluir a `iamResourcesUpdate` declaração que permite ao usuário criar e excluir funções e políticas do IAM.

Para obter mais informações sobre como criar uma função personalizada do IAM como sua função de troca de tokens, consulteConfigurar uma função personalizada de troca de tokens.

`hsm`

Esse recurso está disponível para o IDT v4.5.1 e versões posteriores.

(Opcional) As informações de configuração para testes com um módulo de segurança de AWS IoT Greengrass hardware (HSM). Caso contrário, a propriedade `hsm` deve ser omitida. Para ter mais informações, consulte Integração de segurança de hardware.

Essa propriedade será aplicada somente se `connectivity.protocol` estiver definido como `ssh`.

> **Warning**
>
> A configuração do HSM pode ser considerada um dado confidencial se o módulo de segurança de hardware for compartilhado entre o IDT e outro sistema. Nessa situação, você pode evitar proteger esses valores de configuração em texto simples armazenando-os em um AWS parâmetro do Parameter Store SecureString e configurando o IDT para buscá-los durante a execução do teste. Para mais informações, consulte ???.

`hsm.greengrassPkcsPluginJar`

O caminho completo para o componente do provedor PKCS #11 que você baixa para a máquina host IDT. AWS IoT Greengrass fornece esse componente como arquivo JAR

que você pode baixar para especificar como um plug-in de provisionamento durante a instalação. Você pode baixar a versão mais recente do arquivo JAR do componente na seguinte URL: https://d2s8p88vqu9w66.cloudfront.net/releases/Pkcs11Provider/aws.greengrass.crypto.Pkcs11Provider-latest.jar.

`hsm.pkcs11ProviderLibrary`

O caminho completo para a biblioteca PKCS #11 fornecida pelo fornecedor do módulo de segurança de hardware (HSM) para interagir com o HSM.

`hsm.slotId`

O ID do slot usado para identificar o slot HSM no qual você carrega a chave e o certificado.

`hsm.slotLabel`

A etiqueta do slot usada para identificar o slot HSM no qual você carrega a chave e o certificado.

`hsm.slotUserPin`

O PIN do usuário que o IDT usa para autenticar o software AWS IoT Greengrass Core no HSM.

> **Note**
>
> Como prática recomendada de segurança, não use o mesmo PIN de usuário em dispositivos de produção.

`hsm.keyLabel`

O rótulo usado para identificar a chave no módulo de hardware. Tanto a chave quanto o certificado devem usar o mesmo rótulo de chave.

`hsm.preloadedCertificateArn`

O Amazon Resource Name (ARN) do certificado de dispositivo carregado na AWS IoT nuvem.

Você deve ter gerado esse certificado anteriormente usando a chave no HSM, importado para o HSM e carregado na AWS IoT nuvem. Para obter informações sobre como gerar e importar o certificado, consulte a documentação do seu HSM.

Você deve carregar o certificado na mesma conta e região que você fornece em config.json. . Para obter mais informações sobre como fazer o upload do seu certificado para AWS IoT, consulte Registrar um certificado de cliente manualmente no Guia do AWS IoT desenvolvedor.

`hsm.rootCAPath`

(Opcional) O caminho completo na máquina host do IDT até a autoridade de certificação (CA) raiz que assinou seu certificado. Isso é necessário se o certificado em seu HSM criado não for assinado pela CA raiz da Amazon.

Obter configuração do AWS Parameter Store

AWS IoT O Device Tester (IDT) inclui um recurso opcional para buscar valores de configuração do AWS Systems Manager Parameter Store. AWS O Parameter Store permite o armazenamento seguro e criptografado das configurações. Quando configurado, o IDT pode buscar AWS parâmetros do Parameter Store em vez de armazenar parâmetros em texto simples dentro do arquivo. `userdata.json` Isso é útil para qualquer dado confidencial que deva ser armazenado criptografado, como: senhas, pinos e outros segredos.

1. Para usar esse recurso, você deve atualizar as permissões usadas na criação do usuário do IDT para permitir a GetParameter ação nos parâmetros que o IDT está configurado para usar. Abaixo está um exemplo de uma declaração de permissão que pode ser adicionada ao usuário do IDT. Para obter mais informações, consulte o AWS Systems Manager guia do usuário.

   ```
   {
          "Sid":"parameterStoreResources",
          "Effect": "Allow",
           "Action": [
               "ssm:GetParameter"
           ],
           "Resource": "arn:aws:ssm:*:*:parameter/IDT*"
   }
   ```

   A permissão acima está configurada para permitir a busca de todos os parâmetros com um nome começando com`IDT`, usando o caractere curinga. * Você deve personalizá-lo de acordo com suas necessidades para que o IDT tenha acesso para buscar quaisquer parâmetros configurados com base na nomenclatura dos parâmetros que você está usando.
2. Você precisa armazenar seus valores de configuração dentro do AWS Paramater Store. Isso pode ser feito no AWS console ou na AWS CLI. AWS O Parameter Store permite que

você escolha armazenamento criptografado ou não criptografado. Para armazenar valores confidenciais, como segredos, senhas e pinos, você deve usar a opção criptografada, que é um tipo de parâmetro de SecureString. Para carregar um parâmetro usando a AWS CLI, você pode usar o seguinte comando:

```
aws ssm put-parameter --name IDT-example-name --value IDT-example-value --type
 SecureString
```

Você pode verificar se um parâmetro está armazenado usando o comando a seguir. (Opcional) Use o `--with-decryption` sinalizador para buscar um parâmetro descriptografadoSecureString .

```
aws ssm get-parameter --name IDT-example-name
```

O uso da AWS CLI carregará o parâmetro na AWS região do usuário atual da CLI e o IDT buscará os parâmetros da região configurada em. `config.json` Para verificar sua região na AWS CLI, use o seguinte:

```
aws configure get region
```

3. Depois de ter um valor de configuração na AWS nuvem, você pode atualizar qualquer valor dentro da configuração do IDT para buscá-lo na AWS nuvem. Para fazer isso, use um espaço reservado na configuração do IDT do formulário `{{AWS.Parameter.parameter_name}}` para buscar o parâmetro com esse nome no Parameter Store. AWS

   Por exemplo, suponha que você queira usar o `IDT-example-name` parâmetro da Etapa 2 como o HSM KeyLabel na configuração do HSM. Para fazer isso, você pode atualizar o seu da `userdata.json` seguinte forma:

   ```
   "hsm": {
           "keyLabel": "{{AWS.Parameter.IDT-example-name}}",
           [...]
       }
   ```

   O IDT buscará o valor desse parâmetro no tempo de execução definido `IDT-example-value` na Etapa 2. Essa configuração é semelhante à configuração, `"keyLabel": "IDT-example-value"` mas, em vez disso, esse valor é armazenado como criptografado na AWS nuvem.

Execute o pacote de qualificação do AWS IoT Greengrass

Depois de definir a configuração necessária, você poderá iniciar os testes. O tempo de execução do conjunto de testes completo depende do seu hardware. Como referência, leva aproximadamente 30 minutos para concluir o pacote de testes completo em um Raspberry Pi 3B.

Use o `run-suite` comando a seguir para executar um conjunto de testes.

```
devicetester_[linux | mac | win]_x86-64 run-suite  \\
    --suite-id suite-id  \\
    --group-id group-id  \\
    --pool-id your-device-pool \\
    --test-id test-id  \\
    --update-idt y|n  \\
    --userdata userdata.json
```

Todas as opções são opcionais. Por exemplo, você pode omitir `pool-id` se tiver apenas um pool de dispositivos, que é um conjunto de dispositivos idênticos, definido em seu `device.json` arquivo. Você também pode omitir `suite-id` se quiser executar a versão mais recente do conjunto de testes na pasta `tests`.

> **Note**
>
> O IDT avisará se uma versão mais recente do conjunto de testes estiver disponível online. Para ter mais informações, consulte the section called “Versões do pacote de testes”.

Exemplos de comandos para executar o pacote de qualificação

Os exemplos de linha de comando a seguir mostram como executar os testes de qualificação para um pool de dispositivos. Para obter mais informações sobre `run-suite` e outros comandos IDT, consulte the section called “Comandos do IDT”.

Use o comando a seguir para executar todos os grupos de teste em uma suíte de testes especificada. O `list-suites` comando lista as suítes de teste que estão na `tests` pasta.

```
devicetester_[linux | mac | win]_x86-64 run-suite \
    --suite-id GGV2Q_1.0.0 \
    --pool-id <pool-id> \
    --userdata userdata.json
```

Use o comando a seguir para executar um grupo de teste específico em uma suíte de testes. O `list-groups` comando lista os grupos de teste em uma suíte de testes.

```
devicetester_[linux | mac | win]_x86-64 run-suite \
    --suite-id GGV2Q_1.0.0 \
    --group-id <group-id> \
    --pool-id <pool-id> \
    --userdata userdata.json
```

Use o comando a seguir para executar um caso de teste específico em um grupo de teste.

```
devicetester_[linux | mac | win]_x86-64 run-suite \
    --group-id <group-id> \
    --test-id <test-id> \
    --userdata userdata.json
```

Use o comando a seguir para executar vários casos de teste em um grupo de teste.

```
devicetester_[linux | mac | win]_x86-64 run-suite \
    --group-id <group-id> \
    --test-id <test-id1>,<test-id2>
    --userdata userdata.json
```

Use o comando a seguir para listar todos os casos de teste em um grupo de teste.

```
devicetester_[linux | mac | win]_x86-64 list-test-cases --group-id <group-id>
```

Recomendamos que você execute o conjunto completo de testes de qualificação, que executa as dependências do grupo de teste na ordem correta. Se você optar por executar grupos de teste específicos, recomendamos que primeiro execute o grupo de teste do verificador de dependências para garantir que todas as dependências do Greengrass estejam instaladas antes de executar os grupos de teste relacionados. Por exemplo: .

- Execute `coredependencies` antes de executar grupos de testes de qualificação de núcleo.

IDT para comandos AWS IoT Greengrass V2

Os comandos do IDT estão localizados no diretório *<device-tester-extract-location>*`/bin`. Para executar uma suíte de testes, você fornece o comando no seguinte formato:

`help`

Lista as informações sobre o comando especificado.

`list-groups`

Lista os grupos em um determinado conjunto de teste.

`list-suites`

Lista os conjuntos de teste disponíveis.

`list-supported-products`

Lista os produtos compatíveis, neste caso, versões do AWS IoT Greengrass e versões do conjunto de testes para a versão atual do IDT.

`list-test-cases`

Lista os casos de teste em um grupo de teste. A seguinte opção é compatível:

- `group-id`. O grupo de teste a ser pesquisado. Esta opção é necessária e deve especificar um único grupo.

`run-suite`

Executa um conjunto de testes em um grupo de dispositivos. Algumas opções compatíveis estão listadas a seguir:

- `suite-id`. A versão do conjunto de testes a ser executada. Se não for especificado, o IDT usará a versão mais recente na pasta `tests`.
- `group-id`. Os grupos de teste a serem executados, em uma lista separada por vírgulas. Se não for especificado, o IDT executa todos os grupos de teste apropriados na suíte de testes, dependendo das configurações definidas em`device.json`. O IDT não executa nenhum grupo de teste que o dispositivo não suporte com base nas configurações definidas, mesmo que esses grupos de teste estejam especificados na `group-id` lista.
- `test-id`. Os casos de teste a serem executados, em uma lista separada por vírgulas. Quando especificado, `group-id` deve especificar um único grupo.
- `pool-id`. O grupo de dispositivos a ser testado. Você deve especificar um grupo se tiver vários grupos de dispositivos definidos no arquivo `device.json`.
- `stop-on-first-failure`. Configura o IDT para parar de ser executado na primeira falha. Use essa opção `group-id` quando quiser depurar os grupos de teste especificados. Não use essa opção para gerar um relatório de qualificação ao executar um conjunto de testes completo.

- `update-idt`. Define a resposta para a solicitação de atualização do IDT. A `Y` resposta interrompe a execução do teste se o IDT detectar que há uma versão mais recente. A `N` resposta continua a execução do teste.
- `userdata`. O caminho completo para o `userdata.json` arquivo que contém informações sobre os caminhos dos artefatos de teste. Essa opção é necessária para o `run-suite` comando. *O `userdata.json` arquivo deve estar localizado no diretório `devicetester_extract_location /devicetester_ggv2_ [win|mac|linux] / configs/`.*

Para obter mais informações sobre as opções `run-suite`, use a opção `help`:

```
devicetester_[linux | mac | win]_x86-64 run-suite -h
```

Noções básicas de resultados e logs

Esta seção descreve como visualizar e interpretar os resultados de relatórios e logs do IDT.

Para solucionar erros, consulteSolução de problemas de IDT paraAWS IoT GreengrassV2.

Visualização de resultados

Enquanto em execução, o IDT grava erros no console, arquivos de log e relatórios de teste. Depois de concluir o conjunto de testes de qualificação, o IDT gera dois relatórios de teste. Esses relatórios estão localizados em*<device-tester-extract-location>*`/results/`*<execution-id>*`/`. Ambos os relatórios capturam os resultados da execução do conjunto de testes de qualificação.

`awsiotdevicetester_report.xml`É o relatório de teste de qualificação que você enviaAWS para listar seuAWS Partner dispositivo no Catálogo de dispositivos. O relatório contém os seguintes elementos:

- A versão IDT.
- A versão do AWS IoT Greengrass que foi testada.
- A SKU e o nome de grupo do dispositivo especificados no arquivo `device.json`.
- Os recursos do grupo do dispositivo especificados no arquivo `device.json`.
- O resumo agregado dos resultados de teste.
- Uma análise dos resultados dos testes por bibliotecas que foram testadas com base nos recursos do dispositivo, como acesso a recursos locais, sombra e MQTT.

O relatório `GGV2Q_Result.xml` está no formato JUnit XML. Você pode integrá-lo em plataformas de integração e implantação como Jenkins, Bamboo, e assim por diante. O relatório contém os seguintes elementos:

- Um resumo agregado dos resultados de teste.
- Detalhamento dos resultados do teste pela funcionalidade do AWS IoT Greengrass que foi testada.

Como interpretarAWS IoT Device Tester os resultados

A seção de relatório em `awsiotdevicetester_report.xml` ou `awsiotdevicetester_report.xml` lista os testes que foram executados e os resultados.

A primeira tag XML`<testsuites>` contém o resumo da execução do teste. Por exemplo:

```
<testsuites name="GGQ results" time="2299" tests="28" failures="0" errors="0"
 disabled="0">
```

Atributos usados na tag **`<testsuites>`**

`name`

O nome do conjunto de testes.

`time`

O tempo, em segundos, pelo qual o conjunto de qualificações foi executado.

`tests`

O número de testes que foram executados.

`failures`

O número de testes que foram executados, mas não foram aprovados.

`errors`

O número de testes que o IDT não conseguiu executar.

`disabled`

Ignore esse atributo. Ele não é usado.

O arquivo `awsiotdevicetester_report.xml` contém uma tag `<awsproduct>` com informações sobre o produto que está sendo testado e os recursos do produto que foram validados após a execução de um pacote de testes.

Atributos usados na tag **`<awsproduct>`**

`name`

O nome do produto testado.

`version`

A versão do produto testado.

`features`

Os recursos validados. Recursos marcados como `required` são necessários para enviar sua placa para qualificação. O snippet a seguir mostra como essas informações aparecem no arquivo `awsiotdevicetester_report.xml`.

```
<name="aws-iot-greengrass-v2-core" value="supported" type="required"></feature>
```

Se não há falhas de teste ou erros nos recursos exigidos, isso significa que o dispositivo atende aos requisitos técnicos para executar o AWS IoT Greengrass e pode interoperar com serviços do AWS IoT. Se quiser listar seuAWS Partner dispositivo no Catálogo de dispositivos, você pode usar esse relatório como evidência de qualificação.

Se houver falhas de teste ou erros, você poderá identificar o teste com falha analisando as tags XML `<testsuites>`. As tags XML `<testsuite>` dentro da tag `<testsuites>` mostram o resumo do resultado do teste para um grupo de testes. Por exemplo:

```
<testsuite name="combination" package="" tests="1" failures="0" time="161" disabled="0"
 errors="0" skipped="0">
```

O formato é semelhante à tag `<testsuites>`, mas com um atributo `skipped` que não é usado e pode ser ignorado. Dentro de cada tag`<testsuite>` XML, há`<testcase>` tags para cada teste que foi executado para um grupo de teste. Por exemplo:

```
<testcase classname="Security Combination (IPD + DCM) Test Context" name="Security
 Combination IP Change Tests sec4_test_1: Should rotate server cert when IPD disabled
```

```
 and following changes are made:Add CIS conn info and Add another CIS conn info"
 attempts="1"></testcase>>
```

Atributos usados na tag **<testcase>**

`name`

O nome do teste.

`attempts`

O número de vezes que o IDT executou o caso de teste.

Quando um teste falha ou ocorre um erro, as tags `<failure>` ou `<error>` são adicionadas à tag `<testcase>` com informações para a solução de problemas. Por exemplo:

```
<testcase classname="mcu.Full_MQTT" name="AFQP_MQTT_Connect_HappyCase" attempts="1">
 <failure type="Failure">Reason for the test failure</failure>
 <error>Reason for the test execution error</error>
</testcase>
```

Visualizar logs do

O IDT gera registros a partir de execuções de teste em*<devicetester-extract-location>*/`results/`*<execution-id>*`/logs`. Dois conjuntos de logs são gerados:

`test_manager.log`

Registros gerados a partir do componente Test Manager doAWS IoT Device Tester (por exemplo, registros relacionados à configuração, sequenciamento de testes e geração de relatórios).

<test-case-id>`.log (for example, lambdaDeploymentTest.log)`

Registros do caso de teste dentro do grupo de teste, incluindo registros do dispositivo em teste. A partir do IDT v4.2.0, o IDT agrupa os registros de teste para cada caso de teste em uma pasta *<test-case-id >* separada dentro do*<devicetester-extract-location>*/`results/`*<execution-id>*`/logs/`*<test-group-id>/* diretório.

Usar o IDT para desenvolver e executar os próprios pacotes de testes

A partir do IDT v4.0.1, o IDT for AWS IoT Greengrass V2 combina uma configuração padronizada e um formato de resultado com um ambiente de suíte de testes que permite desenvolver suítes de testes personalizadas para seus dispositivos e software de dispositivos. Você pode adicionar testes personalizados a própria validação interna ou fornecê-los aos seus clientes para verificação de dispositivo.

Use o IDT para desenvolver e executar conjuntos de testes personalizados, dessa forma:

Para desenvolver conjuntos de testes personalizados

- Crie conjuntos de testes com lógica de teste personalizada para o dispositivo Greengrass que você deseja testar.
- Forneça ao IDT seus conjuntos de testes personalizados para os executores de testes. Inclua informações sobre configurações específicas para seus conjuntos de teste.

Para executar conjuntos de testes personalizados

- Configure o dispositivo que deseja testar.
- Implemente as configurações conforme exigido pelos conjuntos de testes que você deseja usar.
- Use o IDT para executar seus conjuntos de testes personalizados.
- Veja os resultados do teste e os logs de execução dos testes executados pelo IDT.

Baixe a versão mais recente do AWS IoT Device Tester for AWS IoT Greengrass

Baixe a versão mais recente do IDT e extraia o software em um local (*< device-tester-extract-location >*) em seu sistema de arquivos onde você tenha permissões de leitura/gravação.

> **Note**
>
> O IDT não oferece suporte a execução por vários usuários em um local compartilhado, como um diretório NFS ou uma pasta compartilhada de rede do Windows. Recomendamos extrair

o pacote do IDT para uma unidade local e executar o binário do IDT na estação de trabalho local.
O Windows tem uma limitação de comprimento de caminho de 260 caracteres. Se você estiver usando o Windows, extraia o IDT para um diretório raiz como `C:\` ou `D:\` para manter os caminhos abaixo do limite de 260 caracteres.

Fluxo de trabalho de criação de conjuntos de testes

Os conjuntos de testes são compostos por três tipos de arquivos:

- Arquivos de configuração que fornecem ao IDT informações sobre como executar a suíte de testes.
- Arquivos executáveis de teste que o IDT usa para executar casos de teste.
- Arquivos adicionais necessários para executar testes.

Conclua as etapas básicas a seguir para criar testes de IDT personalizados:

1. Crie arquivos de configuração para seu pacote de teste.
2. Crie executáveis de casos de teste que contenham a lógica de teste para seu pacote de teste.
3. Verifique e documente as informações de configuração necessárias para que os executores de teste executem o conjunto de testes.
4. Verifique se o IDT pode executar seu conjunto de testes e produzir resultados de teste conforme o esperado.

Para criar rapidamente uma amostra de conjunto personalizado e executá-la, siga as instruções em Tutorial: compile e execute o pacote de amostra de teste de IDT.

Para começar a criar um pacote de testes personalizado em Python, consulte Tutorial: desenvolva um pacote de testes de IDT simples.

Tutorial: compile e execute o pacote de amostra de teste de IDT

O download do AWS IoT Device Tester inclui o código-fonte de uma amostra de conjunto de testes. Você pode concluir este tutorial para criar e executar a suíte de testes de amostra para entender como você pode usar o IDT AWS IoT Greengrass para executar suítes de testes personalizadas.

Neste tutorial, você concluirá as seguintes etapas:

1. Compilação do conjunto de testes de amostra
2. Usar o IDT para executar o conjunto de testes de amostra

Pré-requisitos

Para concluir este tutorial, você precisará do seguinte:

- Requisitos do computador host
  - Versão mais recente do AWS IoT Device Tester
  - Python 3.7 ou posterior

    Para verificar a versão do Python instalada em seu computador, execute o comando a seguir:

    ```
    python3 --version
    ```

    No Windows, se o uso desse comando retornar um erro, use `python --version`. Se o número da versão retornada for 3.7 ou superior, execute o seguinte comando em um terminal do Powershell para definir `python3` como um alias para seu comando `python`.

    ```
    Set-Alias -Name "python3" -Value "python"
    ```

    Se nenhuma informação sobre versão for retornada ou se o número da versão for menor do que 3.7, siga as instruções em Fazer download do Python para instalar o Python 3.7+. Para obter mais informações, consulte a Documentação do Python.
  - urllib3

    Para verificar se `urllib3` está instalado corretamente, execute o comando a seguir:

    ```
    python3 -c 'import urllib3'
    ```

    Se o `urllib3` não estiver instalado, execute o comando a seguir para instalá-lo:

    ```
    python3 -m pip install urllib3
    ```

- Requisitos do dispositivo
  - Um dispositivo com sistema operacional Linux e uma conexão de rede com a mesma rede do seu computador host.

    Recomendamos que você use um Raspberry Pi com o sistema operacional Raspberry Pi. Configure o SSH no seu Raspberry Pi para se conectar remotamente a ele.

Configuração das informações do dispositivo para o IDT

Configure as informações doseu dispositivo para o IDT executar o teste. Você deve atualizar o modelo `device.json` localizado na pasta *<device-tester-extract-location>*`/configs` com as seguintes informações.

```
[
  {
    "id": "pool",
    "sku": "N/A",
    "devices": [
      {
        "id": "<device-id>",
        "connectivity": {
          "protocol": "ssh",
          "ip": "<ip-address>",
          "port": "<port>",
          "auth": {
            "method": "pki | password",
            "credentials": {
              "user": "<user-name>",
              "privKeyPath": "/path/to/private/key",
              "password": "<password>"
            }
          }
        }
      }
    ]
  }
]
```

No objeto `devices`, forneça as seguintes informações:

`id`

Um identificador exclusivo, definido pelo usuário, para o seu dispositivo.

`connectivity.ip`

O endereço IP do seu dispositivo.

`connectivity.port`

Opcional. O número da porta que deve ser usado nas conexões SSH ao seu dispositivo.

`connectivity.auth`

Informações de autenticação da conexão.

Essa propriedade será aplicada somente se `connectivity.protocol` estiver definido como `ssh`.

`connectivity.auth.method`

O método de autenticação usado para acessar um dispositivo pelo protocolo de conectividade indicado.

Os valores compatíveis são:

- `pki`
- `password`

`connectivity.auth.credentials`

As credenciais usadas para autenticação.

`connectivity.auth.credentials.user`

O nome de usuário usado para fazer login no seu dispositivo.

`connectivity.auth.credentials.privKeyPath`

O caminho completo para a chave privada usada para fazer login no dispositivo.

Esse valor se aplica somente se `connectivity.auth.method` estiver definido como `pki`.

`devices.connectivity.auth.credentials.password`

A senha usada para fazer login no dispositivo.

Esse valor se aplica somente se `connectivity.auth.method` estiver definido como `password`.

> **Note**
>
> Especifique `privKeyPath` somente se `method` estiver definido como `pki`.
> Especifique `password` somente se `method` estiver definido como `password`.

Compilação do conjunto de testes de amostra

A pasta *<device-tester-extract-location>*`/samples/python` contém exemplos de arquivos de configuração, código-fonte e o IDT Client SDK que você pode combinar em um conjunto de testes usando os scripts de compilação fornecidos. A árvore de diretórios a seguir mostra a localização desses arquivos de amostra:

```
<device-tester-extract-location>
### ...
### tests
### samples
#   ### ...
#   ### python
#       ### configuration
#       ### src
#       ### build-scripts
#           ### build.sh
#           ### build.ps1
### sdks
    ### ...
    ### python
        ### idt_client
```

Para compilar o conjunto de testes, execute os seguintes comandos em seu computador host:

Windows

```
cd <device-tester-extract-location>/samples/python/build-scripts
./build.ps1
```

Linux, macOS, or UNIX

```
cd <device-tester-extract-location>/samples/python/build-scripts
./build.sh
```

Isso compila o conjunto de testes de amostra na pasta `IDTSampleSuitePython_1.0.0` dentro da pasta *<device-tester-extract-location>*`/tests`. Examine os arquivos na `IDTSampleSuitePython_1.0.0` pasta para entender como a suíte de testes de amostra está estruturada e para ver vários exemplos de executáveis de casos de teste e arquivos JSON de configuração de teste.

> **Note**
>
> O pacote de teste de amostra inclui um código-fonte em Python. Não inclua informações confidenciais no código do seu pacote de teste.

Próxima etapa: use o IDT para executar o pacote de teste de amostra criada.

Usar o IDT para executar o conjunto de testes de amostra

Para executar o conjunto de testes de amostra, execute os seguintes comandos em seu computador host:

```
cd <device-tester-extract-location>/bin
./devicetester_[linux | mac | win_x86-64] run-suite --suite-id IDTSampleSuitePython
```

O IDT executa o conjunto de testes de amostra e transmite os resultados para o console. Quando a execução do teste é concluída, você vê as seguintes informações:

```
========== Test Summary ==========
Execution Time:         5s
Tests Completed:        4
Tests Passed:           4
Tests Failed:           0
Tests Skipped:          0
----------------------------------
Test Groups:
    sample_group:       PASSED
```

```
---------------------------------
Path to IoT Device Tester Report: /path/to/devicetester/
results/87e673c6-1226-11eb-9269-8c8590419f30/awsiotdevicetester_report.xml
Path to Test Execution Logs: /path/to/devicetester/
results/87e673c6-1226-11eb-9269-8c8590419f30/logs
Path to Aggregated JUnit Report: /path/to/devicetester/
results/87e673c6-1226-11eb-9269-8c8590419f30/IDTSampleSuitePython_Report.xml
```

Solução de problemas

Use as informações a seguir para ajudar a resolver os problemas ao concluir o tutorial.

Caso de teste não é executado

Se o teste não for executado, o IDT transmitirá os logs de erro para o console e isso pode ajudar a solucionar o problema de execução do teste. Verifique se você atende a todos os pré-requisitos deste tutorial.

Não é possível se conectar ao dispositivo em teste

Verifique o seguinte:

- Seu arquivo `device.json` contém o endereço IP, a porta e as informações de autenticação corretos.
- Você pode se conectar ao seu dispositivo via SSH a partir do seu computador host.

Tutorial: desenvolva um pacote de testes de IDT simples

Um conjunto de testes combina o seguinte:

- Executáveis de teste que contêm a lógica de teste
- Arquivos de configuração que descrevem o pacote de teste

Este tutorial mostra como usar o IDT para AWS IoT Greengrass para desenvolver um pacote de testes em Python que contém um único caso de teste. Neste tutorial, você completará as seguintes etapas:

1. Criação de um diretório de conjunto de testes
2. Crie arquivos de configuração

3. Criação do executável do caso de teste
4. Execução de conjunto de testes

Pré-requisitos

Para concluir este tutorial, você precisará do seguinte:

- Requisitos do computador host
  - Versão mais recente do AWS IoT Device Tester
  - Python 3.7 ou posterior

    Para verificar a versão do Python instalada em seu computador, execute o comando a seguir:

    ```
    python3 --version
    ```

    No Windows, se o uso desse comando retornar um erro, use `python --version`. Se o número da versão retornada for 3.7 ou superior, execute o seguinte comando em um terminal do Powershell para definir `python3` como um alias para seu comando `python`.

    ```
    Set-Alias -Name "python3" -Value "python"
    ```

    Se nenhuma informação sobre versão for retornada ou se o número da versão for menor do que 3.7, siga as instruções em Fazer download do Python para instalar o Python 3.7+. Para obter mais informações, consulte a Documentação do Python.
  - urllib3

    Para verificar se `urllib3` está instalado corretamente, execute o comando a seguir:

    ```
    python3 -c 'import urllib3'
    ```

    Se o `urllib3` não estiver instalado, execute o comando a seguir para instalá-lo:

    ```
    python3 -m pip install urllib3
    ```
- Requisitos do dispositivo
  - Um dispositivo com sistema operacional Linux e uma conexão de rede com a mesma rede do seu computador host.

Recomendamos que você use um Raspberry Pi com o sistema operacional Raspberry Pi. Configure o SSH no seu Raspberry Pi para se conectar remotamente a ele.

Criação de um diretório de conjunto de testes

O IDT separa logicamente os casos de teste em grupos de teste dentro de cada conjunto de testes. Cada caso de teste deve estar dentro de um grupo de testes. Para este tutorial, crie uma pasta chamada `MyTestSuite_1.0.0` e crie a seguinte árvore de diretórios nesta pasta:

```
MyTestSuite_1.0.0
### suite
    ### myTestGroup
        ### myTestCase
```

Crie arquivos de configuração

Seu pacote de teste deve conter os seguintes arquivos de configuração necessários:

Arquivos de configuração necessária

`suite.json`

Contém informações sobre o pacote de teste. Consulte Configurar suite.json.

`group.json`

Contém informações sobre um grupo de testes. Você deve criar um arquivo `group.json` para cada grupo de testes em seu conjunto de testes. Consulte Configuração de group.json.

`test.json`

Contém informações sobre um caso de testes. Você deve criar um arquivo `test.json` para cada caso de teste em seu conjunto de testes. Consulte Configuração de test.json.

1. Na pasta `MyTestSuite_1.0.0/suite`, crie um arquivo `suite.json` com a seguinte estrutura:

   ```
   {
       "id": "MyTestSuite_1.0.0",
   ```

```
    "title": "My Test Suite",
    "details": "This is my test suite.",
    "userDataRequired": false
}
```

2. Na pasta `MyTestSuite_1.0.0/myTestGroup`, crie um arquivo `group.json` com a seguinte estrutura:

```
{
    "id": "MyTestGroup",
    "title": "My Test Group",
    "details": "This is my test group.",
    "optional": false
}
```

3. Na pasta `MyTestSuite_1.0.0/myTestGroup/myTestCase`, crie um arquivo `test.json` com a seguinte estrutura:

```
{
    "id": "MyTestCase",
    "title": "My Test Case",
    "details": "This is my test case.",
    "execution": {
        "timeout": 300000,
        "linux": {
            "cmd": "python3",
            "args": [
                "myTestCase.py"
            ]
        },
        "mac": {
            "cmd": "python3",
            "args": [
                "myTestCase.py"
            ]
        },
        "win": {
            "cmd": "python3",
            "args": [
                "myTestCase.py"
            ]
        }
    }
```

```
}
```

Agora, a árvore de diretórios da pasta `MyTestSuite_1.0.0` deve ser semelhante à seguinte:

```
MyTestSuite_1.0.0
### suite
    ### suite.json
    ### myTestGroup
        ### group.json
        ### myTestCase
            ### test.json
```

Obter IDT Client SDK

Você usa o IDT Client SDK para permitir que o IDT interaja com o dispositivo em teste e relate os resultados do teste. Neste tutorial, você usará a versão Python do SDK.

Na pasta *<device-tester-extract-location>*`/sdks/python/`, copie a pasta `idt_client` para sua pasta `MyTestSuite_1.0.0/suite/myTestGroup/myTestCase`.

Para verificar se o SDK foi copiado, execute o comando a seguir.

```
cd MyTestSuite_1.0.0/suite/myTestGroup/myTestCase
python3 -c 'import idt_client'
```

Criação do executável do caso de teste

Os executáveis do caso de teste contêm a lógica de teste que você deseja executar. Um conjunto de testes pode conter vários executáveis de casos de teste. Para este tutorial, você criará somente um executável de caso de teste.

1. Criação de arquivo do conjunto de testes.

   Na pasta `MyTestSuite_1.0.0/suite/myTestGroup/myTestCase`, crie um arquivo `myTestCase.py` com o seguinte conteúdo:

   ```
   from idt_client import *

   def main():
       # Use the client SDK to communicate with IDT
   ```

```
    client = Client()

if __name__ == "__main__":
    main()
```

2. Use as funções do SDK de cliente para adicionar a seguinte lógica de teste ao seu arquivo `myTestCase.py`:

   a. Execute um comando SSH no dispositivo em teste.

```
from idt_client import *

def main():
    # Use the client SDK to communicate with IDT
    client = Client()

    # Create an execute on device request
    exec_req = ExecuteOnDeviceRequest(ExecuteOnDeviceCommand("echo 'hello
 world'"))

    # Run the command
    exec_resp = client.execute_on_device(exec_req)

    # Print the standard output
    print(exec_resp.stdout)

if __name__ == "__main__":
    main()
```

   b. Envie o resultado do teste para o IDT.

```
from idt_client import *

def main():
    # Use the client SDK to communicate with IDT
    client = Client()

    # Create an execute on device request
    exec_req = ExecuteOnDeviceRequest(ExecuteOnDeviceCommand("echo 'hello
 world'"))

    # Run the command
    exec_resp = client.execute_on_device(exec_req)
```

```
    # Print the standard output
    print(exec_resp.stdout)

    # Create a send result request
    sr_req = SendResultRequest(TestResult(passed=True))

    # Send the result
    client.send_result(sr_req)

if __name__ == "__main__":
    main()
```

Configuração das informações do dispositivo para o IDT

Configure as informações doseu dispositivo para o IDT executar o teste. Você deve atualizar o modelo `device.json` localizado na pasta *<device-tester-extract-location>*`/configs` com as seguintes informações.

```
[
  {
    "id": "pool",
    "sku": "N/A",
    "devices": [
      {
        "id": "<device-id>",
        "connectivity": {
          "protocol": "ssh",
          "ip": "<ip-address>",
          "port": "<port>",
          "auth": {
            "method": "pki | password",
            "credentials": {
              "user": "<user-name>",
              "privKeyPath": "/path/to/private/key",
              "password": "<password>"
            }
          }
        }
      }
    ]
  }
```

```
]
```

No objeto `devices`, forneça as seguintes informações:

`id`

Um identificador exclusivo, definido pelo usuário, para o seu dispositivo.

`connectivity.ip`

O endereço IP do seu dispositivo.

`connectivity.port`

Opcional. O número da porta que deve ser usado nas conexões SSH ao seu dispositivo.

`connectivity.auth`

Informações de autenticação da conexão.

Essa propriedade será aplicada somente se `connectivity.protocol` estiver definido como `ssh`.

`connectivity.auth.method`

O método de autenticação usado para acessar um dispositivo pelo protocolo de conectividade indicado.

Os valores compatíveis são:

- `pki`
- `password`

`connectivity.auth.credentials`

As credenciais usadas para autenticação.

`connectivity.auth.credentials.user`

O nome de usuário usado para fazer login no seu dispositivo.

`connectivity.auth.credentials.privKeyPath`

O caminho completo para a chave privada usada para fazer login no dispositivo.

Esse valor se aplica somente se `connectivity.auth.method` estiver definido como `pki`.

`devices.connectivity.auth.credentials.password`

A senha usada para fazer login no dispositivo.

Esse valor se aplica somente se `connectivity.auth.method` estiver definido como `password`.

> **Note**
>
> Especifique `privKeyPath` somente se `method` estiver definido como `pki`.
> Especifique `password` somente se `method` estiver definido como `password`.

Execução de conjunto de testes

Depois de criar o conjunto de testes, verifique se ele funciona conforme o esperado. Para isso, conclua as etapas a seguir para executar o conjunto de testes em seu grupo de dispositivos existente.

1. Copie a pasta `MyTestSuite_1.0.0` em *<device-tester-extract-location>*`/tests`.
2. Execute os seguintes comandos:

```
cd <device-tester-extract-location>/bin
./devicetester_[linux | mac | win_x86-64] run-suite --suite-id MyTestSuite
```

O IDT executa seu conjunto de testes e transmite os resultados para o console. Quando a execução do teste é concluída, você vê as seguintes informações:

```
time="2020-10-19T09:24:47-07:00" level=info msg=Using pool: pool
time="2020-10-19T09:24:47-07:00" level=info msg=Using test suite "MyTestSuite_1.0.0"
 for execution
time="2020-10-19T09:24:47-07:00" level=info msg=b'hello world\n'
 suiteId=MyTestSuite groupId=myTestGroup testCaseId=myTestCase deviceId=my-device
 executionId=9a52f362-1227-11eb-86c9-8c8590419f30
time="2020-10-19T09:24:47-07:00" level=info msg=All tests finished.
 executionId=9a52f362-1227-11eb-86c9-8c8590419f30
time="2020-10-19T09:24:48-07:00" level=info msg=

========== Test Summary ==========
```

```
Execution Time:         1s
Tests Completed:        1
Tests Passed:           1
Tests Failed:           0
Tests Skipped:          0
---------------------------------
Test Groups:
    myTestGroup:        PASSED
---------------------------------
Path to IoT Device Tester Report: /path/to/devicetester/
results/9a52f362-1227-11eb-86c9-8c8590419f30/awsiotdevicetester_report.xml
Path to Test Execution Logs: /path/to/devicetester/
results/9a52f362-1227-11eb-86c9-8c8590419f30/logs
Path to Aggregated JUnit Report: /path/to/devicetester/
results/9a52f362-1227-11eb-86c9-8c8590419f30/MyTestSuite_Report.xml
```

Solução de problemas

Use as informações a seguir para ajudar a resolver os problemas ao concluir o tutorial.

Caso de teste não é executado

Se o teste não for executado, o IDT transmitirá os logs de erro para o console e isso pode ajudar a solucionar o problema de execução do teste. Antes de verificar os logs de erros, confira:

- O IDT Client SDK está na pasta correta, conforme descrito nesta etapa.
- Você atende a todos os pré-requisitos deste tutorial.

Não é possível se conectar ao dispositivo em teste

Verifique o seguinte:

- Seu arquivo `device.json` contém o endereço IP, a porta e as informações de autenticação corretos.
- Você pode se conectar ao seu dispositivo via SSH a partir do seu computador host.

Crie arquivos de configuração do pacote de testes do IDT

Esta seção descreve os formatos nos quais você cria arquivos de configuração JSON incluídos ao escrever um pacote de teste personalizado.

Arquivos de configuração necessária

`suite.json`

Contém informações sobre o pacote de teste. Consulte Configurar suite.json.

`group.json`

Contém informações sobre um grupo de testes. Você deve criar um arquivo `group.json` para cada grupo de testes em seu conjunto de testes. Consulte Configuração de group.json.

`test.json`

Contém informações sobre um caso de testes. Você deve criar um arquivo `test.json` para cada caso de teste em seu conjunto de testes. Consulte Configuração de test.json.

Arquivos de configuração opcional

`test_orchestrator.yaml` ou `state_machine.json`

Define como os testes são executados quando o IDT executa o pacote de teste. Consulte Configurar test_orchestrator.yaml.

> **Note**
>
> A partir do IDT v4.5.1, você usa o `test_orchestrator.yaml` arquivo para definir o fluxo de trabalho de teste. Nas versões anteriores do IDT, você usa o arquivo `state_machine.json`. Para obter informações sobre a máquina de estado, consulte Configurar a máquina de estado IDT.

`userdata_schema.json`

Define o esquema do arquivo `userdata.json` que os executores de teste podem incluir na definição de configuração. O arquivo `userdata.json` é usado em toda informação de configuração adicional necessária para executar o teste, mas que não esteja presente no arquivo `device.json`. Consulte Configurar userdata_schema.json.

Os arquivos de configuração JSON são colocados no seu *<custom-test-suite-folder>*, conforme mostrado aqui.

```
<custom-test-suite-folder>
### suite
    ### suite.json
    ### test_orchestrator.yaml
    ### userdata_schema.json
    ### <test-group-folder>
        ### group.json
        ### <test-case-folder>
            ### test.json
```

Configurar suite.json

O arquivo `suite.json` define as variáveis de ambiente e determina se os dados do usuário são necessários para executar o conjunto de testes. Use o modelo a seguir para configurar seu arquivo *<custom-test-suite-folder>*`/suite/suite.json`:

```
{
    "id": "<suite-name>_<suite-version>",
    "title": "<suite-title>",
    "details": "<suite-details>",
    "userDataRequired": true | false,
    "environmentVariables": [
        {
            "key": "<name>",
            "value": "<value>",
        },
        ...
        {
            "key": "<name>",
            "value": "<value>",
        }
    ]
}
```

Todos os campos que contêm valores são necessários, conforme descrito aqui:

`id`

Um ID exclusivo, definido pelo usuário, para o conjunto de testes. O valor de `id` deve corresponder ao nome da pasta do conjunto de testes na qual o arquivo `suite.json` está localizado. O nome e a versão do conjunto devem atender aos seguintes requisitos:

- *<suite-name>* não pode conter sublinhados.
- *<suite-version>* é indicado como *x.x.x*, em que x é um número.

O ID é mostrado nos relatórios de teste gerados pelo IDT.

`title`

Um nome definido pelo usuário para o produto ou atributo que está sendo testado por esse conjunto de testes. O nome é exibido na CLI do IDT para executores de teste.

`details`

Uma descrição breve da finalidade do conjunto de testes.

`userDataRequired`

Define se os executores de teste precisam incluir informações personalizadas em um arquivo `userdata.json`. Se você definir esse valor como `true`, também deverá incluir o arquivo `userdata_schema.json` na pasta do conjunto de testes.

`environmentVariables`

Opcional. Uma matriz de variáveis de ambiente para configurar para esse conjunto de testes.

`environmentVariables.key`

O nome da variável de ambiente.

`environmentVariables.value`

O valor da variável de ambiente.

Configuração de group.json

O arquivo `group.json` define se o grupo de teste é necessário ou opcional. Use o modelo a seguir para configurar seu arquivo *<custom-test-suite-folder>*`/suite/`*<test-group>*`/group.json`:

```
{
    "id": "<group-id>",
    "title": "<group-title>",
    "details": "<group-details>",
    "optional": true | false,
```

```
}
```

Todos os campos que contêm valores são necessários, conforme descrito aqui:

`id`

Um ID exclusivo, definido pelo usuário, para o grupo de testes. O valor de `id` deve corresponder ao nome da pasta do grupo de teste na qual o `group.json` arquivo está localizado e não pode conter sublinhados (_). O ID é usado nos relatórios de teste gerados pelo IDT.

`title`

Um nome descritivo para o grupo de teste. O nome é exibido na CLI do IDT para executores de teste.

`details`

Uma descrição breve da finalidade do grupo de testes.

`optional`

Opcional. Defina `true` para exibir esse grupo de teste como um grupo opcional depois que o IDT terminar de executar os testes necessários. O valor padrão é `false`.

Configuração de test.json

O arquivo `test.json` determina os executáveis do caso de teste e as variáveis de ambiente que são usadas por um caso de teste. Para mais informações sobre como criar executáveis de casos de teste, consulte Crie executáveis de casos de teste do IDT.

Use o modelo a seguir para configurar seu arquivo *<custom-test-suite-folder>*/suite/*<test-group>*/*<test-case>*/test.json:

```
{
    "id": "<test-id>",
    "title": "<test-title>",
    "details": "<test-details>",
    "requireDUT": true | false,
    "requiredResources": [
        {
            "name": "<resource-name>",
            "features": [
                {
```

```
                    "name": "<feature-name>",
                    "version": "<feature-version>",
                    "jobSlots": <job-slots>
                }
            ]
        }
    ],
    "execution": {
        "timeout": <timeout>,
        "mac": {
            "cmd": "/path/to/executable",
            "args": [
                "<argument>"
            ],
        },
        "linux": {
            "cmd": "/path/to/executable",
            "args": [
                "<argument>"
            ],
        },
        "win": {
            "cmd": "/path/to/executable",
            "args": [
                "<argument>"
            ]
        }
    },
    "environmentVariables": [
        {
            "key": "<name>",
            "value": "<value>",
        }
    ]
}
```

Todos os campos que contêm valores são necessários, conforme descrito aqui:

`id`

Um ID exclusivo, definido pelo usuário, para o caso de teste. O valor de `id` deve corresponder ao nome da pasta do caso de teste na qual o `test.json` arquivo está localizado e não pode conter sublinhados (_). O ID é usado nos relatórios de teste gerados pelo IDT.

`title`

Um nome descritivo para o caso de teste. O nome é exibido na CLI do IDT para executores de teste.

`details`

Uma descrição breve da finalidade do caso de teste.

`requireDUT`

Opcional. Defina como `true` se for necessário um dispositivo para executar esse teste, caso contrário, defina como `false`. O valor padrão é `true`. Os executores de teste configurarão os dispositivos que usarão para executar o teste em seus arquivos `device.json`.

`requiredResources`

Opcional. Uma matriz que fornece informações sobre os dispositivos de atributos necessários para executar esse teste.

`requiredResources.name`

O nome exclusivo para dar ao dispositivo de atributo quando esse teste está sendo executado.

`requiredResources.features`

Uma matriz de recursos de dispositivo de atributos definidos pelo usuário.

`requiredResources.features.name`

O nome do atributo. O atributo do dispositivo para o qual você deseja usar este dispositivo. Esse nome é comparado ao nome do atributo fornecido pelo executor de teste no arquivo `resource.json`.

`requiredResources.features.version`

Opcional. A versão do atributo. Esse valor é comparado com a versão do atributo fornecida pelo executor de teste no arquivo `resource.json`. Se uma versão não for fornecida, o atributo não será verificado. Caso o número de versão não seja necessário para o atributo, deixe este campo em branco.

`requiredResources.features.jobSlots`

Opcional. O número de testes simultâneos que esse atributo pode suportar. O valor padrão é 1. Se desejar que o IDT use dispositivos distintos para atributos individuais, recomendamos que definir esse valor como 1.

`execution.timeout`

A período de tempo (em milissegundos) que o IDT aguarda para o teste terminar de ser executado. Para obter mais informações sobre essa configuração, consulte Crie executáveis de casos de teste do IDT.

`execution.os`

Os executáveis do caso de teste a serem executados com base no sistema operacional do computador host que executa o IDT. Os valores compatíveis são `linux`, `mac` e `win`.

`execution.os.cmd`

O caminho para o executável do caso de teste que você deseja executar para o sistema operacional especificado. Esse local deve estar no caminho do sistema.

`execution.os.args`

Opcional. Os argumentos a serem fornecidos para executar o executável do caso de teste.

`environmentVariables`

Opcional. Uma matriz de variáveis de ambiente definidas para esse caso de teste.

`environmentVariables.key`

O nome da variável de ambiente.

`environmentVariables.value`

O valor da variável de ambiente.

> **Note**
>
> Se especificar a mesma variável de ambiente no arquivo `test.json` e no arquivo `suite.json`, o valor no arquivo `test.json` terá precedência.

Configurar test_orchestrator.yaml

Um orquestrador de teste é uma estrutura que controla o fluxo de execução do pacote de teste. Ele determina o estado inicial de um pacote de teste, gerencia as transições de estado com base nas regras definidas pelo usuário e continua a transição por esses estados até atingir o estado final.

Se seu pacote de teste não incluir um orquestrador de testes definido pelo usuário, o IDT gerará um orquestrador de testes para você.

O orquestrador de teste padrão executa as seguintes funções:

- Fornece aos executores de teste a capacidade de selecionar e executar grupos de testes específicos, em vez de todo o pacote de teste.
- Se grupos de teste específicos não forem selecionados, executará cada grupo de teste no conjunto de testes em uma ordem aleatória.
- Gera relatórios e imprime um resumo do console que mostra os resultados do teste para cada grupo de teste e caso de teste.

Para obter mais informações sobre como o orquestrador de testes do IDT funciona, consulte Configurar o orquestrador de teste IDT.

Configurar userdata_schema.json

O arquivo `userdata_schema.json` determina o esquema no qual os executores de teste fornecem dados do usuário. Os dados do usuário são necessários se seu conjunto de testes exigir informações que não estejam presentes no arquivo `device.json`. Por exemplo, seus testes podem precisar de credenciais de rede Wi-Fi, portas abertas específicas ou certificados que um usuário deve fornecer. Essas informações podem ser fornecidas ao IDT como um parâmetro de entrada chamado `userdata`, cujo valor é um arquivo `userdata.json`, que os usuários criam em suas pastas *`<device-tester-extract-location>`*`/config`. O formato do arquivo `userdata.json` é baseado no arquivo `userdata_schema.json` que você inclui no conjunto de testes.

Para indicar que os executores de teste devem fornecer um arquivo `userdata.json`:

1. No arquivo `suite.json`, defina `userDataRequired` como `true`.
2. No seu *`<custom-test-suite-folder>`*, crie um arquivo `userdata_schema.json`.
3. Edite o arquivo `userdata_schema.json` para criar um esquema JSON de IETF Draft v4.

Quando o IDT executa seu pacote de teste, ele lê automaticamente o esquema e o usa para validar o arquivo `userdata.json` fornecido pelo executor do teste. Se válido, o conteúdo do arquivo `userdata.json` estará disponível no contexto do IDT e no contexto do orquestrador de testes.

Configurar o orquestrador de teste IDT

A partir do IDT v4.5.1, o IDT inclui um novoorquestrador de testeComponente. O orquestrador de teste é um componente IDT que controla o fluxo de execução do conjunto de testes e gera o relatório

de teste após o IDT terminar de executar todos os testes. O orquestrador de teste determina a seleção de teste e a ordem em que os testes são executados com base em regras definidas pelo usuário.

Se o conjunto de testes não incluir um orquestrador de teste definido pelo usuário, o IDT gerará um orquestrador de teste para você.

O orquestrador de teste padrão executa as seguintes funções:

- Fornece aos corredores de teste a capacidade de selecionar e executar grupos de teste específicos, em vez de todo o conjunto de testes.
- Se grupos de teste específicos não forem selecionados, executa todos os grupos de teste na suíte de testes em uma ordem aleatória.
- Gera relatórios e imprime um resumo do console que mostra os resultados do teste para cada grupo de teste e caso de teste.

O orquestrador de teste substitui o orquestrador de teste IDT. É altamente recomendável que você use o orquestrador de teste para desenvolver seus conjuntos de testes em vez do orquestrador de teste IDT. O orquestrador de teste fornece os seguintes recursos aprimorados:

- Usa um formato declarativo comparado ao formato imperativo que a máquina de estado IDT usa. Isso permite que vocêespecificamosquais testes você deseja executar equandovocê quer executá-los.
- Gerencia o tratamento de grupos específicos, a geração de relatórios, o tratamento de erros e o rastreamento de resultadospara que você não seja necessáriopara gerenciar manualmente essas ações.
- Usa o formato YAML, que suporta comentários por padrão.
- Requer80 por centomenos espaço em disco do que o orquestrador de teste para definir o mesmo fluxo de trabalho.
- Adiciona validação pré-teste para verificar se a definição do fluxo de trabalho não contém IDs de teste incorretas ou dependências circulares.

Formato do orquestrador de teste

Você pode usar o seguinte modelo para configurar o seu próprio*<custom-test-suite-folder>*/`suite/test_orchestrator.yaml`file:

```
Aliases:
  string: context-expression

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

Order:
  - - group-descriptor
    - group-descriptor

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

Todos os campos que contêm valores são necessários, conforme descrito aqui:

`Aliases`

Opcional. Strings definidas pelo usuário que são mapeadas para expressões de contexto. Aliases permitem que você gere nomes amigáveis paraidentifique expressões de contexto na configuração do orquestrador de teste. Isso será especialmente útil se você estiver criando expressões de contexto complexas ou expressões que você usa em vários lugares.

Você pode usar expressões de contexto para armazenar consultas de contexto que permitem acessar dados de outras configurações IDT. Para obter mais informações, consulteAcesse dados no contexto

Example Exemplo

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

`ConditionalTests`

Opcional. Uma lista de condições e os casos de teste correspondentes que são executados quando cada condição é satisfeita. Cada condição pode ter vários casos de teste; no entanto, você pode atribuir um determinado caso de teste a apenas uma condição.

Por padrão, o IDT executa qualquer caso de teste que não esteja atribuído a uma condição nesta lista. Se você não especificar essa seção, o IDT executa todos os grupos de teste no conjunto de testes.

Cada item no`ConditionalTests`A list inclui os seguintes parâmetros:

`Condition`

Uma expressão de contexto que avalia para umboleanovalue. Se o valor avaliado for verdadeiro, o IDT executará os casos de teste especificados na`Tests`parâmetro .

`Tests`

A lista de descritores de teste.

Cada descritor de teste usa o ID do grupo de teste e um ou mais IDs de caso de teste para identificar os testes individuais a serem executados a partir de um grupo de teste específico. O descritor de teste usa o seguinte formato:

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

Example Exemplo

O exemplo a seguir usa expressões de contexto genéricas que você pode definir como`Aliases`.

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

Com base nas condições definidas, o IDT seleciona grupos de teste da seguinte forma:

- Se`Condition1`é verdade, o IDT executa os testes nos grupos de teste A, B e C.
- Se`Condition2`é verdade, o IDT executa os testes nos grupos de teste C e D.

`Order`

Opcional. A ordem em que os testes serão executados. Você especifica a ordem de teste no nível do grupo de teste. Se você não especificar esta seção, o IDT executará todos os grupos de teste aplicáveis em uma ordem aleatória. O valor de`Order`é uma lista de listas de descritores de grupo. Qualquer grupo de teste que você não listar`Order`, pode ser executado em parallel com qualquer outro grupo de teste listado.

Cada lista de descritores de grupo contém um dos mais descritores de grupo e identifica a ordem na qual executar os grupos especificados em cada descritor. Você pode usar os formatos a seguir para definir descritores de grupo individuais:

- *`group-id`*—O ID de grupo de um grupo de teste existente.
- [*`group-id`*, *`group-id`*]—Lista de grupos de teste que podem ser executados em qualquer ordem em relação um ao outro.
- `"*"`—Curinga. Isso equivale à lista de todos os grupos de teste que ainda não estão especificados na lista de descritores de grupo atual.

O valor para`Order`Também deve atender aos seguintes requisitos:

- As IDs de grupo de teste especificadas em um descritor de grupo devem existir em seu conjunto de testes.
- Cada lista de descritores de grupo deve incluir pelo menos um grupo de teste.
- Cada lista de descritores de grupo deve conter IDs de grupo exclusivas. Você não pode repetir um ID de grupo de teste em descritores de grupo individuais.
- Uma lista de descritores de grupo pode ter no máximo um descritor de grupo curinga. O descritor de grupo curinga deve ser o primeiro ou o último item da lista.

Example Exemplos

Para um conjunto de testes que contém os grupos de teste A, B, C, D e E, a lista de exemplos a seguir mostra maneiras diferentes de especificar que o IDT deve primeiro executar o grupo de teste A, depois executar o grupo de teste B e, em seguida, executar os grupos de teste C, D e E em qualquer ordem.

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

      - - B
        - C

      - - B
        - D

      - - B
        - E
  ```

`Features`

Opcional. A lista de recursos do produto que você deseja que o IDT adicione aoawsiotdevicetester_report.xmlfile. Se você não especificar esta seção, o IDT não adicionará nenhum recurso do produto ao relatório.

Um recurso do produto são informações definidas pelo usuário sobre critérios específicos que um dispositivo pode atender. Por exemplo, o recurso do produto MQTT pode designar que o dispositivo publica mensagens MQTT corretamente. Dentroawsiotdevicetester_report.xml, os recursos do produto são definidos comosupported,not-supported, ou um valor personalizado definido pelo usuário, com base na aprovação dos testes especificados.

Cada item noFeaturesA list consiste nos seguintes parâmetros:

`Name`

O nome do recurso.
  ```

`Value`

Opcional. O valor personalizado que você deseja usar no relatório em vez de`supported`. Se esse valor não for especificado, o IDT baseado definirá o valor do recurso como`supported`ou`not-supported`Com base nos resultados dos testes. Se você testar o mesmo recurso com condições diferentes, poderá usar um valor personalizado para cada instância desse recurso na`Features`list, e o IDT concatena os valores do recurso para as condições suportadas. Para obter mais informações, consulte

`Condition`

Uma expressão de contexto que avalia para umboleanovalue. Se o valor avaliado for verdadeiro, o IDT adicionará o recurso ao relatório de teste depois que ele terminar de executar o conjunto de testes. Se o valor avaliado for falso, o teste não será incluído no relatório.

`Tests`

Opcional. A lista de descritores de teste. Todos os testes especificados nesta lista devem ser aprovados para que o recurso seja suportado.

Cada descritor de teste nesta lista usa o ID do grupo de teste e um ou mais IDs de caso de teste para identificar os testes individuais a serem executados a partir de um grupo de teste específico. O descritor de teste usa o seguinte formato:

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

Especifique um`Tests`ou`OneOfTests`para cada recurso no`Features`Lista.

`OneOfTests`

Opcional. A lista de descritores de teste. Pelo menos um dos testes especificados nesta lista deve ser aprovado para que o recurso seja suportado.

Cada descritor de teste nesta lista usa o ID do grupo de teste e um ou mais IDs de caso de teste para identificar os testes individuais a serem executados a partir de um grupo de teste específico. O descritor de teste usa o seguinte formato:

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

Especifique um`Testsou0ne0fTests`para cada recurso no`Features`Lista.

`IsRequired`

O valor booleano que define se o recurso é necessário no relatório de teste. O valor padrão é `false`.

Example

Teste de contexto de orquestrador

O contexto do orquestrador de teste é um documento JSON somente leitura que contém dados disponíveis para o orquestrador de teste durante a execução. O contexto do orquestrador de teste só é acessível a partir do orquestrador de teste e contém informações que determinam o fluxo de teste. Por exemplo, você pode usar informações configuradas por corredores de teste na`userdata.json`arquivo para determinar se um teste específico é necessário para ser executado.

O contexto do orquestrador de teste usa o seguinte formato:

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

`pool`

Informações sobre o pool de dispositivos selecionado para a execução de teste. Para um pool de dispositivos selecionado, essas informações são recuperadas do elemento de matriz de pool de dispositivos de nível superior correspondente definido na`device.json`file.

`userData`

Informações do no`userdata.json`file.

`config`

Informações do no`config.json`file.

Você pode consultar o contexto usando a notação JSONPath. A sintaxe para consultas JSONPath em definições de estado é`{{query}}`. Ao acessar dados do contexto do orquestrador de teste, certifique-se de que cada valor seja avaliado como uma string, um número ou umboleano.

Para obter mais informações sobre como usar a notação JSONPath para acessar dados do contexto, consulteUsar o contexto IDT.

Configurar a máquina de estado IDT

> **Important**
>
> A partir do IDT v4.5.1, esta máquina de estado está obsoleta. Recomendamos que você utilize o novo orquestrador de teste. Para obter mais informações, consulteConfigurar o orquestrador de teste IDT

Uma máquina de estado é uma construção que controla o fluxo de execução do conjunto de testes. Ele determina o estado inicial de um conjunto de testes, gerencia transições de estado com base em regras definidas pelo usuário e continua a fazer a transição por esses estados até atingir o estado final.

Se o conjunto de testes não incluir uma máquina de estado definida pelo usuário, o IDT gerará uma máquina de estado para você. A máquina de estado padrão executa as seguintes funções:

- Fornece aos corredores de teste a capacidade de selecionar e executar grupos de teste específicos, em vez de todo o conjunto de testes.
- Se grupos de teste específicos não forem selecionados, executa todos os grupos de teste na suíte de testes em uma ordem aleatória.
- Gera relatórios e imprime um resumo do console que mostra os resultados do teste para cada grupo de teste e caso de teste.

A máquina de estado para um conjunto de testes IDT deve atender aos seguintes critérios:

- Cada estado corresponde a uma ação para o IDT executar, como executar um grupo de teste ou produto um arquivo de relatório.
- A transição para um estado executa a ação associada ao estado.
- Cada estado define a regra de transição para o próximo estado.
- O estado final deve ser`Succeed`ou`Fail`.

Formato de máquina de estado

Você pode usar o seguinte modelo para configurar o seu próprio*<custom-test-suite-folder>*/`suite/state_machine.json`file:

```
{
  "Comment": "<description>",
  "StartAt": "<state-name>",
  "States": {
    "<state-name>": {
      "Type": "<state-type>",
      // Additional state configuration
    }

    // Required states
    "Succeed": {
      "Type": "Succeed"
    },
    "Fail": {
      "Type": "Fail"
    }
  }
}
```

Todos os campos que contêm valores são necessários, conforme descrito aqui:

`Comment`

Uma descrição da máquina de estado.

`StartAt`

Nome do estado no qual o IDT começa a executar o conjunto de testes. O valor de`StartAt`Deve ser definido como um dos estados listados no`States`objeto.

`States`

Um objeto que mapeia nomes de estados definidos pelo usuário para estados IDT válidos. Cada estado.*`Nome do estado`*objeto contém a definição de um estado válido mapeado para o*`Nome do estado`*.

O`States`objeto deve incluir o`Succeed`e`Fail`Estados. Para obter informações sobre estados válidos, consulteDefinições de estados e estados válidos.

Definições de estados e estados válidos

Esta seção descreve as definições de estado de todos os estados válidos que podem ser usados na máquina de estado IDT. Alguns dos seguintes estados oferecem suporte a configurações no nível do caso de teste. No entanto, recomendamos que você configure regras de transição de estado no nível do grupo de teste em vez do nível do caso de teste, a menos que seja absolutamente necessário.

Definições de estado

- RunTask
- Choice
- Parallel
- AddProductFeatures
- Relatório
- LogMessage
- SelectGroup
- Fail
- Succeed

RunTask

O`RunTask`O state executa casos de teste de um grupo de teste definido no conjunto de testes.

```
{
    "Type": "RunTask",
    "Next": "<state-name>",
    "TestGroup": "<group-id>",
    "TestCases": [
        "<test-id>"
```

```
    ],
    "ResultVar": "<result-name>"
}
```

Todos os campos que contêm valores são necessários, conforme descrito aqui:

`Next`

Nome do estado para o qual deve mudar após a execução das ações no estado atual.

`TestGroup`

Opcional. O ID do grupo de teste a ser executado. Se esse valor não for especificado, o IDT executará o grupo de teste selecionado pelo executor de teste.

`TestCases`

Opcional. Uma matriz de IDs de caso de teste do grupo especificado em`TestGroup`. Com base nos valores de`TestGroup`e`TestCases`, IDT determina o comportamento de execução do teste da seguinte forma:

- Quando ambos`TestGroup`e`TestCases`são especificados, o IDT executa os casos de teste especificados do grupo de teste.
- Quando`TestCases`são especificados, mas`TestGroup`não é especificado, o IDT executa os casos de teste especificados.
- Quando`TestGroup`é especificado, mas`TestCases`Não é especificado, o IDT executa todos os casos de teste no grupo de teste especificado.
- Quando nenhum dos dois`TestGroup`ou`TestCase`sé especificado, o IDT executa todos os casos de teste do grupo de teste que o executor de teste seleciona da IDT CLI. Para habilitar a seleção de grupo para corredores de teste, você deve incluir ambos`RunTask`e`Choice`estados em seu`state_machine.json`file. Para obter um exemplo de como isso funciona, consulteExemplo de máquina de estado do: Execute grupos de teste selecionados pelo usuário.

  Para obter mais informações sobre como habilitar comandos IDT CLI para executores de teste, consultethe section called “Habilitar comandos da IDT CLI CLI CLI”.

`ResultVar`

O nome da variável de contexto a ser definida com os resultados da execução do teste. Não especifique esse valor se você não especificou um valor para`TestGroup`. IDT define o valor da variável que você define em`ResultVar`para`true`ou`false`Com base no seguinte:

- Se o nome da variável for do formulário*text*_*text*_passed, então o valor é definido para se todos os testes no primeiro grupo de teste passaram ou foram ignorados.
- Em todos os outros casos, o valor é definido para se todos os testes em todos os grupos de teste passaram ou foram ignorados.

Normalmente, você vai usarRunTaskstate para especificar um ID de grupo de teste sem especificar IDs de caso de teste individuais, para que o IDT execute todos os casos de teste no grupo de teste especificado. Todos os casos de teste executados por esse estado são executados em parallel, em uma ordem aleatória. No entanto, se todos os casos de teste exigirem que um dispositivo seja executado e apenas um único dispositivo estiver disponível, os casos de teste serão executados sequencialmente.

Como tratar erros

Se algum dos grupos de teste ou IDs de caso de teste especificados não for válido, esse estado emitirá oRunTaskErrorErro de execução do. Se o estado encontrar um erro de execução, ele também definirá ohasExecutionErrorvariável no contexto da máquina de estado paratrue.

Choice

OChoicestate permite definir dinamicamente o próximo estado para a transição com base nas condições definidas pelo usuário.

```
{
    "Type": "Choice",
    "Default": "<state-name>",
    "FallthroughOnError": true | false,
    "Choices": [
        {
            "Expression": "<expression>",
            "Next": "<state-name>"
        }
    ]
}
```

Todos os campos que contêm valores são necessários, conforme descrito aqui:

`Default`

O estado padrão para o qual deve mudar se nenhuma das expressões definidas no`Choices`pode ser avaliado para`true`.

`FallthroughOnError`

Opcional. Especifica o comportamento quando o estado encontra um erro na avaliação de expressões. Definido como`true`se você quiser pular uma expressão se a avaliação resultar em um erro. Se nenhuma expressão corresponder, a máquina de estado muda para o`Default`Estado. Se o`FallthroughOnError`O valor não é especificado, o padrão é`false`.

`Choices`

Uma matriz de expressões e estados para determinar para qual estado fazer a transição após a execução das ações no estado atual.

`Choices.Expression`

Uma string de expressão que avalia como um valor booleano. Se a expressão for avaliada como`true`, depois a máquina de estado muda para o estado definido no`Choices.Next`. As strings de expressão recuperam valores do contexto da máquina de estado e, em seguida, executam operações nelas para chegar a um valor booleano. Para obter informações sobre como acessar o contexto da máquina de estado, consulteContexto de máquina de estado.

`Choices.Next`

Nome do estado para o qual deve mudar se a expressão definida no`Choices.Expression`avalia para`true`.

Como tratar erros

O`Choice`estate pode exigir tratamento de erros nos seguintes casos:

- Algumas variáveis nas expressões de escolha não existem no contexto da máquina de estado.
- O resultado de uma expressão não é um valor booleano.
- O resultado de uma pesquisa JSON não é uma string, número ou booleano.

Não é possível usar um`Catch`Bloquear para lidar com erros nesse estado. Se você quiser parar de executar a máquina de estado quando ela encontrar um erro, você deve definir`FallthroughOnError`para`false`. No entanto, recomendamos que você

definaFallthroughOnErrorparatrueE, dependendo do seu caso de uso, siga um destes procedimentos:

- Se uma variável que você está acessando não exista em alguns casos, use o valor deDefaulte adicionalChoicesBlocos para especificar o próximo estado.
- Se uma variável que você está acessando sempre existir, defina aDefaultEstado paraFail.

Parallel

OParallelstate permite definir e executar novas máquinas de estado em parallel entre si.

```
{
    "Type": "Parallel",
    "Next": "<state-name>",
    "Branches": [
        <state-machine-definition>
    ]
}
```

Todos os campos que contêm valores são necessários, conforme descrito aqui:

`Next`

Nome do estado para o qual deve mudar após a execução das ações no estado atual.

`Branches`

Uma matriz de definições de máquina de estado a ser executada. Cada definição de máquina de estado deve conter a sua própriaStartAt,Succeed, eFailEstados. As definições de máquina de estado nesta matriz não podem fazer referência a estados fora de sua própria definição.

> **Note**
>
> Como cada máquina de estado de ramificação compartilha o mesmo contexto de máquina de estado, definir variáveis em uma ramificação e depois ler essas variáveis de outra ramificação pode resultar em comportamento inesperado.

OParallelstate se move para o próximo estado somente depois que ele executa todas as máquinas de estado de ramificação. Cada estado que requer um dispositivo aguardará a execução

até que o dispositivo esteja disponível. Se vários dispositivos estiverem disponíveis, esse estado executará casos de teste de vários grupos em parallel. Se dispositivos suficientes não estiverem disponíveis, os casos de teste serão executados sequencialmente. Como os casos de teste são executados em uma ordem aleatória quando são executados em parallel, dispositivos diferentes podem ser usados para executar testes do mesmo grupo de teste.

Como tratar erros

Certifique-se de que tanto a máquina de estado de ramificação quanto a máquina de estado pai façam a transição para o`Fail`state para lidar com erros de execução.

Como as máquinas de estado de ramificação não transmitem erros de execução para a máquina de estado pai, você não pode usar um`Catch`bloco para lidar com erros de execução em máquinas de estado de ramificação. Em vez disso, use o`hasExecutionErrors`valor no contexto da máquina de estado compartilhado. Para obter um exemplo de como isso funciona, consulteExemplo de máquina de estado do: Execute dois grupos de teste em parallel.

AddProductFeatures

O`AddProductFeatures`state permite adicionar recursos do produto ao`awsiotdevicetester_report.xml`arquivo gerado pelo IDT.

Um recurso do produto são informações definidas pelo usuário sobre critérios específicos que um dispositivo pode atender. Por exemplo, as receitas`MQTT`recurso do produto pode designar que o dispositivo publica mensagens MQTT corretamente. No relatório, os recursos do produto são definidos como`supported`,`not-supported`ou um valor personalizado, com base na aprovação dos testes especificados.

> **Note**
>
> O`AddProductFeatures`state não gera relatórios por si só. Esse estado deve fazer a transição para o`Report`estadopara gerar relatórios.

```
{
    "Type": "Parallel",
    "Next": "<state-name>",
    "Features": [
        {
            "Feature": "<feature-name>",
```

```
            "Groups": [
                "<group-id>"
            ],
            "OneOfGroups": [
                "<group-id>"
            ],
            "TestCases": [
                "<test-id>"
            ],
            "IsRequired": true | false,
            "ExecutionMethods": [
                "<execution-method>"
            ]
        }
    ]
}
```

Todos os campos que contêm valores são necessários, conforme descrito aqui:

`Next`

Nome do estado para o qual deve mudar após a execução das ações no estado atual.

`Features`

Uma variedade de recursos do produto para mostrar na`awsiotdevicetester_report.xml`file.

`Feature`

O nome do recurso

`FeatureValue`

Opcional. O valor personalizado a ser usado no relatório em vez de`supported`. Se esse valor não for especificado, com base nos resultados do teste, o valor do recurso será definido como`supported`ou`not-supported`.

Se você usar um valor personalizado para`FeatureValue`, você pode testar o mesmo recurso com condições diferentes, e o IDT concatena os valores de feição para as condições suportadas. Por exemplo, o trecho a seguir mostra o`MyFeature`recurso com dois valores de feição separados:

```
...
```

```
{
    "Feature": "MyFeature",
    "FeatureValue": "first-feature-supported",
    "Groups": ["first-feature-group"]
},
{
    "Feature": "MyFeature",
    "FeatureValue": "second-feature-supported",
    "Groups": ["second-feature-group"]
},
...
```

Se ambos os grupos de teste passarem, o valor do recurso será definido como`first-feature-supported, second-feature-supported`.

`Groups`

Opcional. Uma matriz de IDs de grupos de testes. Todos os testes dentro de cada grupo de teste especificado devem ser aprovados para que o recurso seja suportado.

`OneOfGroups`

Opcional. Uma matriz de IDs de grupos de testes. Todos os testes em pelo menos um dos grupos de teste especificados devem passar para que o recurso seja suportado.

`TestCases`

Opcional. Uma matriz de IDs de casos de teste. Se você especificar esse valor, então o seguinte se aplicará:

- Todos os casos de teste especificados devem ser aprovados para que o recurso seja suportado.
- `Groups`Deve conter apenas um ID do grupo de teste.
- `OneOfGroups`Não deve ser especificado.

`IsRequired`

Opcional. Definido como`false`para marcar esse recurso como um recurso opcional no relatório. O valor padrão é `true`.

`ExecutionMethods`

Opcional. Uma matriz de métodos de execução que correspondem ao`protocol`valor especificado no`device.json`file. Se esse valor for especificado, os corredores de teste

deverão especificar um`protocol`valor que corresponde a um dos valores dessa matriz para incluir o recurso no relatório. Se esse valor não for especificado, o recurso sempre será incluído no relatório.

Para usar a`AddProductFeatures`state, você deve definir o valor de`ResultVar`no`RunTask`State para um dos seguintes valores:

- Se você especificou IDs de caso de teste individuais, defina`ResultVar`para*`group-id_test-id`*`_passed`.
- Se você não especificou IDs de caso de teste individuais, defina`ResultVar`para*`group-id`*`_passed`.

O`AddProductFeatures`Verificações de estado para os resultados do teste da seguinte maneira:

- Se você não especificou nenhuma IDs de caso de teste, o resultado de cada grupo de teste será determinado a partir do valor do*`group-id`*`_passed`variável no contexto da máquina de estado.
- Se você especificou IDs de caso de teste, o resultado de cada um dos testes será determinado a partir do valor do*`group-id_test-id`*`_passed`variável no contexto da máquina de estado.

Como tratar erros

Se uma ID de grupo fornecida nesse estado não for uma ID de grupo válida, esse estado resultará na`AddProductFeaturesError`Erro de execução do. Se o estado encontrar um erro de execução, ele também definirá o`hasExecutionErrors`variável no contexto da máquina de estado para`true`.

Relatório

O`Report`O estado gera o*`suite-name`*`_Report.xml`e`awsiotdevicetester_report.xml`arquivos. Esse estado também transmite o relatório para o console.

```
{
    "Type": "Report",
    "Next": "<state-name>"
}
```

Todos os campos que contêm valores são necessários, conforme descrito aqui:

`Next`

Nome do estado para o qual deve mudar após a execução das ações no estado atual.

Você sempre deve fazer a transição para o`Report`estado no final do fluxo de execução do teste para que os corredores de teste possam visualizar os resultados do teste. Normalmente, o próximo estado após esse estado é`Succeed`.

Como tratar erros

Se esse estado encontrar problemas com a geração dos relatórios, ele emite o`ReportError`Erro de execução do.

LogMessage

O`LogMessage`O estado gera o`test_manager.log`Arquivo e transmite a mensagem de log para o console.

```
{
    "Type": "LogMessage",
    "Next": "<state-name>"
    "Level": "info | warn | error"
    "Message": "<message>"
}
```

Todos os campos que contêm valores são necessários, conforme descrito aqui:

`Next`

Nome do estado para o qual deve mudar após a execução das ações no estado atual.

`Level`

O nível de erro no qual criar a mensagem de registro. Se você especificar um nível que não é válido, esse estado gerará uma mensagem de erro e a descarta.

`Message`

A mensagem para registrar.

SelectGroup

O`SelectGroup`state atualiza o contexto da máquina de estado para indicar quais grupos estão selecionados. Os valores definidos por esse estado são usados por qualquer subseqüente`Choice`Estados.

```
{
    "Type": "SelectGroup",
    "Next": "<state-name>"
    "TestGroups": [
        <group-id>"
    ]
}
```

Todos os campos que contêm valores são necessários, conforme descrito aqui:

`Next`

Nome do estado para o qual deve mudar após a execução das ações no estado atual.

`TestGroups`

Uma matriz de grupos de teste que serão marcados como selecionados. Para cada ID de grupo de teste nesta matriz, o*group-id*`_selected`variável está definida como`true`no contexto. Certifique-se de fornecer IDs de grupo de teste válidos porque o IDT não valida se os grupos especificados existem.

Fail

O`Fail`state indica que a máquina de estado não foi executada corretamente. Este é um estado final para a máquina de estado, e cada definição de máquina de estado deve incluir esse estado.

```
{
    "Type": "Fail"
}
```

Succeed

O`Succeed`state indica que a máquina de estado foi executada corretamente. Este é um estado final para a máquina de estado, e cada definição de máquina de estado deve incluir esse estado.

```
{
```

```
    "Type": "Succeed"
}
```

Contexto de máquina de estado

O contexto da máquina de estado é um documento JSON somente leitura que contém dados disponíveis para a máquina de estado durante a execução. O contexto da máquina de estado só é acessível a partir da máquina de estado e contém informações que determinam o fluxo de teste. Por exemplo, você pode usar informações configuradas por corredores de teste na`userdata.json`arquivo para determinar se um teste específico é necessário para ser executado.

O contexto da máquina de estado usa o seguinte formato:

```
{
    "pool": {
        <device-json-pool-element>
    },
    "userData": {
        <userdata-json-content>
    },
    "config": {
        <config-json-content>
    },
    "suiteFailed": true | false,
    "specificTestGroups": [
        "<group-id>"
    ],
    "specificTestCases": [
        "<test-id>"
    ],
    "hasExecutionErrors": true
}
```

`pool`

Informações sobre o pool de dispositivos selecionado para a execução de teste. Para um pool de dispositivos selecionado, essas informações são recuperadas do elemento de matriz de pool de dispositivos de nível superior correspondente definido na`device.json`file.

`userData`

Informações no`userdata.json`file.

`config`

Informações do fixar o`config.json`file.

`suiteFailed`

O valor é definido como`false`Quando a máquina de estado for iniciada. Se um grupo de teste falhar em um`RunTask`state, então esse valor é definido como`true`Para a duração restante da execução da máquina de estado.

`specificTestGroups`

Se o executor de teste selecionar grupos de teste específicos a serem executados em vez de todo o conjunto de testes, essa chave será criada e contém a lista de IDs de grupos de teste específicos.

`specificTestCases`

Se o executor de teste selecionar casos de teste específicos a serem executados em vez de todo o conjunto de testes, essa chave será criada e contém a lista de IDs de casos de teste específicos.

`hasExecutionErrors`

Não sai quando a máquina de estado é iniciada. Se algum estado encontrar erros de execução, essa variável será criada e definida como`true`Para a duração restante da execução da máquina de estado.

Você pode consultar o contexto usando a notação JSONPath. A sintaxe para consultas JSONPath em definições de estado é`{{$.query}}`. Você pode usar consultas JSONPath como cadeias de espaço reservado em alguns estados. O IDT substitui as strings de espaço reservado pelo valor da consulta JSONPath avaliada a partir do contexto. É possível usar espaços reservados para os seguintes valores:

- O`TestCases`valor em`RunTask`Estados.
- O`Expression`valor`Choice`Estado.

Quando você acessar dados do contexto da máquina de estado, verifique se as seguintes condições são atendidas:

- Seus caminhos JSON devem começar com`$.`

- Cada valor deve ser avaliado para uma string, um número ou um booleano.

Para obter mais informações sobre como usar a notação JSONPath para acessar dados do contexto, consulteUsar o contexto IDT.

Erros de execução

Erros de execução são erros na definição da máquina de estado que a máquina de estado encontra ao executar um estado. O IDT registra informações sobre cada erro no`test_manager.log`Arquivo e transmite a mensagem de log para o console.

É possível usar os seguintes métodos para lidar com erros de execução:

- Adicionar um`Catch`blocona definição de estado.
- Verifique o valor do`hasExecutionErrors`valorNo contexto da máquina de estado.

Captura

Para usar`Catch`, adicione o seguinte à definição de estado:

```
"Catch": [
    {
        "ErrorEquals": [
            "<error-type>"
        ]
        "Next": "<state-name>"
    }
]
```

Todos os campos que contêm valores são necessários, conforme descrito aqui:

`Catch.ErrorEquals`

Uma matriz dos tipos de erro a serem capturados. Se um erro de execução corresponder a um dos valores especificados, a máquina de estado muda para o estado especificado no`Catch.Next`. Consulte cada definição de estado para obter informações sobre o tipo de erro que ele produz.

`Catch.Next`

O próximo estado para a transição para se o estado atual encontrar um erro de execução que corresponda a um dos valores especificados em`Catch.ErrorEquals`.

Blocos de captura são manipulados sequencialmente até que um corresponda. Se os erros não corresponderem aos listados nos blocos Catch, as máquinas de estado continuarão sendo executadas. Como os erros de execução são resultado de definições de estado incorretas, recomendamos que você faça a transição para o estado Falha quando um estado encontrar um erro de execução.

HasExecutionError

Quando alguns estados encontram erros de execução, além de emitir o erro, eles também definem o`hasExecutionError`valor para`true`no contexto da máquina de estado. Você pode usar esse valor para detectar quando ocorre um erro e, em seguida, usar um`Choice`state para fazer a transição da máquina de estado para o`Fail`Estado.

Esse método tem as seguintes características.

- A máquina de estado não inicia com nenhum valor atribuído a`hasExecutionError`, e esse valor não estará disponível até que um determinado estado o defina. Isso significa que você deve definir explicitamente o`Fallthrough0nError`para`false`para a`Choice`afirma que acessam esse valor para impedir que a máquina de estado pare se não ocorrerem erros de execução.
- Uma vez definido como`true`,`hasExecutionError`nunca é definido como falso ou removido do contexto. Isso significa que esse valor é útil somente na primeira vez que é definido como`true`, e para todos os estados subsequentes, ele não fornece um valor significativo.
- O`hasExecutionError`valor é compartilhado com todas as máquinas de estado de ramificação na`Parallel`state, que pode resultar em resultados inesperados dependendo da ordem em que ele é acessado.

Devido a essas características, não recomendamos que você use esse método se puder usar um bloco Catch.

Exemplo de máquinas de estado

Esta seção fornece algumas configurações de máquina de estado de exemplo.

Exemplos

- Exemplo de máquina de estado do: Execute um único grupo de teste
- Exemplo de máquina de estado do: Execute grupos de teste selecionados pelo usuário
- Exemplo de máquina de estado do: Execute um único grupo de teste com recursos do produto
- Exemplo de máquina de estado do: Execute dois grupos de teste em parallel

Exemplo de máquina de estado do: Execute um único grupo de teste

Esta máquina de estado do:

- Executa o grupo de teste com id`GroupA`, que deve estar presente na suíte em um`group.json`file.
- Verifica se há erros de execução e transições para`Fail`se algum for encontrado.
- Gera um relatório e faz a transição para`Succeed`se não houver erros, e`Fail`Caso contrário, .

```
{
    "Comment": "Runs a single group and then generates a report.",
    "StartAt": "RunGroupA",
    "States": {
        "RunGroupA": {
            "Type": "RunTask",
            "Next": "Report",
            "TestGroup": "GroupA",
            "Catch": [
                {
                    "ErrorEquals": [
                        "RunTaskError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "Report": {
            "Type": "Report",
            "Next": "Succeed",
            "Catch": [
                {
                    "ErrorEquals": [
                        "ReportError"
                    ],
                    "Next": "Fail"
                }
```

```
            ]
        },
        "Succeed": {
            "Type": "Succeed"
        },
        "Fail": {
            "Type": "Fail"
        }
    }
}
```

Exemplo de máquina de estado do: Execute grupos de teste selecionados pelo usuário

Esta máquina de estado do:

- Verifica se o executor de teste selecionou grupos de teste específicos. A máquina de estado não verifica casos de teste específicos porque os corredores de teste não podem selecionar casos de teste sem também selecionar um grupo de teste.
- Se os grupos de testes forem selecionados:
  - Executa os casos de teste dentro dos grupos de teste selecionados. Para fazer isso, a máquina de estado não especifica explicitamente nenhum grupo de teste ou casos de teste no`RunTask`Estado.
  - Gera um relatório depois de executar todos os testes e saídas.
- Se os grupos de teste não forem selecionados:
  - Executa testes no grupo de teste`GroupA`.
  - Gera relatórios e saídas.

```
{
    "Comment": "Runs specific groups if the test runner chose to do that, otherwise
 runs GroupA.",
    "StartAt": "SpecificGroupsCheck",
    "States": {
        "SpecificGroupsCheck": {
            "Type": "Choice",
            "Default": "RunGroupA",
            "FallthroughOnError": true,
            "Choices": [
                {
                    "Expression": "{{$.specificTestGroups[0]}} != ''",
```

```
                "Next": "RunSpecificGroups"
            }
        ]
    },
    "RunSpecificGroups": {
        "Type": "RunTask",
        "Next": "Report",
        "Catch": [
            {
                "ErrorEquals": [
                    "RunTaskError"
                ],
                "Next": "Fail"
            }
        ]
    },
    "RunGroupA": {
        "Type": "RunTask",
        "Next": "Report",
        "TestGroup": "GroupA",
        "Catch": [
            {
                "ErrorEquals": [
                    "RunTaskError"
                ],
                "Next": "Fail"
            }
        ]
    },
    "Report": {
        "Type": "Report",
        "Next": "Succeed",
        "Catch": [
            {
                "ErrorEquals": [
                    "ReportError"
                ],
                "Next": "Fail"
            }
        ]
    },
    "Succeed": {
        "Type": "Succeed"
    },
```

```
        "Fail": {
            "Type": "Fail"
        }
    }
}
```

Exemplo de máquina de estado do: Execute um único grupo de teste com recursos do produto

Esta máquina de estado do:

- Executa o grupo de testesGroupA.
- Verifica se há erros de execução e transições paraFailse algum for encontrado.
- Adiciona oFeatureThatDependsOnGroupArecurso para oawsiotdevicetester_report.xmlfile:
  - SeGroupApassa, o recurso está definido comosupported.
  - O recurso não está marcado como opcional no relatório.
- Gera um relatório e faz a transição paraSucceedse não houver erros, eFailde outra forma

```
{
    "Comment": "Runs GroupA and adds product features based on GroupA",
    "StartAt": "RunGroupA",
    "States": {
        "RunGroupA": {
            "Type": "RunTask",
            "Next": "AddProductFeatures",
            "TestGroup": "GroupA",
            "ResultVar": "GroupA_passed",
            "Catch": [
                {
                    "ErrorEquals": [
                        "RunTaskError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "AddProductFeatures": {
            "Type": "AddProductFeatures",
            "Next": "Report",
            "Features": [
```

```
                {
                    "Feature": "FeatureThatDependsOnGroupA",
                    "Groups": [
                        "GroupA"
                    ],
                    "IsRequired": true
                }
            ]
        },
        "Report": {
            "Type": "Report",
            "Next": "Succeed",
            "Catch": [
                {
                    "ErrorEquals": [
                        "ReportError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "Succeed": {
            "Type": "Succeed"
        },
        "Fail": {
            "Type": "Fail"
        }
    }
}
```

Exemplo de máquina de estado do: Execute dois grupos de teste em parallel

Esta máquina de estado do:

- Executa o`GroupA`e`GroupB`grupos de teste em parallel. O`ResultVar`variáveis armazenadas no contexto pelo`RunTask`estados nas máquinas de estado de ramificação por estão disponíveis para o`AddProductFeatures`Estado.
- Verifica se há erros de execução e transições para`Fail`se algum for encontrado. Esta máquina de estado não usa um`Catch`bloco porque esse método não detecta erros de execução em máquinas de estado de ramificação.
- Adiciona recursos ao`awsiotdevicetester_report.xml`arquivo com base nos grupos que passam

- SeGroupApassa, o recurso está definido comosupported.
- O recurso não está marcado como opcional no relatório.

- Gera um relatório e faz a transição paraSucceedse não houver erros, eFailde outra forma

Se dois dispositivos estiverem configurados no pool de dispositivos, ambosGroupAeGroupBPode ser executado ao mesmo tempo. No entanto, se um dos doisGroupAouGroupBtem vários testes nele, então ambos os dispositivos podem ser alocados para esses testes. Se apenas um dispositivo estiver configurado, os grupos de teste serão executados sequencialmente.

```
{
    "Comment": "Runs GroupA and GroupB in parallel",
    "StartAt": "RunGroupAAndB",
    "States": {
        "RunGroupAAndB": {
            "Type": "Parallel",
            "Next": "CheckForErrors",
            "Branches": [
                {
                    "Comment": "Run GroupA state machine",
                    "StartAt": "RunGroupA",
                    "States": {
                        "RunGroupA": {
                            "Type": "RunTask",
                            "Next": "Succeed",
                            "TestGroup": "GroupA",
                            "ResultVar": "GroupA_passed",
                            "Catch": [
                                {
                                    "ErrorEquals": [
                                        "RunTaskError"
                                    ],
                                    "Next": "Fail"
                                }
                            ]
                        },
                        "Succeed": {
                            "Type": "Succeed"
                        },
                        "Fail": {
                            "Type": "Fail"
                        }
```

```
                }
            },
            {
                "Comment": "Run GroupB state machine",
                "StartAt": "RunGroupB",
                "States": {
                    "RunGroupA": {
                        "Type": "RunTask",
                        "Next": "Succeed",
                        "TestGroup": "GroupB",
                        "ResultVar": "GroupB_passed",
                        "Catch": [
                            {
                                "ErrorEquals": [
                                    "RunTaskError"
                                ],
                                "Next": "Fail"
                            }
                        ]
                    },
                    "Succeed": {
                        "Type": "Succeed"
                    },
                    "Fail": {
                        "Type": "Fail"
                    }
                }
            }
        ]
    },
    "CheckForErrors": {
        "Type": "Choice",
        "Default": "AddProductFeatures",
        "FallthroughOnError": true,
        "Choices": [
            {
                "Expression": "{{$.hasExecutionErrors}} == true",
                "Next": "Fail"
            }
        ]
    },
    "AddProductFeatures": {
        "Type": "AddProductFeatures",
        "Next": "Report",
```

```
            "Features": [
                {
                    "Feature": "FeatureThatDependsOnGroupA",
                    "Groups": [
                        "GroupA"
                    ],
                    "IsRequired": true
                },
                {
                    "Feature": "FeatureThatDependsOnGroupB",
                    "Groups": [
                        "GroupB"
                    ],
                    "IsRequired": true
                }
            ]
        },
        "Report": {
            "Type": "Report",
            "Next": "Succeed",
            "Catch": [
                {
                    "ErrorEquals": [
                        "ReportError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "Succeed": {
            "Type": "Succeed"
        },
        "Fail": {
            "Type": "Fail"
        }
    }
}
```

Crie executáveis de casos de teste do IDT

Você pode criar e colocar executáveis de casos de teste em uma pasta de suíte de testes das seguintes formas:

- Para suítes de teste que usam argumentos ou variáveis de ambiente dos`test.json` arquivos para determinar quais testes devem ser executados, você pode criar um único caso de teste executável para toda a suíte de testes ou um executável de teste para cada grupo de testes na suíte de testes.
- Para uma suíte de testes em que você deseja executar testes específicos com base em comandos especificados, você cria um caso de teste executável para cada caso de teste na suíte de testes.

Como redator de testes, você pode determinar qual abordagem é apropriada para seu caso de uso e estruturar seu executável de caso de teste de acordo. Certifique-se de fornecer o caminho executável correto do caso de teste em cada`test.json` arquivo e de que o executável especificado seja executado corretamente.

Quando todos os dispositivos estiverem prontos para a execução de um caso de teste, o IDT lê os seguintes arquivos:

- O`test.json` para o caso de teste selecionado determina os processos a serem iniciados e as variáveis de ambiente a serem definidas.
- O`suite.json` para o conjunto de testes determina as variáveis de ambiente a serem definidas.

O IDT inicia o processo executável de teste necessário com base nos comandos e argumentos especificados no`test.json` arquivo e passa as variáveis de ambiente necessárias para o processo.

Use o SDK do cliente IDT

Os SDKs do cliente IDT permitem que você simplifique a forma como você escreve a lógica de teste em seu executável de teste com comandos de API que você pode usar para interagir com o IDT e seus dispositivos em teste. Atualmente, o IDT fornece os seguintes SDKs:

- SDK do cliente IDT SDK para Python thon thon SDK do cliente
- SDK do cliente IDT SDK for Go SDK do IDT
- SDK do cliente IDT SDK for Java Java SDT SDK

Esses SDKs estão localizados na*<device-tester-extract-location>*`/sdks` pasta. Ao criar um novo executável de caso de teste, você deve copiar o SDK que deseja usar para a pasta que contém o executável do caso de teste e referenciar o SDK em seu código. Esta seção fornece uma breve descrição dos comandos de API disponíveis que você pode usar nos executáveis do seu caso de teste.

Nesta seção

Interação entre dispositivos:

Os comandos a seguir permitem que você se comunique com o dispositivo em teste sem precisar implementar nenhuma função adicional de gerenciamento de conectividade e interação com o dispositivo.

`ExecuteOnDevice`

Permite que suítes de teste executem comandos shell em um dispositivo compatível com conexões SSH ou Docker shell.

`CopyToDevice`

Permite que as suítes de teste copiem um arquivo local da máquina host que executa o IDT para um local especificado em um dispositivo que suporta conexões SSH ou Docker shell.

`ReadFromDevice`

Permite que as suítes de teste leiam a partir da porta serial de dispositivos que suportam conexões UART.

> **Note**
>
> Como o IDT não gerencia conexões diretas com dispositivos que são feitas usando informações de acesso a dispositivos do contexto, recomendamos usar esses comandos da API de interação de dispositivos em seus executáveis de caso de teste. No entanto, se esses comandos não atenderem aos requisitos do seu caso de teste, você poderá recuperar as informações de acesso ao dispositivo do contexto do IDT e usá-las para fazer uma conexão direta com o dispositivo a partir da suíte de testes.
> Para fazer uma conexão direta, recupere as informações nos`resource.devices.connectivity` campos`device.connectivity` e do dispositivo em teste e dos dispositivos de recursos, respectivamente. Para obter mais informações sobre como usar o contexto do IDT, consulteUsar o contexto IDT.

Interação com o IDT,

Os comandos a seguir permitem que suas suítes de teste se comuniquem com o IDT.

`PollForNotifications`

Permite que as suítes de teste verifiquem as notificações do IDT.

`GetContextValue` e `GetContextString`

Permite que os conjuntos de testes recuperem valores do contexto do IDT. Para obter mais informações, consulte Usar o contexto IDT.

`SendResult`

Permite que as suítes de teste relatem os resultados dos casos de teste ao IDT. Esse comando deve ser chamado no final de cada caso de teste em uma suíte de testes.

Interação do anfitrião

O comando a seguir permite que suas suítes de teste se comuniquem com a máquina host.

`PollForNotifications`

Permite que as suítes de teste verifiquem as notificações do IDT.

`GetContextValue` e `GetContextString`

Permite que os conjuntos de testes recuperem valores do contexto do IDT. Para obter mais informações, consulte Usar o contexto IDT.

`ExecuteOnHost`

Permite que as suítes de teste executem comandos na máquina local e permite que o IDT gerencie o ciclo de vida executável do caso de teste.

Habilitar comandos da IDT CLI CLI CLI

O`run-suite` comando IDT CLI fornece várias opções que permitem que o executor de testes personalize a execução do teste. Para permitir que os executores de teste usem essas opções para executar sua suíte de testes personalizada, você implementa o suporte para o IDT CLI. Se você não implementar o suporte, os executores de teste ainda poderão executar testes, mas algumas opções da CLI não funcionarão corretamente. Para fornecer uma experiência ideal ao cliente,

recomendamos que você implemente o suporte para os seguintes argumentos para `orun-suite` comando na CLI do IDT:

`timeout-multiplier`

Especifica um valor maior que 1,0 que será aplicado a todos os tempos limite durante a execução dos testes.

Os executores de teste podem usar esse argumento para aumentar o tempo limite dos casos de teste que desejam executar. Quando um executor de teste especifica esse argumento em seu`run-suite` comando, o IDT o usa para calcular o valor da variável de ambiente IDT_TEST_TIMEOUT e define `oconfig.timeoutMultiplier` campo no contexto do IDT. Para apoiar esse argumento, você deve fazer o seguinte:

- Em vez de usar diretamente o valor de tempo limite do`test.json` arquivo, leia a variável de ambiente IDT_TEST_TIMEOUT para obter o valor de tempo limite calculado corretamente.
- Recupere `oconfig.timeoutMultiplier` valor do contexto do IDT e aplique-o a tempos limite de execução prolongados.

Para obter mais informações sobre como sair mais cedo devido a eventos de tempo limite, consulteEspecifique o comportamento de saída.

`stop-on-first-failure`

Especifica que o IDT deve parar de executar todos os testes se encontrar uma falha.

Quando um executor de teste especifica esse argumento em seu`run-suite` comando, o IDT interrompe a execução dos testes assim que encontrar uma falha. No entanto, se os casos de teste estiverem sendo executados em parallel, isso poderá levar a resultados inesperados. Para implementar o suporte, certifique-se de que, se o IDT encontrar esse evento, sua lógica de teste instrua todos os casos de teste em execução a parar, limpar recursos temporários e relatar o resultado do teste ao IDT. Para obter mais informações sobre como sair em caso de falhas, consulteEspecifique o comportamento de saída.

`group-id` e `test-id`

Especifica que o IDT deve executar somente os grupos de teste ou casos de teste selecionados.

Os executores de teste podem usar esses argumentos com seus`run-suite` comandos para especificar o seguinte comportamento de execução do teste:

- Execute todos os testes dentro dos grupos de teste especificados.
- Execute uma seleção de testes de dentro de um grupo de teste especificado.

Para apoiar esses argumentos, o orquestrador de teste do seu conjunto de testes deve incluir um conjuntoRunTask eChoice estados específicos em seu orquestrador de teste. Se você não estiver usando uma máquina de estado personalizada, o orquestrador de teste padrão do IDT inclui os estados necessários para você e você não precisa realizar nenhuma ação adicional. No entanto, se você estiver usando um orquestrador de teste personalizado, useExemplo de máquina de estado do: Execute grupos de teste selecionados pelo usuário como exemplo para adicionar os estados necessários em seu orquestrador de teste.

Para obter mais informações sobre comandos da IDT CLI, consulteDepure e execute conjuntos de teste personalizados.

Escrever registros de eventos

Enquanto o teste está sendo executado, você envia dados para`stdout` `stderr` para gravar registros de eventos e mensagens de erro no console. Para obter mais informações sobre o formato das mensagens do console, consulteFormato de mensagens do console.

Quando o IDT terminar de executar a suíte de testes, essas informações também estarão disponíveis no`test_manager.log` arquivo localizado na*<devicetester-extract-location>*/`results/`*<execution-id>*`/logs` pasta.

Você pode configurar cada caso de teste para gravar os registros de sua execução de teste, incluindo registros do dispositivo em teste, no*<group-id>*`_`*<test-id>* arquivo localizado na*<device-tester-extract-location>*`/results/`*execution-id*`/logs` pasta. Para fazer isso, recupere o caminho para o arquivo de log do contexto do IDT com a`testData.logFilePath` consulta, crie um arquivo nesse caminho e grave o conteúdo desejado nele. O IDT atualiza automaticamente o caminho com base no caso de teste que está sendo executado. Se você optar por não criar o arquivo de log para um caso de teste, nenhum arquivo será gerado para esse caso de teste.

Você também pode configurar seu executável de texto para criar arquivos de log adicionais, conforme necessário, na*<device-tester-extract-location>*`/logs` pasta. Recomendamos que você especifique prefixos exclusivos para nomes de arquivos de log para que seus arquivos não sejam substituídos.

Reportar resultados ao IDT

O IDT grava os resultados do teste nos*suite-name*`_report.xml` arquivos`awsiotdevicetester_report.xml` e. Esses arquivos do relatório estão localizados

em*<device-tester-extract-location>*/results/*<execution-id>*/. Ambos os relatórios capturam os resultados da execução da suíte de testes. Para obter mais informações sobre os esquemas que o IDT usa para esses relatórios, consulteRevise os resultados e os registros do teste IDT

Para preencher o conteúdo do*suite-name*_report.xml arquivo, você deve usar o`SendResult` comando para relatar os resultados do teste ao IDT antes que a execução do teste termine. Se o IDT não conseguir localizar os resultados de um teste, ele emitirá um erro para o caso de teste. O seguinte trecho do Python mostra os comandos para enviar um resultado de teste para o IDT:

```
request-variable = SendResultRequest(TestResult(result))
client.send_result(request-variable)
```

Se você não relatar os resultados por meio da API, o IDT procura os resultados do teste na pasta de artefatos de teste. O caminho para essa pasta é armazenado no`testData.testArtifactsPath` campo no contexto do IDT. Nessa pasta, o IDT usa o primeiro arquivo XML classificado em ordem alfabética que ele localiza como resultado do teste.

Se sua lógica de teste produzir resultados XML do JUnit, você poderá gravar os resultados do teste em um arquivo XML na pasta de artefatos para fornecer os resultados diretamente ao IDT, em vez de analisar os resultados e usar a API para enviá-los ao IDT.

Se você usar esse método, certifique-se de que sua lógica de teste resuma com precisão os resultados do teste e formate seu arquivo de resultados no mesmo formato do*suite-name*_report.xml arquivo. O IDT não realiza nenhuma validação dos dados que você fornece, com as seguintes exceções:

- O IDT ignora todas as propriedades da`testsuites` tag. Em vez disso, ele calcula as propriedades da tag a partir dos resultados de outros grupos de teste relatados.
- Pelo menos uma`testsuite` tag deve existir dentro de`testsuites`.

Como o IDT usa a mesma pasta de artefatos para todos os casos de teste e não exclui arquivos de resultados entre as execuções de teste, esse método também pode levar a relatórios errôneos se o IDT ler o arquivo incorreto. Recomendamos que você use o mesmo nome para o arquivo de resultados XML gerado em todos os casos de teste para sobrescrever os resultados de cada caso de teste e garantir que os resultados corretos estejam disponíveis para uso do IDT. Embora você possa usar uma abordagem mista para gerar relatórios em sua suíte de testes, ou seja, usar um arquivo

de resultados XML para alguns casos de teste e enviar resultados por meio da API para outros, não recomendamos essa abordagem.

Especifique o comportamento de saída

Configure seu executável de texto para sempre sair com um código de saída de 0, mesmo que um caso de teste reporte uma falha ou um resultado de erro. Use códigos de saída diferentes de zero somente para indicar que um caso de teste não foi executado ou se o executável do caso de teste não conseguiu comunicar nenhum resultado ao IDT. Quando o IDT recebe um código de saída diferente de zero, ele marca que o caso de teste encontrou um erro que o impediu de ser executado.

O IDT pode solicitar ou esperar que um caso de teste pare de ser executado antes de ser concluído nos seguintes eventos. Use essas informações para configurar o executável do caso de teste para detectar cada um desses eventos do caso de teste:

Timeout (Tempo limite)

Ocorre quando um caso de teste é executado por mais tempo do que o valor de tempo limite especificado no`test.json` arquivo. Se o executor do teste usou o`timeout-multiplier` argumento para especificar um multiplicador de tempo limite, o IDT calculará o valor do tempo limite com o multiplicador.

Para detectar esse evento, use a variável de ambiente IDT_TEST_TIMEOUT. Quando um executor de teste inicia um teste, o IDT define o valor da variável de ambiente IDT_TEST_TIMEOUT como o valor de tempo limite calculado (em segundos) e passa a variável para o executável do caso de teste. Você pode ler o valor da variável para definir um cronômetro apropriado.

Interromper

Ocorre quando o executor de teste interrompe o IDT. Por exemplo, pressionandoCtrl+C.

Como os terminais propagam sinais para todos os processos secundários, você pode simplesmente configurar um manipulador de sinal em seus casos de teste para detectar sinais de interrupção.

Como alternativa, você pode pesquisar periodicamente a API para verificar o valor do`CancellationRequested` booleano na resposta da`PollForNotifications` API. Quando o IDT recebe um sinal de interrupção, ele define o valor do`CancellationRequested` booleano como`true`.

Pare na primeira falha

Ocorre quando um caso de teste que está sendo executado em parallel com o caso de teste atual falha e o executor do teste usa o`stop-on-first-failure` argumento para especificar que o IDT deve parar quando encontrar alguma falha.

Para detectar esse evento, você pode pesquisar periodicamente a API para verificar o valor do`CancellationRequested` booleano na resposta da`PollForNotifications` API. Quando o IDT encontra uma falha e é configurado para parar na primeira falha, ele define o valor do`CancellationRequested` booleano como`true`.

Quando qualquer um desses eventos ocorre, o IDT espera 5 minutos para que qualquer caso de teste atualmente em execução termine de ser executado. Se todos os casos de teste em execução não saírem em 5 minutos, o IDT forçará a interrupção de cada um de seus processos. Se o IDT não tiver recebido os resultados do teste antes do término dos processos, ele marcará o tempo limite dos casos de teste. Como melhor prática, você deve garantir que seus casos de teste realizem as seguintes ações quando encontrarem um dos eventos:

1. Pare de executar a lógica de teste normal.
2. Limpe todos os recursos temporários, como artefatos de teste no dispositivo em teste.
3. Relate o resultado de um teste ao IDT, como uma falha no teste ou um erro.
4. Saída.

Usar o contexto IDT

Quando o IDT executa um conjunto de testes, o conjunto de testes pode acessar um conjunto de dados que pode ser usado para determinar como cada teste é executado. Esses dados são chamados de contexto IDT. Por exemplo, configuração de dados do usuário fornecida por corredores de teste em um`userdata.json`arquivo é disponibilizado para suítes de teste no contexto IDT.

O contexto IDT pode ser considerado um documento JSON somente leitura. Os conjuntos de testes podem recuperar dados e gravar dados no contexto usando tipos de dados JSON padrão, como objetos, matrizes, números e assim por diante.

Esquema de contexto

O contexto IDT usa o formato a seguir:

```
{
    "config": {
        <config-json-content>
        "timeoutMultiplier": timeout-multiplier
    },
    "device": {
        <device-json-device-element>
    },
    "devicePool": {
        <device-json-pool-element>
    },
    "resource": {
        "devices": [
            {
                <resource-json-device-element>
                "name": "<resource-name>"
            }
        ]
    },
    "testData": {
        "awsCredentials": {
            "awsAccessKeyId": "<access-key-id>",
            "awsSecretAccessKey": "<secret-access-key>",
            "awsSessionToken": "<session-token>"
        },
        "logFilePath": "/path/to/log/file"
    },
    "userData": {
        <userdata-json-content>
    }
}
```

`config`

Informações da`config.json`arquivo. O`config`também contém o seguinte campo adicional:

`config.timeoutMultiplier`

O multiplicador para o valor de tempo limite qualquer usado pelo conjunto de testes. Esse valor é especificado pelo executor de teste da IDT CLI. O valor padrão é 1.

`device`

Informações sobre o dispositivo selecionado para a execução de teste. Essas informações são equivalentes ao`devices`elemento de matriz no`device.json`arquivopara o dispositivo selecionado.

`devicePool`

Informações sobre o pool de dispositivos selecionado para a execução de teste. Essas informações são equivalentes ao elemento de matriz de pool de dispositivos de nível superior definido na`device.json`arquivo para o pool de dispositivos selecionado.

`resource`

Informações sobre dispositivos de recursos do`resource.json`file.

`resource.devices`

Essas informações são equivalentes ao`devices`array definido na`resource.json`file. EACH`devices`elemento inclui o seguinte campo adicional:

`resource.device.name`

O nome do dispositivo de recursos. Esse valor é configurado como o`requiredResource.name`valor na`test.json`file.

`testData.awsCredentials`

OAWScredenciais usadas pelo teste para se conectar aoAWSnuvem. Essas informações são obtidas a partir do`config.json`file.

`testData.logFilePath`

O caminho para o arquivo de log no qual o caso de teste grava mensagens de log. Se ele não existir, o conjunto de testes cria esse arquivo.

`userData`

Informações fornecidas pelo executor de teste no`userdata.json`arquivo.

Acesse dados no contexto

Você pode consultar o contexto usando a notação JSONPath de seus arquivos JSON e do executável de texto com o`GetContextValue`e`GetContextString`APIs. A sintaxe das strings JSONPath para acessar o contexto IDT varia da seguinte forma:

- Dentro `suite.jsone` `test.json`, você usa `{{query}}`. Ou seja, não use o elemento raiz `$.` para iniciar sua expressão.
- Dentro `test_orchestrator.yaml`, você usa `{{query}}`.

  Se você usar a máquina de estado obsoleta, então em `state_machine.json`, você usa `{{$.query}}`.
- Nos comandos da API, você usa `query` ou `{{$.query}}`, dependendo do comando. Para obter mais informações, consulte a documentação em linha nos SDKs.

A tabela a seguir descreve os operadores em uma expressão JSONPath típica:

Operator	Description
`$`	The root element. Because the top-level context value for IDT is an object, you will typically use `$.` to start your queries.
`.ChildName`	Accesses the child element with name `ChildName` from an object. If applied to an array, yields a new array with this operator applied to each element. The element name is case sensitive. For example, the query to access the `awsRegion` value in the `config` object is `$.config.awsRegion` .
`[start:end]`	Filters elements from an array, retrieving items beginning from the `start` index and going up to the `fim` index, both inclusive.
`[index1, index2,..., IndexN]`	Filters elements from an array, retrieving items from only the specified indices.
`[? (expr)]`	Filters elements from an array using the `expr` expression. This expression must evaluate to a boolean value.

Para criar expressões de filtro, use a seguinte sintaxe:

```
<jsonpath> | <value> operator <jsonpath> | <value>
```

Nesta sintaxe:

- `jsonpath`é um JSONPath que usa sintaxe JSON padrão.
- `value`é qualquer valor personalizado que usa sintaxe JSON padrão.
- `operator`é um dos seguintes operadores:
  - <(Menor que)
  - <=(Menor ou igual a)
  - ==(Igual a)

    Se o JSONPath ou valor em sua expressão for um valor de matriz, booleano ou objeto, esse será o único operador binário suportado que você pode usar.
  - >=(Maior ou igual a)
  - >(Maior que)
  - =~(Correspondência de expressões regulares). Para usar esse operador em uma expressão de filtro, o JSONPath ou valor no lado esquerdo da expressão deve ser avaliado para uma string e o lado direito deve ser um valor padrão que segue oSintaxe RE2.

Você pode usar consultas JSONPath no formulário {{*interrogação*}} como cadeias de espaço reservado dentro do`args`e`environmentVariables`campos em`test.json`arquivos e dentro do`environmentVariables`campos em`suite.json`arquivos. O IDT executa uma pesquisa de contexto e preenche os campos com o valor avaliado da consulta. Por exemplo, na`suite.json`, você pode usar strings de espaço reservado para especificar valores de variáveis de ambiente que mudam com cada caso de teste e o IDT preencherá as variáveis de ambiente com o valor correto para cada caso de teste. No entanto, quando você usa cadeias de espaço reservado no`test.json`e`suite.json`arquivos, as seguintes considerações se aplicam às suas consultas:

- Você deve cada ocorrência do`devicePool`chave em sua consulta em todas as minúsculas. Ou seja, use`devicepool`Em vez disso.
- Para matrizes, você pode usar somente matrizes de strings. Além disso, os arrays usam um não-padrão`item1, item2,...,itemN`format. Se a matriz contiver apenas um elemento, ela será serializada como`item`, tornando-o indistinguível de um campo de string.
- Você não pode usar espaços reservados para recuperar objetos do contexto.

Devido a essas considerações, recomendamos que, sempre que possível, você use a API para acessar o contexto em sua lógica de teste em vez de cadeias de espaço reservado `notest.jsonesuite.json`arquivos. No entanto, em alguns casos, pode ser mais conveniente usar espaços reservados JSONPath para recuperar cadeias de caracteres individuais para definir como variáveis de ambiente.

Definir configurações para executores de teste

Para executar pacotes de testes personalizados, os executores de teste devem definir suas configurações com base no pacote de teste que desejam executar. As configurações são especificadas com base nos modelos de arquivo de configuração localizados na pasta *<device-tester-extract-location>*`/configs/`. Se necessário, os executores de teste também devem configurar as credenciais da AWS que a IDT usará para se conectar à nuvem da AWS.

Como autor de testes, você precisará configurar esses arquivos para depurar o conjunto de testes. Você deve fornecer instruções aos executores de teste para que eles possam definir as seguintes configurações conforme necessário para executar os conjuntos de testes.

Configurar device.json

O arquivo `device.json` contém informações sobre os dispositivos nos quais os testes são executados (por exemplo, endereço IP, informações de login, sistema operacional e arquitetura de CPU).

Os executores de teste podem fornecer essas informações usando o seguinte arquivo `device.json` de modelo localizado na pasta *<device-tester-extract-location>*`/configs/`.

```
[
    {
        "id": "<pool-id>",
        "sku": "<pool-sku>",
        "features": [
            {
                "name": "<feature-name>",
                "value": "<feature-value>",
                "configs": [
                    {
                        "name": "<config-name>",
                        "value": "<config-value>"
                    }
```

```
                ],
            }
        ],
        "devices": [
            {
                "id": "<device-id>",
                "connectivity": {
                    "protocol": "ssh | uart | docker",
                    // ssh
                    "ip": "<ip-address>",
                    "port": <port-number>,
                    "auth": {
                        "method": "pki | password",
                        "credentials": {
                            "user": "<user-name>",
                            // pki
                            "privKeyPath": "/path/to/private/key",

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

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

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

Todos os campos que contêm valores são necessários, conforme descrito aqui:

`id`

Um ID alfanumérico definido pelo usuário que identifica uma coleção de dispositivos chamada de grupo de dispositivos. Os dispositivos que pertencem a um grupo devem ter hardware idêntico. Quando um conjunto de testes é executado, os dispositivos do grupo são usados para paralelizar a carga de trabalho. Vários dispositivos são usados para executar testes diferentes.

`sku`

Um valor alfanumérico que identifica exclusivamente o dispositivo em teste. A SKU é usada para rastrear os dispositivos qualificados.

> **Note**
>
> Se você deseja listar sua placa no AWS Partner Device Catalog, a SKU especificada aqui deve corresponder à SKU que você usa no processo de oferta.

`features`

Opcional. Uma matriz que contém atributos compatíveis com o dispositivo. Os atributos do dispositivo são valores definidos pelo usuário que você configura no conjunto de testes. Você deve fornecer aos executores de teste informações sobre os nomes e valores dos atributos a serem incluídos no arquivo `device.json`. Por exemplo, se desejar testar um dispositivo que funciona como um servidor MQTT em outros dispositivos, você poderá configurar a lógica de teste para validar níveis compatíveis específicos de um atributo chamado `MQTT_QOS`. Os executores de teste fornecem esse nome de atributo e definem o valor do atributo para os níveis de QOS compatíveis com o dispositivo deles. Você pode recuperar as informações fornecidas do contexto do IDT com a `devicePool.features` consulta ou do contexto do orquestrador de teste com a consulta. `pool.features`

`features.name`

O nome do atributo.

`features.value`

Os valores dos atributos compatíveis.

`features.configs`

Definições de configuração, se necessárias, para o atributo.

`features.config.name`

O nome da definição de configuração.

`features.config.value`

Os valores de configuração compatível.

`devices`

Uma matriz de dispositivos no grupo a ser testado. Pelo menos um dispositivo é necessário.

`devices.id`

Um identificador exclusivo, definido pelo usuário, para o dispositivo que está sendo testado.

`connectivity.protocol`

O protocolo de comunicação usado para se comunicar com esse dispositivo. Cada dispositivo em um grupo deve usar o mesmo protocolo.

No momento, os únicos valores compatíveis são `ssh` e `uart` para dispositivos físicos e `docker` para contêineres do Docker.

`connectivity.ip`

O endereço IP do dispositivo que está sendo testado.

Essa propriedade será aplicada somente se `connectivity.protocol` estiver definido como `ssh`.

`connectivity.port`

Opcional. O número da porta a ser usado para conexões SSH.

O valor padrão é 22.

Essa propriedade será aplicada somente se `connectivity.protocol` estiver definido como `ssh`.

`connectivity.auth`

Informações de autenticação da conexão.

Essa propriedade será aplicada somente se `connectivity.protocol` estiver definido como `ssh`.

`connectivity.auth.method`

O método de autenticação usado para acessar um dispositivo pelo protocolo de conectividade indicado.

Os valores compatíveis são:

- `pki`
- `password`

`connectivity.auth.credentials`

As credenciais usadas para autenticação.

`connectivity.auth.credentials.password`

A senha usada para fazer login no dispositivo que está sendo testado.

Esse valor se aplica somente se `connectivity.auth.method` estiver definido como `password`.

`connectivity.auth.credentials.privKeyPath`

O caminho completo para a chave privada usada para fazer login no dispositivo que está sendo testado.

Esse valor se aplica somente se `connectivity.auth.method` estiver definido como `pki`.

`connectivity.auth.credentials.user`

O nome de usuário para fazer login no dispositivo que está sendo testado.

`connectivity.serialPort`

Opcional. A porta serial à qual o dispositivo está conectado.

Essa propriedade será aplicada somente se `connectivity.protocol` estiver definido como `uart`.

`connectivity.containerId`

O ID do contêiner ou o nome do contêiner do Docker que está sendo testado.

Essa propriedade será aplicada somente se `connectivity.protocol` estiver definido como `ssh`.

`connectivity.containerUser`

Opcional. O nome de usuário para o usuário dentro do contêiner. O valor padrão é o usuário fornecido no Dockerfile.

O valor padrão é 22.

Essa propriedade será aplicada somente se `connectivity.protocol` estiver definido como `ssh`.

> **Note**
>
> Para verificar se os executores de teste configuram a conexão incorreta do dispositivo para um teste, você pode recuperá-la do contexto `pool.Devices[0].Connectivity.Protocol` do orquestrador de testes e compará-la com o valor esperado em um estado. `Choice` Se um protocolo incorreto for usado, exiba uma mensagem usando o estado`LogMessage` e faça a transição para o estado `Fail`. Opcionalmente, você pode usar o código de tratamento de erros para relatar uma falha no teste de tipos de dispositivos incorretos.

(Opcional) Configuração de userdata.json

O arquivo `userdata.json` contém toda as informações adicionais exigidas por um conjunto de testes, mas não estão especificadas no arquivo `device.json`. O formato desse arquivo depende do arquivo `userdata_scheme.json` definido no conjunto de testes. Se você é um autor de testes, forneça essas informações aos usuários que executarão os conjuntos de testes que você escreveu.

(Opcional) Configurar resource.json

O arquivo `resource.json` contém informações sobre todos os dispositivos que serão usados como dispositivos de recursos. Os dispositivos de recursos são necessários para testar determinados recursos de um dispositivo em teste. Por exemplo, para testar a capacidade Bluetooth de um dispositivo, você pode usar um dispositivo de recurso para testar se seu dispositivo consegue se conectar com êxito. Os dispositivos de recursos são opcionais e você pode exigir quantos dispositivos de recursos forem necessários. Como autor de testes, você usa o arquivo test.json para definir os recursos do dispositivo de recursos necessários para um teste. Em seguida, os executores de teste usam o arquivo `resource.json` para fornecer um grupo de dispositivos de recursos com os recursos necessários. Forneça essas informações aos usuários que executarão os conjuntos de teste que você escreve.

Os executores de teste podem fornecer essas informações usando o seguinte arquivo `resource.json` de modelo localizado na pasta *`<device-tester-extract-location>`*`/configs/`.

```
[
    {
        "id": "<pool-id>",
        "features": [
            {
                "name": "<feature-name>",
                "version": "<feature-version>",
                "jobSlots": <job-slots>
            }
        ],
        "devices": [
            {
                "id": "<device-id>",
                "connectivity": {
                    "protocol": "ssh | uart | docker",
                    // ssh
                    "ip": "<ip-address>",
                    "port": <port-number>,
                    "auth": {
                        "method": "pki | password",
                        "credentials": {
                            "user": "<user-name>",
                            // pki
                            "privKeyPath": "/path/to/private/key",

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

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

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

Todos os campos que contêm valores são necessários, conforme descrito aqui:

`id`

Um ID alfanumérico definido pelo usuário que identifica uma coleção de dispositivos chamada de grupo de dispositivos. Os dispositivos que pertencem a um grupo devem ter hardware idêntico. Quando um conjunto de testes é executado, os dispositivos do grupo são usados para paralelizar a carga de trabalho. Vários dispositivos são usados para executar testes diferentes.

`features`

Opcional. Uma matriz que contém atributos compatíveis com o dispositivo. As informações necessárias nesse campo são definidas nos arquivos test.json no conjunto de testes e determinam quais testes devem ser executados e como executá-los. Se o conjunto de testes não exigir nenhum atributo, esse campo não será obrigatório.

`features.name`

O nome do atributo.

`features.version`

A versão do atributo.

`features.jobSlots`

Configuração para indicar quantos testes podem usar o dispositivo simultaneamente. O valor padrão é 1.

`devices`

Uma matriz de dispositivos no grupo a ser testado. Pelo menos um dispositivo é necessário.

`devices.id`

Um identificador exclusivo, definido pelo usuário, para o dispositivo que está sendo testado.

`connectivity.protocol`

O protocolo de comunicação usado para se comunicar com esse dispositivo. Cada dispositivo em um grupo deve usar o mesmo protocolo.

No momento, os únicos valores compatíveis são `ssh` e `uart` para dispositivos físicos e `docker` para contêineres do Docker.

`connectivity.ip`

O endereço IP do dispositivo que está sendo testado.

Essa propriedade será aplicada somente se `connectivity.protocol` estiver definido como `ssh`.

`connectivity.port`

Opcional. O número da porta a ser usado para conexões SSH.

O valor padrão é 22.

Essa propriedade será aplicada somente se `connectivity.protocol` estiver definido como `ssh`.

`connectivity.auth`

Informações de autenticação da conexão.

Essa propriedade será aplicada somente se `connectivity.protocol` estiver definido como `ssh`.

`connectivity.auth.method`

O método de autenticação usado para acessar um dispositivo pelo protocolo de conectividade indicado.

Os valores compatíveis são:

- `pki`
- `password`

`connectivity.auth.credentials`

As credenciais usadas para autenticação.

`connectivity.auth.credentials.password`

A senha usada para fazer login no dispositivo que está sendo testado.

Esse valor se aplica somente se `connectivity.auth.method` estiver definido como `password`.

`connectivity.auth.credentials.privKeyPath`

O caminho completo para a chave privada usada para fazer login no dispositivo que está sendo testado.

Esse valor se aplica somente se `connectivity.auth.method` estiver definido como `pki`.

`connectivity.auth.credentials.user`

O nome de usuário para fazer login no dispositivo que está sendo testado.

`connectivity.serialPort`

Opcional. A porta serial à qual o dispositivo está conectado.

Essa propriedade será aplicada somente se `connectivity.protocol` estiver definido como `uart`.

`connectivity.containerId`

O ID do contêiner ou o nome do contêiner do Docker que está sendo testado.

Essa propriedade será aplicada somente se `connectivity.protocol` estiver definido como `ssh`.

`connectivity.containerUser`

Opcional. O nome de usuário para o usuário dentro do contêiner. O valor padrão é o usuário fornecido no Dockerfile.

O valor padrão é 22.

Essa propriedade será aplicada somente se `connectivity.protocol` estiver definido como `ssh`.

(Opcional) Configuração de config.json

O arquivo `config.json` contém as informações de configuração do IDT. Normalmente, os executores de teste não precisarão modificar esse arquivo, exceto para fornecer as credenciais de usuário da AWS para o IDT e, opcionalmente, para uma região AWS. Se as credenciais da AWS com as permissões necessárias forem fornecidas, o AWS IoT Device Tester coletará e enviará as métricas de uso para a AWS. Esse é um atributo opcional usado para melhorar a funcionalidade do IDT. Para ter mais informações, consulte Métricas de uso do IDT.

Os executores de teste podem configurar as credenciais da AWS de uma das seguintes maneiras:

- Arquivo de credenciais

O IDT usa o mesmo arquivo de credenciais que a AWS CLI. Para obter mais informações, consulte Arquivos de configuração e credenciais.

O local do arquivo de credenciais varia de acordo com o sistema operacional que você está usando:

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

- Variáveis de ambiente

  As variáveis de ambiente são variáveis mantidas pelo sistema operacional e usadas pelos comandos do sistema. As variáveis definidas durante uma sessão SSH não ficam disponíveis após o encerramento da sessão. O IDT pode usar as variáveis de ambiente `AWS_ACCESS_KEY_ID` e `AWS_SECRET_ACCESS_KEY` para armazenar as credenciais da AWS

  Para definir essas variáveis no Linux, macOS ou Unix, use :export

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

  Para definir essas variáveis no Windows, use :set

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

Para configurar as credenciais da AWS para o IDT, os executores de teste editam a seção `auth` no arquivo `config.json` localizado na pasta *<device-tester-extract-location>*`/configs/`.

```
{
    "log": {
        "location": "logs"
    },
    "configFiles": {
        "root": "configs",
        "device": "configs/device.json"
    },
    "testPath": "tests",
    "reportPath": "results",
    "awsRegion": "<region>",
```

```
    "auth": {
        "method": "file | environment",
        "credentials": {
            "profile": "<profile-name>"
        }
    }
}
]
```

Todos os campos que contêm valores são necessários, conforme descrito aqui:

> **Note**
>
> Todos os caminhos nesse arquivo são definidos em relação ao *< device-tester-extract-location >*.

`log.location`

O caminho para a pasta de registros em *< device-tester-extract-location >*.

`configFiles.root`

O caminho para o arquivo que contém os arquivos de configuração.

`configFiles.device`

O caminho para o arquivo `device.json`.

`testPath`

O caminho para a pasta que contém os conjuntos de teste.

`reportPath`

O caminho para a pasta que conterá os resultados do teste depois que o IDT executar um conjunto de testes.

`awsRegion`

Opcional. A região da AWS que será usada pelos conjuntos de teste. Se não for definida, os conjuntos de teste usarão a região padrão especificada em cada conjunto de testes.

`auth.method`

O método que o IDT usa para recuperar as credenciais da AWS. Os valores compatíveis são `file` para recuperar credenciais de um arquivo de credenciais e `environment` para recuperar credenciais usando variáveis de ambiente.

`auth.credentials.profile`

O perfil de credenciais a ser usado do arquivo de credenciais. Essa propriedade será aplicada somente se `auth.method` estiver definido como `file`.

Depure e execute conjuntos de teste personalizados

Depois que a configuração necessária for definida, o IDT poderá executar seu pacote de teste. O runtime do pacote de testes completo depende do hardware e da composição do pacote de testes. Como referência, leva aproximadamente 30 minutos para concluir o pacote de qualificação completo AWS IoT Greengrass em um Raspberry Pi 3B.

Ao escrever seu pacote de testes, você pode usar o IDT para executar a conjunto de testes no modo de depuração para verificar seu código antes de executá-lo ou fornecê-lo aos executores de teste.

Execução do IDT no modo de depuração

Como os conjuntos de teste dependem do IDT para interagir com dispositivos, fornecer o contexto e receber resultados, você não pode simplesmente depurar os conjuntos de teste em um IDE sem interagir com o IDT. Para fazer isso, a CLI do IDT fornece o comando `debug-test-suite` que permite executar o IDT no modo de depuração. Execute o seguinte comando para visualizar as opções disponíveis para `debug-test-suite`:

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

Ao executar o IDT no modo de depuração, o IDT na verdade não inicia o pacote de teste nem executa o orquestrador de testes. Em vez disso, ele interage com seu IDE para responder às solicitações feitas do pacote de teste em execução no IDE e imprime os logs no console. O IDT não atinge o tempo limite e espera para sair até ser interrompido manualmente. No modo de depuração, o IDT também não executa o orquestrador de testes e não gera nenhum arquivo de relatório. Para depurar o conjunto de testes, você deve usar seu IDE para fornecer algumas informações que o IDT normalmente obtém dos arquivos JSON de configuração. Forneça as seguintes informações:

- Variáveis de ambiente e argumentos para cada teste. A IDT não lerá essas informações de `test.json` ou `suite.json`.
- Argumentos para selecionar dispositivos de recursos. A IDT não lerá essas informações de `test.json`.

Para depurar os conjuntos de teste, conclua as seguintes etapas:

1. Crie os arquivos de definição de configuração necessários para executar o conjunto de testes. Por exemplo, se o conjunto de testes necessita do `device.json`, `resource.json` e `user data.json`, configure todos eles conforme necessário.
2. Execute o comando a seguir para colocar o IDT no modo de depuração e selecionar todos os dispositivos necessários para executar o teste.

   ```
   devicetester_[linux | mac | win_x86-64] debug-test-suite [options]
   ```

   Depois de executar esse comando, o IDT aguarda as solicitações do conjunto de testes e depois as responde. O IDT também gera as variáveis de ambiente necessárias para o processo de caso do IDT Client SDK.
3. Em seu IDE, use a configuração `run` ou `debug` para fazer o seguinte:

   a. Definir os valores das variáveis de ambiente geradas pelo IDT.

   b. Definir o valor das variáveis de ambiente ou argumentos que você especificou no arquivo `test.json` e `suite.json`.

   c. Definir pontos de interrupção conforme necessário.
4. Executar o conjunto de testes no IDE.

   Você pode depurar e executar novamente o conjunto de testes, sempre que for necessário. O IDT não atinge o tempo limite no modo de depuração.
5. Depois de concluir a depuração, interrompa o IDT para sair do modo de depuração.

Comandos da CLI do IDT para executar testes

A seção a seguir descreve os comandos da CLI do IDT:

IDT v4.0.0

`help`

Lista as informações sobre o comando especificado.

`list-groups`

Lista os grupos em um determinado conjunto de teste.

`list-suites`

Lista os conjuntos de teste disponíveis.

`list-supported-products`

Lista os produtos compatíveis com a sua versão, neste caso, versões do AWS IoT Greengrass e versões do pacote de testes de qualificação AWS IoT Greengrass disponíveis para a versão atual do IDT.

`list-test-cases`

Lista os casos de teste em um grupo de teste. A seguinte opção é compatível:

- `group-id`. O grupo de teste a ser pesquisado. Esta opção é necessária e deve especificar um único grupo.

`run-suite`

Executa um conjunto de testes em um grupo de dispositivos. Confira a seguir algumas opções geralmente usadas:

- `suite-id`. A versão do conjunto de testes a ser executada. Se não for especificado, o IDT usará a versão mais recente na pasta `tests`.
- `group-id`. Os grupos de teste a serem executados, em uma lista separada por vírgulas. Se não for especificado, o IDT executa todos os grupos de teste no conjunto de testes.
- `test-id`. Os casos de teste a serem executados, em uma lista separada por vírgulas. Quando especificado, `group-id` deve especificar um único grupo.
- `pool-id`. O grupo de dispositivos a ser testado. Os executores de testes devem especificar um grupo se tiverem vários grupos de dispositivos definidos no arquivo `device.json`.
- `timeout-multiplier`. Configura o IDT para modificar o tempo limite de execução do teste especificado no arquivo `test.json` para um teste com um multiplicador definido pelo usuário.

- `stop-on-first-failure`. Configura o IDT para interromper a execução na primeira falha. Essa opção deve ser usada com para depurar os grupos de teste especificados `group-id`.
- `userdata`. Define o arquivo que contém as informações de dados do usuário necessárias para executar o conjunto de testes. Isso é necessário somente se `userdataRequired` estiver definido como verdadeiro no arquivo `suite.json` do conjunto de testes.

Para obter mais informações sobre as opções `run-suite`, use a opção `help`:

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

`debug-test-suite`

Execute o conjunto de testes no modo de depuração. Para ter mais informações, consulte Execução do IDT no modo de depuração.

Revise os resultados e os registros do teste IDT

Esta seção descreve o formato no qual o IDT gera logs de console e relatórios de teste.

Formato de mensagens do console

AWS IoT Device Testerusa um formato padrão para imprimir mensagens no console quando ele inicia um conjunto de testes. O trecho a seguir mostra um exemplo de uma mensagem de console gerada pelo IDT.

```
time="2000-01-02T03:04:05-07:00" level=info msg=Using suite: MyTestSuite_1.0.0
executionId=9a52f362-1227-11eb-86c9-8c8590419f30
```

A maioria das mensagens do console são compostas de:

`time`

Um carimbo de data/hora ISO 8601 completo para o evento registrado.

`level`

O nível da mensagem para o evento registrado. Normalmente, o nível da mensagem registrada é um dos`info`,`warn`, ou`error`. O IDT emite um`fatal`ou`panic`mensagem se encontrar um evento esperado que faça com que ele saia mais cedo.

`msg`

A mensagem registrada.

`executionId`

Uma string de ID exclusiva para o processo IDT atual. Esse ID é usado para diferenciar entre execuções IDT individuais.

As mensagens do console geradas a partir de um conjunto de testes fornecem informações adicionais sobre o dispositivo em teste e o conjunto de testes, o grupo de testes e os casos de teste executados pelo IDT. O trecho a seguir mostra um exemplo de uma mensagem de console gerada a partir de um conjunto de testes.

```
time="2000-01-02T03:04:05-07:00" level=info msg=Hello world! suiteId=MyTestSuite
groupId=myTestGroup testCaseId=myTestCase deviceId=my-device
executionId=9a52f362-1227-11eb-86c9-8c8590419f30
```

A parte específica do conjunto de testes da mensagem do console contém os seguintes campos:

`suiteId`

O nome do conjunto de testes em execução no momento.

`groupId`

O ID do grupo de teste em execução no momento.

`testCaseId`

A ID do caso de teste em execução atual.

`deviceId`

Um ID do dispositivo em teste que o caso de teste atual está usando.

Para imprimir um resumo de teste no console quando um IDT terminar de executar um teste, você deve incluir um`Reportestado`em seu orquestrador de testes. O resumo do teste contém informações sobre o conjunto de testes, os resultados do teste para cada grupo que foi executado e os locais dos registros gerados e arquivos de relatório. O exemplo a seguir mostra uma mensagem de resumo de teste.

```
========== Test Summary ==========
Execution Time:     5m00s
Tests Completed:    4
Tests Passed:       3
Tests Failed:       1
Tests Skipped:      0
---------------------------------
Test Groups:
    GroupA:         PASSED
    GroupB:         FAILED
---------------------------------
Failed Tests:
    Group Name: GroupB
        Test Name: TestB1
            Reason: Something bad happened
---------------------------------
Path to IoT Device Tester Report: /path/to/awsiotdevicetester_report.xml
Path to Test Execution Logs: /path/to/logs
Path to Aggregated JUnit Report: /path/to/MyTestSuite_Report.xml
```

AWS IoT Device TesterEsquema de relatório

`awsiotdevicetester_report.xml`É um relatório assinado que contém as seguintes informações:

- A versão IDT.
- A versão do conjunto de testes.
- A assinatura do relatório e a chave usadas para assinar o relatório.
- A SKU do dispositivo e o nome do conjunto de dispositivos especificados no`device.json`file.
- A versão do produto e os recursos do dispositivo que foram testados.
- O resumo agregado dos resultados de teste. Essas informações são as mesmas que as contidas no*`suite-name`*`_report.xml`file.

```
<apnreport>
    <awsiotdevicetesterversion>idt-version</awsiotdevicetesterversion>
    <testsuiteversion>test-suite-version</testsuiteversion>
    <signature>signature</signature>
    <keyname>keyname</keyname>
    <session>
```

```
        <testsession>execution-id</testsession>
        <starttime>start-time</starttime>
        <endtime>end-time</endtime>
    </session>
    <awsproduct>
        <name>product-name</name>
        <version>product-version</version>
        <features>
            <feature name="<feature-name>" value="supported | not-supported | <feature-
value>" type="optional | required"/>
        </features>
    </awsproduct>
    <device>
        <sku>device-sku</sku>
        <name>device-name</name>
        <features>
            <feature name="<feature-name>" value="<feature-value>"/>
        </features>
        <executionMethod>ssh | uart | docker</executionMethod>
    </device>
    <devenvironment>
        <os name="<os-name>"/>
    </devenvironment>
    <report>
        <suite-name-report-contents>
    </report>
</apnreport>
```

O arquivo `awsiotdevicetester_report.xml` contém uma tag `<awsproduct>` com informações sobre o produto que está sendo testado e os recursos do produto que foram validados após a execução de um pacote de testes.

Atributos usados na tag **`<awsproduct>`**

`name`

O nome do produto testado.

`version`

A versão do produto testado.

`features`

Os recursos validados. Recursos marcados como`required`são necessários para que o conjunto de testes valide o dispositivo. O snippet a seguir mostra como essas informações aparecem no arquivo `awsiotdevicetester_report.xml`.

```
<feature name="ssh" value="supported" type="required"></feature>
```

Recursos marcados como`optional`não são necessários para validação. Os seguintes trechos mostram recursos opcionais.

```
<feature name="hsi" value="supported" type="optional"></feature>

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

Esquema de relatório do conjunto de testes

O relatório *`suite-name`*`_Result.xml` está no formato JUnit XML. Você pode integrá-lo em plataformas de integração e implantação como Jenkins, Bamboo, e assim por diante. O relatório contém um resumo agregado dos resultados de testes.

```
<testsuites name="<suite-name> results" time="<run-duration>" tests="<number-of-test>"
 failures="<number-of-tests>" skipped="<number-of-tests>" errors="<number-of-tests>"
 disabled="0">
    <testsuite name="<test-group-id>" package="" tests="<number-of-tests>"
 failures="<number-of-tests>" skipped="<number-of-tests>" errors="<number-of-tests>"
 disabled="0">
        <!--success-->
        <testcase classname="<classname>" name="<name>" time="<run-duration>"/>
        <!--failure-->
        <testcase classname="<classname>" name="<name>" time="<run-duration>">
            <failure type="<failure-type>">
                reason
            </failure>
        </testcase>
        <!--skipped-->
        <testcase classname="<classname>" name="<name>" time="<run-duration>">
            <skipped>
                reason
            </skipped>
```

```
        </testcase>
        <!--error-->
        <testcase classname="<classname>" name="<name>" time="<run-duration>">
            <error>
                reason
            </error>
        </testcase>
    </testsuite>
</testsuites>
```

A seção de relatório em ambos osawsiotdevicetester_report.xmlou*suite-name*_report.xmlLista os testes que foram executados e os resultados.

A primeira tag XML `<testsuites>` contém o resumo da execução do teste. Por exemplo:

```
<testsuites name="MyTestSuite results" time="2299" tests="28" failures="0" errors="0"
 disabled="0">
```

Atributos usados na tag **`<testsuites>`**

`name`

O nome do conjunto de testes.

`time`

O tempo, em segundos, necessário para executar o conjunto de testes.

`tests`

O número de testes executados.

`failures`

O número de testes que foram executados, mas não foram aprovados.

`errors`

O número de testes que não puderam ser executados pelo IDT.

`disabled`

Esse atributo não é usado e pode ser ignorado.

Se houver falhas de teste ou erros, você poderá identificar o teste com falha analisando as tags XML `<testsuites>`. As tags XML `<testsuite>` dentro da tag `<testsuites>` mostram o resumo do resultado do teste para um grupo de testes. Por exemplo:

```
<testsuite name="combination" package="" tests="1" failures="0" time="161" disabled="0"
 errors="0" skipped="0">
```

O formato é semelhante à tag `<testsuites>`, mas com um atributo `skipped` que não é usado e pode ser ignorado. Dentro de cada tag XML `<testsuite>`, há tags `<testcase>` para cada teste executado para um grupo de testes. Por exemplo:

```
<testcase classname="Security Test" name="IP Change Tests" attempts="1"></testcase>>
```

Atributos usados na tag **`<testcase>`**

`name`

O nome do teste.

`attempts`

O número de vezes que o IDT executou o caso de teste.

Quando um teste falha ou ocorre um erro, as tags `<failure>` ou `<error>` são adicionadas à tag `<testcase>` com informações para a solução de problemas. Por exemplo:

```
<testcase classname="mcu.Full_MQTT" name="MQTT_TestCase" attempts="1">
 <failure type="Failure">Reason for the test failure</failure>
 <error>Reason for the test execution error</error>
</testcase>
```

Métricas de uso do IDT

Se você fornecer AWS credenciais com as permissões necessárias, AWS IoT Device Tester coletará e enviará métricas de uso para. AWS Este é um recurso opcional e é usado para melhorar a funcionalidade do IDT. O IDT coleta informações como as seguintes:

- O Conta da AWS ID usado para executar o IDT
- Os AWS CLI comandos IDT usados para executar testes

- As suítes de teste que são executadas
- As suítes de teste na pasta *< device-tester-extract-location >*
- O número de dispositivos configurados no grupo de dispositivos
- Nomes de casos de teste e tempos de execução
- Informações do resultado do teste, como se os testes foram aprovados, falharam, encontraram erros ou foram ignorados
- recursos testados do produto
- Comportamento de saída do IDT, como saídas inesperadas ou antecipadas

Todas as informações enviadas pelo IDT também são registradas em um arquivo `metrics.log` na pasta *<device-tester-extract-location>*`/results/`*<execution-id>*`/`. Você pode visualizar o arquivo de log para ver as informações que foram coletadas durante a execução de um teste. Este arquivo é gerado somente se optar por coletar métricas de uso.

Para desativar a coleta de métricas, não é necessário tomar nenhuma outra medida. Simplesmente não armazene suas AWS credenciais e, se você tiver AWS credenciais armazenadas, não configure o `config.json` arquivo para acessá-las.

Configure suas AWS credenciais

Se você ainda não tem um Conta da AWS, você deve criar um. Se você já tem uma Conta da AWS, basta configurar as permissões necessárias para sua conta, permitindo que a IDT envie métricas de uso AWS em seu nome.

Etapa 1: criar um Conta da AWS

Nesta etapa, crie e configure uma Conta da AWS. Se você já tem uma Conta da AWS, pule para the section called “Etapa 2: Configurar permissões para o IDT”.

Se você não tiver um Conta da AWS, conclua as etapas a seguir para criar um.

Para se inscrever em um Conta da AWS

1. Abra https://portal.aws.amazon.com/billing/signup.
2. Siga as instruções on-line.

   Parte do procedimento de inscrição envolve receber uma chamada telefônica e digitar um código de verificação no teclado do telefone.

Quando você se inscreve em um Conta da AWS, um Usuário raiz da conta da AWSé criado. O usuário-raiz tem acesso a todos os Serviços da AWS e recursos na conta. Como uma prática recomendada de segurança, atribua o acesso administrativo para um usuário e use somente o usuário-raiz para executar tarefas que requerem o acesso de usuário-raiz.

Para criar um usuário administrador, selecione uma das opções a seguir.

Selecionar uma forma de gerenciar o administrador	Para	Por	Você também pode
Centro de Identidade do IAM (Recomendado)	Use credenciais de curto prazo para acessar a AWS. Isso está de acordo com as práticas recomendadas de segurança. Para obter informações sobre as práticas recomendadas, consulte Práticas recomendadas de segurança no IAM no Guia do usuário do IAM.	Seguindo as instruções em Conceitos básicos no Guia do usuário do AWS IAM Identity Center .	Configure o acesso programático configurando o AWS CLI para uso AWS IAM Identity Center no Guia do AWS Command Line Interface usuário.
No IAM (Não recomendado)	Use credenciais de curto prazo para acessar a AWS.	Seguindo as instruções em Criar o seu primeiro usuário administrador e um	Para configurar o acesso programático, consulte Gerenciamento de chaves de

Selecionar uma forma de gerenciar o administrador	Para	Por	Você também pode
		grupo de usuários do IAM no Guia do usuário do IAM.	acesso de usuários do IAM no Guia do usuário do IAM.

Etapa 2: Configurar permissões para o IDT

Nesta etapa, configure as permissões que o IDT para executar testes e coletar dados de uso do IDT. Você pode usar o AWS Management Console or AWS Command Line Interface (AWS CLI) para criar uma política do IAM e um usuário para o IDT e, em seguida, anexar políticas ao usuário.

- Para configurar permissões para IDT (Console)
- Para configurar permissões para o IDT (AWS CLI)

Como configurar permissões para o IDT (console)

Siga estas etapas para usar o console para configurar permissões para IDT para AWS IoT Greengrass.

1. Faça login no console do IAM.
2. Crie uma política gerenciada pelo cliente que conceda permissões para criar funções com permissões específicas.

   a. No painel de navegação, selecione Políticas e, em seguida, Criar política.

   b. Na guia JSON, substitua o conteúdo do espaço reservado pela política a seguir.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
```

```
            "Action": [
                "iot-device-tester:SendMetrics"
            ],
            "Resource": "*"
        }
    ]
}
```

c. Escolha Review policy (Revisar política).

d. Em Name (Nome), insira **`IDTUsageMetricsIAMPermissions`**. Em Summary (Resumo), revise as permissões concedidas pela política.

e. Escolha Create policy (Criar política).

3. Crie um usuário do IAM e anexe permissões ao usuário.

a. Criar um usuário do IAM. Siga as etapas de 1 a 5 em Criando usuários do IAM (console) no Guia do usuário do IAM. Se você já tiver criado um usuário do IAM, vá para a próxima etapa.

b. Anexe as permissões ao usuário do IAM:

i. Na página Set permissions (Definir permissões), selecione Attach existing policies to user directly (Anexar políticas existentes diretamente ao usuário).

ii. Pesquise a política IDT UsageMetrics IAMPermissions que você criou na etapa anterior. Marque a caixa de seleção.

c. Selecione Next: Tags (Próximo: tags).

d. Selecione Next: Review (Próximo: revisar) para exibir um resumo das suas escolhas.

e. Selecione Criar usuário.

f. Para exibir as chaves de acesso do usuário (IDs de chave de acesso e chaves de acesso secretas), selecione Show (Mostrar) ao lado da senha e da chave de acesso. Para salvar as chaves de acesso, selecione Download.csv (Fazer download do .csv) e salve o arquivo em um local seguro. Você usa essas informações posteriormente para configurar seu arquivo de AWS credenciais.

Como configurar permissões para o IDT (AWS CLI)

Siga estas etapas para usar o AWS CLI para configurar as permissões do IDT para AWS IoT Greengrass.

1. No seu computador, instale e configure o, AWS CLI se ainda não estiver instalado. Siga as etapas em Installing the (Instalando a) AWS CLI no Guia do usuário do AWS Command Line Interface .

   > **Note**
   >
   > AWS CLI É uma ferramenta de código aberto que você pode usar para interagir com AWS serviços do seu shell de linha de comando.

2. Crie a seguinte política gerenciada pelo cliente que concede permissões para gerenciar IDT e AWS IoT Greengrass funções.

   Linux or Unix

   ```
   aws iam create-policy --policy-name IDTUsageMetricsIAMPermissions --policy-
   document '{
       "Version": "2012-10-17",
       "Statement": [
           {
               "Effect": "Allow",
               "Action": [
                   "iot-device-tester:SendMetrics"
               ],
               "Resource": "*"
           }
       ]
   }'
   ```

   Windows command prompt

   ```
   aws iam create-policy --policy-name IDTUsageMetricsIAMPermissions --policy-
   document
                                           '{\"Version\": \"2012-10-17\",
    \"Statement\": [{\"Effect\": \"Allow\", \"Action\": [\"iot-device-
   tester:SendMetrics\"], \"Resource\": \"*\"}]}'
   ```

> **Note**
>
> Esta etapa inclui um exemplo de prompt de comando do Windows porque ele usa uma sintaxe JSON diferente dos comandos de terminal Linux, macOS ou Unix.

PowerShell

```
aws iam create-policy --policy-name IDTUsageMetricsIAMPermissions --policy-
document '{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot-device-tester:SendMetrics"
            ],
            "Resource": "*"
        }
    ]
}'
```

3. Crie um usuário do IAM e anexe as permissões exigidas pelo IDT para o AWS IoT Greengrass.

   a. Criar um usuário do IAM.

   ```
   aws iam create-user --user-name user-name
   ```

   b. Anexe a política de `IDTUsageMetricsIAMPermissions` criada para o novo usuário do IAM. Substitua *`user-name`* pelo seu nome de usuário do IAM e *`<account-id>`* no comando pelo ID da sua Conta da AWS.

   ```
   aws iam attach-user-policy --user-name user-name --policy-arn
    arn:aws:iam::<account-id>:policy/IDTGreengrassIAMPermissions
   ```

4. Crie uma chave de acesso secreta para o usuário.

   ```
   aws iam create-access-key --user-name user-name
   ```

Armazene a saída em um local seguro. Você usa essas informações posteriormente para configurar seu arquivo de AWS credenciais.

Forneça AWS credenciais ao IDT

Para permitir que o IDT acesse suas AWS credenciais e envie métricas para AWS, faça o seguinte:

1. Armazene as AWS credenciais do seu usuário do IAM como variáveis de ambiente ou em um arquivo de credenciais:

   a. Para usar variáveis de ambiente, execute os comandos a seguir.

      Linux or Unix

      ```
      export AWS_ACCESS_KEY_ID=access-key
      export AWS_SECRET_ACCESS_KEY=secret-access-key
      ```

      Windows Command Prompt (CMD)

      ```
      set AWS_ACCESS_KEY_ID=access-key
      set AWS_SECRET_ACCESS_KEY=secret-access-key
      ```

      PowerShell

      ```
      $env:AWS_ACCESS_KEY_ID="access-key"
      $env:AWS_SECRET_ACCESS_KEY="secret-access-key"
      ```

   b. Para usar o arquivo de credenciais, adicione as seguintes informações ao `~/.aws/credentials` arquivo.

      ```
      [profile-name]
      aws_access_key_id=access-key
      aws_secret_access_key=secret-access-key
      ```

2. Configure a seção `auth` do arquivo `config.json`. Para ter mais informações, consulte (Opcional) Configuração de config.json.

Solução de problemas de IDT paraAWS IoT GreengrassV2

IDT paraAWS IoT GreengrassA V2 grava erros em vários locais com base no tipo de erro. O IDT grava erros no console, nos arquivos de log e nos relatórios de teste.

Onde procurar erros

Erros de alto nível são exibidos no console enquanto o teste está sendo executado, e um resumo dos testes que falharam é exibido quando todos os testes são concluídos.`awsiotdevicetester_report.xml`contém um resumo de todos os erros que causaram a falha de um teste. O IDT armazena os arquivos de log de cada execução de teste em um diretório com um UUID para a execução do teste, exibido no console durante a execução do teste.

O diretório de registros de teste do IDT é*<device-tester-extract-location>*/`results/`*<execution-id>*`/logs/`. Esse diretório contém os seguintes arquivos exibidos na tabela. Isso é útil para depuração.

Arquivo	Descrição
`test_manager.log`	Os registros gravados no console durante a execução do teste. O resumo dos resultados no final desse arquivo inclui uma lista de quais testes falharam. Os logs de aviso e de erro nesse arquivo podem fornecer algumas informações sobre as falhas.
test-group-id /*test-case-id* /*test-name* .log	Registros detalhados do teste específic o em um grupo de teste. Para testes que implantam componentes do Greengrass, o arquivo de registro do caso de teste é chamado`greengrass-test-run.log` .
test-group-id /*test-case-id* / `greengrass.log`	Registros detalhados paraAWS IoT Greengras sSoftware principal. O IDT copia esse arquivo do dispositivo em teste quando executa testes

Arquivo	Descrição
	que instalamAWS IoT GreengrassSoftware principal no dispositivo. Para obter mais informações sobre as mensagens nesse arquivo de log, consulteSolução de problemas AWS IoT Greengrass V2.
test-group-id /*test-case-id*/*component-name* .log	Registros detalhados dos componentes do Greengrass que são implantados durante as execuções de teste. O IDT copia os arquivos de log de componentes do dispositivo em teste quando executa testes que implantam componentes específicos. O nome do arquivo de log de cada componente corresponde ao nome do componente implantado. Para obter mais informações sobre as mensagens nesse arquivo de log, consulteSolução de problemas AWS IoT Greengrass V2.

Resolvendo o IDT paraAWS IoT GreengrassErros V2

Antes de executar o IDT paraAWS IoT Greengrass, instale os arquivos de configuração corretos. Se você receber erros de análise e configuração, a primeira etapa é localizar e usar um modelo de configuração apropriado para seu ambiente.

Se você ainda estiver com problemas, consulte o processo de depuração a seguir.

Tópicos

- Erros de qualificação de aprendizado de máquina
- Implantações com falha no Open Test Framework (OTF)
- Erros de análise
- Erros de permissão negada
- Erro na geração do relatório de qualificação
- Erro de parâmetro necessário ausente
- Exceção de segurança no macOS
- Erros de conexão SSH
- Erros de qualificação do gerenciador de stream
- Erros de tempo limite
- Erros de verificação de versão

Erros de resolução de aliases

Ao executar suítes de testes personalizadas, você pode ver o seguinte erro no console e `notest_manager.log`.

```
Couldn't resolve placeholders: couldn't do a json lookup: index out of range
```

Esse erro pode ocorrer quando os aliases configurados no orquestrador de testes do IDT não são resolvidos corretamente ou se os valores resolvidos não estão presentes nos arquivos de configuração. Para resolver esse erro, certifique-se de que seu`device.json`ou`userdata.json` contém as informações corretas necessárias para sua suíte de testes. Para obter informações sobre a configuração necessária paraAWS IoT Greengrassqualificação, consulteDefina as configurações de IDT para executar o pacote de AWS IoT Greengrass qualificação.

Erros de conflito

Você pode ver o seguinte erro ao executar oAWS IoT Greengrassconjunto de qualificação simultaneamente em mais de um dispositivo.

```
ConflictException: Component [com.example.IDTHelloWorld : 1.0.0] for account [account-
id] already exists with state: [DEPLOYABLE] { RespMetadata: { StatusCode: 409,
 RequestID: “id” }, Message_: “Component [com.example.IDTHelloWorld : 1.0.0] for
 account [account-id] already exists with state: [DEPLOYABLE]” }
```

A execução simultânea de testes ainda não é suportada peloAWS IoT Greengrasssuíte de qualificação. Execute o conjunto de qualificação sequencialmente para cada dispositivo.

Erro: Não foi possível iniciar teste

Você pode encontrar erros que apontam para falhas que ocorreram quando o teste estava tentando iniciar. Há várias causas possíveis e, portanto, faça o seguinte:

- Certifique-se de que o nome do pool em seu comando de execução realmente exista. O IDT faz referência ao nome do pool diretamente de seu`device.json`arquivo.
- Verifique se os dispositivos no grupo têm os parâmetros de configuração corretos.

Existem erros na imagem de qualificação do Docker

Os testes de qualificação do gerenciador de aplicativos Docker usam o`amazon/amazon-ec2-metadata-mock`imagem de contêiner no Amazon ECR para qualificar o dispositivo em teste.

Você pode receber o seguinte erro se a imagem já estiver presente em um contêiner do Docker no dispositivo em teste.

```
The Docker image amazon/amazon-ec2-metadata-mock:version already exists on the device.
```

Se você baixou essa imagem anteriormente e executou o`amazon/amazon-ec2-metadata-mock`contêiner em seu dispositivo, certifique-se de remover essa imagem do dispositivo em teste antes de executar os testes de qualificação.

Falha ao ler a credencial

Ao testar dispositivos Windows, você pode encontrar o`Failed to read credential`erro no`greengrass.log`arquivo se o usuário que você usa para se conectar ao dispositivo em teste não estiver configurado no gerenciador de credenciais desse dispositivo.

Para resolver esse erro, configure o usuário e a senha do usuário do IDT no gerenciador de credenciais do dispositivo em teste.

Para obter mais informações, consulte Configurar credenciais de usuário para dispositivos Windows.

Erros de guia com PreInstalled Capim verde

Ao executar o IDT com PreInstalled Greengrass, se você encontrar um erro de`GuiceouErrorInCustomProvider`, verifique se o arquivo`userdata.json`tem o`InstalledDirRootOnDevice`defina para a pasta de instalação do Greengrass. O IDT verifica o arquivo`effectiveConfig.yaml`embaixo`<InstallationDirRootOnDevice>/config/effectiveConfig.yaml`.

Para obter mais informações, consulte Configurar credenciais de usuário para dispositivos Windows.

Exceção de assinatura inválida

Ao executar testes de qualificação do Lambda, você pode encontrar o`invalidsignatureexception`erro se sua máquina host IDT tiver problemas de acesso à rede. Reinicie o roteador e execute os testes novamente.

Erros de qualificação de aprendizado de máquina

Ao executar testes de qualificação de aprendizado de máquina (ML), você pode encontrar falhas de qualificação se o dispositivo não atender àsrequisitospara implantar oAWS-componentes de ML fornecidos. Para solucionar erros de qualificação de ML, faça o seguinte:

- Procure detalhes de erro nos registros dos componentes que foram implantados durante a execução do teste. Os registros de componentes estão localizados no*<device-tester-extract-location>*`/results/`*<execution-id>*`/logs/`*<test-group-id>*diretório.
- Adicione o`-Dgg.persist=installed.software`argumento para o`test.json`arquivo para o caso de teste com falha. O`test.json`o arquivo está localizado no*<device-tester-extract-location>*`/tests/GGV2Q_`*version* `directory.`

Implantações com falha no Open Test Framework (OTF)

Se os testes OTF falharem em concluir a implantação, uma causa provável pode ser as permissões definidas para a pasta principal do`TempResourcesDirOnDevice`e`InstallationDirRootOnDevice`. Para definir as permissões dessa pasta corretamente, execute o comando a seguir. Substituir*folder-name*com o nome da pasta principal.

```
sudo chmod 755 folder-name
```

Erros de análise

Erros de digitação em uma configuração JSON podem levar a erros de análise. Na maioria dos casos, o problema é resultado da omissão de um colchete, vírgula ou aspas de seu arquivo JSON. O IDT executa a validação do JSON e imprime as informações de depuração. Ele imprime a linha em que ocorreu o erro, o número da linha e o número da coluna do erro de sintaxe. Essas informações devem ser suficientes para ajudá-lo a corrigir o erro, mas se você ainda não conseguir localizar o erro, poderá realizar a validação manualmente em seu IDE, em um editor de texto como Atom ou Sublime ou por meio de uma ferramenta on-line como o JSONLint.

Erros de permissão negada

O ITD executa operações em vários diretórios e arquivos em um dispositivo em teste. Algumas dessas operações exigem acesso raiz. Para automatizar essas operações, o IDT deverá ser capaz de executar comandos com sudo sem digitar uma senha.

Siga estas etapas para permitir o acesso do sudo sem digitar uma senha.

> **Note**
>
> `user` e `username` se referem ao usuário SSH usado pelo IDT para acessar o dispositivo em teste.

1. Use sudo usermod -aG sudo ***<ssh-username>*** para adicionar o usuário SSH ao grupo sudo.
2. Saia e faça login para que as alterações entrem em vigor.
3. Abra o arquivo `/etc/sudoers` e adicione a linha a seguir ao final do arquivo: *<ssh-username>* `ALL=(ALL) NOPASSWD: ALL`

   > **Note**
   >
   > Como prática recomendada, recomendamos que você use sudo visudo ao editar `/etc/sudoers`.

Erro na geração do relatório de qualificação

O IDT suporta os quatro mais recentes*major*.*minor*versões doAWS IoT GreengrassPacote de qualificação V2 (GGV2Q) para gerar relatórios de qualificação que você pode enviar paraAWS

Partner Networkpara incluir seus dispositivos noAWS PartnerCatálogo de dispositivos. As versões anteriores do pacote de qualificação não geram relatórios de qualificação.

Se você tiver dúvidas sobre a política de suporte, entre em contato comAWS Support.

Erro de parâmetro necessário ausente

Quando o IDT adiciona novos recursos, ele pode introduzir alterações nos arquivos de configuração. O uso de um arquivo de configuração antigo pode danificar sua configuração. Se isso acontecer, o arquivo `<test_case_id>.log`, em `/results/<execution-id>/logs`, listará explicitamente todos os parâmetros ausentes. O IDT também valida os esquemas do arquivo de configuração JSON para verificar se você está usando a versão mais recente compatível.

Exceção de segurança no macOS

Quando você executa o IDT em um computador host macOS, ele impede a execução do IDT. Para executar o IDT, conceda uma exceção de segurança aos executáveis que fazem parte da funcionalidade de tempo de execução do IDT. Ao ver a mensagem de aviso exibida no computador host, faça o seguinte para cada um dos executáveis aplicáveis:

Para conceder uma exceção de segurança aos executáveis do IDT

1. No computador macOS, no menu Apple, abraPreferências do sistema.
2. EscolhaSegurança e privacidade, em seguida, noGeralguia, escolha o ícone de cadeado para fazer alterações nas configurações de segurança.
3. Em caso de bloqueio`devicetester_mac_x86-64`, procure a mensagem`"devicetester_mac_x86-64" was blocked from use because it is not from an identified developer.`e escolhaPermitir mesmo assim.
4. Continue os testes de IDT até concluir todos os executáveis envolvidos.

Erros de conexão SSH

Quando o IDT não consegue se conectar a um dispositivo em teste, ele registra falhas de conexão no`/results/<execution-id>/logs/<test-case-id>.log`. As mensagens SSH aparecem na parte superior desse arquivo de log porque a conexão com um dispositivo em teste é uma das primeiras operações que o IDT executa.

A maioria das configurações do Windows usa o aplicativo de terminal PuTTY para se conectar aos hosts Linux. Esse aplicativo requer que você converta arquivos de chave privada PEM

padrão em um formato proprietário do Windows chamado PPK. Se você configurar o SSH no seu`device.json`arquivo, use arquivos PEM. Se você usa um arquivo PPK, o IDT não pode criar uma conexão SSH com oAWS IoT Greengrassdispositivo e não consigo executar testes.

A partir do IDT v4.4.0, se você não tiver habilitado o SFTP no dispositivo em teste, talvez veja o seguinte erro no arquivo de log.

```
SSH connection failed with EOF
```

Para resolver esse erro, ative o SFTP no seu dispositivo.

Erros de qualificação do gerenciador de stream

Ao executar testes de qualificação do stream manager, você pode ver o seguinte erro no`com.aws.StreamManagerExport.log`arquivo.

```
Failed to upload data to S3
```

Esse erro pode ocorrer quando o gerenciador de fluxo usa oAWScredenciais no`~/root/.aws/credentials`arquivo em seu dispositivo em vez de usar as credenciais de ambiente que o IDT exporta para o dispositivo em teste. Para evitar esse problema, exclua o`credentials`arquive no seu dispositivo e execute novamente o teste de qualificação.

Erros de tempo limite

Você pode aumentar o tempo limite de cada teste especificando um multiplicador de tempo limite aplicado ao valor padrão do tempo limite de cada teste. Qualquer valor configurado para esse sinalizador deve ser maior que ou igual a 1.0.

Para usar o multiplicador de tempo limite, use o sinalizador `--timeout-multiplier` ao executar os testes. Por exemplo:

```
./devicetester_linux run-suite --suite-id GGV2Q_1.0.0 --pool-id DevicePool1 --timeout-
multiplier 2.5
```

Para obter mais informações, execute `run-suite --help`.

Alguns erros de tempo limite ocorrem quando os casos de teste do IDT não podem ser concluídos devido a problemas de configuração. Você não pode resolver esses erros aumentando o

multiplicador de tempo limite. Use os registros da execução do teste para solucionar os problemas de configuração subjacentes.

- Se os registros do componente MQTT ou Lambda contiveremAccess deniederros, sua pasta de instalação do Greengrass pode não ter as permissões de arquivo corretas. Execute o comando a seguir para cada pasta no caminho de instalação que você definiu no seuuserdata.jsonarquivo.

  ```
  sudo chmod 755 folder-name
  ```

- Se os registros do Greengrass indicarem que a implantação da CLI do Greengrass não foi concluída, faça o seguinte:
  - Verifique issobashestá instalado no dispositivo em teste.
  - Se o seuuserdata.jsono arquivo inclui oGreengrassCliVersionparâmetro de configuração, remova-o. Esse parâmetro está obsoleto no IDT v4.1.0 e versões posteriores. Para obter mais informações, consulte Configurar userdata.json.
- Se o teste de implantação do Lambda falhar com uma mensagem de erro de “Validando a publicação do Lambda: tempo limite” e você receber um erro no arquivo de log de teste (`idt-gg2-lambda-function-idt-<resource-id>.log`) que diz`Error: Could not find or load main class com.amazonaws.greengrass.runtime.LambdaRuntime.`, faça o seguinte:
  - Verifique para qual pasta foi usada`InstallationDirRootOnDevice`no`userdata.json`arquivo.
  - Verifique se as permissões de usuário corretas estão configuradas no seu dispositivo. Para obter mais detalhes, consulteConfigure as permissões do usuário em seu dispositivo.

Erros de verificação de versão

O IDT emite o seguinte erro quando oAWSas credenciais do usuário do IDT não têm as permissões necessárias do IAM.

```
Failed to check version compatibility
```

OAWSusuário que não tem as permissões necessárias do IAM.

Política de suporte AWS IoT Device Tester para AWS IoT Greengrass

AWS IoT Device Tester for AWS IoT Greengrass é uma ferramenta de automação de testes usada para validar e qualificar seus AWS IoT Greengrass dispositivos para inclusão no Catálogo de AWS Partner dispositivos. Recomendamos que você use a versão mais recente do AWS IoT Greengrass e AWS IoT Device Tester para testar ou qualificar seus dispositivos.

Pelo menos uma versão do AWS IoT Device Tester está disponível para cada versão compatível do AWS IoT Greengrass. Para ver as versões compatíveis do AWS IoT Greengrass, consulte as versões do Greengrass nucleus. Para ver as versões compatíveis do AWS IoT Device Tester, consulteVersões compatíveis do AWS IoT Device Tester for AWS IoT Greengrass V2.

Você também pode usar qualquer uma das versões compatíveis do AWS IoT Greengrass e AWS IoT Device Tester para testar ou qualificar seus dispositivos. Embora você possa continuar usando versões não suportadas do AWS IoT Device Tester, essas versões não recebem correções de erros ou atualizações. Se você tiver dúvidas sobre a política de suporte, entre em contato com AWS Support.

Soluções de IoT baseadas em Greengrass

O Everyware da Eurotech GreenEdge está em uma versão prévia AWS IoT Greengrass e está sujeito a alterações. Essa solução não é compatível com o AWS. Você deve entrar em contato com a Eurotech para qualquer problema com este dispositivo.

AWS IoT Greengrassoferece soluções de parceiros para otimizar sua experiência na instalação do Greengrass. A seguir está uma solução que AWS foi oferecida em parceria com a Eurotech. Essa solução vem com o tempo de execução AWS IoT Greengrass do Core Edge e recursos adicionais pré-instalados.

Eurotech

AWSfez parceria com a Eurotech para oferecer uma solução de IoT para clientes que procuram um dispositivo que vem com AWS IoT Greengrass o software Core pré-instalado. O Everyware da Eurotech GreenEdge é um software de ponta de IoT pré-configurado e pré-qualificado pela. AWS Essa solução combina os recursos do Greengrass e do Eurotech Everyware Software Framework (ESF) para oferecer aos clientes ampla conectividade para o sul por meio de adaptadores de protocolo como: Modbus, OPC-UA Client/Server, S7, TwinCat, J1939, DNP3 Master/Outstation e muito mais. Com essa solução, você também pode enviar dados para o Nuvem AWS e se conectar a todos os AWS serviços do norte (comoAWS IoT Core,, AWS IoT SiteWiseAWS IoT Analytics, Amazon S3 e Amazon Kinesis Video Streams). Combinada com o Everyware Cloud, a solução de gerenciamento de dispositivos da Eurotech, essa solução apresenta um novo serviço de provisionamento Zero-Touch, que simplifica a integração de dispositivos e a implantação em massa.

Para obter mais informações sobre a Eurotech, consulte Eurotech.

Solução de problemas AWS IoT Greengrass V2

Use as informações e soluções de solução de problemas desta seção para ajudar a resolver problemas com AWS IoT Greengrass Version 2.

Tópicos

Veja os AWS IoT Greengrass principais registros de software e componentes

O software AWS IoT Greengrass Core grava registros no sistema de arquivos local que você pode usar para visualizar informações em tempo real sobre o dispositivo principal. Você também pode configurar dispositivos principais para gravar registros em CloudWatch registros, para que você possa solucionar problemas remotamente nos dispositivos principais. Esses registros podem ajudar você a identificar problemas com componentes, implantações e dispositivos principais. Para ter mais informações, consulte Monitore AWS IoT Greengrass os registros.

AWS IoT Greengrass Principais problemas de software

Solucione os AWS IoT Greengrass principais problemas do software.

Tópicos

- Não é possível configurar o dispositivo principal
- Não é possível iniciar o software AWS IoT Greengrass principal como um serviço do sistema
- Não é possível configurar o nucleus como um serviço do sistema
- Não é possível se conectar a AWS IoT Core
- Erro de falta de memória
- Não é possível instalar o Greengrass CLI
- User root is not allowed to execute
- com.aws.greengrass.lifecyclemanager.GenericExternalService: Could not determine user/group to run with
- Failed to map segment from shared object: operation not permitted
- Falha ao configurar o serviço Windows
- com.aws.greengrass.util.exceptions.TLSAuthException: Failed to get trust manager
- com.aws.greengrass.deployment.IotJobsHelper: No connection available during subscribing to Iot Jobs descriptions topic. Will retry in sometime
- software.amazon.awssdk.services.iam.model.IamException: The security token included in the request is invalid
- software.amazon.awssdk.services.iot.model.IotException: User: <user> is not authorized to perform: iot:GetPolicy
- Error: com.aws.greengrass.shadowmanager.sync.model.FullShadowSyncRequest: Could not execute cloud shadow get request
- Operation aws.greengrass#<operation> is not supported by Greengrass
- java.io.FileNotFoundException: <stream-manager-store-root-dir>/stream_manager_metadata_store (Permission denied)
- com.aws.greengrass.security.provider.pkcs11.PKCS11CryptoKeyService: Private key or certificate with label <label> does not exist
- software.amazon.awssdk.services.secretsmanager.model.SecretsManagerException: User: <user> is not authorized to perform: secretsmanager:GetSecretValue on resource: <arn>
- software.amazon.awssdk.services.secretsmanager.model.SecretsManagerException: Access to KMS is not allowed
- java.lang.NoClassDefFoundError: com/aws/greengrass/security/CryptoKeySpi
- com.aws.greengrass.security.provider.pkcs11.PKCS11CryptoKeyService: CKR_OPERATION_NOT_INITIALIZED

- Greengrass core device stuck on nucleus v2.12.3

Não é possível configurar o dispositivo principal

Se o instalador do software AWS IoT Greengrass Core falhar e você não conseguir configurar um dispositivo principal, talvez seja necessário desinstalar o software e tentar novamente. Para ter mais informações, consulte Desinstale o software AWS IoT Greengrass principal.

Não é possível iniciar o software AWS IoT Greengrass principal como um serviço do sistema

Se o software AWS IoT Greengrass principal não for iniciado, verifique os registros de serviço do sistema para identificar o problema. Um problema comum é quando o Java não está disponível na variável de ambiente PATH (Linux) ou na variável de sistema PATH (Windows).

Não é possível configurar o nucleus como um serviço do sistema

Você pode ver esse erro quando o instalador do software AWS IoT Greengrass Core não consegue ser configurado AWS IoT Greengrass como um serviço do sistema. Em dispositivos Linux, esse erro geralmente ocorre se o dispositivo principal não tiver o sistema de inicialização systemd. O instalador pode configurar com êxito o software AWS IoT Greengrass Core, mesmo que não consiga configurar o serviço do sistema.

Execute um destes procedimentos:

- Configure e execute o software AWS IoT Greengrass Core como um serviço do sistema. Você deve configurar o software como um serviço do sistema para usar todos os recursos do AWS IoT Greengrass. Você pode instalar o systemd ou usar um sistema init diferente. Para ter mais informações, consulte Configurar o núcleo do Greengrass como um serviço do sistema.
- Execute o software AWS IoT Greengrass Core sem um serviço de sistema. Você pode executar o software usando um script de carregamento que o instalador configura na pasta raiz do Greengrass. Para ter mais informações, consulte Execute o software AWS IoT Greengrass Core sem um serviço de sistema.

Não é possível se conectar a AWS IoT Core

Você pode ver esse erro quando o software AWS IoT Greengrass Core não consegue se conectar AWS IoT Core para recuperar trabalhos de implantação, por exemplo. Faça o seguinte:

- Verifique se seu dispositivo principal pode se conectar à Internet AWS IoT Core e. Para obter mais informações sobre o AWS IoT Core endpoint ao qual seu dispositivo se conecta, consulteConfigurar o software AWS IoT Greengrass principal.
- Verifique se o dispositivo AWS IoT principal usa um certificado que permite as `iot:Subscribe` permissões `iot:Connect` `iot:Publish``iot:Receive`,, e.
- Se seu dispositivo principal usa um proxy de rede, verifique se seu dispositivo principal tem uma função de dispositivo e se essa função permite as `iot:Subscribe` permissões `iot:Connect` `iot:Publish``iot:Receive`,, e.

Erro de falta de memória

Esse erro geralmente ocorre se o dispositivo não tiver memória suficiente para alocar um objeto no heap Java. Em dispositivos com memória limitada, talvez seja necessário especificar um tamanho máximo de pilha para controlar a alocação de memória. Para ter mais informações, consulte Controle a alocação de memória com opções de JVM.

Não é possível instalar o Greengrass CLI

Você pode ver a seguinte mensagem do console ao usar o `--deploy-dev-tools` argumento no comando de instalação AWS IoT Greengrass do Core.

```
Thing group exists, it could have existing deployment and devices, hence NOT creating
 deployment for Greengrass first party dev tools, please manually create a deployment
 if you wish to
```

Isso ocorre quando o componente CLI do Greengrass não está instalado porque seu dispositivo principal é membro de um grupo de coisas que tem uma implantação existente. Se você ver essa mensagem, poderá implantar manualmente o componente da CLI do Greengrass (`aws.greengrass.Cli`) no dispositivo para instalar a CLI do Greengrass. Para ter mais informações, consulte Instale a CLI do Greengrass.

User root is not allowed to execute

Você pode ver esse erro quando o usuário que executa o software AWS IoT Greengrass Core (normalmente`root`) não tem permissão para executar `sudo` com nenhum usuário e nenhum grupo. Para o usuário padrão do `ggc_user` sistema, esse erro é semelhante ao seguinte:

```
Sorry, user root is not allowed to execute <command> as ggc_user:ggc_group.
```

Verifique se seu `/etc/sudoers` arquivo dá permissão ao usuário para ser executado `sudo` como outros grupos. A permissão para o usuário entrar `/etc/sudoers` deve ser semelhante ao exemplo a seguir.

```
root    ALL=(ALL:ALL) ALL
```

com.aws.greengrass.lifecyclemanager.GenericExternalService: Could not determine user/group to run with

Você pode ver esse erro quando o dispositivo principal tenta executar um componente e o núcleo do Greengrass não especifica um usuário padrão do sistema a ser usado para executar componentes.

Para corrigir esse problema, configure o núcleo do Greengrass para especificar o usuário padrão do sistema que executa os componentes. Para obter mais informações, consulte Configurar o usuário que executa os componentes e Configurar o usuário padrão do componente.

Failed to map segment from shared object: operation not permitted

Você pode ver esse erro quando o software AWS IoT Greengrass Core falha ao iniciar porque a `/tmp` pasta está montada com `noexec` permissões. A biblioteca AWS Common Runtime (CRT) usa a `/tmp` pasta por padrão.

Execute um destes procedimentos:

- Execute o comando a seguir para remontar a `/tmp` pasta com `exec` permissões e tente novamente.

  ```
  sudo mount -o remount,exec /tmp
  ```

- Se você executar o Greengrass nucleus v2.5.0 ou posterior, poderá definir uma opção de JVM para alterar a pasta que a biblioteca CRT usa. AWS Você pode especificar o `jvmOptions` parâmetro na configuração do componente nucleus do Greengrass em uma implantação ou ao instalar o AWS IoT Greengrass software Core. Substitua *`/path/to/use`* pelo caminho para uma pasta que a biblioteca CRT possa usar. AWS

  ```
  {
  ```

```
  "jvmOptions": "-Daws.crt.lib.dir=\"/path/to/use\""
}
```

Falha ao configurar o serviço Windows

Você pode ver esse erro se você instalar o software AWS IoT Greengrass Core em um dispositivo Microsoft Windows 2016. O software AWS IoT Greengrass Core não é compatível com o Windows 2016. Para obter uma lista dos sistemas operacionais compatíveis, consulte Plataformas compatíveis.

Se você precisar usar o Windows 2016, você pode fazer o seguinte:

1. Descompacte o arquivo de instalação AWS IoT Greengrass do Core baixado
2. No `Greengrass` diretório, abra o `bin/greengrass.xml.template` arquivo.
3. Adicione a `<autoRefresh>` tag ao final do arquivo logo antes da `</service>` tag.

```
  </log>
  <autoRefresh>false</autoRefresh>
</service>
```

com.aws.greengrass.util.exceptions.TLSAuthException: Failed to get trust manager

Você pode ver esse erro ao instalar o software AWS IoT Greengrass Core sem um arquivo de autoridade de certificação (CA) raiz.

```
2022-06-05T10:00:39.556Z [INFO] (main) com.aws.greengrass.lifecyclemanager.Kernel:
 service-loaded. {serviceName=DeploymentService}
2022-06-05T10:00:39.943Z [WARN] (main)
 com.aws.greengrass.componentmanager.ClientConfigurationUtils: configure-greengrass-
mutual-auth. Error during configure greengrass client mutual auth. {}
com.aws.greengrass.util.exceptions.TLSAuthException: Failed to get trust manager
```

Verifique se você especificou um arquivo CA raiz válido com o `rootCaPath` parâmetro no arquivo de configuração fornecido ao instalador. Para ter mais informações, consulte Instalar o software do AWS IoT Greengrass Core.

com.aws.greengrass.deployment.IotJobsHelper: No connection available during subscribing to Iot Jobs descriptions topic. Will retry in sometime

Você pode ver essa mensagem de aviso quando o dispositivo principal não consegue se conectar AWS IoT Core para assinar as notificações de trabalho de implantação. Faça o seguinte:

- Verifique se o dispositivo principal está conectado à Internet e pode alcançar o ponto final de AWS IoT dados que você configurou. Para obter mais informações sobre endpoints que os dispositivos principais usam, consulte Permitir o tráfego de dispositivos por meio de um proxy ou firewall.
- Verifique os registros do Greengrass em busca de outros erros que revelem outras causas principais.

software.amazon.awssdk.services.iam.model.IamException: The security token included in the request is invalid

Você pode ver esse erro ao instalar o software AWS IoT Greengrass Core com provisionamento automático, e o instalador usa um token de AWS sessão que não é válido. Faça o seguinte:

- Se você usar credenciais de segurança temporárias, verifique se o token da sessão está correto e se você está copiando e colando o token de sessão completo.
- Se você usa credenciais de segurança de longo prazo, verifique se o dispositivo não tem um token de sessão de uma época em que você usava credenciais temporárias anteriormente. Faça o seguinte:
  1. Execute o comando a seguir para cancelar a configuração da variável de ambiente do token de sessão.

     Linux or Unix

     ```
     unset AWS_SESSION_TOKEN
     ```

     Windows Command Prompt (CMD)

     ```
     set AWS_SESSION_TOKEN=
     ```

     PowerShell

     ```
     Remove-Item Env:\AWS_SESSION_TOKEN
     ```

2. Verifique se o arquivo de AWS credenciais,`~/.aws/credentials`, contém um token de sessão,`aws_session_token`. Em caso afirmativo, remova essa linha do arquivo.

```
aws_session_token = AQoEXAMPLEH4aoAH0gNCAPyJxz4BlCFFxWNE1OPTgk5TthT
+FvwqnKwRc0IfrRh3c/LTo6UDdyJwOOvEVPvLXCrrrUtdnniCEXAMPLE/
IvU1dYUg2RVAJBanLiHb4IgRmpRV3zrkuWJOgQs8IZZaIv2BXIa2R4Olgk
```

Você também pode instalar o software AWS IoT Greengrass Core sem fornecer AWS credenciais. Para obter mais informações, consulte Instale o software AWS IoT Greengrass principal com provisionamento manual de recursos ou Instale o software AWS IoT Greengrass principal com provisionamento de AWS IoT frota.

software.amazon.awssdk.services.iot.model.IotException: User: <user> is not authorized to perform: iot:GetPolicy

Você pode ver esse erro ao instalar o software AWS IoT Greengrass Core com provisionamento automático, e o instalador usa AWS credenciais que não têm as permissões necessárias. Para obter mais informações sobre as permissões necessárias, consultePolítica mínima de IAM para o instalador provisionar recursos.

Verifique as permissões da identidade do IAM das credenciais e conceda à identidade do IAM todas as permissões necessárias que estejam faltando.

Error: com.aws.greengrass.shadowmanager.sync.model.FullShadowSyncRequest: Could not execute cloud shadow get request

Você pode ver esse erro ao usar o componente gerenciador de sombras para sincronizar as sombras do dispositivo. AWS IoT Core O código de status HTTP 403 indica que esse erro ocorreu porque a AWS IoT política do dispositivo principal não concede permissão para fazer chamadasGetThingShadow.

```
com.aws.greengrass.shadowmanager.sync.model.FullShadowSyncRequest: Could not execute
 cloud shadow get request. {thing name=MyGreengrassCore, shadow name=MyShadow}
2021-07-14T21:09:02.456Z [ERROR] (pool-2-thread-109)
 com.aws.greengrass.shadowmanager.sync.SyncHandler: sync. Skipping sync request. {thing
 name=MyGreengrassCore, shadow name=MyShadow}
```

```
com.aws.greengrass.shadowmanager.exception.SkipSyncRequestException:
 software.amazon.awssdk.services.iotdataplane.model.IotDataPlaneException:
 null (Service: IotDataPlane, Status Code: 403, Request ID:
 f6e713ba-1b01-414c-7b78-5beb3f3ad8f6, Extended Request ID: null)
```

Para sincronizar sombras locais com AWS IoT Core, a AWS IoT política do dispositivo principal deve conceder as seguintes permissões:

- `iot:GetThingShadow`
- `iot:UpdateThingShadow`
- `iot:DeleteThingShadow`

Verifique a AWS IoT política do dispositivo principal e adicione as permissões necessárias que estão faltando. Para mais informações, consulte:

- AWS IoT Core ações políticas no Guia do AWS IoT desenvolvedor
- Atualizar a AWS IoT política de um dispositivo principal

Operation aws.greengrass#<operation> is not supported by Greengrass

Você pode ver esse erro ao usar uma operação de comunicação entre processos (IPC) em um componente personalizado do Greengrass, e o componente AWS fornecido necessário não está instalado no dispositivo principal.

Para corrigir esse problema, adicione o componente necessário como uma dependência em sua receita de componentes, para que o software AWS IoT Greengrass Core instale o componente necessário quando você implanta seu componente.

- Recupere valores secretos — `aws.greengrass.SecretManager`
- Interaja com sombras locais — `aws.greengrass.ShadowManager`
- Gerencie implantações e componentes locais — `aws.greengrass.Cli` v2.6.0 ou posterior
- Autenticar e autorizar dispositivos clientes — `aws.greengrass.clientdevices.Auth` v2.2.0 ou posterior

java.io.FileNotFoundException: <stream-manager-store-root-dir>/stream_manager_metadata_store (Permission denied)

Você pode ver esse erro no arquivo de log do gerenciador de stream (`aws.greengrass.StreamManager.log`) ao configurar o stream manager para usar uma pasta raiz que não existe ou tem as permissões corretas. Para obter mais informações sobre como configurar essa pasta, consulte configuração do gerenciador de fluxo.

com.aws.greengrass.security.provider.pkcs11.PKCS11CryptoKeyService: Private key or certificate with label <label> does not exist

Esse erro ocorre quando o componente do provedor PKCS #11 não consegue encontrar ou carregar a chave privada ou o certificado que você especifica ao configurar o software AWS IoT Greengrass Core para usar um módulo de segurança de hardware (HSM). Faça o seguinte:

- Verifique se a chave privada e o certificado estão armazenados no HSM usando o slot, o PIN do usuário e a etiqueta do objeto que você configura para usar no software AWS IoT Greengrass Core.
- Verifique se a chave privada e o certificado usam o mesmo rótulo de objeto no HSM.
- Se o seu HSM suportar IDs de objeto, verifique se a chave privada e o certificado usam o mesmo ID de objeto no HSM.

Consulte a documentação do seu HSM para saber como consultar detalhes sobre os tokens de segurança no HSM. Se você precisar alterar o slot, o rótulo do objeto ou o ID do objeto de um token de segurança, consulte a documentação do seu HSM para saber como fazer isso.

software.amazon.awssdk.services.secretsmanager.model.SecretsManagerExcep User: <user> is not authorized to perform: secretsmanager:GetSecretValue on resource: <arn>

Esse erro pode ocorrer quando você usa o componente gerenciador de segredos para implantar um AWS Secrets Manager segredo. Se a função IAM de troca de tokens do dispositivo principal não conceder permissão para obter o segredo, a implantação falhará e os registros do Greengrass incluirão esse erro.

Para autorizar um dispositivo principal a baixar um segredo

1. Adicione a `secretsmanager:GetSecretValue` permissão à função de troca de tokens do dispositivo principal. O exemplo de declaração de política a seguir concede permissão para obter o valor de um segredo.

   ```
   {
       "Effect": "Allow",
       "Action": [
           "secretsmanager:GetSecretValue"
       ],
       "Resource": [
           "arn:aws:secretsmanager:us-west-2:123456789012:secret:MyGreengrassSecret-
   abcdef"
       ]
   }
   ```

   Para ter mais informações, consulte Autorize os dispositivos principais a interagir com os serviços AWS.
2. Reaplique a implantação no dispositivo principal. Execute um destes procedimentos:
   - Revise a implantação sem nenhuma alteração. O dispositivo principal tenta baixar o segredo novamente quando recebe a implantação revisada. Para ter mais informações, consulte Revise as implantações.
   - Reinicie o software AWS IoT Greengrass principal para tentar a implantação novamente. Para mais informações, consulte Execute o software AWS IoT Greengrass Core.

   A implantação será bem-sucedida se o gerenciador secreto baixar o segredo com êxito.

software.amazon.awssdk.services.secretsmanager.model.SecretsManagerExcep Access to KMS is not allowed

Esse erro pode ocorrer quando você usa o componente gerenciador de segredos para implantar um AWS Secrets Manager segredo criptografado por uma AWS Key Management Service chave. Se a função do IAM de troca de tokens do dispositivo principal não conceder permissão para descriptografar o segredo, a implantação falhará e os registros do Greengrass incluirão esse erro.

Para corrigir o problema, adicione a `kms:Decrypt` permissão à função de troca de tokens do dispositivo principal. Para mais informações, consulte:

- Criptografia e decodificação secretas no Guia do Usuário AWS Secrets Manager
- Autorize os dispositivos principais a interagir com os serviços AWS

java.lang.NoClassDefFoundError: com/aws/greengrass/security/CryptoKeySpi

Você pode ver esse erro ao tentar instalar o software AWS IoT Greengrass Core com segurança de hardware e usar uma versão anterior do Greengrass nucleus que não oferece suporte à integração de segurança de hardware. Para usar a integração de segurança de hardware, você deve usar o Greengrass nucleus v2.5.3 ou posterior.

com.aws.greengrass.security.provider.pkcs11.PKCS11CryptoKeyService: CKR_OPERATION_NOT_INITIALIZED

Você pode ver esse erro ao usar a biblioteca TPM2 ao executar o AWS IoT Greengrass Core como um serviço do sistema.

Esse erro indica que você precisa adicionar uma variável de ambiente que forneça a localização do armazenamento PKCS #11 no arquivo de serviço AWS IoT Greengrass Core systemd.

Para obter mais informações, consulte a seção Requisitos da documentação do Fornecedor PKCS #11 componente.

Greengrass core device stuck on nucleus v2.12.3

Se seu dispositivo principal do Greengrass não revisar sua implantação a partir da versão 2.12.3 do nucleus, talvez seja necessário baixar e substituir o arquivo `Greengrass.jar` pela versão 2.12.2 do núcleo do Greengrass. Faça o seguinte:

1. Em seu dispositivo principal do Greengrass, execute o comando a seguir para interromper o software Greengrass Core.

   Linux or Unix

   ```
   sudo systemctl stop greengrass
   ```

Windows Command Prompt (CMD)

```
sc stop "greengrass"
```

PowerShell

```
Stop-Service -Name "greengrass"
```

2. Em seu dispositivo principal, baixe o AWS IoT Greengrass software em um arquivo chamado`greengrass-2.12.2.zip`.

   Linux or Unix

   ```
   curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-2.12.2.zip >
    greengrass-2.12.2.zip
   ```

   Windows Command Prompt (CMD)

   ```
   curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-2.12.2.zip >
    greengrass-2.12.2.zip
   ```

   PowerShell

   ```
   iwr -Uri https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-2.12.2.zip -
   OutFile greengrass-2.12.2.zip
   ```

3. Descompacte o software AWS IoT Greengrass Core em uma pasta no seu dispositivo. *`GreengrassInstaller`*Substitua pela pasta que você deseja usar.

   Linux or Unix

   ```
   unzip greengrass-2.12.2.zip -d GreengrassInstaller && rm greengrass-2.12.2.zip
   ```

   Windows Command Prompt (CMD)

   ```
   mkdir GreengrassInstaller && tar -xf greengrass-2.12.2.zip -
   C GreengrassInstaller && del greengrass-2.12.2.zip
   ```

PowerShell

```
Expand-Archive -Path greengrass-2.12.2.zip -DestinationPath .\
\GreengrassInstaller
rm greengrass-2.12.2.zip
```

4. Execute o comando a seguir para substituir o arquivo JAR do Greengrass versão 2.12.3 do núcleo pelo arquivo JAR do Greengrass versão 2.12.2 do núcleo.

   Linux or Unix

   ```
   sudo cp ./GreengrassInstaller/lib/Greengrass.jar /greengrass/v2/packages/
   artifacts-unarchived/aws.greengrass.Nucleus/2.12.3/aws.greengrass.nucleus/lib
   ```

   Windows Command Prompt (CMD)

   ```
   robocopy ./GreengrassInstaller/lib/Greengrass.jar /greengrass/v2/packages/
   artifacts-unarchived/aws.greengrass.Nucleus/2.12.3/aws.greengrass.nucleus/lib /E
   ```

   PowerShell

   ```
   cp -Path ./GreengrassInstaller/lib/Greengrass.jar -Destination /
   greengrass/v2/packages/artifacts-unarchived/aws.greengrass.Nucleus/2.12.3/
   aws.greengrass.nucleus/lib
   ```

5. Execute o comando a seguir para iniciar o software Greengrass Core.

   Linux or Unix

   ```
   sudo systemctl start greengrass
   ```

   Windows Command Prompt (CMD)

   ```
   sc start "greengrass"
   ```

   PowerShell

   ```
   Start-Service -Name "greengrass"
   ```

AWS IoT Greengrass problemas de nuvem

Use as informações a seguir para solucionar problemas com o AWS IoT Greengrass console e a API. Cada entrada corresponde a uma mensagem de erro que você pode ver ao executar uma ação.

An error occurred (AccessDeniedException) when calling the CreateComponentVersion operation: User: arn:aws:iam::123456789012:user/<username> is not authorized to perform: null

Você pode ver esse erro ao criar uma versão do componente a partir do AWS IoT Greengrass console ou com a CreateComponentVersionoperação.

Esse erro indica que sua receita não é um JSON ou YAML válido. Verifique a sintaxe da sua receita, corrija os problemas de sintaxe e tente novamente. Você pode usar um verificador de sintaxe JSON ou YAML online para identificar problemas de sintaxe em sua receita.

Invalid Input: Encountered following errors in Artifacts: {<s3ArtifactUri> = Specified artifact resource cannot be accessed}

Você pode ver esse erro ao criar uma versão do componente a partir do AWS IoT Greengrass console ou com a CreateComponentVersionoperação. Esse erro indica que um artefato S3 na receita do componente não é válido.

Faça o seguinte:

- Verifique se o bucket do S3 está no mesmo Região da AWS local em que você criou o componente. AWS IoT Greengrass não oferece suporte a solicitações entre regiões de artefatos de componentes.
- Verifique se o URI do artefato é um URL válido do objeto do S3 e verifique se o artefato existe nesse URL do objeto do S3.
- Verifique se você Conta da AWS tem permissão para acessar o artefato na URL do objeto do S3.

INACTIVE deployment status

Você pode obter um status de `INACTIVE` implantação ao chamar a ListDeploymentsAPI sem as AWS IoT políticas dependentes necessárias. Você deve ter as permissões necessárias

para obter um status de implantação preciso. Você pode encontrar as ações dependentes consultando as Ações definidas por AWS IoT Greengrass V2 e seguindo as permissões necessárias para`ListDeployments`. Sem as AWS IoT permissões dependentes necessárias, você ainda verá o status de implantação, mas poderá ver um status de implantação impreciso de`INACTIVE`.

Principais problemas de implantação de dispositivos

Solucione problemas de implantação nos dispositivos principais do Greengrass. Cada entrada corresponde a uma mensagem de registro que você pode ver no seu dispositivo principal.

Tópicos

- Error: com.aws.greengrass.componentmanager.exceptions.PackageDownloadException: Failed to download artifact
- Error: com.aws.greengrass.componentmanager.exceptions.ArtifactChecksumMismatchException: Integrity check for downloaded artifact failed. Probably due to file corruption.
- Error: com.aws.greengrass.componentmanager.exceptions.NoAvailableComponentVersionException: Failed to negotiate component <name> version with cloud and no local applicable version satisfying requirement <requirements>
- software.amazon.awssdk.services.greengrassv2data.model.ResourceNotFoundException: The latest version of Component <componentName> doesn't claim platform <coreDevicePlatform> compatibility
- com.aws.greengrass.componentmanager.exceptions.PackagingException: The deployment attempts to update the nucleus from aws.greengrass.Nucleus-<version> to aws.greengrass.Nucleus-<version> but no component of type nucleus was included as target component
- Error: com.aws.greengrass.deployment.exceptions.DeploymentException: Unable to process deployment. Greengrass launch directory is not set up or Greengrass is not set up as a system service
- Info: com.aws.greengrass.deployment.exceptions.RetryableDeploymentDocumentDownloadException: Greengrass Cloud Service returned an error when getting full deployment configuration
- Warn: com.aws.greengrass.deployment.DeploymentService: Failed to get thing group hierarchy
- Info: com.aws.greengrass.deployment.DeploymentDocumentDownloader: Calling Greengrass cloud to get full deployment configuration

- Caused by: software.amazon.awssdk.services.greengrassv2data.model.GreengrassV2DataException: null (Service: GreengrassV2Data, Status Code: 403, Request ID: <some_request_id>, Extended Request ID: null)

Error: com.aws.greengrass.componentmanager.exceptions.PackageDownloadExceptio Failed to download artifact

Você pode ver esse erro quando o software AWS IoT Greengrass Core não consegue baixar um artefato de componente quando o dispositivo principal aplica uma implantação. A implantação falha como resultado desse erro.

Quando você recebe esse erro, o registro também inclui um rastreamento de pilha que você pode usar para identificar o problema específico. Cada uma das entradas a seguir corresponde a uma mensagem que você pode ver no rastreamento de pilha da mensagem de `Failed to download artifact` erro.

Tópicos

- software.amazon.awssdk.services.s3.model.S3Exception: null (Service: S3, Status Code: 403, Request ID: null, ...)
- software.amazon.awssdk.services.s3.model.S3Exception: Access Denied (Service: S3, Status Code: 403, Request ID: <requestID>

software.amazon.awssdk.services.s3.model.S3Exception: null (Service: S3, Status Code: 403, Request ID: null, ...)

O PackageDownloadException erro pode incluir esse rastreamento de pilha nos seguintes casos:

- O artefato do componente não está disponível na URL do objeto do S3 que você especifica na receita do componente. Verifique se você carregou o artefato no bucket do S3 e se o URI do artefato corresponde ao URL do objeto do S3 do artefato no bucket.
- A função de troca de tokens do dispositivo principal não permite que o software AWS IoT Greengrass Core baixe o artefato do componente da URL do objeto S3 que você especifica na receita do componente. Verifique se a função de troca de tokens permite `s3:GetObject` a URL do objeto S3 em que o artefato está disponível.

software.amazon.awssdk.services.s3.model.S3Exception: Access Denied (Service: S3, Status Code: 403, Request ID: <requestID>

O PackageDownloadException erro pode incluir esse rastreamento de pilha quando o dispositivo principal não tem permissão para fazer chamadas`s3:GetBucketLocation`. A mensagem de erro também inclui a seguinte mensagem.

```
reason: Failed to determine S3 bucket location
```

Verifique se a função de troca de tokens do dispositivo principal permite `s3:GetBucketLocation` o bucket S3 em que o artefato está disponível.

Error: com.aws.greengrass.componentmanager.exceptions.ArtifactChecksumMismatchI Integrity check for downloaded artifact failed. Probably due to file corruption.

Você pode ver esse erro quando o software AWS IoT Greengrass Core não consegue baixar um artefato de componente quando o dispositivo principal aplica uma implantação. A implantação falha porque a soma de verificação do arquivo de artefato baixado não corresponde à soma de verificação AWS IoT Greengrass calculada quando você criou o componente.

Faça o seguinte:

- Verifique se o arquivo de artefato foi alterado no bucket do S3 em que você o hospeda. Se o arquivo tiver sido alterado desde que você criou o componente, restaure-o para a versão anterior que o dispositivo principal espera. Se você não conseguir restaurar o arquivo para a versão anterior ou se quiser usar a nova versão do arquivo, crie uma nova versão do componente com o arquivo de artefato.
- Verifique a conexão com a Internet do seu dispositivo principal. Esse erro pode ocorrer se o arquivo do artefato for corrompido durante o download. Crie uma nova implantação e tente novamente.

Error: com.aws.greengrass.componentmanager.exceptions.NoAvailableComponentVers Failed to negotiate component <name> version with cloud and no local applicable version satisfying requirement <requirements>

Você pode ver esse erro quando um dispositivo principal não consegue encontrar uma versão do componente que atenda aos requisitos das implantações desse dispositivo principal. O dispositivo principal verifica o componente no AWS IoT Greengrass serviço e no dispositivo local. A mensagem de erro inclui o destino de cada implantação e os requisitos de versão dessa implantação para o componente. O destino da implantação pode ser uma coisa, um grupo de coisas ou`LOCAL_DEPLOYMENT`, que representa a implantação local no dispositivo principal.

Esse problema pode ocorrer nos seguintes casos:

- O dispositivo principal é o alvo de várias implantações que têm requisitos de versão de componentes conflitantes. Por exemplo, o dispositivo principal pode ser o alvo de várias implantações que incluem um `com.example.HelloWorld` componente, em que uma implantação requer a versão 1.0.0 e a outra requer a versão 1.0.1. É impossível ter um componente que atenda aos dois requisitos, então a implantação falha.
- A versão do componente não existe no AWS IoT Greengrass serviço ou no dispositivo local. O componente pode ter sido excluído, por exemplo.
- Existem versões de componentes que atendem aos requisitos de versão, mas nenhuma é compatível com a plataforma do dispositivo principal.
- A AWS IoT política do dispositivo principal não concede a `greengrass:ResolveComponentCandidates` permissão. Procure `Status Code: 403` no registro de erros para identificar esse problema. Para resolver esse problema, adicione a `greengrass:ResolveComponentCandidates` permissão à AWS IoT política do dispositivo principal. Para ter mais informações, consulte AWS IoTPolítica mínima para dispositivos AWS IoT Greengrass V2 principais.

Para resolver esse problema, revise as implantações para incluir versões de componentes compatíveis ou remover as incompatíveis. Para obter mais informações sobre como revisar as implantações na nuvem, consulte. Revise as implantações Para obter mais informações sobre como revisar implantações locais, consulte o comando AWS IoT Greengrass CLI deployment create.

software.amazon.awssdk.services.greengrassv2data.model.ResourceNotFoundE The latest version of Component <componentName> doesn't claim platform <coreDevicePlatform> compatibility

Você pode ver esse erro ao implantar um componente em um dispositivo principal, e o componente não lista uma plataforma compatível com a plataforma do dispositivo principal. Execute um destes procedimentos:

- Se o componente for um componente personalizado do Greengrass, você poderá atualizar o componente para que seja compatível com o dispositivo principal. Adicione um novo manifesto que corresponda à plataforma do dispositivo principal ou atualize um manifesto existente para corresponder à plataforma do dispositivo principal. Para ter mais informações, consulte AWS IoT Greengrass referência da receita do componente.
- Se o componente for fornecido pela AWS, verifique se outra versão do componente é compatível com o dispositivo principal. Se nenhuma versão for compatível, entre em contato conosco AWS re:Postusando a AWS IoT Greengrass tag ou entre em contato AWS Support.

com.aws.greengrass.componentmanager.exceptions.PackagingException: The deployment attempts to update the nucleus from aws.greengrass.Nucleus-<version> to aws.greengrass.Nucleus-<version> but no component of type nucleus was included as target component

Você pode ver esse erro ao implantar um componente que depende do núcleo do Greengrass, e o dispositivo principal executa uma versão anterior do núcleo do Greengrass do que a versão secundária mais recente disponível. Esse erro ocorre porque o software AWS IoT Greengrass Core tenta atualizar automaticamente os componentes para a versão compatível mais recente. No entanto, o software AWS IoT Greengrass Core impede que o núcleo do Greengrass seja atualizado para uma nova versão secundária, porque vários componentes AWS fornecidos dependem de versões secundárias específicas do núcleo do Greengrass. Para ter mais informações, consulte Comportamento de atualização do núcleo Greengrass.

Você deve revisar a implantação para especificar a versão do núcleo do Greengrass que você deseja usar. Execute um destes procedimentos:

- Revise a implantação para especificar a versão do núcleo do Greengrass que o dispositivo principal executa atualmente.

- Revise a implantação para especificar uma versão secundária posterior do núcleo do Greengrass. Se você escolher essa opção, também deverá atualizar as versões de todos os componentes AWS fornecidos que dependem de versões secundárias específicas do núcleo do Greengrass. Para ter mais informações, consulte AWS-componentes fornecidos.

Error: com.aws.greengrass.deployment.exceptions.DeploymentException: Unable to process deployment. Greengrass launch directory is not set up or Greengrass is not set up as a system service

Você pode ver esse erro ao mover um dispositivo Greengrass de um grupo de coisas para outro e depois voltar para o grupo original com implantações que exigem a reinicialização do Greengrass.

Para resolver esse problema, recrie o diretório de inicialização do dispositivo. Também é altamente recomendável atualizar para a versão 2.9.6 ou posterior do núcleo Greengrass.

A seguir está um script Linux para recriar o diretório de inicialização. Salve o script em um arquivo chamado`fix_directory.sh`.

```
#!/bin/bash

set -e

GG_ROOT=$1
GG_VERSION=$2

CURRENT="$GG_ROOT/alts/current"

if [ ! -L "$CURRENT" ]; then
  mkdir -p $GG_ROOT/alts/directory_fix
  echo "Relinking $GG_ROOT/alts/directory_fix to $CURRENT"
  ln -sf $GG_ROOT/alts/directory_fix $CURRENT
fi

TARGET=$(readlink $CURRENT)

if [[ ! -d "$TARGET" ]]; then
  echo "Creating directory: $TARGET"
  mkdir -p "$TARGET"
fi
```

```
DISTRO_LINK="$TARGET/distro"
DISTRO="$GG_ROOT/packages/artifacts-unarchived/aws.greengrass.Nucleus/$GG_VERSION/
aws.greengrass.nucleus/"
echo "Relinking Nucleus artifacts to $DISTRO_LINK"
ln -sf $DISTRO $DISTRO_LINK
```

Para executar o script, execute o seguinte comando:

```
[root@ip-172-31-27-165 ~]# ./fix_directory.sh /greengrass/v2 2.9.5
Relinking /greengrass/v2/alts/directory_fix to /greengrass/v2/alts/current
Relinking Nucleus artifacts to /greengrass/v2/alts/directory_fix/distro
```

Info: com.aws.greengrass.deployment.exceptions.RetryableDeploymentDocumentDow Greengrass Cloud Service returned an error when getting full deployment configuration

Você pode ver esse erro quando o dispositivo principal recebe um documento de implantação grande, que é um documento de implantação maior que 7 KB (para implantações que visam coisas) ou 31 KB (para implantações que visam grupos de coisas). Para recuperar um documento de implantação grande, a AWS IoT política de um dispositivo principal deve permitir a `greengrass:GetDeploymentConfiguration` permissão. Esse erro pode ocorrer quando o dispositivo principal não tem essa permissão. Quando esse erro ocorre, a implantação tenta novamente indefinidamente e seu status é Em andamento ()`IN_PROGRESS`.

Para resolver esse problema, adicione a `greengrass:GetDeploymentConfiguration` permissão à AWS IoT política do dispositivo principal. Para ter mais informações, consulte Atualizar a AWS IoT política de um dispositivo principal.

Warn: com.aws.greengrass.deployment.DeploymentService: Failed to get thing group hierarchy

Você pode ver esse aviso quando o dispositivo principal recebe uma implantação e a AWS IoT política do dispositivo principal não permite a `greengrass:ListThingGroupsForCoreDevice` permissão. Quando você cria uma implantação, o dispositivo principal usa essa permissão para identificar seus grupos de coisas e remover componentes de qualquer grupo de coisas do qual você removeu o dispositivo principal. Se o dispositivo principal executar o Greengrass nucleus v2.5.0, a

implantação falhará. Se o dispositivo principal executar o Greengrass nucleus v2.5.1 ou posterior, a implantação prosseguirá, mas não removerá os componentes. Para obter mais informações sobre o comportamento de remoção de grupos de coisas, consulteImplemente AWS IoT Greengrass componentes em dispositivos.

Para atualizar o comportamento do dispositivo principal para remover componentes de grupos de coisas dos quais você remove o dispositivo principal, adicione a `greengrass:ListThingGroupsForCoreDevice` permissão à AWS IoT política do dispositivo principal. Para ter mais informações, consulte Atualizar a AWS IoT política de um dispositivo principal.

Info: com.aws.greengrass.deployment.DeploymentDocumentDownloader: Calling Greengrass cloud to get full deployment configuration

Você pode ver essa mensagem informativa impressa várias vezes sem erros, porque o dispositivo principal registra o erro no nível do `DEBUG` registro. Esse problema pode ocorrer quando o dispositivo principal recebe um grande documento de implantação. Quando esse problema ocorre, a implantação tenta novamente indefinidamente e seu status é Em andamento ()`IN_PROGRESS`. Para obter mais informações sobre como resolver esse problema, consulte esta entrada de solução de problemas.

Caused by: software.amazon.awssdk.services.greengrassv2data.model.GreengrassV2DataE null (Service: GreengrassV2Data, Status Code: 403, Request ID: <some_request_id>, Extended Request ID: null)

Talvez você veja esse erro quando uma API de plano de dados não tem `iot:Connect` permissão. Se você não tiver a política correta, você receberá uma`GreengrassV2DataException: 403`. Para criar uma política de permissão, siga estas instruções:Criar uma política do AWS IoT.

Problemas com os principais componentes do dispositivo

Solucione problemas com componentes do Greengrass nos dispositivos principais.

Tópicos

- Warn: '<command>' is not recognized as an internal or external command
- O script Python não registra mensagens

- A configuração do componente não é atualizada ao alterar a configuração padrão
- awsiot.greengrasscoreipc.model.UnauthorizedError
- com.aws.greengrass.authorization.exceptions.AuthorizationException: Duplicate policy ID "<id>" for principal "<componentList>"
- com.aws.greengrass.tes.CredentialRequestHandler: Error in retrieving AwsCredentials from TES (HTTP 400)
- com.aws.greengrass.tes.CredentialRequestHandler: Error in retrieving AwsCredentials from TES (HTTP 403)
- com.aws.greengrass.tes.CredentialsProviderError: Could not load credentials from any providers
- Received error when attempting to retrieve ECS metadata: Could not connect to the endpoint URL: "<tokenExchangeServiceEndpoint>"
- copyFrom: <configurationPath> is already a container, not a leaf
- com.aws.greengrass.componentmanager.plugins.docker.exceptions.DockerLoginException: Error logging into the registry using credentials - 'The stub received bad data.'
- java.io.IOException: Cannot run program "cmd" ...: [LogonUser] The password for this account has expired.
- aws.greengrass.StreamManager: Instant exceeds minimum or maximum instant

Warn: '<command>' is not recognized as an internal or external command

Você pode ver esse erro nos registros de um componente do Greengrass quando o software AWS IoT Greengrass principal falha ao executar um comando no script de ciclo de vida do componente. O estado do componente `BROKEN` se torna resultado desse erro. Esse erro pode ocorrer se o usuário do sistema que executa o componente, por exemplo`ggc_user`, não conseguir encontrar o executável do comando nas pastas do PATH.

Em dispositivos Windows, verifique se a pasta que contém o executável está na do `PATH` usuário do sistema que executa o componente. Se estiver ausente do`PATH`, faça o seguinte:

- Adicione a pasta do executável à variável do `PATH` sistema, que está disponível para todos os usuários. Em seguida, reinicie o componente.

  Se você executar o Greengrass nucleus 2.5.0, depois de atualizar a variável do `PATH` sistema, você deverá reiniciar o software AWS IoT Greengrass Core para executar os componentes com a atualização. `PATH` Se o software AWS IoT Greengrass Core não usar a atualização `PATH`

depois de reiniciar o software, reinicie o dispositivo e tente novamente. Para ter mais informações, consulte Execute o software AWS IoT Greengrass Core.

- Adicione a pasta do executável à variável de `PATH` usuário do sistema que executa o componente.

O script Python não registra mensagens

Os principais dispositivos do Greengrass coletam registros que você pode usar para identificar problemas com componentes. Se os scripts `stdout` e `stderr` as mensagens do Python não aparecerem nos registros do componente, talvez seja necessário limpar o buffer ou desativar o buffer para esses fluxos de saída padrão em Python. Faça o seguinte:

- Execute o Python com o argumento -u para desativar o buffer em e. `stdout` `stderr`

  Linux or Unix

  ```
  python3 -u hello_world.py
  ```

  Windows

  ```
  py -3 -u hello_world.py
  ```

- Use Setenv na receita do seu componente para definir a variável de ambiente PYTHONUNBUFFERED como uma string não vazia. Essa variável de ambiente desativa o buffer em e. `stdout` `stderr`
- Limpe o buffer para os fluxos `stdout` ou`stderr`. Execute um destes procedimentos:
  - Limpe uma mensagem ao imprimir.

    ```
    import sys

    print('Hello, error!', file=sys.stderr, flush=True)
    ```

  - Limpe uma mensagem depois de imprimir. Você pode enviar várias mensagens antes de liberar o stream.

    ```
    import sys

    print('Hello, error!', file=sys.stderr)
    sys.stderr.flush()
    ```

Para obter mais informações sobre como verificar se seu script Python gera mensagens de log, consulte. Monitore AWS IoT Greengrass os registros

A configuração do componente não é atualizada ao alterar a configuração padrão

Quando você altera o `DefaultConfiguration` na receita de um componente, a nova configuração padrão não substituirá a configuração existente do componente durante uma implantação. Para aplicar a nova configuração padrão, você deve redefinir a configuração do componente para as configurações padrão. Ao implantar o componente, especifique uma única string vazia como atualização de redefinição.

Console

Redefinir caminhos

```
[""]
```

AWS CLI

O comando a seguir cria uma implantação em um dispositivo principal.

```
aws greengrassv2 create-deployment --cli-input-json file://reset-configuration-
deployment.json
```

O `reset-configuration-deployment.json` arquivo contém o seguinte documento JSON.

```
{
  "targetArn": "arn:aws:iot:us-west-2:123456789012:thing/MyGreengrassCore",
  "deploymentName": "Deployment for MyGreengrassCore",
  "components": {
    "com.example.HelloWorld": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {,
        "reset": [""]
      }
    }
  }
}
```

Greengrass CLI

O comando CLI do Greengrass a seguir cria uma implantação local em um dispositivo principal.

```
sudo greengrass-cli deployment create \
  --recipeDir recipes \
  --artifactDir artifacts \
  --merge "com.example.HelloWorld=1.0.0" \
  --update-config reset-configuration-deployment.json
```

O `reset-configuration-deployment.json` arquivo contém o seguinte documento JSON.

```
{
  "com.example.HelloWorld": {
    "RESET": [""]
  }
}
```

awsiot.greengrasscoreipc.model.UnauthorizedError

Você pode ver esse erro nos registros de um componente do Greengrass quando o componente não tem permissão para realizar uma operação de IPC em um recurso. Para conceder permissão a um componente para chamar uma operação de IPC, defina uma política de autorização de IPC na configuração do componente. Para ter mais informações, consulte Autorize componentes a realizar operações de IPC.

> **Tip**
>
> Se você alterar o `DefaultConfiguration` na receita de um componente, deverá redefinir a configuração do componente para sua nova configuração padrão. Ao implantar o componente, especifique uma única string vazia como atualização de redefinição. Para ter mais informações, consulte A configuração do componente não é atualizada ao alterar a configuração padrão.

com.aws.greengrass.authorization.exceptions.AuthorizationException: Duplicate policy ID "<id>" for principal "<componentList>"

Você pode ver esse erro se várias políticas de autorização de IPC, inclusive em todos os componentes do dispositivo principal, usarem a mesma ID de política.

Verifique as políticas de autorização de IPC de seus componentes, corrija quaisquer duplicatas e tente novamente. Para criar IDs de política exclusivos, recomendamos combinar o nome do componente, o nome do serviço IPC e um contador. Para ter mais informações, consulte Autorize componentes a realizar operações de IPC.

> **Tip**
>
> Se você alterar o `DefaultConfiguration` na receita de um componente, deverá redefinir a configuração do componente para sua nova configuração padrão. Ao implantar o componente, especifique uma única string vazia como atualização de redefinição. Para ter mais informações, consulte A configuração do componente não é atualizada ao alterar a configuração padrão.

com.aws.greengrass.tes.CredentialRequestHandler: Error in retrieving AwsCredentials from TES (HTTP 400)

Você pode ver esse erro quando um dispositivo principal não consegue obter AWS credenciais do serviço de troca de tokens. O código de status HTTP 400 indica que esse erro ocorreu porque a função IAM de troca de tokens do dispositivo principal não existe ou não tem uma relação de confiança que permita que o provedor de AWS IoT credenciais a assuma.

Faça o seguinte:

1. Identifique a função de troca de tokens que o dispositivo principal usa. A mensagem de erro inclui o alias da AWS IoT função do dispositivo principal, que aponta para a função de troca de tokens. Execute o comando a seguir em seu computador de desenvolvimento e *`MyGreengrassCoreTokenExchangeRoleAlias`*substitua pelo nome do alias da AWS IoT função na mensagem de erro.

   ```
   aws iot describe-role-alias --role-alias MyGreengrassCoreTokenExchangeRoleAlias
   ```

A resposta inclui o Amazon Resource Name (ARN) da função IAM de troca de tokens.

```
{
  "roleAliasDescription": {
    "roleAlias": "MyGreengrassCoreTokenExchangeRoleAlias",
    "roleAliasArn": "arn:aws:iot:us-west-2:123456789012:rolealias/
MyGreengrassCoreTokenExchangeRoleAlias",
    "roleArn": "arn:aws:iam::123456789012:role/MyGreengrassV2TokenExchangeRole",
    "owner": "123456789012",
    "credentialDurationSeconds": 3600,
    "creationDate": "2021-02-05T16:46:18.042000-08:00",
    "lastModifiedDate": "2021-02-05T16:46:18.042000-08:00"
  }
}
```

2. Verifique se a função existe. Execute o comando a seguir e substitua *MyGreengrassV2TokenExchangeRole* pelo nome da função de troca de tokens.

```
aws iam get-role --role-name MyGreengrassV2TokenExchangeRole
```

Se o comando retornar um `NoSuchEntity` erro, a função não existe e você deve criá-la. Para obter mais informações sobre como criar e configurar essa função, consulteAutorize os dispositivos principais a interagir com os serviços AWS.

3. Verifique se a função tem uma relação de confiança que permita que o provedor de AWS IoT credenciais a assuma. A resposta da etapa anterior contém um`AssumeRolePolicyDocument`, que define as relações de confiança da função. A função deve definir uma relação de confiança que `credentials.iot.amazonaws.com` permita assumi-la. Esse documento deve ser semelhante ao exemplo a seguir.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "credentials.iot.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
```

```
}
```

Se as relações de confiança da função não `credentials.iot.amazonaws.com` permitirem assumi-la, você deverá adicionar essa relação de confiança à função. Para obter mais informações, consulte Modificar um perfil no Guia do usuário do AWS Identity and Access Management .

com.aws.greengrass.tes.CredentialRequestHandler: Error in retrieving AwsCredentials from TES (HTTP 403)

Você pode ver esse erro quando um dispositivo principal não consegue obter AWS credenciais do serviço de troca de tokens. O código de status HTTP 403 indica que esse erro ocorreu porque as AWS IoT políticas do dispositivo principal não concedem a `iot:AssumeRoleWithCertificate` permissão para o alias de AWS IoT função do dispositivo principal.

Analise as AWS IoT políticas do dispositivo principal e adicione a `iot:AssumeRoleWithCertificate` permissão para o alias de AWS IoT função do dispositivo principal. A mensagem de erro inclui o alias da AWS IoT função atual do dispositivo principal. Para obter mais informações sobre essa permissão e como atualizar as AWS IoT políticas do dispositivo principal, consulte AWS IoTPolítica mínima para dispositivos AWS IoT Greengrass V2 principais Atualizar a AWS IoT política de um dispositivo principal e.

com.aws.greengrass.tes.CredentialsProviderError: Could not load credentials from any providers

Você pode ver esse erro quando o componente tenta solicitar AWS credenciais e não consegue se conectar ao serviço de troca de tokens.

Faça o seguinte:

- Verifique se o componente declara uma dependência do componente do serviço de troca de tokens,. `aws.greengrass.TokenExchangeService` Caso contrário, adicione a dependência e reimplante o componente.
- Se o componente for executado no docker, certifique-se de aplicar as configurações de rede e as variáveis de ambiente corretas, de acordo com. Use AWS credenciais em componentes de contêiner do Docker (Linux)

- Se o componente estiver escrito em NodeJS, defina dns. setDefaultResultFaça o pedido para**`ipv4first`**.
- Verifique se `/etc/hosts` há uma entrada que comece com `::1` e contenha`localhost`. Remova a entrada para ver se ela fez com que o componente se conectasse ao serviço de troca de tokens no endereço errado.

Received error when attempting to retrieve ECS metadata: Could not connect to the endpoint URL: "<tokenExchangeServiceEndpoint>"

Você pode ver esse erro quando o componente não executa o serviço de troca de tokens e um componente tenta solicitar AWS credenciais.

Faça o seguinte:

- Verifique se o componente declara uma dependência do componente do serviço de troca de tokens,. `aws.greengrass.TokenExchangeService` Caso contrário, adicione a dependência e reimplante o componente.
- Verifique se o componente usa AWS credenciais em seu ciclo de `install` vida. AWS IoT Greengrass não garante a disponibilidade do serviço de troca de tokens durante o `install` ciclo de vida. Atualize o componente para mover o código que usa AWS credenciais para o `run` ciclo de vida `startup` ou e, em seguida, reimplante o componente.

copyFrom: <configurationPath> is already a container, not a leaf

Você pode ver esse erro ao alterar um valor de configuração de um tipo de contêiner (uma lista ou objeto) para um tipo que não seja de contêiner (uma string, número ou booleano). Faça o seguinte:

1. Verifique a receita do componente para ver se sua configuração padrão define esse valor de configuração como uma lista ou um objeto. Em caso afirmativo, remova ou altere esse valor de configuração.
2. Crie uma implantação para redefinir esse valor de configuração para o valor padrão. Para obter mais informações, consulte Criar implantações e Atualizar configurações de componentes.

Em seguida, você pode definir esse valor de configuração como uma string, número ou booleano.

com.aws.greengrass.componentmanager.plugins.docker.exceptions.DockerLoginException: Error logging into the registry using credentials - 'The stub received bad data.'

Você pode ver esse erro nos registros do núcleo do Greengrass quando o componente gerenciador de aplicativos Docker tenta baixar uma imagem do Docker de um repositório privado no Amazon Elastic Container Registry (Amazon ECR). Esse erro ocorre se você usar o auxiliar de credenciais do `wincred` Docker (). `docker-credential-wincred` Como resultado, o Amazon ECR não consegue armazenar as credenciais de login.

Execute uma das seguintes ações:

- Se você não usa o auxiliar de credenciais do `wincred` Docker, remova o `docker-credential-wincred` programa do dispositivo principal.
- Se você usar o auxiliar de credenciais do `wincred` Docker, faça o seguinte:
  1. Renomeie o `docker-credential-wincred` programa no dispositivo principal. `wincred`Substitua por um novo nome para o auxiliar de credenciais do Windows Docker. Por exemplo, você pode renomeá-lo para`docker-credential-wincredreal`.
  2. Atualize a `credsStore` opção no arquivo de configuração do Docker (`.docker/config.json`) para usar o novo nome para o auxiliar de credenciais do Docker do Windows. Por exemplo, se você renomeou o programa para`docker-credential-wincredreal`, atualize a `credsStore` opção para`wincredreal`.

     ```
     {
       "credsStore": "wincredreal"
     }
     ```

java.io.IOException: Cannot run program "cmd" ...: [LogonUser] The password for this account has expired.

Você pode ver esse erro em um dispositivo principal do Windows quando o usuário do sistema que executa os processos do componente, como`ggc_user`, tem uma senha expirada. Como resultado, o software AWS IoT Greengrass Core não consegue executar processos de componentes como esse usuário do sistema.

Para atualizar a senha de um usuário do sistema Greengrass

1. Execute o comando a seguir como administrador para definir a senha do usuário. Substitua *ggc_user pelo usuário* do sistema e substitua a *senha pela senha* a ser definida.

   ```
   net user ggc_user password
   ```

2. Use o PsExec utilitário para armazenar a nova senha do usuário na instância do Credential Manager da LocalSystem conta. Substitua a *senha* pela senha do usuário que você definiu.

   ```
   psexec -s cmd /c cmdkey /generic:ggc_user /user:ggc_user /pass:password
   ```

> **Tip**
>
> Dependendo da configuração do Windows, a senha do usuário pode ser definida para expirar em uma data futura. Para garantir que seus aplicativos Greengrass continuem operando, monitore quando a senha expira e atualize-a antes que ela expire. Você também pode definir a senha do usuário para nunca expirar.
>
> - Para verificar quando um usuário e sua senha expiram, execute o comando a seguir.
>
>   ```
>   net user ggc_user | findstr /C:expires
>   ```
>
> - Para definir que a senha de um usuário nunca expire, execute o comando a seguir.
>
>   ```
>   wmic UserAccount where "Name='ggc_user'" set PasswordExpires=False
>   ```
>
> - Se você estiver usando o Windows 10 ou posterior, onde o `wmic`comando está obsoleto, execute o comando a seguir. PowerShell
>
>   ```
>   Get-CimInstance -Query "SELECT * from Win32_UserAccount WHERE name =
>    'ggc_user'" | Set-CimInstance -Property @{PasswordExpires="False"}
>   ```

aws.greengrass.StreamManager: Instant exceeds minimum or maximum instant

Ao atualizar o stream manager v2.0.7 para uma versão entre v2.0.8 e v2.0.11, você pode ver o seguinte erro nos registros do componente do gerenciador de stream se o componente falhar ao iniciar.

```
2021-07-16T00:54:58.568Z [INFO] (Copier) aws.greengrass.StreamManager:
 stdout. Caused by: com.fasterxml.jackson.databind.JsonMappingException:
 Instant exceeds minimum or maximum instant (through reference chain:
 com.amazonaws.iot.greengrass.streammanager.export.PersistedSuccessExportStatesV1["lastExportTi
 {scriptName=services.aws.greengrass.StreamManager.lifecycle.startup.script,
 serviceName=aws.greengrass.StreamManager, currentState=STARTING}
2021-07-16T00:54:58.579Z [INFO] (Copier) aws.greengrass.StreamManager: stdout.
 Caused by: java.time.DateTimeException: Instant exceeds minimum or maximum instant.
 {scriptName=services.aws.greengrass.StreamManager.lifecycle.startup.script,
 serviceName=aws.greengrass.StreamManager, currentState=STARTING}
```

Se você implantou o stream manager v2.0.7 e deseja atualizar para uma versão posterior, você deve atualizar diretamente para o stream manager v2.0.12. Para obter mais informações sobre o componente do gerenciador de fluxo, consulteGerenciador de fluxo.

Problemas com o componente da função Lambda do dispositivo principal

Solucione problemas de componentes da função Lambda em dispositivos principais.

Tópicos

- The following cgroup subsystems are not mounted: devices, memory
- ipc_client.py:64,HTTP Error 400:Bad Request, b'No subscription exists for the source <label-or-lambda-arn> and subject <label-or-lambda-arn>

The following cgroup subsystems are not mounted: devices, memory

Você pode ver esse erro ao executar uma função Lambda em contêiner nos seguintes casos:

- O dispositivo principal não tem o cgroup v1 habilitado para a memória ou os cgroups do dispositivo.

- O dispositivo principal tem cgroups v2 habilitado. As funções Lambda do Greengrass exigem cgroups v1, e cgroups v1 e v2 são mutuamente exclusivos.

Para habilitar o cgroups v1, inicialize o dispositivo com os seguintes parâmetros do kernel Linux.

```
cgroup_enable=memory cgroup_memory=1 systemd.unified_cgroup_hierarchy=0
```

> **Tip**
>
> Em um Raspberry Pi, edite o `/boot/cmdline.txt` arquivo para definir os parâmetros do kernel do dispositivo.

ipc_client.py:64,HTTP Error 400:Bad Request, b'No subscription exists for the source <label-or-lambda-arn> and subject <label-or-lambda-arn>

Você pode ver esse erro ao executar uma função Lambda V1, que usa AWS IoT Greengrass o SDK principal, em um dispositivo V2 core sem especificar uma assinatura no componente antigo do roteador de assinatura. Para corrigir esse problema, implante e configure o roteador de assinatura antigo para especificar as assinaturas necessárias. Para ter mais informações, consulte Importar funções V1 Lambda.

Versão do componente descontinuada

Você pode ver uma notificação em seu Personal Health Dashboard (PHD) quando uma versão do componente em seu dispositivo principal for descontinuada. A versão do componente envia essa notificação ao seu PHD dentro de 60 minutos após ser descontinuada.

Para ver quais implantações você precisa revisar, faça o seguinte usando o: AWS Command Line Interface

1. Execute o comando a seguir para obter uma lista dos seus dispositivos principais.

   ```
   aws greengrassv2 list-core-devices
   ```

2. Execute o comando a seguir para recuperar o status dos componentes em cada dispositivo principal da Etapa 1. *coreDeviceName*Substitua pelo nome de cada dispositivo principal a ser consultado.

```
aws greengrassv2 list-installed-components --core-device-thing-name coreDeviceName
```

3. Reúna os dispositivos principais com a versão descontinuada do componente instalada nas etapas anteriores.
4. Execute o comando a seguir para recuperar o status de todos os trabalhos de implantação de cada dispositivo principal da Etapa 3. *coreDeviceName*Substitua pelo nome do dispositivo principal a ser consultado.

   ```
   aws greengrassv2 list-effective-deployments --core-device-thing-name coreDeviceName
   ```

   A resposta contém a lista de trabalhos de implantação para o dispositivo principal. Você pode revisar a implantação para escolher outra versão do componente. Para obter mais informações sobre como revisar uma implantação, consulte Revisar implantações.

Problemas na interface de linha de comando do Greengrass

Solucione problemas com a CLI do Greengrass.

Tópicos

- java.lang.RuntimeException: Unable to create ipc client

java.lang.RuntimeException: Unable to create ipc client

Você pode ver esse erro ao executar um comando da CLI do Greengrass e especificar uma pasta raiz diferente daquela em que o software AWS IoT Greengrass Core está instalado.

Siga um destes procedimentos para definir o caminho raiz e */greengrass/v2* substituí-lo pelo caminho para a instalação AWS IoT Greengrass do software Core:

- Defina a variável de ambiente `GGC_ROOT_PATH` como */greengrass/v2*.
- Adicione o `--ggcRootPath` */greengrass/v2* argumento ao seu comando conforme mostrado no exemplo a seguir.

  ```
  greengrass-cli --ggcRootPath /greengrass/v2 <command> <subcommand> [arguments]
  ```

AWS Command Line Interface problemas

Solucione AWS CLI problemas para AWS IoT Greengrass V2.

Tópicos

Error: Invalid choice: 'greengrassv2'

Você pode ver esse erro ao executar um AWS IoT Greengrass V2 comando com o AWS CLI (por exemplo,`aws greengrassv2 list-core-devices`).

Esse erro indica que você tem uma versão do AWS CLI que não é compatível AWS IoT Greengrass V2. Para usar AWS IoT Greengrass V2 com o AWS CLI, você deve ter uma das seguintes versões ou posteriores:

- Versão AWS CLI V1 mínima: v1.18.197
- Versão AWS CLI V2 mínima: v2.1.11

> **Tip**
>
> Você pode executar o comando a seguir para verificar a versão do AWS CLI que você tem.
>
> ```
> aws --version
> ```

Para resolver esse problema, atualize o AWS CLI para uma versão posterior que ofereça suporte AWS IoT Greengrass V2. Para obter mais informações, consulte Como instalar, atualizar e desinstalar a AWS CLI no Guia do usuário da AWS Command Line Interface .

Códigos de erro de implantação detalhados

Use os códigos de erro e as soluções nessas seções para ajudar a resolver problemas com a implantação de componentes ao usar o núcleo Greengrass versão 2.8.0 ou posterior.

O núcleo do Greengrass relata os erros de implantação como uma hierarquia do código menos específico ao mais específico disponível. Você pode usar essa hierarquia para ajudar a identificar o motivo de um erro de implantação. Por exemplo, a seguir está uma possível hierarquia de erros:

- FALHA_DE_IMPLANTAÇÃO
  - ERRO DE DOWNLOAD DO ARTEFATO
    - IO_ERROR
      - ESPAÇO EM DISCO CRÍTICO

Os códigos de erro são organizados em tipos. Cada tipo representa uma classe de erros que podem ocorrer.AWS IoT Greengrassrelata esses tipos de erros no console, na API eAWS CLI. Pode haver mais de um tipo de erro, dependendo dos erros relatados na hierarquia de erros. Para o exemplo anterior, o tipo de erro retornado é`DEVICE_ERROR`.

Os tipos são:

- ERRO_DE_PERMISSÃO— O acesso a uma operação que requer permissão foi negado.
- ERRO_PEDIDO— Ocorreu um erro devido a um problema no documento de implantação.
- ERRO_RECEITA_COMPONENTE— Ocorreu um erro devido a um problema na receita de um componente.
- AWS_COMPONENT_ERROR— Ocorreu um erro ao iniciar ou remover umAWScomponente fornecido.
- ERRO DO COMPONENTE DO USUÁRIO— Ocorreu um erro ao iniciar ou remover um componente do usuário.
- ERRO_COMPONENTE— Ocorreu um erro ao iniciar ou remover um componente, mas o núcleo do Greengrass não conseguiu determinar se o componente é umAWScomponente fornecido ou componente do usuário.
- ERRO_DE_DISPOSITIVO— Ocorreu um erro com a E/S local ou ocorreu outro erro do dispositivo.
- ERRO_DE_DEPENDÊNCIA— Uma implantação falhou ao baixar um artefato do Amazon S3 ou ao extrair uma imagem de um registro de ECR.
- HTTP_ERROR— Ocorreu um erro com uma solicitação HTTP.
- ERRO_DE_REDE— Ocorreu um erro com a rede do dispositivo.
- ERRO_NÚCLEO— O núcleo Greengrass não conseguiu localizar um componente ou não conseguiu encontrar a versão do núcleo ativo.

- ERRO_SERVIDOR— Um servidor retornou um erro 500 em resposta a uma solicitação.
- ERRO_SERVIÇO_NUVEM— Ocorreu um erro com oAWS IoT Greengrassserviço em nuvem.
- ERRO_DESCONHECIDO— Uma exceção não verificada foi lançada pelo componente.

Muitos dos erros nesta seção relatam informações adicionais naAWS IoT GreengrassRegistros principais. Esses registros são armazenados no sistema de arquivos local do dispositivo principal. Existem registros para oAWS IoT GreengrassSoftware principal e para cada componente individual. Para obter informações sobre como acessar os registros, consulteAcesse os registros do sistema de arquivos.

Erro de permissão

ACESSO_NEGADO

Você pode receber esse erro quando umAWSa operação de serviço retorna um erro 403 porque as permissões não estão configuradas corretamente. Verifique o código de erro mais específico para obter detalhes.

GET_DEPLOYMENT_CONFIGURATION_ACCESS_DENIED

Você pode receber esse erro quando oAWS IoTa política não permite permissão para chamar o`GetDeploymentConfiguration`operação. Adicione o`greengrass::GetDeploymentConfiguration`permissão para a política do dispositivo principal.

GET_COMPONENT_VERSION_ARTIFACT_ACCESS_DENIED

Você pode receber esse erro quando o dispositivo principalAWS IoTa política não permite o`greengrass:GetComponentVersionArtifact`permissão. Adicione a permissão à política do dispositivo principal.

RESOLVE_COMPONENT_CANDIDATES_ACES_NEGADO

Você pode receber esse erro quando o dispositivo principalAWS IoTa política não permite o`greengrass:ResolveComponentCandidates`permissão. Adicione a permissão à política do dispositivo principal.

GET_ECR_CREDENTIAL_ERROR

Você pode receber esse erro quando a implantação não conseguiu se autenticar com um registro privado no ECR. Verifique se há um erro específico no log e tente a implantação novamente.

USUÁRIO_NÃO_AUTORIZADO_PARA_DOCKER

Você pode receber esse erro quando o usuário do Greengrass não está autorizado a usar o Docker. Verifique se você está executando o Greengrass como root ou se o usuário foi adicionado ao`docker`grupo. Em seguida, tente a implantação novamente.

S3_ACESSO_NEGADO

Você pode receber esse erro quando uma operação do Amazon S3 retorna um erro 403. Verifique todos os códigos de erro ou registros adicionais para obter detalhes.

S3_HEAD_OBJECT_ACCESS_DENIED

Você também pode receber esse erro quando a função de troca de tokens do dispositivo não permite aAWS IoT GreengrassSoftware principal para baixar o artefato do componente do URL do objeto do S3 que você especifica na receita do componente ou que o artefato do componente não está disponível. Verifique se a função de troca de tokens permites`3:GetObject`para o URL do objeto do S3 em que o artefato está disponível e em que o artefato está presente.

S3_GET_BUCKET_LOCATION_ACCESS_DENIED

Você pode receber esse erro quando a função de troca de tokens do dispositivo não permite as`3:GetBucketLocation`permissão para o bucket do Amazon S3 em que o artefato está disponível. Verifique se o dispositivo permite a permissão e tente a implantação novamente.

S3_GET_OBJECT_ACCESS_DENIED

Você também pode receber esse erro quando a função de troca de tokens do dispositivo não permite aAWS IoT GreengrassSoftware principal para baixar o artefato do componente do URL do objeto do S3 que você especifica na receita do componente ou que o artefato do componente não está disponível. Verifique se a função de troca de tokens permites`3:GetObject`para o URL do objeto do S3 em que o artefato está disponível e em que o artefato está presente.

Erro de solicitação

NUCLEUS_AUSENTES_CAPACIDADES_REQUERIDAS

Você pode receber esse erro quando a versão do núcleo na implantação não é capaz de realizar a operação solicitada, como baixar uma configuração grande ou definir limites de recursos do Linux. Tente novamente a implantação com uma versão do núcleo que suporte a operação.

ERRO_MÚLTIPLO NUCLEUS_RESOLVIDO

Você pode receber esse erro quando uma implantação tenta implantar vários componentes do núcleo. Verifique o registro para ver o que causou o erro e, em seguida, verifique a página de atualização do software do núcleo para ver se o problema foi corrigido em uma versão posterior do núcleo, ou entre em contato comAWS Support.

ERRO_CIRCULAR_DEPENDÊNCIA_COMPONENTE

Você pode receber esse erro quando dois componentes em sua implantação dependem um do outro. Revise a configuração do componente para que os componentes em sua implantação não dependam uns dos outros.

ATUALIZAÇÃO_NUCLEUS_MINOR_VERSION_NÃO AUTORIZADA

Você pode receber esse erro quando um componente em sua implantação exige uma atualização da versão secundária do nucleus, mas essa versão não está especificada na implantação. Isso ajuda a reduzir atualizações acidentais de versões secundárias para componentes que dependem de uma versão diferente. Inclua a nova versão do núcleo secundário na implantação.

AUSENTE_DOCKER_APPLICATION_MANAGER

Você pode receber esse erro ao implantar um componente do Docker sem implantar o gerenciador de aplicativos do Docker. Certifique-se de que sua implantação inclua o gerenciador de aplicativos Docker.

SERVIÇO_DE_TROCA DE TOKENS AUSENTES

Você pode receber esse erro quando a implantação quiser baixar um artefato de imagem do Docker de um registro ECR privado sem implantar o serviço de troca de tokens. Certifique-se de que sua implantação inclua o serviço de troca de tokens.

REQUISITOS DE VERSÃO DO COMPONENTE NÃO ATENDIDOS

Você pode receber esse erro quando há um conflito de restrição de versão ou quando uma versão do componente não existe. Para obter mais informações, consulte Error: com.aws.greengrass.componentmanager.exceptions.NoAvailableComponentVersionException: Failed to negotiate component <name> version with cloud and no local applicable version satisfying requirement <requirements>.

ERRO DE LIMITAÇÃO

Você pode receber esse erro quando umAWSa operação de serviço excedeu uma cota tarifária. Repita a implantação.

SOLICITAÇÃO_CONFLITANTE

Você pode receber esse erro quando umAWSA operação de serviço retorna um erro 409 porque sua implantação está tentando realizar mais de uma operação por vez. Repita a implantação.

RECURSO_NÃO_ENCONTRADO

Você pode receber esse erro quando umAWSa operação de serviço retorna um erro 404 porque não foi possível encontrar um recurso. Verifique o registro em busca do recurso ausente.

EXECUTE_WITH_CONFIG_NOT_VALID

Você pode receber esse erro quando o`posixUser`,`posixGroup`, ou`windowsUser`as informações especificadas para executar o componente não são válidas. Verifique se o usuário é válido e tente a implantação novamente.

REGIÃO_SEM SUPORTE

Você pode receber esse erro quando a região especificada para a implantação não é suportada peloAWS IoT Greengrass. Verifique a região e tente a implantação novamente.

IOT_CRED_ENDPOINT_NÃO_VÁLIDO

Você pode receber esse erro quando oAWS IoTo endpoint de credencial especificado na configuração não é válido. Verifique o endpoint e tente fazer sua solicitação novamente.

IOT_DATA_ENDPOINT_NÃO_VÁLIDO

Você pode receber esse erro quando oAWS IoTo endpoint de dados especificado na configuração não é válido. Verifique o endpoint e tente fazer sua solicitação novamente.

S3_HEAD_OBJECT_RESOURCE_NOT_FOUND

Você pode receber esse erro quando o artefato do componente não está disponível na URL do objeto do S3 que você especifica na receita do componente. Verifique se você carregou o artefato no bucket do S3 e se o URI do artefato corresponde ao URL do objeto do S3 do artefato no bucket.

S3_GET_BUCKET_LOCATION_RESOURCE_NOT_FOUND

Você pode receber esse erro quando o bucket do Amazon S3 não for encontrado. Verifique se o bucket existe e tente a implantação novamente.

S3_GET_OBJECT_RESOURCE_NOT_FOUND

Você pode receber esse erro quando o artefato do componente não está disponível na URL do objeto do S3 que você especifica na receita do componente. Verifique se você carregou o

artefato no bucket do S3 e se o URI do artefato corresponde ao URL do objeto do S3 do artefato no bucket.

IO_MAPPING_ERROR

Você pode receber esse erro quando ocorre um erro de E/S ao analisar o documento ou a receita de implantação. Verifique todos os códigos de erro ou registros adicionais para obter detalhes.

Erro na receita do componente

ERRO DE ANÁLISE DE RECEITA

Você pode receber esse erro quando a receita de implantação não pôde ser analisada porque há um erro na estrutura da receita. Verifique se a receita está formatada corretamente e tente a implantação novamente.

RECEITA_METADATA_PARSE_ERROR

Você pode receber esse erro quando os metadados da receita de implantação baixados da nuvem não puderam ser analisados. Entre em contato com a AWS Support.

ARTIFACT_URI_NÃO_VÁLIDO

Você pode receber esse erro quando um URI de artefato em uma receita não está formatado corretamente. Verifique no log o URI que não é válido, atualize o URI na receita e tente a implantação novamente.

S3_ARTIFACT_URI_NÃO_VÁLIDO

Você pode receber esse erro quando o URI do Amazon S3 de um artefato em uma receita não é válido. Verifique no log o URI que não é válido, atualize o URI na receita e tente a implantação novamente.

DOCKER_ARTIFACT_URI_NÃO_VÁLIDO

Você pode receber esse erro quando o URI do Docker de um artefato em uma receita não é válido. Verifique no log o URI que não é válido, atualize o URI na receita e tente a implantação novamente.

_ARTIFACT_URI VAZIO

Você pode receber esse erro quando o URI de um artefato não for especificado em uma receita. Verifique no log o artefato que não tem um URI, atualize o URI na receita e tente a implantação novamente.

ESQUEMA DE ARTIFATO_VAZIO

Você pode receber esse erro quando um esquema de URI não está definido para um artefato. Verifique no log o URI que não é válido, atualize o URI na receita e tente a implantação novamente.

ESQUEMA DE ARTIFACTO_NÃO SUPORTADO

Você pode receber esse erro quando um esquema de URI não é suportado pela versão do núcleo em execução. Ou um URI não é válido ou você precisa atualizar a versão do núcleo. Se o URI não for válido, verifique no log o URI que não é válido, atualize o URI na receita e tente a implantação novamente.

MANIFESTO FALTANTE DA RECEITA

Você pode receber esse erro quando a seção do manifesto não estiver incluída na receita. Adicione o manifesto à receita e tente a implantação novamente.

RECEITA_FALTA_ARTIFACT_HASH_ALGORITHM

Você pode receber esse erro quando um artefato que não é local é especificado em uma receita sem um algoritmo de hash. Adicione o algoritmo ao artefato e tente fazer a solicitação novamente.

ARTIFACT_CHECKSUM_MISMATCH

Você pode receber esse erro quando um artefato baixado tem um resumo diferente do especificado na receita. Certifique-se de que a receita contenha o resumo correto e tente a implantação novamente. Para ter mais informações, consulte Error: com.aws.greengrass.componentmanager.exceptions.ArtifactChecksumMismatchException: Integrity check for downloaded artifact failed. Probably due to file corruption..

DEPENDÊNCIA_COMPONENTE NÃO_VÁLIDA

Você pode receber esse erro quando o tipo de dependência especificado em uma receita de implantação não é válido. Verifique a receita e tente fazer sua solicitação novamente.

CONFIG_INTERPOLATE_ERROR

Você pode receber esse erro ao interpolar uma variável de receita. Verifique o registro para obter detalhes.

IO_MAPPING_ERROR

Você pode receber esse erro quando ocorre um erro de E/S ao analisar o documento ou a receita de implantação. Verifique todos os códigos de erro ou registros adicionais para obter detalhes.

AWSerro do componente, erro do componente do usuário, erro do componente

Os códigos de erro a seguir são retornados quando há um problema com um componente. O tipo de erro real relatado depende do componente específico que gerou o erro. Se o núcleo do Greengrass identificar o componente como um fornecido porAWS IoT Greengrass, ele retornaAWS_COMPONENT_ERROR. Se o componente for identificado como um componente do usuário, o núcleo do Greengrass retornaráUSER_COMPONENT_ERROR. Se o núcleo Greengrass não souber, ele retornaCOMPONENT_ERROR.

ERRO DE ATUALIZAÇÃO DO COMPONENTE

Você pode receber esse erro quando um componente não é atualizado durante uma implantação. Verifique quaisquer códigos de erro adicionais ou verifique o registro para ver o que causou o erro.

COMPONENT_QUEBRADO

Você pode receber esse erro quando um componente é quebrado durante uma implantação. Verifique os detalhes do erro no registro do componente e tente a implantação novamente.

REMOVE_COMPONENT_ERROR

Você pode receber esse erro quando o núcleo não consegue remover um componente durante uma implantação. Verifique os detalhes do erro no log e tente a implantação novamente.

TIMEOUT DO COMPONENT_BOOTSTRAP_TIMEOUT

Você pode receber esse erro quando a tarefa de bootstrap de um componente demorou mais do que o tempo limite configurado. Aumente o tempo limite ou reduza o tempo de execução da tarefa de bootstrap e tente a implantação novamente.

ERRO DO COMPONENTE BOOTSTRAP_ERROR

Você pode receber esse erro quando a tarefa de bootstrap de um componente tem um erro. Verifique os detalhes do erro no log e tente a implantação novamente.

CONFIGURAÇÃO_COMPONENTE NÃO_VÁLIDA

Você pode receber esse erro quando o núcleo não consegue validar a configuração implantada para o componente. Verifique os detalhes do erro no log e tente a implantação novamente.

Erro do dispositivo

IO_WRITE_ERROR

Você pode receber esse erro ao gravar em um arquivo. Verifique o registro para obter detalhes.

IO_READ_ERROR

Você pode receber esse erro ao ler de um arquivo. Verifique o registro para obter detalhes.

ESPAÇO EM DISCO CRÍTICO

Você pode receber esse erro quando não há espaço em disco suficiente para concluir uma solicitação de implantação. Você deve ter pelo menos 20 MB de espaço disponível ou o suficiente para armazenar um artefato maior. Libere algum espaço em disco e tente novamente a implantação.

IO_FILE_ATTRIBUTE_ERROR

Você pode receber esse erro quando o tamanho do arquivo existente não puder ser recuperado do sistema de arquivos. Verifique o registro para obter detalhes.

SET_PERMISSION_ERROR

Você pode receber esse erro quando as permissões não puderem ser definidas em um artefato baixado ou diretório de artefatos. Verifique o registro para obter detalhes.

IO_UNZIP_ERROR

Você pode receber esse erro quando um artefato não pode ser descompactado. Verifique o registro para obter detalhes.

RECEITA_LOCAL NÃO ENCONTRADA

Você pode receber esse erro quando a cópia local de um arquivo de receita não foi encontrada. Tente a implantação novamente.

LOCAL_RECIPE_CORROMPIDO

Você pode receber esse erro quando a cópia local da receita for alterada desde o download. Exclua a cópia existente da receita e tente a implantação novamente.

LOCAL_RECIPE_METADATA_NOT_FOUND

Você pode receber esse erro quando a cópia local do arquivo de metadados da receita não foi encontrada. Tente a implantação novamente.

LAUNCH_DIRECTORY_CORROMPIDO

Você pode receber esse erro quando o diretório usado para iniciar o núcleo do Greengrass (`/greengrass/v2/alts/current`) foi modificado desde a última vez em que o núcleo foi iniciado. Reinicie o núcleo e tente novamente a implantação.

HASHING_ALGORITHM_INDISPONÍVEL

Você pode receber esse erro quando a distribuição Java do dispositivo não suporta o algoritmo de hash necessário ou quando o algoritmo de hash especificado em uma receita de componente não é válido.

DEVICE_CONFIG_NOT_VALID_FOR_ARTIFACT_DOWNLOAD

Você pode receber esse erro quando há um erro na configuração do dispositivo que impediu a implantação de baixar o artefato do Amazon S3 ou da nuvem do Greengrass. Verifique se há um erro de configuração específico no log e, em seguida, tente a implantação novamente.

Erro de dependência

DOCKER_ERROR

Você pode receber esse erro ao extrair uma imagem do Docker. Verifique todos os códigos de erro ou registros adicionais para obter detalhes.

DOCKER_SERVICE_INDISPONÍVEL

Você pode receber esse erro quando o Greengrass não conseguiu fazer login no registro do Docker. Verifique se há um erro específico no log e tente a implantação novamente.

DOCKER_LOGIN_ERROR

Você pode receber esse erro quando ocorre um erro inesperado ao fazer login no Docker. Verifique se há um erro específico no log e tente a implantação novamente.

DOCKER_PULL_ERROR

Você pode receber esse erro quando ocorre um erro inesperado ao extrair uma imagem do Docker do registro. Verifique se há um erro específico no log e tente a implantação novamente.

DOCKER_IMAGE_NOT_VALID

Você pode receber esse erro quando a imagem do Docker solicitada não existe. Verifique se há um erro específico no log e tente a implantação novamente.

DOCKER_IMAGE_QUERY_ERROR

Você pode receber esse erro quando ocorre uma falha inesperada ao consultar o Docker para ver as imagens disponíveis. Verifique o erro específico no log e tente a implantação novamente.

S3_ERROR

Você pode receber esse erro ao baixar um artefato do Amazon S3. Verifique todos os códigos de erro ou registros adicionais para obter detalhes.

S3_RESOURCE_NÃO_ENCONTRADO

Você pode receber esse erro quando uma operação do Amazon S3 retorna um erro 404. Verifique todos os códigos de erro ou registros adicionais para obter detalhes.

S3_BAD_REQUEST

Você pode receber esse erro quando uma operação do Amazon S3 retorna um erro 400. Verifique se há um erro específico no log e tente fazer a solicitação novamente.

Erro HTTP

HTTP_REQUEST_ERROR

Você pode receber esse erro quando ocorre um erro ao fazer uma solicitação HTTP. Verifique o registro para ver o erro específico.

DOWNLOAD_DEPLOYMENT_DOCUMENT_ERROR

Você pode receber esse erro quando ocorreu um erro HTTP ao baixar o documento de implantação. Verifique o log para ver o erro HTTP específico.

GET_GREENGRASS_ARTIFACT_SIZE_ERROR

Você pode receber esse erro quando ocorre um erro HTTP ao obter o tamanho de um artefato de componente público. Verifique o log para ver o erro HTTP específico.

DOWNLOAD_GREENGRASS_ARTIFACT_ERROR

Você pode receber esse erro quando ocorre um erro HTTP ao baixar um artefato de componente público. Verifique o log para ver o erro HTTP específico.

Erro de rede

ERRO_DE_REDE

Você pode receber esse erro quando há um problema de conexão durante uma implantação. Verifique a conexão do dispositivo com a Internet e tente a implantação novamente.

Erro do núcleo

SOLICITAÇÃO_INCORRETA

Você pode receber esse erro quando umAWSa operação na nuvem retorna um erro 400. Verifique o registro para ver qual API causou o erro e, em seguida, verifique a página de atualização do software nucleus para ver se o problema foi corrigido em uma versão posterior do núcleo, ou entre em contato comAWS Support.

VERSÃO_NUCLEUS_NÃO_ENCONTRADA

Você pode receber esse erro quando um dispositivo central não consegue encontrar a versão do núcleo ativo. Verifique o registro para ver o que causou o erro e, em seguida, verifique a página de atualização do software do núcleo para ver se o problema foi corrigido em uma versão posterior do núcleo, ou entre em contato comAWS Support.

FALHA DO REINÍCIO DO NÚCLEO

Você pode receber esse erro quando o núcleo não for reiniciado durante qualquer implantação que exija a reinicialização do núcleo. Verifique o registro do carregador para ver o que causou o erro e, em seguida, verifique a página de atualização do software do núcleo para ver se o problema foi corrigido em uma versão posterior do núcleo, ou entre em contato comAWS Support.

COMPONENT_NÃO_ENCONTRADO INSTALADO

Você pode receber esse erro quando o núcleo não consegue localizar um componente instalado. Verifique o registro para ver o que causou o erro e, em seguida, verifique a página de atualização do software do núcleo para ver se o problema foi corrigido em uma versão posterior do núcleo, ou entre em contato comAWS Support.

DOCUMENTO_DE_IMPLANTAÇÃO NÃO_VÁLIDO

Você pode receber esse erro quando o dispositivo receber um documento de implantação que não é válido. Verifique quaisquer códigos de erro adicionais ou verifique o registro para ver o que causou o erro.

SOLICITAÇÃO_DE_IMPLANTAÇÃO VAZIA

Você pode receber esse erro quando um dispositivo recebe uma solicitação de implantação vazia. Verifique o registro para ver o que causou o erro e, em seguida, verifique a página de atualização do software do núcleo para ver se o problema foi corrigido em uma versão posterior do núcleo, ou entre em contato comAWS Support.

ERRO DE ANÁLISE DO DOCUMENTO_IMPLANTAÇÃO

Você pode receber esse erro quando o formato da solicitação de implantação não corresponder ao formato esperado. Verifique o registro para ver o que causou o erro e, em seguida, verifique a página de atualização do software do núcleo para ver se o problema foi corrigido em uma versão posterior do núcleo, ou entre em contato comAWS Support.

COMPONENT_METADATA_NOT_VALID_IN_DEPLOYMENT

Você pode receber esse erro quando a solicitação de implantação contém metadados de componentes que não são válidos. Verifique o registro para ver o que causou o erro e, em seguida, verifique a página de atualização do software do núcleo para ver se o problema foi corrigido em uma versão posterior do núcleo, ou entre em contato comAWS Support.

LAUNCH_DIRECTORY_CORROMPIDO

Você pode receber esse erro ao mover um dispositivo Greengrass de um grupo para outro e depois voltar para o grupo original com implantações que exigem a reinicialização do Greengrass. Para resolver o erro, recrie o diretório de inicialização do Greengrass no dispositivo.

Para obter mais informações, consulte Error: com.aws.greengrass.deployment.exceptions.DeploymentException: Unable to process deployment. Greengrass launch directory is not set up or Greengrass is not set up as a system service.

Erro do servidor

ERRO_SERVIDOR

Você pode receber esse erro quando umAWSa operação de serviço retorna um erro 500 porque o serviço não pode processar a solicitação no momento. Tente fazer a implantação novamente mais tarde.

ERRO_S3_SERVER_ERROR

Você pode receber esse erro quando uma operação do Amazon S3 retorna um erro 500. Verifique todos os códigos de erro ou registros adicionais para obter detalhes.

Erro no serviço de nuvem

RESOLVE_COMPONENT_CANDIDATES_BAD_RESPONSE

Você pode receber esse erro quando o serviço de nuvem do Greengrass envia uma resposta incompatível para o`ResolveComponentCandidates`operação. Verifique o registro para ver o que causou o erro e, em seguida, verifique a página de atualização do software do núcleo para ver se o problema foi corrigido em uma versão posterior do núcleo, ou entre em contato comAWS Support.

TAMANHO_DOCUMENTO_IMPLANTAÇÃO EXCEDIDO

Você pode receber esse erro quando o documento de implantação solicitado excedeu a cota de tamanho máximo. Reduza o tamanho do documento de implantação e tente a implantação novamente.

TAMANHO_DO_ARTEFATO VERDE NÃO ENCONTRADO

Você pode receber esse erro quando o Greengrass não consegue obter o tamanho de um artefato de componente público. Verifique o registro para ver o que causou o erro e, em seguida, verifique a página de atualização do software do núcleo para ver se o problema foi corrigido em uma versão posterior do núcleo, ou entre em contato comAWS Support.

DOCUMENTO_DE_IMPLANTAÇÃO NÃO_VÁLIDO

Você pode receber esse erro quando o dispositivo receber um documento de implantação que não é válido. Verifique quaisquer códigos de erro adicionais ou verifique o registro para ver o que causou o erro.

SOLICITAÇÃO_DE_IMPLANTAÇÃO VAZIA

Você pode receber esse erro quando um dispositivo recebe uma solicitação de implantação vazia. Verifique o registro para ver o que causou o erro e, em seguida, verifique a página de atualização do software do núcleo para ver se o problema foi corrigido em uma versão posterior do núcleo, ou entre em contato comAWS Support.

ERRO DE ANÁLISE DO DOCUMENTO_IMPLANTAÇÃO

Você pode receber esse erro quando o formato da solicitação de implantação não corresponder ao formato esperado. Verifique o registro para ver o que causou o erro e, em seguida, verifique a página de atualização do software do núcleo para ver se o problema foi corrigido em uma versão posterior do núcleo, ou entre em contato comAWS Support.

COMPONENT_METADATA_NOT_VALID_IN_DEPLOYMENT

Você pode receber esse erro quando a solicitação de implantação contém metadados de componentes que não são válidos. Verifique o registro para ver o que causou o erro e, em seguida, verifique a página de atualização do software do núcleo para ver se o problema foi corrigido em uma versão posterior do núcleo, ou entre em contato comAWS Support.

Erros genéricos

Esses erros genéricos não têm um tipo de erro associado.

IMPLANTAÇÃO_INTERROMPIDA

Você pode receber esse erro quando uma implantação não pode ser concluída devido a um desligamento do núcleo ou outro evento externo. Verifique todos os códigos de erro ou registros adicionais para obter detalhes.

ERRO DE DOWNLOAD DO ARTEFATO

Você pode receber esse erro quando há um problema ao baixar um artefato. Verifique todos os códigos de erro ou registros adicionais para obter detalhes.

VERSÃO_INDISPONÍVEL_COMPONENTE_

Você pode receber esse erro quando uma versão do componente não existe na nuvem ou localmente, ou se houver um conflito de resolução de dependência. Verifique todos os códigos de erro ou registros adicionais para obter detalhes.

ERRO DE CARREGAMENTO DO PACOTE DE COMPONENTES

Você pode receber esse erro ao processar os artefatos baixados. Verifique todos os códigos de erro ou registros adicionais para obter detalhes.

CLOUD_API_ERROR

Você pode receber esse erro quando ocorre um erro ao chamar umAWSAPI de serviço. Verifique todos os códigos de erro ou registros adicionais para obter detalhes.

IO_ERROR

Você pode receber esse erro quando ocorre um erro de E/S durante uma implantação. Verifique todos os códigos de erro ou registros adicionais para obter detalhes.

ERRO DE ATUALIZAÇÃO DO COMPONENTE

Você pode receber esse erro quando um componente não é atualizado durante uma implantação. Verifique quaisquer códigos de erro adicionais ou verifique o registro para ver o que causou o erro.

Erro desconhecido

FALHA_DE_IMPLANTAÇÃO

Você pode receber esse erro quando uma implantação falhar porque uma exceção não verificada foi lançada. Verifique o registro para ver o que causou o erro e, em seguida, verifique a página de atualização do software do núcleo para ver se o problema foi corrigido em uma versão posterior do núcleo, ou entre em contato comAWS Support.

TIPO_DE_IMPLANTAÇÃO NÃO_VÁLIDO

Você pode receber esse erro quando o tipo de implantação não é válido. Verifique o registro para ver o que causou o erro e, em seguida, verifique a página de atualização do software do núcleo para ver se o problema foi corrigido em uma versão posterior do núcleo, ou entre em contato comAWS Support.

Códigos detalhados de status do componente

Use os códigos de status e as soluções nessas seções para ajudar a resolver problemas com componentes ao usar o núcleo Greengrass versão 2.8.0 ou posterior.

Muitos dos status neste tópico relatam informações adicionais nos registros AWS IoT Greengrass principais. Esses registros são armazenados no sistema de arquivos local do dispositivo principal. Há registros para cada componente individual. Para obter informações sobre como acessar os registros, consulteAcesse os registros do sistema de arquivos.

ERRO DE INSTALAÇÃO

Você pode receber isso quando ocorre um erro ao executar um script de instalação. O código de erro é relatado no registro do componente. Verifique se há erros no script de instalação e implante seu componente novamente.

INSTALL_CONFIG_NOT_VALID

Você pode receber esse erro quando a instalação de um componente não pôde ser concluída porque a `Install` seção da receita não é válida. Verifique se há erros na seção de instalação da sua receita e tente a implantação novamente.

INSTALL_IO_ERROR

Isso pode ocorrer quando ocorre um erro de E/S durante a instalação de um componente. Confira o log de log de log para ter detalhes sobre o erro.

INSTALL_MISSING_DEFAULT_RUN COM

Você pode receber esse erro quando não AWS IoT Greengrass consegue determinar o usuário ou grupo a ser usado ao instalar um componente. Verifique se a `runWith` seção da sua receita de instalação inclui um usuário ou grupo válido.

TIMEOUT DE INSTALAÇÃO

Você pode receber esse erro quando o script de instalação não for concluído dentro do período de tempo limite configurado. Aumente o `Timeout` período especificado na `Install` seção da receita ou modifique seu script de instalação para terminar dentro do tempo limite configurado.

ERRO_DE_INICIALIZAÇÃO

Você pode receber isso quando ocorre um erro ao executar um script de inicialização. O código de erro é relatado no registro do componente. Verifique se há erros no script de instalação e implante seu componente novamente.

STARTUP_CONFIG_NÃO_VÁLIDO

Você pode receber esse erro quando a instalação de um componente não pôde ser concluída porque a `Startup` seção da receita não é válida. Verifique se há erros na seção de inicialização da sua receita e tente a implantação novamente.

STARTUP_IO_ERROR

Isso pode ocorrer quando ocorre um erro de E/S durante a inicialização de um componente. Confira o log de log de log para ter detalhes sobre o erro.

STARTUP_MISSING_DEFAULT_RUN COM

Você pode receber esse erro quando não AWS IoT Greengrass consegue determinar o usuário ou grupo a ser usado ao executar um componente. Verifique se a `runWith` seção da sua receita de inicialização inclui um usuário ou grupo válido.

TIMEOUT DE INICIALIZAÇÃO

Você pode receber esse erro quando o script de inicialização não for concluído dentro do período de tempo limite configurado. Aumente o `Timeout` período especificado na `Startup` seção da receita ou modifique seu script de inicialização para terminar dentro do tempo limite configurado.

ERRO_DE_EXECUÇÃO

Você pode receber isso quando ocorre um erro ao executar um script de componente. O código de erro é relatado no registro do componente. Verifique se há erros no script de execução e implante seu componente novamente.

RUN_MISSING_DEFAULT_RUN COM

Você pode receber esse erro quando não AWS IoT Greengrass consegue determinar o usuário ou grupo a ser usado ao executar um componente. Verifique se a `runWith` seção da sua receita de execução inclui um usuário ou grupo válido.

RUN_CONFIG_NÃO_VÁLIDO

Você pode receber esse erro quando um componente não pôde ser executado porque a `Run` seção da receita não é válida. Verifique se há erros na seção de execução de sua receita e tente a implantação novamente.

RUN_IO_ERROR

Isso pode ocorrer quando ocorre um erro de E/S enquanto o componente está em execução. Confira o log de log de log para ter detalhes sobre o erro.

TEMPO LIMITE DE EXECUÇÃO

Você pode receber esse erro quando o script de execução não for concluído dentro do período de tempo limite configurado. Aumente o `Timeout` período especificado na `Run` seção da receita ou modifique seu script de execução para terminar dentro do tempo limite configurado.

ERRO DE DESLIGAMENTO

Você pode receber isso quando ocorre um erro ao encerrar um script de componente. O código de erro é relatado no registro do componente. Verifique se há erros no script de desligamento e implante seu componente novamente.

TIMEOUT DE DESLIGAMENTO

Você pode receber esse erro quando o script de desligamento não for concluído dentro do período de tempo limite configurado. Aumente o `Timeout` período especificado na `Shutdown` seção da receita ou modifique seu script de execução para terminar dentro do tempo limite configurado.

Marcar com tag os recursos do AWS IoT Greengrass Version 2

Com tags, você pode organizar e gerenciar seus recursos no AWS IoT Greengrass. Você pode usar tags para atribuir metadados aos seus recursos e usar tags nas políticas do IAM para definir o acesso condicional aos seus recursos.

> **Note**
>
> No momento, as tags de recurso do Greengrass não são compatíveis com relatórios de alocação de custos ou grupos de faturamento da AWS IoT.

Usar tags no AWS IoT Greengrass V2

Você pode usar tags para categorizar seus recursos do AWS IoT Greengrass por finalidade, proprietário, ambiente ou qualquer outra classificação para seu caso de uso. Quando você tem tags do mesmo tipo; tags tags lhe ajudem a identificar um recurso específico com mais facilidade um recurso específico com base tags tags que tags lhe ajudem a identificar um recurso específico com mais facilidade.

Cada tag consiste em uma chave e um valor opcional, ambos definidos por você. Por exemplo, você pode definir um conjunto de tags para os dispositivos principais que lhe ajudem a rastreá-los pelos clientes que possuem os dispositivos. Recomendamos que você crie um conjunto de chave de tags que atenda às suas necessidades para cada tipo de recurso. Usando um conjunto consistente de chaves de tag, você pode gerenciar os recursos com mais facilidade.

Tag com oAWS Management Console

O Tag Editor (Editor de tags) no AWS Management Console fornece uma maneira central e unificada para criar e gerenciar suas tags para recursos de todos os serviços da AWS. Para obter mais informações, consulte Tag Editor (Editor de tags) no Guia do usuário do AWS Resource Groups.

Tag com aAWS IoT Greengrass V2 API

Você também podeAWS IoT Greengrass V2 usar a tags tags. Antes de criar tags, esteja ciente dessas restrições de marcação. Para obter mais informações, consulte Convenções de nomenclatura e uso de tags na Referência geral da AWS.

- Para adicionar tags ao criar um recurso, defina-as na propriedade `tags` do recurso.
- Para adicionar tags a um recurso existente ou atualizar valores de tags, use a TagResourceoperação.
- Para tags tags de um recurso da, a UntagResourceoperação.
- Para recuperar as tags associadas a um recurso, use a ListTagsForResourceoperação ou descreva o recurso e inspecione sua`tags` propriedade.

A tabela a seguir lista os recursos que você pode marcar usando aAWS IoT Greengrass V2 API e suas`Get` operações`Describe` e/ou correspondentes`Create`.

Recursos do AWS IoT Greengrass V2 que podem ser marcados

Recurso	Criar operação	Descreva ou obtenha a operação
Dispositivo principal	Nenhum. Execute o softwareAWS IoT Greengrass Core em um dispositivo para criar um dispositivo principal.	GetCoreDevice
Componente	CreateComponentVersion	DescribeComponent, GetComponent
Implantação	CreateDeployment	GetDeployment

Use as operações a seguir para visualizar e gerenciar as tags de recursos que são compatíveis com a marcação:

- TagResource— Adiciona tags a um recurso ou atualiza o valor de uma tag existente.
- ListTagsForResource— Lista as tags de um recurso.
- UntagResource— tags tags de um recurso.

Você pode adicionar ou remover tags de um recurso a qualquer momento. Para alterar o valor de uma chave de tag, adicione uma tag ao recurso que defina a mesma chave e o novo valor. O novo valor substitui o valor anterior. Você pode definir um valor a uma string vazia, mas não pode definir o valor como nulo.

Ao excluir um recurso, as tags associadas ao recurso também são excluídas.

Utilização de tags com políticas do IAM

Em suas políticas do IAM tags de recurso para controlar o acesso e as permissões do usuário. Por exemplo, as políticas podem permitir que os usuários criem somente os recursos que tenham uma tag específica. As políticas também podem restringir os usuários de criar ou modificar recursos que tenham determinadas tags.

> **Note**
>
> Se você usar tags para permitir ou negar o acesso de usuários a recursos, negue aos usuários a capacidade de adicionar ou remover essas tags para os mesmos recursos. Caso contrário, um usuário poderá contornar suas restrições e obter acesso a um recurso modificando as tags.

Você pode usar as seguintes chaves e valores de contexto de condição no`Condition` elemento, também chamado de`Condition` bloco, de uma declaração de política.

`greengrassv2:ResourceTag/`*`tag-key`*`: `*`tag-value`*

Permitir ou negar ações em recursos com tags específicas.

`aws:RequestTag/`*`tag-key`*`: `*`tag-value`*

Exija que uma tag específica seja usada, ou não, ao criar ou modificar um recurso marcável.

`aws:TagKeys: [`*`tag-key`*`, ...]`

Exija que um conjunto específico de chaves de tag seja usado, ou não, ao criar ou modificar um recurso marcável.

> **Note**
>
> As chaves e valores do contexto da condição em uma política do IAM se aplicam somente às ações que têm um recurso marcável como parâmetro obrigatório. Por exemplo, você pode definir o acesso condicional baseado em tag para ListCoreDevices.

Para obter mais informações, consulte Controlar o acesso aosAWS recursos usando tags de recursos e a referência de política JSON do IAM no Guia do usuário do IAM.

Criar recursos do AWS IoT Greengrass com AWS CloudFormation

O AWS IoT Greengrass está integrado ao AWS CloudFormation, um serviço que ajuda você a modelar e configurar seus recursos da AWS para que você possa gastar menos tempo criando e gerenciando seus recursos e infraestrutura. Você cria um modelo que descreve todas asAWSrecursos que você deseja (como versões e implantações de componentes) eAWS CloudFormationprovisiona e configura esses recursos para você.

Ao usar o AWS CloudFormation, você poderá reutilizar seu modelo para configurar seus recursos do AWS IoT Greengrass de forma repetida e consistente. Descreva seus recursos uma vez e depois provisione os mesmos recursos repetidamente em várias regiões e Contas da AWS.

AWS IoT GreengrassModelos do AWS CloudFormation e

Para provisionar e configurar recursos para o AWS IoT Greengrass e serviços relacionados, você deve entender os modelos do AWS CloudFormation. Os modelos são arquivos de texto formatados em JSON ou YAML. Esses modelos descrevem os recursos que você deseja provisionar nas suas pilhas do AWS CloudFormation. Se você não estiver familiarizado com JSON ou YAML, poderá usar o AWS CloudFormation Designer para ajudá-lo a começar a usar os modelos do AWS CloudFormation. Para obter mais informações, consulte O que é o Designer? (O que é o AWS CloudFormation Designer) no Manual do usuário do AWS CloudFormation.

AWS IoT Greengrasssuporta a criação de versões de componentes e implantações noAWS CloudFormation. Para obter mais informações, incluindo exemplos de modelos JSON e YAML para versões e implantações doAWS IoT GreengrassReferência donaAWS CloudFormationGuia do usuário do.

ComponentVersion Exemplo de modelo do

A seguir está o modelo YAML para uma versão de um componente simples. A receita JSON contém quebras de linha para facilitar a leitura.

```
Parameters:
  ComponentVersion:
    Type: String
Resources:
```

```
  TestSimpleComponentVersion:
    Type: AWS::GreengrassV2::ComponentVersion
    Properties:
      InlineRecipe: !Sub
        - "{\n
            \"RecipeFormatVersion\": \"2020-01-25\",\n
            \"ComponentName\": \"component1\",\n
            \"ComponentVersion\": \"${ComponentVersion}\",\n
            \"ComponentType\": \"aws.greengrass.generic\",\n
            \"ComponentDescription\": \"This\",\n
            \"ComponentPublisher\": \"You\",\n
            \"Manifests\": [\n
            {\n
              \"Platform\": {\n
              \"os\": \"darwin\"\n
            },\n
            \"Lifecycle\": {},\n
            \"Artifacts\": []\n
           },\n
           {\n
             \"Lifecycle\": {},\n
             \"Artifacts\": []\n
           }\n
          ],\n
          \"Lifecycle\": {\n
          \"install\": {\n
            \"script\": \"yuminstallpython\"\n
          }\n
         }\n
        }"
        - { ComponentVersion: !Ref ComponentVersion }
```

Exemplo de modelo do

A seguir está um arquivo YAML definindo um modelo simples para uma implantação.

```
Parameters:
  ComponentVersion:
    Type: String
  TargetArn:
    Type: String
Resources:
  TestDeployment:
```

```
    Type: AWS::GreengrassV2::Deployment
    Properties:
      Components:
        component1:
          ComponentVersion: !Ref ComponentVersion
      TargetArn: !Ref TargetArn
      DeploymentName: CloudFormationIntegrationTest
      DeploymentPolicies:
        FailureHandlingPolicy: DO_NOTHING
        ComponentUpdatePolicy:
          TimeoutInSeconds: 5000
          Action: SKIP_NOTIFY_COMPONENTS
        ConfigurationValidationPolicy:
          TimeoutInSeconds: 30000
Outputs:
  TestDeploymentArn:
    Value: !Sub
      - arn:${AWS::Partition}:greengrass:${AWS::Region}:${AWS::AccountId}:deployments:
${DeploymentId}
      - DeploymentId: !GetAtt TestDeployment.DeploymentId
```

Saiba mais sobre o AWS CloudFormation

Para saber mais sobre o AWS CloudFormation, consulte os seguintes recursos:

- AWS CloudFormation
- Manual do usuário do AWS CloudFormation
- AWS CloudFormation Referência da API
- Guia do usuário da interface de linha de comando do AWS CloudFormation

Software AWS IoT Greengrass principal de código aberto

O AWS IoT Greengrass Version 2 edge runtime (núcleo) e outros componentes do software AWS IoT Greengrass Core são de código aberto. Isso significa que você pode revisar o código para solucionar problemas de interações com seus aplicativos. Você também pode personalizar e estender o software AWS IoT Greengrass Core para atender às suas necessidades específicas de software e hardware.

Para obter informações sobre os repositórios de código aberto do software AWS IoT Greengrass Core, consulte a organização aws-greengrass em. GitHub Seu uso de software de código aberto é regido pela licença de código aberto no GitHub repositório correspondente.

Seu uso do software e dos componentes AWS IoT Greengrass principais não sujeitos a uma licença de código aberto é regido pela licença de software principal do AWS Greengrass.

Histórico de documentos do AWS IoT Greengrass V2 Developer Guide

A tabela a seguir descreve a documentação desta versão do AWS IoT Greengrass Version 2.

- Versão da API: 2020-11-30

Alteração	Descrição	Data
Lançado o Shadow manager v2.3.8	O Shadow manager v2.3.8 está disponível. Esta versão corrige um problema em que o shadow manager cria uma situação de impasse durante a conexão do cliente MQTT.	5 de junho de 2024
Lançado o Greengrass CLI v2.12.6	O componente Greengrass CLI v2.12.6 está disponível.	24 de maio de 2024
AWS IoT Greengrass Atualização do software Core v2.12.6	Esta versão fornece a versão 2.12.6 do componente nucleus do Greengrass e atualiza os componentes fornecidos. AWS	24 de maio de 2024
AWS IoT Device Tester v4.9.4 com GGV2Q v2.5.4 lançado	A versão 4.9.4 do IDT para AWS IoT Greengrass V2 está disponível. Essa versão inclui o pacote de qualificação AWS IoT Greengrass V2 (GGV2Q) v2.5.4 e é compatível com as versões 2.12.0, 2.11.0, 2.10.0, 2.9.5 do núcleo Greengrass.	3 de maio de 2024
Lançamento do Secure Tunneling v1.0.19	O tunelamento seguro v1.0.19 está disponível. Essa versão atualiza o AWS IoT Device	1º de maio de 2024

	Client subjacente invocado pelo componente da versão 1.8.0 para a versão 1.9.0. O tunelamento seguro v1.0.19 aumenta o limite de túneis simultâneos para 20 túneis em um nível de componente. Essa nova versão também aumenta o tempo limite do AWS IoT Greengrass Core IPC de 3 segundos para 10 segundos.	
Lançado o conector Edge para o componente Kinesis Video Streams v1.0.5	A versão 1.0.5 do conector edge para o component e Kinesis Video Streams está disponível. Esta versão inclui correções de erros e melhorias gerais.	29 de abril de 2024
Lançado o Greengrass CLI v2.12.5	O componente Greengrass CLI v2.12.5 está disponível.	25 de abril de 2024
Lançado o componente de autenticação do dispositivo cliente v2.5.0	O componente de autentica ção do dispositivo cliente v2.5.0 está disponível. Esta versão adiciona suporte a variáveis de política para nomes de coisas. Essa versão também permite recursos de política com curingas.	25 de abril de 2024
AWS IoT Greengrass Atualização do software Core v2.12.5	Esta versão fornece a versão 2.12.5 do componente nucleus do Greengrass e atualiza os componentes fornecidos. AWS	25 de abril de 2024

AWS IoT Device Tester v4.9.3 com GGV2Q v2.5.3 lançado	A versão 4.9.3 do IDT for AWS IoT Greengrass V2 está disponível. Essa versão inclui o pacote de qualificação AWS IoT Greengrass V2 (GGV2Q) v2.5.3 e é compatível com as versões 2.12.0, 2.11.0, 2.10.0, 2.9.5 do núcleo Greengrass.	5 de abril de 2024
Lançado o Greengrass CLI v2.12.4	O componente Greengrass CLI v2.12.4 está disponível.	2 de abril de 2024
AWS IoT Greengrass Atualização do software Core v2.12.4	Esta versão fornece a versão 2.12.4 do componente nucleus do Greengrass e atualiza os componentes fornecidos. AWS	2 de abril de 2024
Lançado o Shadow manager v2.3.7	O Shadow manager v2.3.7 está disponível. Esta versão corrige um problema em que o shadow manager registra periodicamente um `NullPointerException` erro durante a sincronização do shadow manager.	27 de março de 2024
Lançado o broker Moquette MQTT 3.1.1 v2.3.6	O componente de corretor Moquette MQTT 3.1.1 v2.3.6 está disponível. Esta versão inclui correções de erros e melhorias gerais.	27 de março de 2024
Lançado o console de depuração local v2.4.2	O componente v2.4.2 do console de depuração local está disponível. Esta versão inclui correções de erros e melhorias gerais.	27 de março de 2024

Lançado o Lambda manager v2.3.3	O componente Lambda Manager v2.3.3 está disponível. Esta versão inclui correções de erros e melhorias gerais.	27 de março de 2024
Lançado o detector IP v2.1.9	O componente detector de IP v2.1.9 está disponível. Esta versão ajusta a etapa de aquisição de IP para enviar registros somente no nível do registro de depuração.	27 de março de 2024
AWS IoT Lançado o plug-in de provisionamento de frota v1.2.1	AWS IoT o plug-in de provisionamento de frota v1.2.1 está disponível. Esta versão corrige um problema em que o plug-in de provisionamento de frotas fica off-line durante a inicialização do Greengrass Nucleus. O plug-in de provisionamento de frota agora repete indefinidamente as chamadas de conexão MQTT.	27 de março de 2024
AWS IoT Greengrass Atualização do software Core v2.12.3	Esta versão fornece a versão 2.12.3 do componente nucleus do Greengrass e atualiza os componentes fornecidos. AWS	27 de março de 2024
Lançado o Greengrass CLI v2.12.3	O componente Greengrass CLI v2.12.3 está disponível.	25 de março de 2024

AWS IoT Device Tester v4.9.2 com GGV2Q v2.5.2 lançado	A versão 4.9.2 do IDT for AWS IoT Greengrass V2 está disponível. Essa versão inclui o pacote de qualificação AWS IoT Greengrass V2 (GGV2Q) v2.5.2 e é compatível com as versões 2.12.0, 2.11.0, 2.10.0, 2.9.5 do núcleo Greengrass.	18 de março de 2024
Lançado o agente Lookout for Vision edge v1.2.0	O agente Lookout for Vision edge v1.2.0 está disponível.	11 de março de 2024
AWS IoT Greengrass Atualização do software Core v2.12.2	Esta versão fornece a versão 2.12.2 do componente nucleus do Greengrass e atualiza os componentes fornecidos. AWS	15 de fevereiro de 2024
Lançado o Shadow manager v2.3.6	O Shadow manager v2.3.6 está disponível. Esta versão corrige um problema em que as propriedades de sombra que são excluídas por meio de Nuvem AWS atualizações enquanto o dispositivo está off-line continuam existindo na sombra local após recuperar a conectividade.	14 de fevereiro de 2024
Lançado o lançador Lambda v2.0.13	A versão 2.0.13 do component e Lambda launcher está disponível. Esta versão inclui correções de erros e melhorias gerais.	14 de fevereiro de 2024

Lançado o spooler de disco v1.0.3	O componente spooler de disco v1.0.3 está disponíve l. Essa versão melhora o desempenho ao reutilizar conexões de banco de dados.	14 de fevereiro de 2024
Lançado o agente Lookout for Vision edge v1.1.9	O agente Lookout for Vision edge v1.1.9 está disponível.	17 de janeiro de 2024
Kit de desenvolvimento Greengrass CLI v1.6.2	A versão 1.6.2 da CLI do Greengrass Development Kit está disponível. Esta versão corrige um problema em que o Windows gradlew.bat não funciona devido ao caminho relativo. Essa versão também contém melhorias adicionais.	16 de janeiro de 2024
Novos eventos CloudTrail de dados	Agora você pode registrar eventos de AWS CloudTrail dados para obter informações sobre operações de recursos, como obter um component e ou a configuração de uma implantação. Use esses eventos para obter informaçõ es sobre a operação de seus dispositivos Greengrass.	20 de dezembro de 2023
Lançado o agente Lookout for Vision edge v1.1.8	O agente Lookout for Vision edge v1.1.8 está disponível.	12 de dezembro de 2023

Lançado o Stream Manager v2.1.12	O Stream Manager v2.1.12 já está disponível. Essa versão altera a ordem que o Greengrass usa para selecionar um conjunto de credenciais para AWS chamadas de serviço.	8 de dezembro de 2023
Lançada a ponte MQTT v2.3.1	A ponte MQTT v2.3.1 está disponível. Esta versão corrige um problema raro em que o cliente MQTT local entra em um loop de desconexão.	8 de dezembro de 2023
Lançado o spooler de disco v1.0.2	O componente spooler de disco v1.0.2 está disponíve l. Esta versão corrige um problema em que o campo de formato de mensagem MQTT não persiste em alguns casos.	8 de dezembro de 2023
Lançado o componente de autenticação do dispositivo cliente v2.4.5	O componente de autentica ção do dispositivo cliente v2.4.5 está disponível. Esta versão adiciona suporte para curingas no final dos nomes em uma regra de seleção e corrige um problema em que os certificados não são atualizados com novas informações de conectividade em certos casos.	8 de dezembro de 2023
AWS IoT Greengrass Atualização do software Core v2.12.1	Esta versão fornece a versão 2.12.1 do componente nucleus do Greengrass e atualiza os componentes fornecidos. AWS	8 de dezembro de 2023

Kit de desenvolvimento Greengrass CLI v1.6.1	A versão 1.6.1 da CLI do Greengrass Development Kit está disponível. Esta versão contém correções de erros e melhorias.	6 de dezembro de 2023
Validação da receita	Foi adicionado um recurso de validação de receita que validará uma receita de componente ao criar uma versão de componente.	16 de novembro de 2023
Componentes compatíveis com o editor	AWS IoT Greengrass agora oferece componentes compatíveis com o Publisher . Esses componentes são desenvolvidos, oferecidos e atendidos por fornecedores terceirizados.	16 de novembro de 2023
Lançado o Greengrass Testing Framework v1.2.0	O Greengrass Testing Framework v1.2.0 está disponível.	15 de novembro de 2023

Kit de desenvolvimento Greengrass CLI v1.6.0	A versão 1.6.0 da CLI do Greengrass Development Kit está disponível. Esta versão adiciona uma verificação de validação de receita em relação ao esquema de receitas do Greengrass durante os comandos e. `component build` `component publish` Essa atualização ajuda os desenvolvedores a identificar problemas acionáveis em suas receitas de componentes no início do processo de criação de componentes. Essa versão também adiciona um conjunto de testes de confiança ao modelo que pode ser baixado pelo `test-e2e init` comando. Esse conjunto de testes de confiança inclui oito testes genéricos que podem ser usados e ampliados para atender às necessidades básicas de testes de componentes.	15 de novembro de 2023
AWS IoT Device Tester v4.9.1 suporta a versão 2.12.0 do núcleo Greengrass	A versão 4.9.1 do IDT for AWS IoT Greengrass V2 agora oferece suporte à versão 2.12.0 do núcleo Greengrass.	7 de novembro de 2023

AWS IoT Greengrass Atualização do software Core v2.12.0	Esta versão fornece a versão 2.12.0 do componente nucleus do Greengrass e atualiza os componentes fornecidos. AWS	7 de novembro de 2023
Opere um dispositivo principal do Greengrass na VPC	A operação de um dispositivo principal do Greengrass na VPC está disponível. Esse recurso permite que você execute implantações em VPC sem acesso público à Internet.	3 de novembro de 2023
Lançado o Greengrass CLI v2.12.0	O componente Greengrass CLI v2.12.0 está disponível.	30 de outubro de 2023
Lançado o Stream Manager v2.1.10	O Stream Manager v2.1.10 já está disponível. Esta versão corrige um problema em que a configuração do proxy HTTPS não confia na cadeia de certificados CA do Greengrass.	26 de outubro de 2023
Lançado o lançador Lambda v2.0.12	A versão 2.0.12 do componente Lambda launcher está disponível. Esta versão corrige um problema em que o lançador Lambda poderia gerar um erro se o processo anterior não fosse interrompido corretamente.	26 de outubro de 2023

Kit de desenvolvimento Greengrass CLI v1.5.0	A versão 1.5.0 da CLI do Greengrass Development Kit está disponível. Esta versão atualiza os padrões reconhecidos pela opção de `excludes` construção quando `build_system` está`zip`. Esta versão agora reconhecerá padrões globais que correspondem aos nomes de caminho com base em seus caracteres curinga. Isso permite a especific ação personalizada de quais diretórios excluir.	26 de outubro de 2023
Lançado o agente Lookout for Vision edge v1.1.7	O agente Lookout for Vision edge v1.1.7 está disponível.	24 de outubro de 2023
Lançado o Shadow manager v2.3.4	O Shadow manager v2.3.4 está disponível. Esta versão adiciona suporte para documentos de estado de sombra nulos e vazios.	18 de outubro de 2023
Lançado o gerenciador de registros v2.3.6	O componente gerenciador de registros v2.3.6 está disponíve l.	18 de outubro de 2023
Lançado o console de depuração local v2.4.0	O componente v2.4.0 do console de depuração local está disponível.	18 de outubro de 2023
Lançado o Lambda manager v2.3.1	O componente Lambda Manager v2.3.1 está disponíve l.	18 de outubro de 2023

Lançado o Greengrass CLI v2.11.3	O componente Greengrass CLI v2.11.3 está disponível.	18 de outubro de 2023
AWS IoT Greengrass Atualização do software Core v2.11.3	Esta versão fornece a versão 2.11.3 do componente nucleus do Greengrass e atualiza os componentes fornecidos. AWS	18 de outubro de 2023
Lançamento do Secure Tunneling v1.0.17	O tunelamento seguro v1.0.17 está disponível.	4 de outubro de 2023
Kit de desenvolvimento Greengrass CLI v1.4.0	A versão 1.4.0 da CLI do Greengrass Development Kit está disponível. Essa versão adiciona um novo `config` comando que inicia um prompt interativo para modificar campos em um arquivo de configuração existente do GDK. Essa versão também modifica os `gdk component publish` comandos `gdk component build` e para verificar se o tamanho da receita está dentro dos requisitos do Greengrass (<=16000 bytes) antes de continuar.	2 de outubro de 2023
Lançado o broker Moquette MQTT 3.1.1 v2.3.5	O componente de corretor v2.3.5 do Moquette MQTT 3.1.1 está disponível. Esta versão atualiza o Moquette para a versão 0.17.	28 de setembro de 2023

Lançada a ponte MQTT v2.3.0	A ponte MQTT v2.3.0 está disponível. Esta versão adiciona suporte ao MQTT 5 para fazer a ponte entre fontes MQTT locais AWS IoT Core e fontes MQTT.	28 de setembro de 2023
Lançado o agente Lookout for Vision edge v1.1.6	O agente Lookout for Vision edge v1.1.6 está disponível.	27 de setembro de 2023
Lançado o Lambda manager v2.3.0	O componente Lambda Manager v2.3.0 está disponível.	15 de setembro de 2023
Lançado o lançador Lambda v2.0.11	A versão 2.0.11 do componente Lambda launcher está disponível. Esta versão é compatível com o Lambda Manager 2.3.0.	15 de setembro de 2023
Lançado o broker Moquette MQTT 3.1.1 v2.3.4	O componente de corretor Moquette MQTT 3.1.1 v2.3.4 está disponível.	1º de setembro de 2023
Estrutura de testes do Greengrass	O GTF é uma coleção de blocos de construção para apoiar a end-to-end automação. Ele permite que os clientes AWS IoT Greengrass Version 2 internos usem a mesma estrutura de testes que a equipe de serviço usa para qualificar alterações de software, aceitação automatizada e fins de garantia de qualidade.	11 de agosto de 2023

AWS IoT Greengrass Atualização do software Core v2.11.2	Esta versão fornece a versão 2.11.2 do componente nucleus do Greengrass e atualiza os componentes fornecidos. AWS	9 de agosto de 2023
Kit de desenvolvimento Greengrass CLI v1.3.0	A versão 1.3.0 da CLI do Greengrass Development Kit está disponível. Esta versão adiciona um novo `test-e2e` comando para oferecer suporte ao end-to-end teste de componentes usando o Open Test Framework.	21 de julho de 2023
AWS IoT Greengrass Atualização do software Core v2.11.1	Esta versão fornece a versão 2.11.1 do componente nucleus do Greengrass e atualiza os componentes fornecidos. AWS	21 de julho de 2023
Lançado o spooler de disco v1.0.0	O componente spooler de disco v1.0.0 está disponível.	28 de junho de 2023
AWS IoT Greengrass Atualização do software Core v2.11.0	Esta versão fornece a versão 2.11.0 do componente nucleus do Greengrass e atualiza os componentes fornecidos. AWS	28 de junho de 2023
AWS IoT Greengrass Atualização do software Core v2.10.3	Esta versão fornece a versão 2.10.3 do componente nucleus do Greengrass e atualiza os componentes fornecidos. AWS	21 de junho de 2023
AWS IoT Greengrass Atualização do software Core v2.10.2	Esta versão fornece a versão 2.10.2 do componente nucleus do Greengrass e atualiza os componentes fornecidos. AWS	5 de junho de 2023

AWS IoT Greengrass Atualização do software Core v2.10.1	Esta versão fornece a versão 2.10.1 do componente nucleus do Greengrass e atualiza os componentes fornecidos. AWS	11 de maio de 2023
AWS IoT Greengrass Atualização do software Core v2.10.0	Esta versão fornece a versão 2.10.0 do componente nucleus do Greengrass e atualiza os componentes fornecidos. AWS	9 de maio de 2023
SageMaker Edge Manager descontinuado	O componente Amazon SageMaker Edge Manager será descontinuado em 26 de abril de 2024.	28 de abril de 2023
AWS IoT Greengrass Atualização do software Core v2.9.6	Esta versão fornece a versão 2.9.6 do componente nucleus do Greengrass e atualiza os componentes fornecidos. AWS	20 de abril de 2023
Lançado o gerenciador de registros v2.3.2	O componente gerenciador de registros v2.3.2 está disponível.	19 de abril de 2023

Lançado o Stream Manager v2.1.4	O Stream Manager v2.1.4 já está disponível. Esta versão corrige um problema em que as entradas do mesmo ativo de propriedade com o mesmo carimbo de data/hora em um único lote retornam `ConflictingOperationException` da SiteWise API, o que faz com que o gerenciador de fluxo tente novamente continuamente. Essa versão também atualiza o tempo limite de conexão padrão de 3 segundos para 1 minuto.	13 de abril de 2023
Kit de desenvolvimento Greengrass CLI v1.2.3	A versão 1.2.3 da CLI do Greengrass Development Kit está disponível. Esta versão contém correções de erros.	13 de abril de 2023
Lançado o componente de autenticação do dispositivo cliente v2.4.0	O componente de autenticação do dispositivo cliente v2.4.0 está disponível. Esta versão adiciona suporte à autenticação do dispositivo cliente para emitir métricas operacionais que podem ser exibidas no painel do dispositivo cliente do Greengrass.	10 de abril de 2023
Kit de desenvolvimento Greengrass CLI v1.2.2	A versão 1.2.2 da CLI do Greengrass Development Kit está disponível. Esta versão contém melhorias e correções de erros.	7 de abril de 2023

AWS IoT Greengrass Atualização do software Core v2.9.5	Esta versão fornece a versão 2.9.5 do componente nucleus do Greengrass e atualiza os componentes fornecidos. AWS	30 de março de 2023
Lançado o Stream Manager v2.1.3	O Stream Manager v2.1.3 já está disponível. Esta versão corrige um problema de inicialização no sistema operacional Windows quando executado como usuário SYSTEM.	7 de março de 2023
Lançado o adaptador de protocolo Modbus-RTU v2.1.5	O componente adaptador de protocolo Modbus-RTU v2.1.5 está disponível. Esta versão corrige um problema com a `ReadDiscreteInput` operação.	7 de março de 2023
Lançado o componente de autenticação do dispositivo cliente v2.3.2	O componente de autentica ção do dispositivo cliente v2.3.2 está disponível. Esta versão adiciona suporte para armazenar em cache as informações do nome do host para que o componente gere corretamente os assuntos do certificado quando reiniciado enquanto estiver off-line.	7 de março de 2023
AWS IoT Device Tester v4.7.0 suporta a versão 2.9.4 do núcleo Greengrass	A versão 4.7.0 do IDT for AWS IoT Greengrass V2 agora oferece suporte à versão 2.9.4 do núcleo Greengrass.	2 de março de 2023

Lançada a interface de linha de comando do Greengrass v1.2.0	A interface de linha de comando do Greengrass v1.2.0 está disponível.	28 de fevereiro de 2023
AWS IoT Greengrass Atualização do software Core v2.9.4	Esta versão fornece a versão 2.9.4 do componente nucleus do Greengrass e atualiza os componentes fornecidos. AWS	24 de fevereiro de 2023
Lançado o Shadow Manager v2.3.1	O Shadow manager v2.3.1 está disponível. Esta versão corrige uma condição que pode impedir a sincronização das atualizações do Cloud Shadow. Essa versão também corrige um problema em que as alterações na configuração de sincronização de sombra nomeada se aplicam somente a uma sombra nomeada.	21 de fevereiro de 2023
AWS IoT Device Tester v4.7.0 suporta a versão 2.9.3 do núcleo Greengrass	A versão 4.7.0 do IDT for AWS IoT Greengrass V2 agora oferece suporte à versão 2.9.3 do núcleo Greengrass.	9 de fevereiro de 2023
As melhores práticas do IAM foram atualizadas	Guia atualizado para alinhamento com as práticas recomendadas do IAM. Para obter mais informações, consulte Práticas recomendadas de segurança no IAM.	3 de fevereiro de 2023
AWS IoT Greengrass Atualização do software Core v2.9.3	Esta versão fornece a versão 2.9.3 do componente nucleus do Greengrass e atualiza os componentes fornecidos. AWS	1° de fevereiro de 2023

Lançado o gerenciador de registros v2.3.1	O gerenciador de registros v2.3.1 está disponível.	27 de janeiro de 2023
AWS IoT Device Tester v4.7.0 suporta a versão 2.9.2 do núcleo Greengrass	A versão 4.7.0 do IDT for AWS IoT Greengrass V2 agora oferece suporte à versão 2.9.2 do núcleo Greengrass.	3 de janeiro de 2023
Lançado o Shadow Manager v2.3.0	O Shadow manager v2.3.0 está disponível. Esta versão corrige um problema que pode impedir a sincronização das sombras quando um dispositivo armazena a chave privada do dispositivo Greengrass em um módulo de segurança de hardware.	29 de dezembro de 2022
AWS IoT Lançado o plug-in de provisionamento de frota v1.2.0	AWS IoT o plug-in de provisionamento de frota v1.2.0 está disponível. Esta versão adiciona suporte para provisionamento de dispositivos por meio de solicitação de assinatura de certificado com caminho de chave privada configurável.	22 de dezembro de 2022
AWS IoT Greengrass Atualização do software Core v2.9.2	Esta versão fornece a versão 2.9.2 do componente nucleus do Greengrass e atualiza os componentes fornecidos. AWS	22 de dezembro de 2022

AWS IoT Device Tester v4.7.0 com GGV2Q v2.5.0 lançado	A versão 4.7.0 do IDT for AWS IoT Greengrass V2 está disponível. Essa versão inclui o pacote de qualificação AWS IoT Greengrass V2 (GGV2Q) v2.5.0 e é compatível com as versões 2.9.1, 2.9.0, 2.8.1, 2.8.0, 2.7.0 e 2.6.0 do núcleo Greengrass.	13 de dezembro de 2022
Lançado o Shadow Manager v2.2.4	Corrige um problema em que a validação do tamanho da sombra não era consistente com a nuvem ao atualizar o documento de sombra local. Isso também corrige um problema em que o gerenciador de sombra para de ouvir as atualizações de configuração se uma implantação executa uma `RESET` nos nós de configuração.	08 de dezembro de 2022
Lançado o Lookout for Vision Edge Agent 1.1.1	O componente v1.1.1 do Lookout for Vision Edge Agent está disponível.	5 de dezembro de 2022
Lançado o gerenciador de registros v2.3.0	O componente gerenciador de registros v2.3.0 está disponível.	18 de novembro de 2022
AWS IoT Device Tester v4.5.11 suporta a versão 2.9.1 do núcleo Greengrass	A versão 4.5.11 do IDT for AWS IoT Greengrass V2 agora oferece suporte à versão 2.9.1 do núcleo Greengrass.	18 de novembro de 2022

AWS IoT Greengrass Atualização do software Core v2.9.1	Esta versão fornece a versão 2.9.1 do componente nucleus do Greengrass e atualiza os componentes fornecidos. AWS	18 de novembro de 2022
AWS IoT Device Tester v4.5.11 suporta a versão 2.9.0 do núcleo Greengrass	A versão 4.5.11 do IDT for AWS IoT Greengrass V2 agora oferece suporte à versão 2.9.0 do núcleo Greengrass.	17 de novembro de 2022
Lançado o Stream Manager v2.1.2	O Stream Manager v2.1.2 já está disponível. Esta versão corrige um problema no sistema operacional Windows que usa um idioma diferente do inglês.	15 de novembro de 2022
AWS IoT Greengrass Atualização do software Core v2.9.0	Esta versão fornece a versão 2.9.0 do componente nucleus do Greengrass e atualiza os componentes fornecidos. AWS	15 de novembro de 2022
AWS IoT Device Tester v4.5.11 suporta o núcleo Greengrass versão 2.8.1	A versão 4.5.11 do IDT for AWS IoT Greengrass V2 agora oferece suporte à versão 2.8.1 do núcleo Greengrass.	19 de outubro de 2022
AWS IoT Device Tester Lançamento da v4.5.11 com o GGV2Q v2.4.1	A versão 4.5.11 do IDT para AWS IoT Greengrass V2 está disponível. Essa versão inclui o pacote de qualificação AWS IoT Greengrass V2 (GGV2Q) v2.4.1 e é compatível com as versões 2.8.0, 2.7.0 e 2.6.0 do núcleo Greengrass.	13 de outubro de 2022

AWS IoT Greengrass Atualização do software Core v2.8.1	Esta versão fornece a versão 2.8.1 do componente nucleus do Greengrass e AWS atualiza os componentes fornecidos.	13 de outubro de 2022
AWS IoT Greengrass Atualização do software Core v2.8.0	Esta versão fornece a versão 2.8.0 do componente nucleus do Greengrass e atualiza os componentes fornecidos. AWS	7 de outubro de 2022
AWS CloudFormation Suporte adicional para implantações	AWS CloudFormation agora oferece suporte a AWS IoT Greengrass implantações como um recurso.	6 de outubro de 2022
SageMaker Lançado o Edge Manager v1.3.0	O componente Amazon SageMaker Edge Manager v1.3.0 está disponível. Esta versão adiciona suporte a esse componente para definir o tamanho do disco para o cache do modelo TensorRT e melhora a simultaneidade de previsão para fazer melhor uso dos mecanismos aceleradores de dispositivos, como GPUs.	1º de setembro de 2022
Use o cliente V2 de comunicação entre processos (IPC)	Foram adicionadas informações sobre o cliente IPC V2, o que reduz a quantidade de código que você precisa escrever para usar operações IPC e ajuda a evitar erros comuns que podem ocorrer com o cliente IPC V1.	12 de agosto de 2022

AWS IoT Device Tester v4.5.8 com GGV2Q v2.4.0 lançado	A versão 4.5.8 do IDT para AWS IoT Greengrass V2 está disponível. Essa versão inclui o pacote de qualificação AWS IoT Greengrass V2 (GGV2Q) v2.4.0 e é compatível com as versões 2.7.0, 2.6.0 e 2.5.6 do núcleo Greengrass.	12 de agosto de 2022
SageMaker Lançado o Edge Manager v1.2.0	O componente Amazon SageMaker Edge Manager v1.2.0 está disponível. Esta versão adiciona suporte a esse componente para recuperar automaticamente os modelos SageMaker compilados pelo NEO que você carrega no Amazon S3, para que você possa implantar novos modelos sem precisar criar uma implantação. AWS IoT Greengrass	3 de agosto de 2022
AWS IoT Device Tester v4.5.3 suporta o Greengrass nucleus versão 2.7.0	A versão 4.5.3 do IDT for AWS IoT Greengrass V2 agora oferece suporte à versão 2.7.0 do núcleo Greengrass.	1º de agosto de 2022
Lançado o Stream Manager v2.1.0	O Stream Manager v2.1.0 já está disponível. Esta versão inclui suporte para você enviar métricas de telemetria para a Amazon. EventBridge	28 de julho de 2022

AWS IoT Greengrass Atualização do software Core v2.7.0	Esta versão fornece a versão 2.7.0 do componente nucleus do Greengrass e atualiza os componentes fornecidos. AWS Inclui suporte para você enviar métricas de telemetria para a Amazon. EventBridge	28 de julho de 2022
Lançado o SiteWise editor de IoT v2.2.0	O componente SiteWise Editor de IoT v2.2.0 está disponíve l. Esta versão atualiza o componente para compactar dados antes de enviá-los ao AWS IoT SiteWise serviço, o que reduz o uso da largura de banda em até 75%.	19 de julho de 2022
Tutorial: Desenvolva um componente que interaja com as sombras do dispositivo cliente	Foi adicionado um novo módulo ao Tutorial: Interaja com dispositivos IoT locais por meio do MQTT, que você pode seguir para aprender a desenvolver um componente que interage com as sombras do dispositivo cliente.	18 de julho de 2022
Escolha um corretor MQTT local	Foram adicionadas informaçõ es sobre como escolher um agente MQTT local no qual os dispositivos cliente se conectam a um dispositivo principal.	18 de julho de 2022
AWS IoT Device Tester v4.5.3 suporta o Greengrass nucleus versão 2.6.0	A versão 4.5.3 do IDT for AWS IoT Greengrass V2 agora oferece suporte à versão 2.6.0 do núcleo Greengrass.	29 de junho de 2022

AWS IoT Greengrass Atualização do software Core v2.6.0	Esta versão fornece a versão 2.6.0 do componente nucleus do Greengrass e atualiza os componentes fornecidos. AWS Ele inclui suporte para sombras de dispositivos clientes e um agente MQTT 5 local para dispositivos clientes. Também inclui suporte para curingas em tópicos locais de publicação/assinatura, variáveis de receita em configurações de componentes e curingas em políticas de autorização de IPC. Esses recursos permitem que você desenvolva e configure com mais facilidade os componentes que você implanta em frotas de dispositivos principais. Essa versão também inclui suporte para componentes usarem operações IPC que gerenciam implantações e componentes locais em um dispositivo principal.	27 de junho de 2022

Atualizações de componentes do dispositivo cliente	A autenticação do dispositivo cliente v2.1.0, o agente MQTT (Moquette) v2.1.0, a ponte MQTT v2.1.1 e o detector de IP v2.1.2 estão disponíveis. Esta versão melhora a rotação de certificados, melhora o desempenho do agente MQTT e corrige problemas com a forma como esses component es lidam com as atualizações de redefinição de configura ção.	14 de junho de 2022
AWS IoT Device Tester v4.5.3 suporta o núcleo Greengrass versão 2.5.6	A versão 4.5.3 do IDT for AWS IoT Greengrass V2 agora oferece suporte à versão 2.5.6 do núcleo Greengrass.	1º de junho de 2022
AWS IoT Greengrass Atualização do software Core v2.5.6	Esta versão fornece a versão 2.5.6 do componente nucleus do Greengrass e atualiza os componentes fornecido s. AWS Ele inclui suporte para módulos de segurança de hardware com chaves ECC. Também inclui outras correções de bugs e melhorias .	31 de maio de 2022

AWS IoT Lançado o plug-in de provisionamento de frota v1.1.0	AWS IoT o plug-in de provisionamento de frota v1.1.0 está disponível. Esta versão adiciona suporte para formatos adicionais de caminho de arquivo quando você configura o plug-in em dispositivos Windows.	12 de maio de 2022
Lançados novos tempos de execução do Lambda	Foi adicionado suporte para novos tempos de execução do Lambda: Python 3.9, Java 11 e NodeJS 14.	10 de maio de 2022
Desenvolva um componente do Greengrass que adia as atualizações de componentes	Foi adicionado um tutorial que você pode seguir para aprender a desenvolver um componente do Greengrass que adia as atualizações de componentes das implantações. Talvez você queira atrasar uma atualização quando um dispositivo está com pouca bateria ou enquanto executa um processo que não pode ser interrompido, por exemplo.	4 de maio de 2022

CloudWatch métricas v3.1.0 e v3.1.0 lançadas AWS IoT Device Defender	CloudWatch o componente de métricas v3.1.0 e o AWS IoT Device Defender component e v3.1.0 estão disponíveis. Essas versões adicionam suporte para configurações de proxy de rede HTTPS. Para obter mais informações, consulte Conectar na porta 443 ou por meio de um proxy de rede e Habilitar o dispositivo principal para confiar em um proxy HTTPS.	27 de abril de 2022
Migrar de AWS IoT Greengrass Version 1	Foi adicionado um guia que você pode seguir para AWS IoT Greengrass V1 migrar de a. AWS IoT Greengrass V2	26 de abril de 2022
AWS IoT Device Tester v4.5.3 com GGV2Q v2.3.1 atualizado e IDT v4.5.1 com GGV2Q v2.3.0 adicionado às versões suportadas	A versão 4.5.3 do IDT para AWS IoT Greengrass V2 com o pacote de qualificação AWS IoT Greengrass V2 (GGV2Q) v2.3.1 foi atualizada para incluir suporte às versões 2.5.5, 2.5.4 e 2.5.3 do núcleo Greengrass. Essa atualização também inclui o IDT 4.5.1 com o pacote de qualificação AWS IoT Greengrass V2 (GGV2Q) v2.3.0 como versão compatível. O IDT 4.5.1 com o pacote de qualificação AWS IoT Greengrass V2 (GGV2Q) v2.3.0 suporta o Greengrass nucleus versão 2.5.3.	25 de abril de 2022

Lançado o adaptador de protocolo Modbus-RTU v2.1.0	O componente adaptador de protocolo Modbus-RTU v2.1.0 está disponível. Esta versão adiciona novos parâmetros que você pode especificar para configurar a comunicaç ão serial com dispositivos Modbus RTU.	20 de abril de 2022
CloudWatch Metrics v2.1.0, Firehose v2.1.0 e Amazon SNS v2.1.0 lançados	CloudWatch O component e de métricas v2.1.0, o componente Firehose v2.1.0 e o componente Amazon SNS v2.1.0 estão disponíve is. Essas versões adicionam suporte para configurações de proxy de rede HTTPS. Para obter mais informações, consulte Conectar na porta 443 ou por meio de um proxy de rede e Habilitar o dispositi vo principal para confiar em um proxy HTTPS.	19 de abril de 2022
AWS IoT Device Tester v4.5.3 com GGV2Q v2.3.1 lançado	A versão 4.5.3 do IDT para AWS IoT Greengrass V2 está disponível. Essa versão inclui o pacote de qualificação AWS IoT Greengrass V2 (GGV2Q) v2.3.1 e é compatível com o Greengrass nucleus versão 2.5.5.	15 de abril de 2022

AWS IoT Greengrass Atualização do software Core v2.5.5	Esta versão fornece a versão 2.5.5 do componente nucleus do Greengrass e AWS atualiza os componentes fornecidos. Ele adiciona suporte para dispositivos Windows que usam um idioma de exibição diferente do inglês. Também corrige um problema em que o dispositivo principal não reportava seu status ao serviço de AWS IoT Greengrass nuvem após o provisionamento em determinados cenários.	6 de abril de 2022
AWS IoT Greengrass Atualização do software Core v2.5.4	Esta versão fornece a versão 2.5.4 do componente nucleus do Greengrass e atualiza os componentes fornecidos. AWS Inclui correções de erros e melhorias.	23 de março de 2022
Faça o download AWS IoT Device Tester programaticamente	Foram adicionadas informações sobre como baixar o IDT de forma AWS IoT Greengrass V2 programática.	15 de março de 2022

Kit de desenvolvimento Greengrass CLI v1.1.0	A versão 1.1.0 da CLI do Greengrass Development Kit está disponível. Esta versão adiciona novos argumentos aos `component publish` comandos `component init` e. Essa versão também atualiza o `component publish` comando para criar o componente se ele não estiver compilado.	24 de fevereiro de 2022
Lançado o Shadow Manager v2.1.0	O componente Shadow Manager v2.1.0 está disponível. Esta versão adiciona a opção de configurar o intervalo com o qual o component e sincroniza as sombras. AWS IoT Core Por exemplo, você pode especificar um intervalo maior para reduzir o uso da largura de banda e as cobranças.	3 de fevereiro de 2022
Dockerfile e imagens Docker para AWS IoT Greengrass o software Core v2.5.3	O Dockerfile e a imagem Docker para o software AWS IoT Greengrass Core v2.5.3 já estão disponíveis.	12 de janeiro de 2022

AWS IoT Device Tester v4.5.1 com GGV2Q v2.3.0 lançado	A versão 4.5.1 do IDT para AWS IoT Greengrass V2 está disponível. Esta versão inclui o pacote de qualificação AWS IoT Greengrass V2 (GGV2Q) v2.3.0 e oferece suporte à validação e qualificação de dispositivos baseados em Linux que usam um módulo de segurança de hardware (HSM) para armazenar a chave privada e o certificado usados pelo software Core. AWS IoT Greengrass	11 de janeiro de 2022
AWS IoT Greengrass Atualização do software Core v2.5.3	Esta versão fornece a versão 2.5.3 do componente nucleus do Greengrass e atualiza os componentes fornecidos. AWS Ele inclui suporte para você configurar o software AWS IoT Greengrass Core para usar uma chave privada e um certificado que você armazena com segurança em um módulo de segurança de hardware (HSM).	6 de janeiro de 2022
Dockerfile e imagens Docker para AWS IoT Greengrass o software Core v2.5.2	O Dockerfile e a imagem Docker para o software AWS IoT Greengrass Core v2.5.2 já estão disponíveis.	20 de dezembro de 2021

AWS IoT Device Tester v4.4.1 com GGV2Q v2.2.1 lançado	A versão 4.4.1 do IDT para AWS IoT Greengrass V2 está disponível. Essa versão inclui o pacote de qualificação AWS IoT Greengrass V2 (GGV2Q) v2.2.1 e é compatível com o Greengrass nucleus versão 2.5.2 para qualificação de dispositivos.	12 de dezembro de 2021
Execute inferências de aprendizado de máquina usando o Amazon Lookout for Vision	Foram adicionadas informaçõ es sobre como realizar inferência de aprendizado de máquina usando o Lookout for Vision nos dispositivos principais do Greengrass. O Lookout for Vision usa visão computacional para encontrar defeitos visuais em produtos industriais.	8 de dezembro de 2021
AWS IoT Device Tester v4.4.1 com GGV2Q v2.2.0 lançado	A versão 4.4.1 do IDT para AWS IoT Greengrass V2 está disponível. Essa versão inclui o pacote de qualificação AWS IoT Greengrass V2 (GGV2Q) v2.2.0 e é compatível com o Greengrass nucleus versão 2.5.2 para qualificação de dispositivos.	6 de dezembro de 2021

AWS IoT Greengrass Atualização do software Core v2.5.2	Esta versão fornece a versão 2.5.2 do componente nucleus do Greengrass e atualiza os componentes fornecidos. AWS Ele corrige um problema com o serviço Windows que ocorre após as atualizações do núcleo do Greengrass. Também inclui suporte para o AWS IoT Device Defender componente em dispositivos Windows.	3 de dezembro de 2021
Novo conector edge para o componente Kinesis Video Streams	A versão 1.0.0 do conector edge para o component e Kinesis Video Streams está disponível. Isso AWS fornecido lê feeds de vídeo de câmeras locais e publica os streams no Kinesis Video Streams. Este component e se integra com AWS IoT TwinMaker, o que permite visualizar e gerenciar fluxos de vídeo e outros dados nos painéis da Grafana.	30 de novembro de 2021

Gerencie os principais dispositivos do Greengrass com AWS Systems Manager	Foram adicionadas informaçõ es sobre como gerenciar os principais dispositivos do Greengrass com o. AWS Systems Manager O Systems Manager é um AWS serviço que permite visualizar dados operacionais, automatizar tarefas operacionais e manter a segurança e a conformid ade.	29 de novembro de 2021
Kit de desenvolvimento do Greengrass (CLI)	Foram adicionadas informaçõ es sobre a interface de linha de comando do kit de AWS IoT Greengrass desenvolv imento (GDK CLI), que é uma ferramenta que você pode baixar em seu computado r de desenvolvimento local para ajudá-lo a desenvolver componentes personalizados do Greengrass. Você pode usar a CLI do GDK para criar, criar e publicar componentes personalizados.	29 de novembro de 2021

Componentes do Greengrass fornecidos pela comunidade	Foram adicionadas informaçõ es sobre o Catálogo de Software do Greengras s, que é um índice dos componentes do Greengrass desenvolvidos pela comunidad e do Greengrass. A partir desse catálogo, você pode baixar, modificar e implantar componentes para criar seus aplicativos Greengrass.	29 de novembro de 2021
AWS IoT Greengrass Atualização do software Core v2.5.1	Esta versão fornece a versão 2.5.1 do componente nucleus do Greengrass e AWS atualiza os componentes fornecido s. Ele inclui suporte para Java de 32 bits em dispositi vos Windows. Ele também corrige problemas com o novo comportamento de remoção de grupos de coisas e com o carregamento de variáveis de ambiente do sistema em dispositivos Windows.	23 de novembro de 2021

AWS IoT Device Tester v4.4.0 com GGV2Q v2.1.0 lançado	A versão 4.4.0 do IDT para AWS IoT Greengrass V2 está disponível. Essa versão inclui o pacote de qualificação AWS IoT Greengrass V2 (GGV2Q) v2.1.0 e oferece suporte à qualificação de dispositivos Greengrass baseados em Windows que executam o Greengrass nucleus versão 2.5.0.	19 de novembro de 2021
AWS IoT Greengrass Atualização do software Core v2.5.0	Esta versão fornece a versão 2.5.0 do componente nucleus do Greengrass e AWS atualiza os componentes fornecidos. Ele inclui suporte para executar o software AWS IoT Greengrass Core em dispositivos Windows. Também altera o comportamento de remoção de grupos de coisas e adiciona suporte para proxies HTTPS.	12 de novembro de 2021

SageMaker Lançado o Edge Manager v1.1.0	O componente Amazon SageMaker Edge Manager v1.1.0 está disponível. Esta versão adiciona suporte aos dispositivos principais do Greengrass que executam o Amazon Linux 2 e adiciona um novo parâmetro de configuração para especific ar a localização da pasta de dados de captura em seu dispositivo.	3 de novembro de 2021
Atualização da prevenção contra o problema confused deputy entre serviços	AWS IoT Greengrass V2 suporta o uso das chaves de contexto de condição `aws:SourceAccount` global `aws:SourceArn` e das políticas de recursos do IAM para evitar o confuso problema adjunto.	1º de novembro de 2023
Atualizações de componentes do dispositivo cliente	A autenticação do dispositivo cliente v2.0.3, o detector de IP v2.1.0, a ponte MQTT v2.1.0 e o agente MQTT (Moquette) v2.0.2 estão disponíveis. Esta versão adiciona suporte completo para portas de broker MQTT não padrão e inclui outras correções de erros e melhorias.	28 de outubro de 2021

Lançado o Shadow Manager v2.0.4	O componente Shadow Manager v2.0.4 está disponível. Esta versão corrige um problema que fazia com que o gerenciador de sombras excluísse versões recém-criadas de qualquer sombra que tenha sido excluída anteriormente. A partir desta versão, a operação `DeleteThingShadow` IPC incrementa a versão sombra.	20 de outubro de 2021
Lançado o gerenciador de registros v2.2.0	O componente gerenciador de registros v2.2.0 está disponível. O gerenciador de registros agora suporta o uso de um mapa de configuração para fornecer configurações de registro de componentes.	20 de outubro de 2021
Lançado o Lambda manager v2.1.4	O componente Lambda Manager v2.1.4 está disponível. Esta versão corrige um problema que fazia com que as funções Lambda que usam tempos de execução do NodeJS processassem somente uma mensagem.	20 de outubro de 2021
Use comunicação entre processos, AWS credenciais e gerenciador de fluxo nos componentes de contêiner do Docker	Foram adicionadas informações sobre como usar a comunicação entre processos (IPC), AWS as credenciais e o gerenciador de streams em seus componentes personalizados de contêiner do Docker.	19 de outubro de 2021

Novo componente emissor de telemetria de núcleo	A versão 1.0.0 do component e emissor de telemetria do núcleo está disponível. Esse componente AWS fornecido reúne dados de telemetria de integridade do sistema e os publica continuamente em um tópico local e em um tópico do MQTT. AWS IoT Core	30 de setembro de 2021
Permitir o tráfego de dispositivos por meio de um proxy ou firewall	Foram adicionadas informaçõ es sobre os endpoints e portas que o dispositivo principal do Greengrass usa, para que você possa restringir o tráfego como medida de segurança.	16 de setembro de 2021
AWS IoT Device Tester v4.2.0 com GGV2Q v2.0.1 lançado	A versão 4.2.0 do IDT para AWS IoT Greengrass V2 foi atualizada com o pacote de qualificação V2 (AWS IoT Greengrass GGV2Q) v2.0.1. Esta versão é compatível com o Greengrass nucleus versão 2.4.0 para qualificação de dispositivos.	31 de agosto de 2021
Componentes atualizados do instalador de aprendizado de máquina	O componente instalador DLR v1.6.5 e o component e instalador TensorFlow Lite v2.5.4 estão disponíveis. Essas versões de component es incluem o novo parâmetro de `UseInstaller` configura ção que permite desativar o script de instalação padrão.	30 de agosto de 2021

Suporte Linux incorporado para AWS IoT Greengrass	A BitBake receita do AWS IoT Greengrass V2 está disponível no `meta-aws` projeto em GitHub. Você pode usar essa receita para criar um sistema operacional personalizado baseado em Linux usando o Projeto Yocto.	20 de agosto de 2021
Integridade do código	Foram adicionadas informações sobre como AWS IoT Greengrass V2 verificar a integridade do software que os dispositivos principais do Greengrass baixam do. Nuvem AWS	19 de agosto de 2021
Endpoints da VPC (AWS PrivateLink)	AWS IoT Greengrass agora suporta a interface VPC endpoints (AWS PrivateLink) para o AWS IoT Greengrass plano de controle. Você pode estabelecer uma conexão privada entre sua VPC e o plano de AWS IoT Greengrass controle.	16 de agosto de 2021
Lançado o Stream Manager v2.0.12	O Stream Manager v2.0.12 já está disponível. Esta versão corrige um problema que impedia atualizações da versão 2.0.7 do componente gerenciador de stream para uma versão entre v2.0.8 e v2.0.11.	10 de agosto de 2021

Dockerfile e imagens Docker para AWS IoT Greengrass o software Core v2.4.0	O Dockerfile e a imagem Docker para o software AWS IoT Greengrass Core v2.4.0 já estão disponíveis.	9 de agosto de 2021
AWS IoT Greengrass Atualização do software Core v2.4.0	Esta versão fornece a versão 2.4.0 do componente nucleus do Greengrass e AWS atualiza os componentes fornecidos. Ele inclui suporte para limites de recursos do sistema de componentes, operações de IPC para pausar e retomar componentes e plug-ins de provisionamento.	3 de agosto de 2021
Novos AWS IoT SiteWise componentes	Foram adicionados os seguintes componentes AWS fornecidos para AWS IoT SiteWise: coletor OPC-UA de SiteWise IoT, editor de IoT e processador de SiteWise IoT. SiteWise	29 de julho de 2021
AWS IoT Device Tester v4.2.0 com GGV2Q v2.0.0 lançado	A versão 4.2.0 do IDT para AWS IoT Greengrass V2 está disponível. Esta versão inclui o pacote de qualificação AWS IoT Greengrass V2 (GGV2Q) v2.0.0 e inclui suporte para testes de qualificação opcionais para componentes do Docker, aprendizado de máquina e gerenciador de stream.	14 de julho de 2021

AWS IoT Greengrass Biblioteca IPC principal disponível AWS IoT Device SDK para C++ v2	A versão 1.13.0 do AWS IoT Device SDK for C++ v2 oferece suporte ao AWS IoT Greengrass Core IPC, para que você possa desenvolver componentes em C++ que interajam com o software Core. AWS IoT Greengrass	14 de julho de 2021
SageMaker Lançado o componente Edge Manager v1.0.2	O componente Amazon SageMaker Edge Manager v1.0.2 está disponível. Esta versão atualiza o script de instalação no ciclo de vida do componente. Seus dispositivos principais agora devem ter o Python 3.6 ou posterior, inclusive `pip` para sua versão do Python, instalado no dispositivo antes de você implantar esse componente.	12 de julho de 2021
Support update AWS IoT Device Tester for AWS IoT Greengrass V2	O IDT for AWS IoT Greengrass V2 versão 4.1.0 agora suporta o uso do Greengrass nucleus versão 2.3.0 para qualificação de dispositivos.	8 de julho de 2021
Dockerfile e imagens Docker para AWS IoT Greengrass o software Core v2.3.0	O Dockerfile e a imagem Docker para o software AWS IoT Greengrass Core v2.3.0 já estão disponíveis.	7 de julho de 2021
AWS políticas gerenciadas	Foram adicionadas informações sobre políticas AWS gerenciadas para AWS IoT Greengrass.	2 de julho de 2021

Novas opções recomendadas de JVM	Foram adicionadas informaçõ es sobre as opções recomendadas de JVM para controlar a alocação de memória para AWS IoT Greengrass o software Core.	30 de junho de 2021
AWS IoT Greengrass Atualização do software Core v2.3.0	Esta versão fornece a versão 2.3.0 do componente nucleus do Greengrass e AWS atualiza os componentes fornecido s. Ele inclui suporte para grandes documentos de configuração de componentes em implantações.	29 de junho de 2021
Dockerfile e imagens Docker para AWS IoT Greengrass o software Core v2.2.0	O Dockerfile e a imagem Docker para o software AWS IoT Greengrass Core v2.2.0 já estão disponíveis.	28 de junho de 2021
AWS IoT Device Tester v4.1.0 com GGV2Q v1.1.1 lançado	A versão 4.1.0 do IDT para AWS IoT Greengrass V2 está disponível. Esta versão inclui o pacote de qualificação AWS IoT Greengrass V2 (GGV2Q) v1.1.1 e suporta o uso do Greengrass nucleus v2.2.0, v2.1.0 e v2.0.5 para qualifica ção de dispositivos.	18 de junho de 2021

AWS IoT Greengrass Atualização do software Core v2.2.0	Esta versão fornece a versão 2.2.0 do componente nucleus do Greengrass e AWS atualiza os componentes fornecidos. Ele inclui componentes que você pode implantar para adicionar suporte a dispositi vos cliente e adicionar o serviço paralelo local.	18 de junho de 2021
Lançado o lançador Lambda v2.0.6	A versão 2.0.6 do component e Lambda launcher está disponível. Esta versão inclui melhorias de desempenho e correções de erros.	13 de junho de 2021
Novo componente SageMaker do Edge Manager lançado	A versão 1.0.0 do component e Amazon SageMaker Edge Manager está disponível para AWS IoT Greengrass. Esse componente instala o binário do agente SageMaker Edge Manager nos dispositivos principais do Greengrass.	10 de junho de 2021
Tipos de componentes	Foram adicionadas informaçõ es sobre os tipos de componentes em AWS IoT Greengrass. O tipo de componente especifica como o software AWS IoT Greengrass Core executa um componente.	3 de junho de 2021

AWS IoT Device Tester v4.0.2 com GGV2Q v1.1.0 lançado	A versão 4.0.2 do IDT para AWS IoT Greengrass V2 está disponível. Esta versão inclui o pacote de qualificação AWS IoT Greengrass V2 (GGV2Q) v1.1.0 e suporta o uso do Greengrass nucleus v2.1.0 com o Greengrass CLI v2.1.0 para qualificação de dispositivos. Isso também inclui novos grupos de teste necessários para MQTT e Lambda, além de outras pequenas correções de erros e melhorias.	5 de maio de 2021
Dockerfile e imagens Docker para AWS IoT Greengrass o software Core v2.1.0	O Dockerfile e a imagem Docker para o software AWS IoT Greengrass Core v2.1.0 já estão disponíveis. A imagem do Docker permite que você execute o software AWS IoT Greengrass Core em um contêiner Docker que usa o Amazon Linux 2 como sistema operacional básico.	27 de abril de 2021

AWS IoT Greengrass Atualização do software Core v2.1.0	Esta versão fornece a versão 2.1.0 do componente nucleus do Greengrass e AWS atualiza os componentes fornecidos. Ele inclui um novo component e que você pode usar para baixar imagens do Docker de repositórios privados do Amazon ECR e novos componentes de amostra para realizar inferências de aprendizado de máquina usando o Lite. TensorFlow	26 de abril de 2021
Exemplo de componente que usa o Secrets Manager	Foi adicionado um component e de exemplo que imprime o valor de um AWS Secrets Manager segredo que você implanta em um dispositivo principal.	8 de abril de 2021
AWS IoT Política mínima para os dispositivos principais do Greengrass	Foram adicionadas informaçõ es sobre o conjunto mínimo de permissões necessárias para oferecer suporte à funcional idade básica do Greengrass em um dispositivo principal.	2 de abril de 2021
Inscreva-se nos streams de eventos do IPC	Foram adicionadas informaçõ es sobre como usar operações de comunicação entre processos (IPC) para assinar fluxos de eventos em um dispositivo principal do Greengrass.	1º de abril de 2021

Atualização de suporte para AWS IoT Device Tester for AWS IoT Greengrass	O IDT for AWS IoT Greengrass V2 versão 4.0.1 agora suporta o uso do Greengrass nucleus versão 2.0.5 com o Greengrass CLI versão 2.0.5 para qualificação de dispositivos.	17 de março de 2021
Crie componentes personalizados que usam o gerenciador de fluxo	Foram adicionadas informações sobre como configurar receitas e artefatos de componentes para desenvolver aplicativos que gerenciam fluxos de dados.	9 de março de 2021
AWS IoT Greengrass Atualização do software Core v2.0.5	Esta versão fornece a versão 2.0.5 do componente nucleus do Greengrass e AWS atualiza os componentes fornecidos. Ele corrige um problema com o suporte de proxy de rede e um problema com o endpoint do plano de dados Greengrass nas AWS regiões da China.	9 de março de 2021
Referência da variável de ambiente do componente	Foram adicionadas informações sobre as variáveis de ambiente que o software AWS IoT Greengrass Core define para os componentes. Você pode usar essas variáveis de ambiente para obter o nome da coisa e a Região da AWS versão do núcleo do Greengrass.	23 de fevereiro de 2021

Instalação manual	Foram adicionadas informaçõ es sobre como criar AWS os recursos necessários manualmente ou instalá-l os atrás de um firewall ou proxy de rede. Ao usar uma instalação manual, você não precisa dar permissão ao instalador para criar recursos no seu Conta da AWS, pois você cria os recursos necessários AWS IoT e do IAM. Você também pode configurar seu dispositivo para se conectar na porta 443 ou por meio de um proxy de rede.	17 de fevereiro de 2021
AWS IoT Greengrass Atualização da biblioteca Core IPC AWS IoT Device SDK para Python v2	A versão 1.5.4 do AWS IoT Device SDK for Python v2 simplifica as etapas necessári as para se conectar ao serviço Core IPC. AWS IoT Greengras s	11 de fevereiro de 2021
Atualização de suporte para AWS IoT Device Tester for AWS IoT Greengrass	O IDT for AWS IoT Greengras s V2 versão 4.0.1 agora suporta o uso do Greengras s nucleus versão 2.0.4 com o Greengrass CLI versão 2.0.4 para qualificação de dispositi vos.	5 de fevereiro de 2021

Novo tutorial para importar funções Lambda	Foi adicionado um novo tutorial baseado em console para importar uma função Lambda como um component e executado no dispositivo principal do Greengrass.	5 de fevereiro de 2021
AWS IoT Greengrass Atualização do software Core v2.0.4	Esta versão fornece a versão 2.0.4 do component e núcleo do Greengrass. Ele inclui o novo `greengrassDataPlanePort` parâmetro para configurar a comunicação HTTPS pela porta 443 e corrige bugs. A política mínima do IAM agora exige o `iam:GetPolicy` e `sts:GetCallerIdentity` quando o instalador do software AWS IoT Greengrass Core é executado `--provision true` .	4 de fevereiro de 2021
Novo componente de tunelamento seguro lançado	A versão 1.0.0 do component e de tunelamento seguro está disponível para. AWS IoT Greengrass Esse componente AWS fornecido usa tunelamento AWS IoT seguro para estabelecer uma comunicação bidirecional segura com um dispositivo central do Greengrass que está protegido por firewalls restritos .	21 de janeiro de 2021

AWS IoT Device Tester para a AWS IoT Greengrass v4.0.1 lançada	A versão 4.0.1 do IDT para AWS IoT Greengrass V2 está disponível. Essa versão permite que você use o IDT para desenvolver e executar seus conjuntos de testes personalizados para validação de dispositivos. Isso também inclui aplicativos IDT com assinatura de código para macOS e Windows.	22 de dezembro de 2020
Lançamento inicial do AWS IoT Greengrass Version 2	AWS IoT Greengrass V2 é uma nova versão principal do AWS IoT Greengrass. Essa versão adiciona vários recursos, como component es modulares de software e implantações contínuas . Esses recursos facilitam o desenvolvimento e o gerenciamento de aplicativos de ponta.	15 de dezembro de 2020

AWS Glossário

Para obter a AWS terminologia mais recente, consulte o AWS glossário na Glossário da AWS Referência.

As traduções são geradas por tradução automática. Em caso de conflito entre o conteúdo da tradução e da versão original em inglês, a versão em inglês prevalecerá.