aws

Guia do desenvolvedor, Versão 1

AWS IoT Greengrass



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

AWS IoT Greengrass: Guia do desenvolvedor, Versão 1

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

As marcas comerciais e imagens de marcas da Amazon não podem ser usadas no contexto de nenhum produto ou serviço que não seja da Amazon, nem de qualquer maneira que possa gerar confusão entre os clientes ou que deprecie ou desprestigie a Amazon. Todas as outras marcas comerciais que não pertencem à Amazon pertencem a seus respectivos proprietários, que podem ou não ser afiliados, patrocinados pela Amazon ou ter conexão com ela.

Table of Contents

..... xxi
O que é AWS IoT Greengrass? 1
AWS IoT Greengrass Software principal 3
AWS IoT Greengrass Versões principais do software 4
AWS IoT Greengrass groups 14
Dispositivos em AWS IoT Greengrass 17
SDKs 20
Plataformas compatíveis e requisitos 21
AWS IoT Greengrass downloads 35
AWS IoT Greengrass Software principal 35
AWS IoT Greengrass software instantâneo 43
AWS IoT Greengrass Software Docker 44
AWS IoT Greengrass SDK principal 46
Bibliotecas e tempos de execução de machine learning compatíveis 47
AWS IoT Greengrass Software ML SDK 48
Deixe seu comentário 48
Instale o AWS IoT Greengrass Software principal 49
Fazer download e extrair um arquivo tar.gz 49
Executar o script de configuração do dispositivo do Greengrass 49
Instalar de um repositório do APT 50
Executar AWS IoT Greengrass em um contêiner Docker 52
Corra AWS IoT Greengrass em um piscar de olhos 52
Arquive uma instalação do software de núcleo 64
Configurar o AWS IoT Greengrass núcleo 66
AWS IoT Greengrass arquivo de configuração principal 67
Os endpoints do serviço devem corresponder ao tipo de certificado 132
Conectar-se à porta 443 ou por meio de um proxy de rede 133
Configurar um diretório de gravação 144
Definir configurações MQTT 148
Ativar detecção automática de IP 166
Iniciar o Greengrass na inicialização do sistema 170
Consulte também 171
AWS IoT Greengrass V1 política de manutenção 172
AWS IoT Greengrass esquema de controle de versão 172

Fases do ciclo de vida do software principal AWS IoT Greengrass 173
Política de manutenção para AWS IoT Greengrass Software principal 173
Cronograma da fase de manutenção 174
Programação de suspensão 174
Política de suporte para funções do Lambda 175
Política de suporte para AWS IoT Testador de dispositivos para AWS IoT Greengrass V1 175
Cronograma do fim da manutenção 175
Fim da manutenção para AWS IoT Greengrass Imagens do Docker v1.x do software principal 45
Fim da manutenção para AWS IoT Greengrass Repositório APT v1.x do software principal 177
Fim da manutenção para AWS IoT Greengrass Software principal v1.11.x Snap 177
Começando com AWS IoT Greengrass 178
Selecione como começar a usar 178
Requisitos 181
Inscreva-se para um Conta da AWS 182
Crie um Conta da AWS 183
Início rápido: Configuração do dispositivo do Greengrass 183
Requisitos 184
Executar a configuração do dispositivo do Greengrass 185
Solução de problemas 188
Opções de configuração do dispositivo do Greengrass 189
Módulo 1: Configuração do ambiente para o Greengrass 200
Configurar um Raspberry Pi 201
Configurando uma instância do Amazon EC2 209
Configurar outros dispositivos 215
Módulo 2: Instalando o AWS IoT Greengrass Software principal 219
Provisão e AWS IoT coisa para usar como núcleo do Greengrass 220
Crie um grupo do Greengrass. 223
Instale e execute AWS IoT Greengrass no dispositivo principal 224
Módulo 3 (parte 1): funções Lambda em AWS IoT Greengrass 231
Crie e empacote uma função do Lambda 232
Configure a função Lambda para AWS IoT Greengrass 237
Implantar configurações de nuvem em um dispositivo de núcleo 241
Verificar se a função do Lambda está em execução no dispositivo de núcleo 242
Módulo 3 (parte 2): funções Lambda em AWS IoT Greengrass 243

Criar e empacotar a função do Lambda ... 244
Configure funções Lambda de longa duração para AWS IoT Greengrass ... 248
Testar funções de longa duração do Lambda ... 249
Teste as funções do Lambda sob demanda ... 252
Módulo 4: Interagindo com dispositivos clientes em um AWS IoT Greengrass grupo ... 256
Crie dispositivos cliente em um AWS IoT Greengrass grupo ... 258
Configurar assinaturas ... 261
Instale o AWS IoT Device SDK para Python ... 262
Testar comunicações ... 268
Módulo 5: Interagir com sombras de dispositivos ... 273
Configurar dispositivos e assinaturas ... 274
Fazer download de arquivos necessários ... 277
Testar comunicações (sincronizações de dispositivos desativadas) ... 278
Testar comunicações (sincronizações de dispositivos ativadas) ... 281
Módulo 6: Acessando outros AWS serviços ... 282
Configurar a função do grupo ... 284
Crie e configure a função do Lambda ... 286
Configurar assinaturas ... 289
Testar comunicações ... 290
Módulo 7: Simular a integração de segurança de hardware ... 292
Instale o SoftHSM ... 293
Configure o SoftHSM ... 293
Importar a chave privada ... 295
Configurar o núcleo do Greengrass ... 296
Testar a configuração ... 300
Consulte também ... 300
Atualizações OTA do software AWS IoT Greengrass Core ... 302
Requisitos ... 302
Permissões do IAM para atualizações OTA ... 304
Considerações ... 306
Agente de atualizações OTA do Greengrass ... 307
Integração com sistemas Init ... 308
Respawn gerenciado com atualizações OTA ... 309
Criar uma atualização OTA ... 311
CreateSoftwareUpdateJob API ... 314
Implemente AWS IoT Greengrass grupos ... 317

Implantar grupos (console) ... 318
Implantar grupos (API) ... 320
Obter o ID do grupo ... 321
Visão geral do modelo de objeto de grupo ... 322
Groups (Grupos) ... 323
Versões do grupo ... 323
Componentes do grupo ... 324
Atualizar grupos ... 325
Consulte também ... 326
Obter notificações de implantação ... 327
Evento de alteração de status da implantação do grupo ... 328
Pré-requisitos para criar regras EventBridge ... 329
Configurar notificações de implantação (console) ... 330
Configurar notificações de implantação (CLI) ... 331
Configurar notificações de implantação (CloudFormation) ... 332
Consulte também ... 332
Redefinir implantações ... 332
Redefina as implantações a partir do AWS IoT console ... 333
Redefina as implantações com o AWS IoT Greengrass solicitações de ... 334
Consulte também ... 335
Criar implantações em massa ... 335
Pré-requisitos ... 336
Criar e fazer upload do arquivo de entrada de implementação em massa ... 336
Criar e configurar uma função de execução do IAM para implantação em lote ... 339
Permitir o acesso da função de execução ao bucket do S3 ... 342
Implantar os grupos ... 343
Teste a implantação ... 346
Solução de problemas de implantações em massa ... 348
Consulte também ... 350
Execute funções locais do Lambda ... 351
SDKs ... 352
Migrando funções do Lambda baseadas em nuvem ... 355
Fazer referência a funções por alias ou por versão ... 356
Controlando a execução da função do Lambda do Greengrass ... 357
Group-specific definições de configuração ... 357
Executar uma função do Lambda como raiz ... 361

Considerações ao escolher a conteinerização de função do Lambda 363
Definir a identidade de acesso padrão para as funções do Lambda em um grupo 367
Definir a conteinerização padrão para funções do Lambda em um grupo 368
Fluxos de comunicação 370
Comunicação usando mensagens MQTT 370
Outros fluxos de comunicação 371
Recuperar o tópico de entrada (ou assunto) 371
Configuração do ciclo de vida 374
Executáveis do Lambda 375
Crie um executável do Lambda 376
Executar AWS IoT Greengrass em um contêiner Docker 378
Pré-requisitos 380
Obtenha a imagem do AWS IoT Greengrass contêiner do Amazon ECR 381
Criar e configurar o núcleo e o grupo do Greengrass 385
Execute AWS IoT Greengrass localmente 385
Configurar a conteinerização "Sem contêiner" para o grupo 389
Implantar funções do Lambda para o contêiner do Docker 389
(Opcional) Implantar dispositivos cliente que interagem com o Greengrass no contêiner do Docker 390
Parando o AWS IoT Greengrass Contêiner do docker 390
Solução de problemas AWS IoT Greengrass em um contêiner Docker 390
Acesso aos recursos locais 394
Tipos de recursos compatíveis 394
Requisitos 396
Recursos de volume no diretório /proc 396
Permissão de acesso a arquivo do proprietário do grupo 397
Consulte também 397
Usar a CLI 397
Criar recursos locais 398
Criar a função do Greengrass 400
Adicionar a função do Lambda ao grupo 401
Solução de problemas 404
Utilizar o console 405
Pré-requisitos 405
Crie um pacote de implantação para a função do Lambda 406
Crie e publique uma função do Lambda 407

Adicionar a função do Lambda ao grupo 410
Adicionar um recurso local ao grupo 411
Adicionar assinaturas ao grupo 412
Implantar o grupo 413
Testar o acesso aos recursos locais 414
Executar a inferência de machine learning 418
Como AWS IoT Greengrass A inferência de ML funciona 418
Recursos de machine learning 419
Fontes de modelo compatíveis 419
Requisitos 422
Tempos de execução e bibliotecas para inferência de ML 422
SageMaker Tempo de execução de aprendizado profundo do AI Neo 423
Versionamento do MXNet 423
MXNet no Raspberry Pi 424
TensorFlow limitações de serviço de modelos no Raspberry Pi 424
Acesse os recursos de machine learning 425
Acessar as permissões para recursos de machine learning 425
Definição das permissões de acesso às funções do Lambda (console) 428
Definindo as permissões de acesso para funções do Lambda (API) 429
Acessando os recursos de machine learning do código de função do Lambda 432
Solução de problemas 433
Consulte também 435
Como configurar a inferência de machine learning 435
Pré-requisitos 436
Configurar o Raspberry Pi 437
Instalar a estrutura de trabalho do MXNet. 439
Criar um pacote de modelo 440
Crie e publique uma função do Lambda 440
Adicionar a função do Lambda ao grupo 443
Adicionar recursos ao grupo 445
Adicionar uma assinatura ao grupo 448
Implantar o grupo 448
Testar o aplicativo 450
Próximas etapas 454
Como configurar um Intel Atom 454
Configuração NVIDIA Jetson TX2 457

Como configurar a inferência otimizada de Machine Learning 462
Pré-requisitos 436
Configurar o Raspberry Pi 464
Instalar o tempo de execução de aprendizagem profunda do Neo 466
Criar uma função do Lambda de inferência 467
Adicionar a função do Lambda ao grupo 470
Adicionar um recurso Neo-optimized modelo ao grupo 472
Adicionar um recurso de dispositivo de câmera ao grupo 475
Adicionar assinaturas ao grupo 476
Implantar o grupo 476
Testar o exemplo 478
Como configurar um Intel Atom 479
Configuração NVIDIA Jetson TX2 482
Solução de problemas AWS IoT Greengrass Inferência de ML 451
Próximas etapas 488
Gerenciamento de fluxos de dados 489
Fluxo de trabalho do gerenciamento de streams 490
Requisitos 492
Segurança de dados 493
Segurança de dados locais 493
Autenticação de cliente 494
Consulte também 495
Configurar o gerenciador de fluxo 495
Parâmetros do gerenciador de fluxo 495
Definir configurações (console) 498
Definir configurações (CLI) 501
Consulte também 511
Use StreamManagerClient para trabalhar com streams 511
Criar stream de mensagens 513
Anexar mensagem 517
Ler Mensagens 524
Listar fluxos 526
Descrever stream de mensagens 528
Atualize o fluxo de mensagens 530
Excluir stream de mensagens 534
Consulte também 536

Exporte configurações para suporte Nuvem AWS destinos 536
Exportar streams de dados (console) 553
Pré-requisitos 553
Crie um pacote de implantação para a função do Lambda 556
Criar uma função do Lambda 559
Adicionar uma função ao grupo 562
Habilitar o gerenciador de fluxo 562
Configurar o registro em log local 563
Implantar o grupo 563
Teste a aplicação 565
Consulte também 566
Exportar streams de dados (CLI) 566
Pré-requisitos 567
Crie um pacote de implantação para a função do Lambda 570
Criar uma função do Lambda 574
Criar uma definição e uma versão de função 576
Criar uma definição e versão do logger 578
Obter o ARN da sua versão de definição de núcleo 579
Criar uma versão de grupo 580
Crie um implantação do 581
Teste a aplicação 582
Consulte também 584
Implantar segredos no núcleo 585
Criptografia de segredos 586
Requisitos 588
Especificar a chave privada para criptografia de segredos 589
Permitir AWS IoT Greengrass para obter valores secretos 590
Consulte também 592
Trabalhar com recursos de segredos 592
Criar e gerenciar segredos 592
Usar segredos locais 597
Como criar um recurso de segredo (console) 600
Pré-requisitos 601
Crie um segredo do Secrets Manager 602
Adicionar um recurso de segredo a um grupo 603
Crie um pacote de implantação para a função do Lambda 604

Criar uma função do Lambda 606
Adicionar a função ao grupo 608
Anexar o recurso de segredo à função 610
Adicionar assinaturas ao grupo 610
Implantar o grupo 611
Testar a função do Lambda 612
Consulte também 613
Integrar a serviços e protocolos usando conectores 614
Requisitos 615
Usar conectores do Greengrass 616
Parâmetros de configuração 618
Parâmetros usados para acessar recursos de grupo 619
Atualizar parâmetros de conector 619
Entradas e saídas 620
Tópicos de entrada 620
Suporte a conteinerização 621
Atualizar a versões do conector 622
Registro em log 623
Conectores do Greengrass fornecidos pela AWS 623
CloudWatch Métricas 627
Device Defender 643
Implantação de aplicativo do Docker 650
IoT Analytics 693
Adaptador do protocolo IP Ethernet IoT 694
IoT SiteWise 699
Kinesis Firehose 716
Feedback do ML 734
Classificação de imagens do ML 752
Detecção de objetos do ML 779
Modbus-RTU Adaptador de protocolo 797
Modbus-TCP Adaptador de protocolo 815
Raspberry Pi GPIO 821
Fluxo serial 832
ServiceNow MetricBase Integração 845
SNS 861
Integração Splunk 873

Notificações Twilio ... 887
Conceitos básicos de conectores (console) ... 904
Pré-requisitos ... 906
Crie um segredo do Secrets Manager ... 907
Adicionar um recurso de segredo a um grupo ... 908
Adicionar um conector ao grupo ... 909
Crie um pacote de implantação para a função do Lambda ... 909
Criar uma função do Lambda ... 911
Adicionar uma função ao grupo ... 913
Adicionar assinaturas ao grupo ... 913
Implantar o grupo ... 914
Testar a solução ... 916
Consulte também ... 917
Conceitos básicos de conectores (CLI) ... 917
Pré-requisitos ... 919
Crie um segredo do Secrets Manager ... 921
Criar uma definição e uma versão de recurso ... 921
Criar uma definição e uma versão de conector ... 922
Crie um pacote de implantação para a função do Lambda ... 924
Criar uma função do Lambda ... 925
Criar uma definição e uma versão de função ... 927
Criar uma definição e uma versão de assinatura ... 928
Criar uma versão de grupo ... 929
Crie um implantação do 931
Testar a solução ... 932
Consulte também ... 934
API RESTful de descoberta do Greengrass ... 935
Solicitação ... 935
Resposta ... 936
Autorização de descoberta ... 937
Exemplos de documentos de resposta de descoberta ... 937
Segurança ... 940
Visão geral da AWS IoT Greengrass segurança ... 941
Fluxo de trabalho de conexão de dispositivo ... 942
Configurar AWS IoT Greengrass security ... 943
Entidades principais de segurança ... 944

Assinaturas gerenciadas no fluxo de trabalho de mensagens MQTT 947
Suporte a pacotes de criptografia TLS 948
Proteção de dados 950
Criptografia de dados 952
Integração de segurança de hardware 955
Autorização e autenticação do dispositivo 976
X.509 certificados 977
AWS IoT políticas 979
AWS IoT Política mínima para o dispositivo principal 982
Gerenciamento de identidade e acesso 986
Público 987
Autenticação com identidades 987
Gerenciar o acesso usando políticas 988
Consulte também 990
Como AWS IoT Greengrass funciona com o IAM 990
Função de serviço do Greengrass 998
Função do grupo do Greengrass. 1007
Cross-service prevenção delegada confusa 1017
Identity-based exemplos de políticas 1018
Solução de problemas de identidade e acesso 1021
Validação de conformidade 1024
Resiliência 1025
Segurança da infraestrutura 1026
Análise de configuração e vulnerabilidade 1026
Endpoints da VPC (AWS PrivateLink) 1027
Considerações para AWS IoT Greengrass Endpoints da VPC 1028
Crie uma interface VPC endpoint para AWS IoT Greengrass operações do plano de controle 1029
Criação de uma política de VPC endpoint para AWS IoT Greengrass 1029
Práticas recomendadas de segurança 1030
Conceder o mínimo possível de permissões 1030
Não codificar credenciais em funções do Lambda 1030
Não registrar em log informações confidenciais 1031
Criar assinaturas direcionadas 1031
Manter o relógio do dispositivo sincronizado 1031
Gerenciar a autenticação de dispositivos com o núcleo do Greengrass 1032

Consulte também 1033
Registro em log e monitoramento 1034
Ferramentas de monitoramento 1034
Consulte também 1035
Monitoramento com logs do AWS IoT Greengrass 1035
Acessando CloudWatch registros 1035
Acessar os logs do sistema de arquivos 1038
Configuração de registro em log padrão 1039
Configurar o registro para AWS IoT Greengrass 1039
Limitações de registro em log 1042
CloudTrail troncos 1044
Registro em log AWS IoT Greengrass Chamadas de API com AWS CloudTrail 1044
AWS IoT Greengrass informações em CloudTrail 1045
Noções básicas AWS IoT Greengrass entradas do arquivo de log 1046
Consulte também 1049
Coletar dados de telemetria da integridade do sistema 1049
Definindo configurações de telemetria 1052
Fazendo a assinatura para receber dados de telemetria 1057
Solução de problemas AWS IoT Greengrass telemetria 1063
Chamada da API de verificação de integridade local 1064
Obter informações de integridade sobre todos os operadores 1064
Obtenha informações de integridade sobre determinados operadores 1065
Informações de integridade dos operadores 1068
Marcar os recursos do Greengrass 1071
Conceitos Básicos de Tags 1071
Suporte à marcação (console) 1071
Suporte à atribuição de tags (API) 1072
Utilização de tags com políticas do IAM 1074
Políticas de exemplo do IAM 1075
Consulte também 1077
CloudFormation suporte para AWS IoT Greengrass 1078
Criar recursos do 1078
Implantar recursos. 1079
Exemplo de modelo 1080
Compatível Região da AWS s 1093
Utilizar AWS IoT Testador de dispositivos para AWS IoT Greengrass V1 1094

AWS IoT Greengrass suíte de qualificação ... 1094
Conjuntos de teste personalizados ... 1095
Versões compatíveis do AWS IoT Testador de dispositivos para AWS IoT Greengrass V1 ... 1095
Versões IDT não suportadas para for AWS IoT Greengrass ... 1096
Use o IDT para executar o AWS IoT Greengrass suíte de qualificação ... 1101
Versões do pacote de testes ... 1102
Descrições dos grupos de testes ... 1103
Pré-requisitos ... 1108
Configure seu dispositivo para executar testes de IDT ... 1116
Defina as configurações do IDT ... 1138
Execute o AWS IoT Greengrass suíte de qualificação ... 1154
Noções básicas de resultados e logs ... 1159
Usar o IDT para desenvolver e executar os próprios pacotes de testes ... 1163
Fazer download da versão mais recente do IDT para o AWS IoT Greengrass ... 1108
Fluxo de trabalho de criação de pacotes de teste ... 1164
Tutorial: compile e execute o pacote de amostra de teste de IDT ... 1164
Tutorial: desenvolva um pacote de testes de IDT simples ... 1170
Crie arquivos de configuração do pacote de testes do IDT ... 1179
Configure a máquina de estados do IDT ... 1187
Criar executáveis de casos de teste do IDT ... 1212
Use o contexto do IDT ... 1219
Definir configurações para executores de teste ... 1224
Depure e execute conjuntos de teste personalizados ... 1236
Analise os resultados e logs dos testes do IDT ... 1239
Métricas de uso do IDT ... 1245
IDT para AWS IoT Greengrass solução de problemas ... 1250
Códigos de erro ... 1250
Resolvendo o IDT para AWS IoT Greengrass erros ... 1274
Política de suporte para AWS IoT Testador de dispositivos para AWS IoT Greengrass V1 ... 1279
Solução de problemas ... 1280
AWS IoT Greengrass Questões principais ... 1280
Erro: O arquivo de configuração não contém o CaPath, CertPath ou KeyPath. O processo do daemon Greengrass com [pid = < pid>] morreu. ... 1282
Erro: Falha ao analisar/<greengrass-root/.json>. config/config ... 1283
Erro: Ocorreu um erro ao gerar a configuração TLS: ErrUnknownURIScheme ... 1283

Erro: Falha ao iniciar o tempo de execução: não foi possível iniciar os operadores: o teste de contêiner expirou. 1284
Erro: Falha ao invocar PutLogEvents no Cloudwatch local, LogGroup:/GreengrassSystem/connection_manager, erro:: falha na solicitação de envio causada por RequestError: Post http://<path>/cloudwatch/logs/: dial tcp < address: getsockopt: connection recused, response>: {}. 1284
Erro: Não foi possível criar o servidor devido a: falha ao carregar o grupo: chmod/<greengrass-root>/ggc/deployment/:aws:lambda: region: > < account-id:function: function-namelambda/arn: version/<file-name: arquivo ou diretório não existe. > < > < > < > 1285
A ferramenta AWS IoT Greengrass O software principal não inicia depois que você deixou de ser executado sem conteinerização para ser executado em um contêiner do Greengrass. 1285
Erro: Tamanho do spool deve ser pelo menos 262.144 bytes. 1285
Erro: [ERRO] – erro de mensagens na nuvem: Ocorreu um erro ao tentar publicar uma mensagem. {"errorString": "operation timed out"} 1286
Erro: container_linux.go:344: iniciar o processo do contêiner causou “process_linux.go:424: a inicialização do contêiner causou\” permissão negada\\\ "\"”. rootfs_linux.go:64: mounting \\\"/greengrass/ggc/socket/greengrass_ipc.sock\\\" to rootfs \\\"/greengrass/ggc/packages/<version>/rootfs/merged\\\" at \\\"/greengrass_ipc.sock\\\" caused \\\"stat /greengrass/ggc/socket/greengrass_ipc.sock: 1287
Erro: daemon do Greengrass em execução com PID: process-id. < > Não foi possível iniciar alguns componentes do sistema. Verifique "runtime.log" para erros. 1287
O shadow do dispositivo não sincroniza com a nuvem. 1023
ERRO: não foi possível aceitar a conexão TCP. accept tcp [::]:8000: accept4: muitos arquivos abertos. 1288
Erro: Erro de execução do tempo de execução: não foi possível iniciar o contêiner lambda. container_linux.go:259: o início do processo do contêiner causou "process_linux.go:345: init do contêiner causou \"rootfs_linux.go:50: rootfs de preparação causaram \\\"permissão negada\\\"\"". 1288
Aviso: [WARN] - [5] GK Remote: Erro ao recuperar dados da chave pública: ErrPrincipalNotConfigured: a chave privada para MqttCertificate não está definida. 1288
Erro: permissão negada ao tentar usar a função arn:aws:iam::<account-id>:role/<role-name> para acessar o URL s3. https://<region>-greengrass-updates.s3.<region>.amazonaws.com/core/<architecture>/greengrass-core-<distribution-version>.tar.gz 1022

A ferramenta AWS IoT Greengrass o núcleo está configurado para usar um proxy de rede e sua função Lambda não pode fazer conexões de saída. 1289
O núcleo está em um loop infinito de desconexão e conexão. O arquivo runtime.log contém uma série contínua de entradas de conexão e desconexão. 1290
Erro: não foi possível iniciar o contêiner do lambda. container_linux.go:259: o início do processo do contêiner causou "process_linux.go:345: o init do contêiner causou \"rootfs_linux.go:62: montagem de \\\"proc\\\" para rootfs \\\" 1291
[ERROR] -erro de execução em tempo de execução: não é possível iniciar o contêiner lambda. {"errorString": “falha ao inicializar montagens de contêiner: falha ao mascarar a raiz do greengrass no diretório superior de sobreposição: falha ao criar o dispositivo de máscara no diretório < ggc-path: o arquivo existe"} > 1291
[ERRO] - Falha na implantação. {"deploymentId": "<deployment-id > “, “errorString”: “>falha no processo de teste do contêiner com pid < pid: estado do processo do contêiner: status de saída 1"} 1292
Erro: [ERROR] -erro de execução em tempo de execução: não é possível iniciar o contêiner lambda. {"errorString": “falha ao inicializar montagens de contêiner: falha ao criar sobreposição fs para contêiner: sobreposição de montagem em//packages/ ggc-version/ falha: rootfs/merged falha na montagem com args source=\" no_source\” dest=\”greengrass/ggc/>/packages/ < ggc-version/\” fstype=\ "overlay\” flags=\ "0\” data= “lowerdir=/ /packages/ ggc-version greengrass/ggc /dns:/, upperdir=/ /packages/ < ggc-version>/rootfs/merged, workdir=/ /packages/ ggc-version greengrass/ggc < > greengrass/ggc < > rootfs/upper greengrass/ggc < >/rootfs/work\”: muitos níveis de links simbólicos "} 1293
Erro: [DEPURAÇÃO] - Falha ao obter rotas. Descarte da mensagem. 1294
Erro: [Errno 24] Muitas < funções lambda abertas>, [Errno 24] Muitos arquivos abertos 1294
Erro: o servidor ds falhou ao iniciar a escuta do soquete: listen unix < ggc-path>/ggc/socket/ greengrass_ipc.sock: bind: argumento inválido 1294
[INFO] (Copiadora) aws.greengrass. StreamManager: robusto. Causado por: com.fasterxml.jackson.databind. JsonMappingException: Instantâneo excede o instante mínimo ou máximo 1295
Erro de GPG: https://dnw9lb6lzp2d8.cloudfront.net estável: as seguintes assinaturas eram inválidas InRelease: EXPKEYSIG 68D644ABD2327D47 Chave mestra do AWS Greengrass 1295
Problemas de implantação 1296
Sua implantação atual não funciona e você deseja reverter para uma implantação anterior que funcione. 1297
Você verá o erro de implantação 403 Forbidden nos logs. 1299

Ocorre um ConcurrentDeployment erro quando você executa o comando create-deployment pela primeira vez. 1300
Erro: O Greengrass não está autorizado a assumir o perfil de serviço associado a essa conta ou o erro: Falha: perfil de serviço TES não está associado a essa conta. 1022
Erro: não é possível executar a etapa de download na implantação. erro ao fazer download: error ao fazer download do arquivo de definição de grupo: ... x509: o certificado expirou ou ainda não é válido 1300
A implantação não é concluída. 1300
Erro: Não foi possível encontrar executáveis java ou java8, ou o erro: Falha na identificação de < implantação > do tipo id de grupo NewDeployment para ID de < grupo: > falha na inicialização do trabalhador com ID de < trabalho, motivo pelo qual a versão instalada do Java deve ser maior > ou igual a 8 1301
A implantação não é concluída, e runtime.log contém várias entradas "esperar 1s para o contêiner ser interrompido". 1302
A implantação não termina e `runtime.log` contém “[ERROR] -Erro de implantação do Greengrass: falha ao relatar o status de implantação à nuvem {" deploymentId”: "<deployment-id > “, “errorString”: “Falha ao iniciar PUT, endpoint: https://deployment-status, error: Put https://<deployment-status>: proxyconnect tcp: x509: certificado assinado por autoridade < > desconhecida"}” 1302
Erro: > Falha no < id de implantação > do tipo id NewDeployment de < grupo Erro: Erro ao processar. A configuração do grupo é inválida: 112 ou [119 0] não têm permissão rw no caminho file:. < > 1303
Erro: < list-of-function-arns > está configurado para ser executado como root, mas o Greengrass não está configurado para executar funções do Lambda com permissões de root. 1304
Erro: ID de < implantação > do tipo NewDeployment de ID de < grupo > falhou erro: erro de implantação do Greengrass: não foi possível executar a etapa de download na implantação. erro durante o processamento: não foi possível carregar o arquivo de grupo baixado: não foi possível encontrar o UID com base no nome de usuário, nome de usuário: ggc_user: usuário desconhecido ggc_user. 1304
Erro: [ERROR] -erro de execução em tempo de execução: não é possível iniciar o contêiner lambda. {"errorString”: “falha ao inicializar montagens de contêiner: falha ao mascarar a raiz do greengrass no diretório superior de sobreposição: falha ao criar o dispositivo de máscara no diretório < ggc-path: o arquivo existe"} > 1304
Erro: Falha na identificação de < implantação > do tipo NewDeployment ID de < grupo erro: falha no início do processo: > falha no início do processo: container_linux.go:259: iniciar o

processo de contêiner causou “process_linux.go:250: a execução do processo exec setns para init causou\" wait: nenhum processo filho\ "”. 1305
Erro: [WARN] -MQTT [client] dial tcp: lookup host-prefix -ats.iot. < > <region > .amazonaws.com: esse host não existe... [ERRO] -Erro de implantação do Greengrass: falha ao relatar o status da implantação para a nuvem... net/http: solicitação cancelada enquanto aguarda a conexão (Client.Timeout excedida enquanto aguarda os cabeçalhos) 1305
Problemas para criar grupo/criar função 1306
Erro: Sua configuração 'IsolationMode' para o grupo é inválida. 1307
Erro: Sua configuração 'IsolationMode' para função com arn < function-arn é > inválida. ... 1307
Erro: a MemorySize configuração da função com arn < function-arn não > é permitida em =. IsolationMode NoContainer 1307
Erro: acessar a configuração do Sysfs para a função com arn < function-arn > não é permitido em =. IsolationMode NoContainer 1307
Erro: a MemorySize configuração da função com arn < function-arn > é necessária em =. IsolationMode GreengrassContainer 1308
Erro: A função < function-arn > se refere ao recurso do tipo < resource-type > que não é permitido em =. IsolationMode NoContainer 1308
Erro: A configuração de execução da função com arn < function-arn > não é permitida. 1308
Problemas de descoberta 1308
Erro: o dispositivo é membro de muitos grupos; os dispositivos não podem estar em mais de 10 grupos 1309
Problemas com recursos de machine learning 1309
InvalidMLModelOwner - GroupOwnerSetting é fornecido no recurso do modelo ML, mas GroupOwner ou não GroupPermission está presente 434
NoContainer A função não pode configurar a permissão ao anexar recursos de aprendizado de máquina. <function-arn > se refere ao recurso de aprendizado de máquina < resource-id com permissão na política de acesso > a recursos. < ro/rw > 434
A função < function-arn > se refere ao recurso de aprendizado de máquina < resource-id sem permissão em ambos > os recursos. ResourceAccessPolicy OwnerSetting 434
A função < function-arn > se refere ao recurso de aprendizado de máquina < resource-id > com permissão\ "rw\”, enquanto a configuração do proprietário do recurso permite GroupPermission apenas\ "ro\”. 434
NoContainer A função < function-arn > se refere aos recursos do caminho de destino aninhado. 435
O Lambda < function-arn > obtém acesso ao recurso < resource-id compartilhando o mesmo id > do proprietário do grupo 435

AWS IoT Greengrass núcleo dos problemas do Docker 1311
Erro: Opções desconhecidas: -no-include-email 391
Aviso: IPv4 está desabilitado. As redes não funcionarão. 391
Erro: Um firewall está bloqueando o compartilhamento de arquivos entre janelas e os contêineres. 391
Erro: Ocorreu um erro (AccessDeniedException) ao chamar a GetAuthorizationToken operação: Usuário: não arn:aws:iam::<account-id>:user/<user-name> está autorizado a executar: ecr: GetAuthorizationToken on resource: * 391
Erro: Não é possível criar um contêiner para o serviço do greengrass: Conflito. O nome do contêiner "/aws-iot-greengrass" já está em uso. 1313
Erro: [FATAL]-Falha ao redefinir o namespace de montagem do thread devido a um erro inesperado: "operação não permitida". Para manter a consistência, o GGC travará e precisará ser reiniciado manualmente. 1313
Solução de problemas com logs 1314
Solução de problemas de armazenamento 1315
Solução de problemas com mensagens 1316
Solução de problemas de intervalo de sincronização de shadow 1316
Verificar AWS Re: post 1317
Histórico do documento 1318
Atualizações anteriores 1343

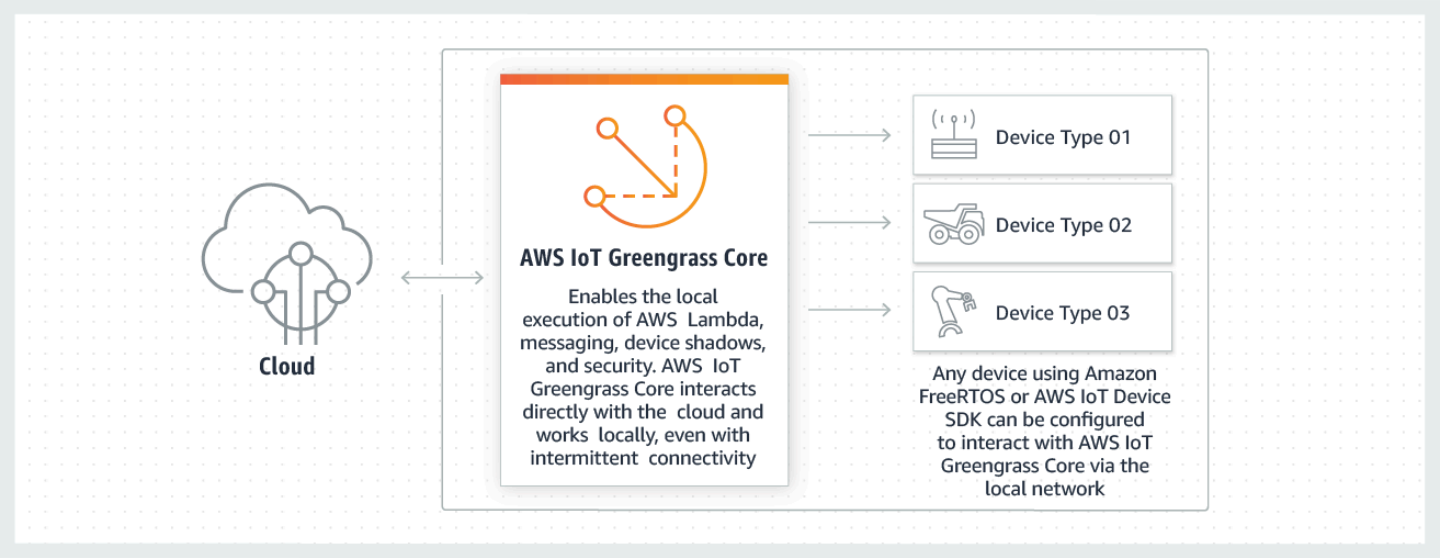
Aviso de fim do suporte: em 7 de outubro de 2026, AWS o suporte para o. AWS IoT Greengrass Version 1 Depois de 7 de outubro de 2026, você não poderá mais acessar os AWS IoT Greengrass V1 recursos. Para obter mais informações, visite Migrar de AWS IoT Greengrass Version 1.

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.

O que é AWS IoT Greengrass?

AWS IoT Greengrass é um software que estende os recursos da nuvem para dispositivos locais. Isso permite que os dispositivos coletem e analisem dados mais próximos da fonte de informações, reajam de maneira autônoma a eventos locais e se comuniquem com segurança uns com os outros em redes locais. Os dispositivos locais também podem se comunicar com segurança AWS IoT Core e exportar dados de IoT para o. Nuvem AWS AWS IoT Greengrass os desenvolvedores podem usar AWS Lambda funções e conectores pré-construídos para criar aplicativos sem servidor que são implantados em dispositivos para execução local.

O diagrama a seguir mostra a arquitetura básica do AWS IoT Greengrass.



AWS IoT Greengrass possibilita que os clientes criem dispositivos de IoT e lógica de aplicativos. Especificamente, AWS IoT Greengrass fornece gerenciamento baseado em nuvem da lógica do aplicativo que é executado em dispositivos. As funções do Lambda e os conectores implantados localmente são acionados por eventos locais, mensagens na nuvem ou outras origens.

Em AWS IoT Greengrass, os dispositivos se comunicam com segurança em uma rede local e trocam mensagens entre si sem precisar se conectar à nuvem. AWS IoT Greengrass fornece um gerenciador de pub/sub mensagens local que pode armazenar mensagens em buffer de forma inteligente se a conectividade for perdida, para que as mensagens de entrada e saída para a nuvem sejam preservadas.

AWS IoT Greengrass protege os dados do usuário:

- Por meio da autenticação e autorização seguras dos dispositivos.

- Por meio da conectividade segura na rede local.
- Entre dispositivos locais e a nuvem.

As credenciais de segurança de dispositivos funcionam em um grupo até que sejam revogadas, mesmo se a conectividade com a nuvem for interrompida, para que os dispositivos possam continuar se comunicando com segurança localmente.

AWS IoT Greengrass fornece atualizações seguras e remotas das funções do Lambda.

AWS IoT Greengrass consiste em:

- Distribuições de software
  - AWS IoT Greengrass Software principal
  - AWS IoT Greengrass SDK principal
- Serviços em nuvem
  - AWS IoT Greengrass API
- Recursos
  - Runtime do Lambda
  - Implementação de sombras
  - Gerenciador de mensagens
  - Gerenciamento de grupos
  - Serviço de descoberta
  - Over-the-air agente de atualização
  - Gerenciador de fluxos
  - Acesso aos recursos locais
  - Inferência de machine learning local
  - Secrets manager local
  - Conectores com a integração incorporada com serviços, protocolos e software

Tópicos

- AWS IoT Greengrass Software principal
- AWS IoT Greengrass groups
- Dispositivos em AWS IoT Greengrass

- SDKs
- Plataformas compatíveis e requisitos
- AWS IoT Greengrass downloads
- Deixe seu comentário
- Instale o AWS IoT Greengrass Software principal
- Configure o AWS IoT Greengrass core

AWS IoT Greengrass Software principal

O software AWS IoT Greengrass Core fornece as seguintes funcionalidades:

- Implantação e execução local de conectores e funções do Lambda.
- Processe fluxos de dados localmente com exportações automáticas para o. Nuvem AWS
- Mensagens MQTT pela rede local entre dispositivos, conectores e funções do Lambda usando assinaturas gerenciadas.
- Mensagens MQTT entre AWS IoT dispositivos, conectores e funções do Lambda usando assinaturas gerenciadas.
- Conexões seguras entre dispositivos e o Nuvem AWS uso da autenticação e autorização do dispositivo.
- Sincronização da shadow local de dispositivos. As shadows podem ser configuradas para serem sincronizadas com a Nuvem AWS.
- Acesso controlado ao dispositivo local e recursos de volume.
- Implantação de modelos de machine learning treinados em nuvem para executar inferência local.
- Detecção automática de endereço IP que permite aos dispositivos descobrirem o dispositivo de núcleo do Greengrass.
- Implantação central de configuração de grupo nova ou atualizada. Depois que os dados de configuração forem obtidos por download, o dispositivo de núcleo será reiniciado automaticamente.
- Atualizações seguras de software por rede sem fio (OTA) das funções do Lambda definidas pelo usuário.
- Armazenamento criptografado e seguro de segredos locais e acesso controlado por conectores e funções do Lambda.

AWS IoT Greengrass as instâncias principais são configuradas por meio de AWS IoT Greengrass APIs que criam e atualizam definições de AWS IoT Greengrass grupos armazenadas na nuvem.

AWS IoT Greengrass Versões principais do software

AWS IoT Greengrass fornece várias opções para instalar o software AWS IoT Greengrass Core, incluindo arquivos de download tar.gz, um script de início rápido e `apt` instalações em plataformas Debian suportadas. Para obter mais informações, consulte the section called “Instale o AWS IoT Greengrass Software principal”.

As guias a seguir descrevem o que há de novo e alterado nas versões AWS IoT Greengrass do software Core.

GGC v1.11

1.11.6

Correções de bugs e melhorias:

- Aprimoramento da resiliência se ocorrer uma queda repentina de energia durante uma implantação.
- Foi corrigido um problema em que a corrupção de dados do gerenciador de streams poderia impedir a inicialização do software AWS IoT Greengrass Core.
- Correção de um problema em que novos dispositivos cliente não conseguiam se conectar ao núcleo em determinados cenários.
- Correção de um problema em que os nomes do fluxo do gerenciador de fluxo não podiam conter `.log`.

1.11.5

Correções de bugs e melhorias:

- Melhorias na performance geral e correções de erros.

1.11.4

Correções de bugs e melhorias:

- Foi corrigido um problema com o gerenciador de streams que impedia atualizações para o software AWS IoT Greengrass Core v1.11.3. Se você estiver usando o gerenciador de stream para exportar dados para a nuvem, agora você pode usar uma atualização OTA

para atualizar uma versão anterior v1.x do software AWS IoT Greengrass Core para a v1.11.4.

- Melhorias na performance geral e correções de erros.

1.11.3

Correções de bugs e melhorias:

- Foi corrigido um problema que fazia com que o software AWS IoT Greengrass Core executado em um piscar de olhos em um dispositivo Ubuntu parasse de responder após uma perda repentina de energia no dispositivo.
- Correção de um problema que causava atrasos na entrega de mensagens MQTT para funções do Lambda de longa duração.
- Correção de um problema que fazia com que as mensagens MQTT não fossem enviadas corretamente quando o valor `maxWorkItemCount` era definido como maior que `1024`.
- Correção de um problema que fazia com que o atendente de atualização OTA ignorasse o período `KeepAlive` do MQTT especificado na propriedade `keepAlive` no `config.json`.
- Melhorias na performance geral e correções de erros.

> **Important**
>
> Se você estiver usando o gerenciador de stream para exportar dados para a nuvem, não atualize para o software AWS IoT Greengrass Core v1.11.3 de uma versão anterior v1.x. Se você estiver ativando o stream manager pela primeira vez, é altamente recomendável instalar primeiro a versão mais recente do software AWS IoT Greengrass Core.

1.11.1

Correções de bugs e melhorias:

- Correção de um problema que causava um aumento na utilização da memória do gerenciador de fluxo.
- Correção de um problema que fazia com que o gerenciador de fluxo redefinisse o número de sequência do fluxo para `0` se o dispositivo principal do Greengrass estivesse desligado por mais tempo do que a vida útil (TTL) especificada dos dados do fluxo.
- Correção de um problema que impedia o gerenciador de fluxo de interromper corretamente as tentativas de exportar dados para o Nuvem AWS.

1.11.0

Novos atributos:

- Um agente de telemetria no núcleo do Greengrass coleta dados de telemetria local e os publica no. Nuvem AWS Para recuperar os dados de telemetria para processamento adicional, os clientes podem criar uma EventBridge regra da Amazon e assinar uma meta. Para obter mais informações, consulte Coleta de dados de telemetria de integridade do sistema dos AWS IoT Greengrass principais dispositivos.
- Uma API HTTP local retorna um instantâneo do estado atual dos processos de trabalho locais iniciados por AWS IoT Greengrass. Para obter mais informações, consulte Chamada da API de verificação de integridade local.
- Um gerenciador de streams exporta dados automaticamente para o Amazon S3 e. AWS IoT SiteWise

  Os novos parâmetros do gerenciador de fluxo permitem que você atualize os fluxos existentes e pause ou retome a exportação de dados.
- Suporte para execução Python 3.8.x core das funções do Lambda no núcleo.
- Uma nova propriedade `ggDaemonPort` no `config.json` usada para configurar o número da porta do IPC do núcleo do Greengrass. O número da porta padrão é 8000.

  Uma nova propriedade `systemComponentAuthTimeout` no `config.json` que você usa para configurar o tempo limite para a autenticação do IPC do núcleo do Greengrass. O padrão de tempo de saída é 5000 milissegundos.
- Aumentou o número máximo de AWS IoT dispositivos por AWS IoT Greengrass grupo de 200 para 2500.

  Aumento do número máximo de assinaturas por grupo, de 1.000 para 10.000.

  Para saber mais, consulte Endpoints e cotas do AWS IoT Greengrass.

Correções de bugs e melhorias:

- Otimização geral que pode reduzir a utilização da memória dos processos de serviço do Greengrass.
- Um novo parâmetro de configuração de runtime (`mountAllBlockDevices`) permite que o Greengrass use montagens bind para montar todos os dispositivos de blocos em um contêiner após configurar o OverlayFS. Esse atributo resolveu um problema que causava a falha na implantação do Greengrass se `/usr` não estivesse sob a hierarquia `/`.

- Corrigido um problema que causava falha no AWS IoT Greengrass núcleo `/tmp` se fosse um link simbólico.
- Correção de um problema que permitia que o atendente de implantação do Greengrass removesse artefatos do modelo de machine learning não utilizados da pasta `mlmodel_public`.
- Melhorias na performance geral e correções de erros.

Extended life versions

1,10.5

Correções de bugs e melhorias:

- Melhorias na performance geral e correções de erros.

1.10.4

Correções de bugs e melhorias:

- Foi corrigido um problema que fazia com que o software AWS IoT Greengrass Core executado em um piscar de olhos em um dispositivo Ubuntu parasse de responder após uma perda repentina de energia no dispositivo.
- Correção de um problema que causava atrasos na entrega de mensagens MQTT para funções do Lambda de longa duração.
- Correção de um problema que fazia com que as mensagens MQTT não fossem enviadas corretamente quando o valor `maxWorkItemCount` era definido como maior que `1024`.
- Correção de um problema que fazia com que o atendente de atualização OTA ignorasse o período `KeepAlive` do MQTT especificado na propriedade `keepAlive` no `config.json`.
- Melhorias na performance geral e correções de erros.

1.10.3

Correções de bugs e melhorias:

- Uma nova propriedade `systemComponentAuthTimeout` no `config.json` que você usa para configurar o tempo limite para a autenticação do IPC do núcleo do Greengrass. O padrão de tempo de saída é 5000 milissegundos.
- Correção de um problema que causava um aumento na utilização da memória do gerenciador de fluxo.

1.10.2

Correções de bugs e melhorias:

- Uma nova `mqttOperationTimeout` propriedade em config.json que você usa para definir o tempo limite para operações de publicação, assinatura e cancelamento de assinatura em conexões MQTT com. AWS IoT Core
- Melhorias na performance geral e correções de erros.

1.10.1

Correções de bugs e melhorias:

- O gerenciador de fluxos é mais resiliente a danos nos dados dos arquivos.
- Correção de um problema que causa uma falha de montagem sysfs em dispositivos que usam o kernel Linux 5.1 e posterior.
- Melhorias na performance geral e correções de erros.

1.10.0

Novos atributos:

- Um gerenciador de fluxo que processa fluxos de dados localmente e os exporta para a Nuvem AWS automaticamente. Esse atributo requer Java 8 no dispositivo de núcleo do Greengrass. Para obter mais informações, consulte Gerenciamento de fluxos de dados.
- Um novo conector de implantação do aplicativo Docker do Greengrass que executa um aplicativo Docker em um dispositivo de núcleo. Para obter mais informações, consulte the section called “Implantação de aplicativo do Docker”.
- Um novo SiteWise conector de IoT que envia dados de dispositivos industriais de OPC-UA servidores para propriedades de ativos em AWS IoT SiteWise. Para obter mais informações, consulte the section called “IoT SiteWise”.
- Funções do Lambda executadas sem conteinerização podem acessar recursos de machine learning no grupo do Greengrass. Para obter mais informações, consulte the section called “Acesse os recursos de machine learning”.
- Suporte para sessões persistentes do MQTT com AWS IoT. Para obter mais informações, consulte the section called “Sessões persistentes do MQTT com AWS IoT Core”.
- O tráfego MQTT local pode passar por uma porta diferente da padrão 8883. Para obter mais informações, consulte the section called “Porta MQTT para mensagens locais”.
- Novas opções da `queueFullPolicy` no SDK do AWS IoT Greengrass Core. para publicação confiável de mensagens a partir de funções do Lambda

- Suporte para executar funções do Lambda Node.js 12.x no núcleo.
- Over-the-air As atualizações (OTA) com integração de segurança de hardware podem ser configuradas com o OpenSSL 1.1.
- Melhorias na performance geral e correções de erros.

1.9.4

Correções de bugs e melhorias:

- Melhorias na performance geral e correções de erros.

1.9.3

Novos atributos:

- Suporte para ARMv6L. AWS IoT Greengrass O software principal v1.9.3 ou posterior pode ser instalado em distribuições Raspbian em arquiteturas ARMv6l (por exemplo, em dispositivos Raspberry Pi Zero).
- Atualizações OTA na porta 443 com ALPN. Os núcleos do Greengrass que usam a porta 443 para tráfego MQTT agora oferecem suporte a atualizações de software over-the-air (OTA). AWS IoT Greengrass usa a extensão TLS da Application Layer Protocol Network (ALPN) para habilitar essas conexões. Para obter mais informações, consulte Atualizações OTA do software AWS IoT Greengrass Core e the section called “Conectar-se à porta 443 ou por meio de um proxy de rede”.

Correções de bugs e melhorias:

- Corrige um erro introduzido no v1.9.0 que impedia as funções do Lambda do Python 2.7 de enviar cargas úteis binárias para outras funções do Lambda.
- Melhorias na performance geral e correções de erros.

1.9.2

Novos atributos:

- Suporte para OpenWrt. AWS IoT Greengrass O software principal v1.9.2 ou posterior pode ser instalado em OpenWrt distribuições com as arquiteturas Armv8 (AArch64) e ARMv7l. Atualmente, OpenWrt não oferece suporte à inferência de ML.

1.9.1

Correções de bugs e melhorias:

- Corrige um erro apresentado na v1.9.0 que interrompe mensagens do `cloud` que contêm caracteres curinga no tópico.

1.9.0

Novos atributos:

- Suporte para os tempos de execução Python 3.7 e Node.js 8.10 Lambda. As funções do Lambda que usam os tempos de execução do Python 3.7 e Node.js 8.10 agora podem ser executadas em um núcleo. AWS IoT Greengrass (AWS IoT Greengrass continua a oferecer suporte aos tempos de execução do Python 2.7 e Node.js 6.10.)
- Conexões MQTT otimizadas. O núcleo do Greengrass estabelece um número menor de conexões com o AWS IoT Core. Essa alteração pode reduzir os custos operacionais para cobranças com base no número de conexões.
- Chave de curva elíptica (EC) para o servidor MQTT local. O servidor MQTT local oferece suporte a chaves EC, além de chaves RSA. (O certificado do servidor MQTT tem uma assinatura SHA-256 RSA, independentemente do tipo de chave.) Para obter mais informações, consulte the section called “Entidades principais de segurança”.

Correções de bugs e melhorias:

- Melhorias na performance geral e correções de erros.

1.8.4

Corrigido um problema com sincronização de shadow e reconexão do gerenciador de certificados de dispositivo.

Melhorias na performance geral e correções de erros.

1.8.3

Melhorias no desempenho geral e correções de erros.

1.8.2

Melhorias no desempenho geral e correções de erros.

1.8.1

Melhorias no desempenho geral e correções de erros.

1.8.0

Novos atributos:

- Identidade de acesso padrão configurável para as funções do Lambda no grupo. Essa configuração no nível de grupo determina as permissões padrão que são usadas para executar as funções do Lambda. Você pode definir o ID de usuário, ID de grupo, ou ambos. As funções do Lambda individuais podem substituir a identidade de acesso padrão de seu grupo. Para obter mais informações, consulte the section called “Definir a identidade de acesso padrão para as funções do Lambda em um grupo”.
- Tráfego HTTPS pela porta 443. A comunicação HTTPS pode ser configurada para viajar pela porta 443, em vez da porta padrão 8443. Isso complementa o AWS IoT Greengrass suporte à extensão TLS da Application Layer Protocol Network (ALPN) e permite que todo o tráfego de mensagens do Greengrass — tanto MQTT quanto HTTPS — use a porta 443. Para obter mais informações, consulte the section called “Conectar-se à porta 443 ou por meio de um proxy de rede”.
- IDs de cliente com nomes previsíveis para AWS IoT conexões. Essa alteração permite oferecer suporte para AWS IoT Device Defender e Eventos de ciclo de vida do AWS IoT, para que você possa receber notificações para se conectar, desconectar, assinar e cancelar a assinatura de eventos. A nomenclatura previsível também facilita a criação lógica em torno de IDs de conexão (por exemplo, para criar modelos de política de assinatura com base em atributos de certificado). Para obter mais informações, consulte the section called “IDs de cliente para conexões MQTT com AWS IoT”.

Correções de bugs e melhorias:

- Corrigido um problema com sincronização de shadow e reconexão do gerenciador de certificados de dispositivo.
- Melhorias na performance geral e correções de erros.

1.7.1

Novos atributos:

- Os conectores do Greengrass fornecem integração integrada com infraestrutura local AWS, protocolos de dispositivos e outros serviços em nuvem. Para obter mais informações, consulte Integrar a serviços e protocolos usando conectores.
- AWS IoT Greengrass se estende AWS Secrets Manager aos principais dispositivos, o que disponibiliza suas senhas, tokens e outros segredos para conectores e funções do Lambda. Os segredos são criptografados em repouso e em trânsito. Para obter mais informações, consulte Implantar segredos no núcleo.
- Suporte para um hardware raiz da opção de segurança de confiança. Para obter mais informações, consulte the section called “Integração de segurança de hardware”.

- Configurações de permissões e isolamento que permitem que funções do Lambda sejam executadas sem contêineres do Greengrass e usem as permissões de um usuário e grupo especificado. Para obter mais informações, consulte the section called “Controlando a execução da função do Lambda do Greengrass”.
- Você pode executar AWS IoT Greengrass em um contêiner do Docker (no Windows, macOS ou Linux) configurando seu grupo do Greengrass para ser executado sem conteinerização. Para obter mais informações, consulte the section called “Executar AWS IoT Greengrass em um contêiner Docker”.
- O sistema de mensagens MQTT na porta 443 com Negociação de protocolo da camada de aplicativos (ALPN) ou conexão por meio de um proxy de rede. Para obter mais informações, consulte the section called “Conectar-se à porta 443 ou por meio de um proxy de rede”.
- O tempo de execução de aprendizado profundo SageMaker AI Neo, que suporta modelos de aprendizado de máquina que foram otimizados pelo compilador de aprendizado profundo SageMaker AI Neo. Para obter informações sobre runtime de aprendizado profundo do Neo, consulte the section called “Tempos de execução e bibliotecas para inferência de ML”.
- Suporte para Raspbian Stretch (27-06-2018) nos dispositivos de núcleo Raspberry Pi.

Correções de bugs e melhorias:

- Melhorias na performance geral e correções de erros.

Além disso, os seguintes atributos estão disponíveis nesta versão:

- O AWS IoT Device Tester for AWS IoT Greengrass, que você pode usar para verificar se a arquitetura da CPU, a configuração do kernel e os drivers funcionam com ele. AWS IoT Greengrass Para obter mais informações, consulte Utilizar AWS IoT Testador de dispositivos para AWS IoT Greengrass V1.
- Os pacotes de software AWS IoT Greengrass Core, AWS IoT Greengrass Core SDK e AWS IoT Greengrass Machine Learning SDK estão disponíveis para download na Amazon. CloudFront Para obter mais informações, consulte the section called “AWS IoT Greengrass downloads”.

1.6.1

Novos atributos:

- Executáveis do Lambda que executam código binário no núcleo do Greengrass. Use o novo AWS IoT Greengrass Core SDK para C para escrever executáveis do Lambda em C e C++. Para obter mais informações, consulte the section called “Executáveis do Lambda”.

- Cache de mensagens de armazenamento opcional que pode persistir entre reinicializações. Você pode definir as configurações de armazenamento para mensagens MQTT enfileiradas para processamento. Para obter mais informações, consulte the section called “Fila de mensagens MQTT”.
- Intervalo para repetição de reconexão máxima configurável quando o dispositivo básico é desconectado. Para obter mais informações, consulte a propriedade `mqttMaxConnectionRetryInterval` em the section called “AWS IoT Greengrass arquivo de configuração principal”.
- Acesso de recurso local para o diretório host/proc. Para obter mais informações, consulte Acesso aos recursos locais.
- Diretório de gravação configurável. O software AWS IoT Greengrass Core pode ser implantado em locais somente de leitura e leitura e gravação. Para obter mais informações, consulte the section called “Configurar um diretório de gravação”.

Correções de bugs e melhorias:

- Melhoria de desempenho para publicar mensagens no núcleo do Greengrass e entre dispositivos e o núcleo.
- Redução nos recursos de computação exigidos para processar logs gerados por funções do Lambda definidas pelo usuário.

1.5.0

Novos atributos:

- AWS IoT Greengrass A inferência de aprendizado de máquina (ML) geralmente está disponível. Você pode realizar a inferência de ML localmente nos dispositivos do AWS IoT Greengrass usando modelos que são criados e treinados na nuvem. Para obter mais informações, consulte Executar a inferência de machine learning.
- As funções do Lambda para Greengrass já oferecem suporte a dados binários como carga de entrada, além de JSON. Para usar esse recurso, você deve atualizar para a versão 1.1.0 do AWS IoT Greengrass Core SDK, que pode ser baixada na página de downloads do AWS IoT Greengrass Core SDK.

Correções de bugs e melhorias:

- Reduziu o espaço geral de memória.
- Melhorias no desempenho para enviar mensagens para a nuvem.
- Melhorias de desempenho e estabilidade para o atendente de download, o Device Certificate Manager e o atendente de atualização OTA.

- Correções de erros secundárias.

1.3.0

Novos atributos:

- Over-the-air Agente de atualização (OTA) capaz de lidar com trabalhos de atualização do Greengrass implantados na nuvem. O atendente é encontrado no novo diretório `/greengrass/ota`. Para obter mais informações, consulte Atualizações OTA de AWS IoT Greengrass Software principal.
- O recurso de acesso a atributos locais permite que as funções do Greengrass Lambda acessem recursos locais, como dispositivos periféricos e volumes. Para obter mais informações, consulte Acesso aos recursos locais com funções e conectores do Lambda.

1.1.0

Novos atributos:

- AWS IoT Greengrass Os grupos implantados podem ser redefinidos excluindo funções, assinaturas e configurações do Lambda. Para obter mais informações, consulte the section called “Redefinir implantações”.
- Suporte para os tempos de execução Node.js 6.10 e Java 8 Lambda, além do Python 2.7.

Para migrar da versão anterior do AWS IoT Greengrass núcleo:

- Copie os certificados da pasta `/greengrass/configuration/certs` em `/greengrass/certs`.
- Copie `/greengrass/configuration/config.json` para `/greengrass/config/config.json`.
- Execute `/greengrass/ggc/core/greengrassd` em vez de `/greengrass/greengrassd`.
- Implante o grupo no novo núcleo.

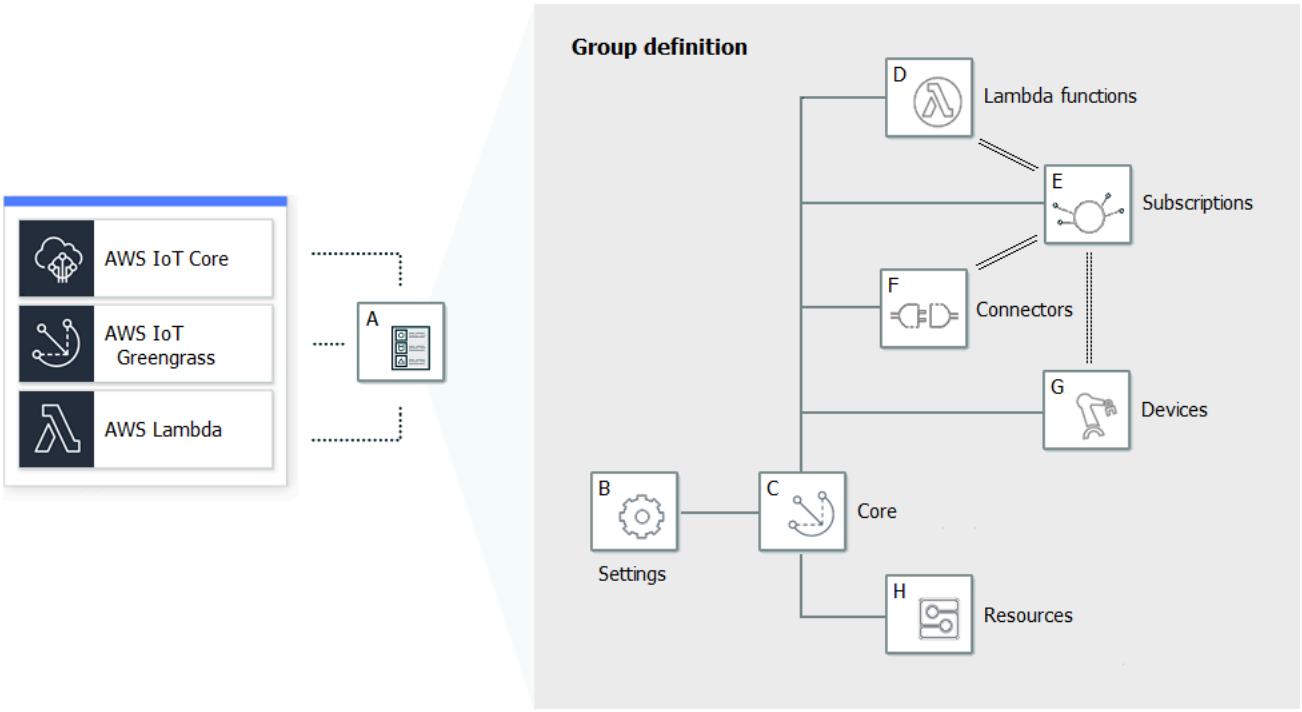
1.0.0

Versão inicial

AWS IoT Greengrass groups

Um grupo do Greengrass é uma coleção de configurações e componentes, como núcleo do Greengrass, dispositivos e assinaturas. Os grupos são usados para definir um escopo de interação.

Por exemplo, um grupo pode representar um andar de um prédio, um caminhão ou um site de mineração inteiro. O diagrama a seguir mostra os componentes que podem formar um grupo do Greengrass.



No diagrama anterior:

A: Definição do grupo do Greengrass

Informações sobre configurações e componentes do grupo.

B: Configurações do grupo do Greengrass

Isso inclui:

- Função do grupo do Greengrass.
- Autoridade de certificação e configuração de conexão local.
- Informações da conectividade do núcleo do Greengrass.
- O ambiente de runtime padrão do Lambda. Para obter mais informações, consulte the section called “Definir a conteinerização padrão para funções do Lambda em um grupo”.
- CloudWatch e configuração de registros locais. Para obter mais informações, consulte the section called “Monitoramento com logs do AWS IoT Greengrass ”.

C: Núcleo do Greengrass

A AWS IoT coisa (dispositivo) que representa o núcleo do Greengrass. Para obter mais informações, consulte the section called “Configurar o AWS IoT Greengrass núcleo”.

D: Definição de função do Lambda

Uma lista das funções do Lambda que são executadas localmente no núcleo, com os dados de configuração associados. Para obter mais informações, consulte Execute funções locais do Lambda.

E: Definição de assinatura

Uma lista de assinaturas que permitem comunicação usando mensagens MQTT. Uma inscrição define:

- A origem e o destino da mensagem. Podem ser dispositivos cliente, funções do Lambda AWS IoT Core, conectores e o serviço de sombra local.
- Um tópico (ou assunto) que é usado para filtrar mensagens.

Para obter mais informações, consulte the section called “Assinaturas gerenciadas no fluxo de trabalho de mensagens MQTT”.

F: Definição do connector

Uma lista dos conectores que são executados localmente no núcleo, com os dados de configuração associados. Para obter mais informações, consulte Integrar a serviços e protocolos usando conectores.

G: definição do dispositivo

Uma lista de AWS IoT itens (conhecidos como dispositivos ou dispositivos cliente) que são membros do grupo Greengrass, com dados de configuração associados. Para obter mais informações, consulte the section called “Dispositivos em AWS IoT Greengrass”.

H: definição do recurso

Uma lista dos recursos locais, recursos de machine learning e recursos de segredos no núcleo do Greengrass, com os dados de configuração associados. Para obter mais informações, consulte Acesso aos recursos locais, Executar a inferência de machine learning e Implantar segredos no núcleo.

Quando implantados, a definição do grupo do Greengrass, as funções do Lambda, os conectores, os recursos e a tabela de assinaturas são copiados para um dispositivo do núcleo. Para obter mais informações, consulte Implemente AWS IoT Greengrass grupos.

Dispositivos em AWS IoT Greengrass

Um grupo do Greengrass pode conter dois tipos de AWS IoT dispositivos:

Núcleo do Greengrass

Um núcleo do Greengrass é um dispositivo que executa o software AWS IoT Greengrass Core, o que permite que ele se comunique diretamente com o serviço AWS IoT Core e com o AWS IoT Greengrass serviço. Um núcleo tem seu próprio certificado de dispositivo usado para autenticação com AWS IoT Core. Ele tem uma sombra de dispositivo e uma entrada no AWS IoT Core registro. Os núcleos do Greengrass executam um runtime Lambda local, um agente de implantação e um rastreador de endereço IP que envia informações de endereço IP ao AWS IoT Greengrass serviço para permitir que os dispositivos clientes descubram automaticamente suas informações de grupo e de conexão principal. Para obter mais informações, consulte the section called “Configurar o AWS IoT Greengrass núcleo”.

> **Note**
>
> Um grupo do Greengrass deve conter exatamente um núcleo.

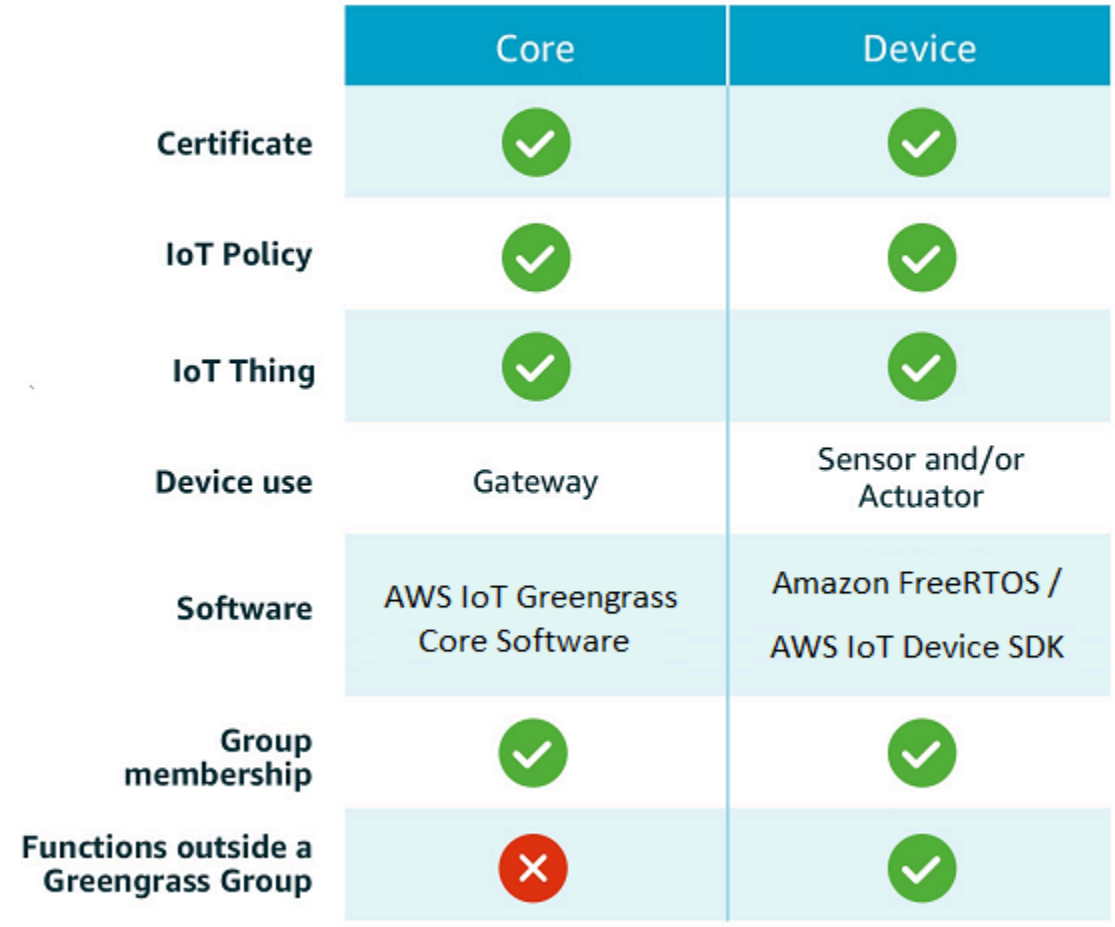
Dispositivo cliente

Os dispositivos cliente (também chamados dispositivos conectados, dispositivos Greengrass ou dispositivos) se conectam a um núcleo do Greengrass por meio do MQTT. Eles têm seu próprio certificado de dispositivo para AWS IoT Core autenticação, uma sombra do dispositivo e uma entrada no AWS IoT Core registro. Os dispositivos cliente do Greengrass podem executar FreeRTOS ou usar o SDK do dispositivo da AWS IoT ou a API de descoberta do AWS IoT Greengrass para obter informações de descoberta usadas para conexão e autenticação com o núcleo no mesmo grupo do Greengrass. Para saber como usar o AWS IoT console para criar e configurar um dispositivo cliente AWS IoT Greengrass, consultethe section called “Módulo 4: Interagindo com dispositivos clientes em um AWS IoT Greengrass grupo”. Ou, para obter exemplos que mostram como usar o AWS CLI para criar e configurar um dispositivo cliente AWS IoT Greengrass, consulte https://docs.aws.amazon.com/cli/latest/reference/greengrass/create-device-definition.html create-device-definition na Referência de Comandos. AWS CLI

Em um grupo do Greengrass, você pode criar assinaturas que permitem que dispositivos cliente se comuniquem pelo MQTT com funções, conectores e outros dispositivos cliente do Lambda no grupo e com AWS IoT Core ou com o serviço paralelo local. As mensagens MQTT são roteadas por meio do núcleo. Se o dispositivo de núcleo perder a conectividade com a nuvem, os dispositivos poderão continuar a comunicação por meio da rede local. Os dispositivos podem ter vários tamanhos, de dispositivos pequenos baseados em microcontroladores a grandes dispositivos. Atualmente, um grupo do Greengrass pode conter até 2500 dispositivos de clientes. Um dispositivo cliente pode ser membro de até 10 grupos.

> **Note**
>
> OPC-UA é um padrão de troca de informações para comunicação industrial. Para implementar o suporte para OPC-UA no núcleo do Greengrass, você pode usar o conector IoT SiteWise . O conector envia dados de dispositivos industriais de OPC-UA servidores para propriedades de ativos em AWS IoT SiteWise.

A tabela a seguir mostra como esses tipos de dispositivo estão relacionados.



O dispositivo AWS IoT Greengrass principal armazena certificados em dois locais:

- Certificado de dispositivo de núcleo em `/greengrass-root/certs`. Normalmente, o certificado de dispositivo de núcleo é nomeado `hash.cert.pem` (por exemplo, `86c84488a5.cert.pem`). Esse certificado é usado pelo AWS IoT cliente para autenticação mútua quando o núcleo se conecta aos AWS IoT Greengrass serviços AWS IoT Core e.
- Certificado do servidor MQTT em `/greengrass-root/ggc/var/state/server`. O certificado de servidor MQTT é nomeado `server.crt`. Este certificado é usado para autenticação mútua entre o servidor MQTT local (no núcleo do Greengrass) e Dispositivos Greengrass.

> **Note**
>
> *greengrass-root*representa o caminho em que o software AWS IoT Greengrass Core está instalado em seu dispositivo. Normalmente, esse é o diretório `/greengrass`.

SDKs

Os seguintes SDKs AWS fornecidos são usados para trabalhar com: AWS IoT Greengrass

AWS SDK

Use o AWS SDK para criar aplicativos que interagem com qualquer AWS serviço, incluindo Amazon S3, Amazon DynamoDB e muito mais. AWS IoT AWS IoT Greengrass No contexto de AWS IoT Greengrass, você pode usar o AWS SDK em funções do Lambda implantadas para fazer chamadas diretas para qualquer serviço. AWS Para obter mais informações, consulte AWS SDKs.

> **Note**
>
> As operações específicas do Greengrass que estão disponíveis nos AWS SDKs também estão disponíveis na AWS IoT Greengrass API e. AWS CLI

AWS IoT SDK do dispositivo

O AWS IoT Device SDK ajuda os dispositivos a se conectarem ao AWS IoT Core e. AWS IoT Greengrass Para obter mais informações, consulte Dispositivos SDK AWS IoT no Guia do desenvolvedor do AWS IoT .

Os dispositivos cliente podem usar qualquer uma das plataformas AWS IoT Device SDK v2 para descobrir informações de conectividade para um núcleo do Greengrass. As informações de conectividade incluem o seguinte:

- As IDs dos grupos do Greengrass aos quais o dispositivo cliente pertence.
- Os endereços IP do núcleo do Greengrass em cada grupo. Eles também são chamados endpoints de núcleo.
- O certificado CA do grupo, que os dispositivos usam para autenticação mútua com o núcleo. Para obter mais informações, consulte the section called “Fluxo de trabalho de conexão de dispositivo”.

> **Note**
>
> Na v1 dos AWS IoT Device SDKs, somente as plataformas C++ e Python fornecem suporte de descoberta integrado.

AWS IoT Greengrass SDK principal

O AWS IoT Greengrass Core SDK permite que as funções do Lambda interajam com o núcleo do Greengrass, publiquem mensagens AWS IoT, interajam com o serviço paralelo local, invoquem outras funções do Lambda implantadas e acessem recursos secretos. Este SDK é usado por funções do Lambda executadas em um AWS IoT Greengrass core. Para obter mais informações, consulte AWS IoT Greengrass SDK principal.

AWS IoT Greengrass SDK de aprendizado de máquina

O SDK do AWS IoT Greengrass Machine Learning permite que as funções do Lambda consumam modelos de aprendizado de máquina que são implantados no núcleo do Greengrass como recursos de aprendizado de máquina. Esse SDK é usado por funções do Lambda que são executadas em um AWS IoT Greengrass núcleo e interagem com um serviço de inferência local. Para obter mais informações, consulte AWS IoT Greengrass SDK de Machine Learning.

Plataformas compatíveis e requisitos

As guias a seguir listam as plataformas suportadas e os requisitos do software AWS IoT Greengrass Core.

> **Note**
>
> Você pode baixar o software AWS IoT Greengrass Core a partir dos downloads do AWS IoT Greengrass Core Software.

GGC v1.11

Plataformas compatíveis:

- Arquitetura: Armv7l
  - SO: Linux
  - Sistema operacional: Linux (OpenWrt)
- Arquitetura: Armv8 (AArch64)
  - SO: Linux
  - Sistema operacional: Linux (OpenWrt)
- Arquitetura: Armv6l

    - SO: Linux
- Arquitetura: x86_64
    - SO: Linux
- As plataformas Windows, macOS e Linux podem ser executadas AWS IoT Greengrass em um contêiner Docker. Para obter mais informações, consulte the section called “Executar AWS IoT Greengrass em um contêiner Docker”.

Requisitos:

- Mínimo de 128 MB de espaço em disco disponível para o software AWS IoT Greengrass Core. Se você usa o atendente de atualização OTA, o mínimo é 400 MB.
- Mínimo de 128 MB de RAM alocados para o software AWS IoT Greengrass Core. Com o gerenciador de fluxo habilitado, o mínimo é de 198 MB de RAM.

  > **Note**
  >
  > O gerenciador de streams é ativado por padrão se você usar a opção de criação de grupo padrão no AWS IoT console para criar seu grupo do Greengrass.

- Versão do kernel do Linux:
    - A versão 4.4 ou posterior do kernel Linux é necessária para suportar a execução AWS IoT Greengrass com contêineres.
    - A versão 3.17 ou posterior do kernel Linux é necessária para suportar a execução AWS IoT Greengrass sem contêineres. Nessa configuração, a conteinerização da função do Lambda padrão para o grupo do Greengrass deve ser definida como Sem contêiner. Para instruções, consulte the section called “Definir a conteinerização padrão para funções do Lambda em um grupo”.
- Biblioteca GNU C (glibc) versão 2.14 ou posterior. OpenWrt as distribuições requerem a versão 1.1.16 ou posterior da biblioteca musl C.
- O diretório `/var/run` precisa estar presente no dispositivo.
- Os arquivos `/dev/stdin`, `/dev/stdout` e `/dev/stderr` devem estar disponíveis.
- A proteção hardlink e softlink deve ser ativada no dispositivo. Caso contrário, só AWS IoT Greengrass pode ser executado no modo inseguro, usando o `-i` sinalizador.
- As seguintes configurações do kernel do Linux precisam ser ativadas no dispositivo:

- Namespace:
  - CONFIG_IPC_NS
  - CONFIG_UTS_NS
  - CONFIG_USER_NS
  - CONFIG_PID_NS
- Cgroups:
  - CONFIG_CGROUP_DEVICE
  - CONFIG_CGROUPS
  - CONFIG_MEMCG

  O kernel precisa ser compatível com cgroups. Os seguintes requisitos se aplicam ao executar AWS IoT Greengrass com contêineres:
  - O cgroup de memória precisa estar habilitado e instalado para permitir que o AWS IoT Greengrass defina o limite de memória para as funções do Lambda.
  - O cgroup de dispositivos deve ser ativado e montado se as funções do Lambda com acesso a recursos locais forem usadas para abrir arquivos no dispositivo AWS IoT Greengrass principal.
- Outros:
  - CONFIG_POSIX_MQUEUE
  - CONFIG_OVERLAY_FS
  - CONFIG_HAVE_ARCH_SECCOMP_FILTER
  - CONFIG_SECCOMP_FILTER
  - CONFIG_KEYS
  - CONFIG_SECCOMP
  - CONFIG_SHMEM
- O certificado raiz do Amazon S3 AWS IoT deve estar presente no repositório confiável do sistema.
- O gerenciador de streams requer o tempo de execução do Java 8 e um mínimo de 70 MB de RAM, além do requisito básico de memória do software AWS IoT Greengrass Core. O gerenciador de streams é ativado por padrão quando você usa a opção de criação de grupo padrão no AWS IoT console. O gerenciador de streams não é suportado em OpenWrt distribuições.

- Bibliotecas que oferecem suporte ao runtime do AWS Lambda exigido pelas funções do Lambda que você deseja executar localmente. As bibliotecas obrigatórias devem ser instaladas no núcleo e adicionadas à variável de ambiente `PATH`. Várias bibliotecas podem ser instaladas no mesmo núcleo.
  - Python versão 3.8 para funções que usam o runtime Python 3.8.
  - Python versão 3.7 para funções que usam o runtime Python 3.7.
  - Python versão 2.7 para funções que usam o runtime Python 2.7.
  - Node.jsversão 12.x para funções que usam o tempo de execução Node.js 12.x.
  - Java versão 8 ou posterior para funções que usam o runtime Java 8.

    > **Note**
    >
    > A execução do Java em uma OpenWrt distribuição não é oficialmente suportada. No entanto, se sua OpenWrt compilação tiver suporte para Java, você poderá executar funções do Lambda criadas em Java em seus dispositivos. OpenWrt

    Para obter mais informações sobre AWS IoT Greengrass suporte para tempos de execução do Lambda, consulte. Execute funções locais do Lambda
- Os seguintes comandos de shell (não as BusyBox variantes) são exigidos pelo agente de atualização over-the-air (OTA):
  - `wget`
  - `realpath`
  - `tar`
  - `readlink`
  - `basename`
  - `dirname`
  - `pidof`
  - `df`
  - `grep`
  - `umount`
  - `mv`
  - `gzip`

- `mkdir`
- `rm`
- `ln`
- `cut`
- `cat`
- `/bin/bash`

GGC v1.10

Plataformas compatíveis:

- Arquitetura: Armv7l
  - SO: Linux
  - Sistema operacional: Linux (OpenWrt)
- Arquitetura: Armv8 (AArch64)
  - SO: Linux
  - Sistema operacional: Linux (OpenWrt)
- Arquitetura: Armv6l
  - SO: Linux
- Arquitetura: x86_64
  - SO: Linux
- As plataformas Windows, macOS e Linux podem ser executadas AWS IoT Greengrass em um contêiner Docker. Para obter mais informações, consulte the section called “Executar AWS IoT Greengrass em um contêiner Docker”.

Requisitos:

- Mínimo de 128 MB de espaço em disco disponível para o software AWS IoT Greengrass Core. Se você usa o atendente de atualização OTA, o mínimo é 400 MB.
- Mínimo de 128 MB de RAM alocados para o software AWS IoT Greengrass Core. Com o gerenciador de fluxo habilitado, o mínimo é de 198 MB de RAM.

> **Note**
>
> O gerenciador de streams é ativado por padrão se você usar a opção de criação de grupo padrão no AWS IoT console para criar seu grupo do Greengrass.

- Versão do kernel do Linux:
  - A versão 4.4 ou posterior do kernel Linux é necessária para suportar a execução AWS IoT Greengrass com contêineres.
  - A versão 3.17 ou posterior do kernel Linux é necessária para suportar a execução AWS IoT Greengrass sem contêineres. Nessa configuração, a conteinerização da função do Lambda padrão para o grupo do Greengrass deve ser definida como Sem contêiner. Para instruções, consulte the section called “Definir a conteinerização padrão para funções do Lambda em um grupo”.
- Biblioteca GNU C (glibc) versão 2.14 ou posterior. OpenWrt as distribuições requerem a versão 1.1.16 ou posterior da biblioteca musl C.
- O diretório `/var/run` precisa estar presente no dispositivo.
- Os arquivos `/dev/stdin`, `/dev/stdout` e `/dev/stderr` devem estar disponíveis.
- A proteção hardlink e softlink deve ser ativada no dispositivo. Caso contrário, só AWS IoT Greengrass pode ser executado no modo inseguro, usando o `-i` sinalizador.
- As seguintes configurações do kernel do Linux precisam ser ativadas no dispositivo:
  - Namespace:
    - CONFIG_IPC_NS
    - CONFIG_UTS_NS
    - CONFIG_USER_NS
    - CONFIG_PID_NS
  - Cgroups:
    - CONFIG_CGROUP_DEVICE
    - CONFIG_CGROUPS
    - CONFIG_MEMCG

    O kernel precisa ser compatível com cgroups. Os seguintes requisitos se aplicam ao executar AWS IoT Greengrass com contêineres:

    - O cgroup de memória precisa estar habilitado e instalado para permitir que o AWS IoT Greengrass defina o limite de memória para as funções do Lambda.
    - O cgroup de dispositivos deve ser ativado e montado se as funções do Lambda com acesso a recursos locais forem usadas para abrir arquivos no dispositivo AWS IoT Greengrass principal.
  - Outros:
    - CONFIG_POSIX_MQUEUE
    - CONFIG_OVERLAY_FS
    - CONFIG_HAVE_ARCH_SECCOMP_FILTER
    - CONFIG_SECCOMP_FILTER
    - CONFIG_KEYS
    - CONFIG_SECCOMP
    - CONFIG_SHMEM
- O certificado raiz do Amazon S3 AWS IoT deve estar presente no repositório confiável do sistema.
- O gerenciador de streams requer o tempo de execução do Java 8 e um mínimo de 70 MB de RAM, além do requisito básico de memória do software AWS IoT Greengrass Core. O gerenciador de streams é ativado por padrão quando você usa a opção de criação de grupo padrão no AWS IoT console. O gerenciador de streams não é suportado em OpenWrt distribuições.
- Bibliotecas que oferecem suporte ao runtime do AWS Lambda exigido pelas funções do Lambda que você deseja executar localmente. As bibliotecas obrigatórias devem ser instaladas no núcleo e adicionadas à variável de ambiente `PATH`. Várias bibliotecas podem ser instaladas no mesmo núcleo.
  - Python versão 3.7 para funções que usam o runtime Python 3.7.
  - Python versão 2.7 para funções que usam o runtime Python 2.7.
  - Node.jsversão 12.x para funções que usam o tempo de execução Node.js 12.x.
  - Java versão 8 ou posterior para funções que usam o runtime Java 8.

> **Note**
>
> A execução do Java em uma OpenWrt distribuição não é oficialmente suportada. No entanto, se sua OpenWrt compilação tiver suporte para Java, você poderá executar funções do Lambda criadas em Java em seus dispositivos. OpenWrt

Para obter mais informações sobre AWS IoT Greengrass suporte para tempos de execução do Lambda, consulte. Execute funções locais do Lambda

- Os seguintes comandos de shell (não as BusyBox variantes) são exigidos pelo agente de atualização over-the-air (OTA):
  - `wget`
  - `realpath`
  - `tar`
  - `readlink`
  - `basename`
  - `dirname`
  - `pidof`
  - `df`
  - `grep`
  - `umount`
  - `mv`
  - `gzip`
  - `mkdir`
  - `rm`
  - `ln`
  - `cut`
  - `cat`
  - `/bin/bash`

GGC v1.9

Plataformas compatíveis:

- Arquitetura: Armv7l
  - SO: Linux
  - Sistema operacional: Linux (OpenWrt)
- Arquitetura: Armv8 (AArch64)
  - SO: Linux
  - Sistema operacional: Linux (OpenWrt)
- Arquitetura: Armv6l
  - SO: Linux
- Arquitetura: x86_64
  - SO: Linux
- As plataformas Windows, macOS e Linux podem ser executadas AWS IoT Greengrass em um contêiner Docker. Para obter mais informações, consulte the section called “Executar AWS IoT Greengrass em um contêiner Docker”.

Requisitos:

- Mínimo de 128 MB de espaço em disco disponível para o software AWS IoT Greengrass Core. Se você usa o atendente de atualização OTA, o mínimo é 400 MB.
- Mínimo de 128 MB de RAM alocados para o software AWS IoT Greengrass Core.
- Versão do kernel do Linux:
  - A versão 4.4 ou posterior do kernel Linux é necessária para suportar a execução AWS IoT Greengrass com contêineres.
  - A versão 3.17 ou posterior do kernel Linux é necessária para suportar a execução AWS IoT Greengrass sem contêineres. Nessa configuração, a conteinerização da função do Lambda padrão para o grupo do Greengrass deve ser definida como Sem contêiner. Para instruções, consulte the section called “Definir a conteinerização padrão para funções do Lambda em um grupo”.
- Biblioteca GNU C (glibc) versão 2.14 ou posterior. OpenWrt as distribuições requerem a versão 1.1.16 ou posterior da biblioteca musl C.
- O diretório `/var/run` precisa estar presente no dispositivo.

- Os arquivos `/dev/stdin`, `/dev/stdout` e `/dev/stderr` devem estar disponíveis.
- A proteção hardlink e softlink deve ser ativada no dispositivo. Caso contrário, só AWS IoT Greengrass pode ser executado no modo inseguro, usando o `-i` sinalizador.
- As seguintes configurações do kernel do Linux precisam ser ativadas no dispositivo:
  - Namespace:
    - CONFIG_IPC_NS
    - CONFIG_UTS_NS
    - CONFIG_USER_NS
    - CONFIG_PID_NS
  - Cgroups:
    - CONFIG_CGROUP_DEVICE
    - CONFIG_CGROUPS
    - CONFIG_MEMCG

    O kernel precisa ser compatível com cgroups. Os seguintes requisitos se aplicam ao executar AWS IoT Greengrass com contêineres:
    - O cgroup de memória precisa estar habilitado e instalado para permitir que o AWS IoT Greengrass defina o limite de memória para as funções do Lambda.
    - O cgroup de dispositivos deve ser ativado e montado se as funções do Lambda com acesso a recursos locais forem usadas para abrir arquivos no dispositivo AWS IoT Greengrass principal.
  - Outros:
    - CONFIG_POSIX_MQUEUE
    - CONFIG_OVERLAY_FS
    - CONFIG_HAVE_ARCH_SECCOMP_FILTER
    - CONFIG_SECCOMP_FILTER
    - CONFIG_KEYS
    - CONFIG_SECCOMP
    - CONFIG_SHMEM
- O certificado raiz do Amazon S3 AWS IoT deve estar presente no repositório confiável do sistema.

- Bibliotecas que oferecem suporte ao runtime do AWS Lambda exigido pelas funções do Lambda que você deseja executar localmente. As bibliotecas obrigatórias devem ser instaladas no núcleo e adicionadas à variável de ambiente `PATH`. Várias bibliotecas podem ser instaladas no mesmo núcleo.
  - Python versão 2.7 para funções que usam o runtime Python 2.7.
  - Python versão 3.7 para funções que usam o runtime Python 3.7.
  - Node.jsversão 6.10 ou posterior para funções que usam o tempo de execução Node.js 6.10.
  - Node.jsversão 8.10 ou posterior para funções que usam o tempo de execução Node.js 8.10.
  - Java versão 8 ou posterior para funções que usam o runtime Java 8.

    > **Note**
    >
    > A execução do Java em uma OpenWrt distribuição não é oficialmente suportada. No entanto, se sua OpenWrt compilação tiver suporte para Java, você poderá executar funções do Lambda criadas em Java em seus dispositivos. OpenWrt

    Para obter mais informações sobre AWS IoT Greengrass suporte para tempos de execução do Lambda, consulte. Execute funções locais do Lambda
- Os seguintes comandos de shell (não as BusyBox variantes) são exigidos pelo agente de atualização over-the-air (OTA):
  - `wget`
  - `realpath`
  - `tar`
  - `readlink`
  - `basename`
  - `dirname`
  - `pidof`
  - `df`
  - `grep`
  - `umount`
  - `mv`
  - `gzip`

- `mkdir`
- `rm`
- `ln`
- `cut`
- `cat`

GGC v1.8

- Plataformas compatíveis:
  - Arquitetura: ARMv7l; SO: Linux
  - Arquitetura: x86_64; SO: Linux
  - Arquitetura: Armv8 (AArch64); SO: Linux
  - As plataformas Windows, macOS e Linux podem ser executadas AWS IoT Greengrass em um contêiner Docker. Para obter mais informações, consulte the section called “Executar AWS IoT Greengrass em um contêiner Docker”.
  - As plataformas Linux podem executar uma versão do AWS IoT Greengrass com funcionalidade limitada usando o snap Greengrass, que está disponível por meio do Snapcraft. Para obter mais informações, consulte the section called “AWS IoT Greengrass software instantâneo”.
- Os seguintes itens são necessários:
  - Mínimo de 128 MB de espaço em disco disponível para o software AWS IoT Greengrass Core. Se você usa o atendente de atualização OTA, o mínimo é 400 MB.
  - Mínimo de 128 MB de RAM alocados para o software AWS IoT Greengrass Core.
  - Versão do kernel do Linux:
    - A versão 4.4 ou posterior do kernel Linux é necessária para suportar a execução AWS IoT Greengrass com contêineres.
    - A versão 3.17 ou posterior do kernel Linux é necessária para suportar a execução AWS IoT Greengrass sem contêineres. Nessa configuração, a conteinerização da função do Lambda padrão para o grupo do Greengrass deve ser definida como Sem contêiner. Para instruções, consulte the section called “Definir a conteinerização padrão para funções do Lambda em um grupo”.
  - Biblioteca do GNU C (glibc) versão 2.14 ou posterior.
  - O diretório `/var/run` precisa estar presente no dispositivo.

- Os arquivos `/dev/stdin`, `/dev/stdout` e `/dev/stderr` devem estar disponíveis.
- A proteção hardlink e softlink deve ser ativada no dispositivo. Caso contrário, só AWS IoT Greengrass pode ser executado no modo inseguro, usando o `-i` sinalizador.
- As seguintes configurações do kernel do Linux precisam ser ativadas no dispositivo:
  - Namespace:
    - CONFIG_IPC_NS
    - CONFIG_UTS_NS
    - CONFIG_USER_NS
    - CONFIG_PID_NS
  - Cgroups:
    - CONFIG_CGROUP_DEVICE
    - CONFIG_CGROUPS
    - CONFIG_MEMCG

    O kernel precisa ser compatível com cgroups. Os seguintes requisitos se aplicam ao executar AWS IoT Greengrass com contêineres:
    - O cgroup de memória precisa estar habilitado e instalado para permitir que o AWS IoT Greengrass defina o limite de memória para as funções do Lambda.
    - O cgroup de dispositivos deve ser ativado e montado se as funções do Lambda com acesso a recursos locais forem usadas para abrir arquivos no dispositivo AWS IoT Greengrass principal.
  - Outros:
    - CONFIG_POSIX_MQUEUE
    - CONFIG_OVERLAY_FS
    - CONFIG_HAVE_ARCH_SECCOMP_FILTER
    - CONFIG_SECCOMP_FILTER
    - CONFIG_KEYS
    - CONFIG_SECCOMP
    - CONFIG_SHMEM
- O certificado raiz do Amazon S3 AWS IoT deve estar presente no repositório confiável do sistema.

- Os seguintes itens são necessários condicionalmente:

- Bibliotecas que oferecem suporte ao runtime do AWS Lambda exigido pelas funções do Lambda que você deseja executar localmente. As bibliotecas obrigatórias devem ser instaladas no núcleo e adicionadas à variável de ambiente PATH. Várias bibliotecas podem ser instaladas no mesmo núcleo.
  - Python versão 2.7 para funções que usam o runtime Python 2.7.
  - Node.jsversão 6.10 ou posterior para funções que usam o tempo de execução Node.js 6.10.
  - Java versão 8 ou posterior para funções que usam o runtime Java 8.
- Os seguintes comandos de shell (não as BusyBox variantes) são exigidos pelo agente de atualização over-the-air (OTA):
  - `wget`
  - `realpath`
  - `tar`
  - `readlink`
  - `basename`
  - `dirname`
  - `pidof`
  - `df`
  - `grep`
  - `umount`
  - `mv`
  - `gzip`
  - `mkdir`
  - `rm`
  - `ln`
  - `cut`
  - `cat`

Para obter informações sobre AWS IoT Greengrass cotas (limites), consulte Cotas de serviço no. Referência geral da Amazon Web Services

Para obter informações sobre definição de preço, consulte Definição de preços do AWS IoT Greengrass e Definição de preços do AWS IoT Core.

AWS IoT Greengrass downloads

Você pode usar as seguintes informações para localizar e fazer download do software para uso com o AWS IoT Greengrass.

Tópicos

- AWS IoT Greengrass Software principal
- AWS IoT Greengrass software instantâneo
- AWS IoT Greengrass Software Docker
- AWS IoT Greengrass SDK principal
- Bibliotecas e tempos de execução de machine learning compatíveis
- AWS IoT Greengrass Software ML SDK

AWS IoT Greengrass Software principal

O software AWS IoT Greengrass Core estende a AWS funcionalidade a um dispositivo AWS IoT Greengrass principal, possibilitando que dispositivos locais atuem localmente nos dados que geram.

v1.11

1.11.6

Correções de bugs e melhorias:

- Aprimoramento da resiliência se ocorrer uma queda repentina de energia durante uma implantação.
- Foi corrigido um problema em que a corrupção de dados do gerenciador de streams poderia impedir a inicialização do software AWS IoT Greengrass Core.
- Correção de um problema em que novos dispositivos cliente não conseguiam se conectar ao núcleo em determinados cenários.
- Correção de um problema em que os nomes do fluxo do gerenciador de fluxo não podiam conter `.log`.

1.11.5

Correções de bugs e melhorias:

- Melhorias na performance geral e correções de erros.

1.11.4

Correções de bugs e melhorias:

- Foi corrigido um problema com o gerenciador de streams que impedia atualizações para o software AWS IoT Greengrass Core v1.11.3. Se você estiver usando o gerenciador de stream para exportar dados para a nuvem, agora você pode usar uma atualização OTA para atualizar uma versão anterior v1.x do software AWS IoT Greengrass Core para a v1.11.4.
- Melhorias na performance geral e correções de erros.

1.11.3

Correções de bugs e melhorias:

- Foi corrigido um problema que fazia com que o software AWS IoT Greengrass Core executado em um piscar de olhos em um dispositivo Ubuntu parasse de responder após uma perda repentina de energia no dispositivo.
- Correção de um problema que causava atrasos na entrega de mensagens MQTT para funções do Lambda de longa duração.
- Correção de um problema que fazia com que as mensagens MQTT não fossem enviadas corretamente quando o valor `maxWorkItemCount` era definido como maior que `1024`.
- Correção de um problema que fazia com que o atendente de atualização OTA ignorasse o período `KeepAlive` do MQTT especificado na propriedade `keepAlive` no `config.json`.
- Melhorias na performance geral e correções de erros.

> **Important**
>
> Se você estiver usando o gerenciador de stream para exportar dados para a nuvem, não atualize para o software AWS IoT Greengrass Core v1.11.3 de uma versão anterior v1.x. Se você estiver ativando o stream manager pela primeira vez, é altamente recomendável instalar primeiro a versão mais recente do software AWS IoT Greengrass Core.

1.11.1

Correções de bugs e melhorias:

- Correção de um problema que causava um aumento na utilização da memória do gerenciador de fluxo.
- Correção de um problema que fazia com que o gerenciador de fluxo redefinisse o número de sequência do fluxo para `0` se o dispositivo principal do Greengrass estivesse desligado por mais tempo do que a vida útil (TTL) especificada dos dados do fluxo.
- Correção de um problema que impedia o gerenciador de fluxo de interromper corretamente as tentativas de exportar dados para o Nuvem AWS.

1.11.0

Novos atributos:

- Um agente de telemetria no núcleo do Greengrass coleta dados de telemetria local e os publica no. Nuvem AWS Para recuperar os dados de telemetria para processamento adicional, os clientes podem criar uma EventBridge regra da Amazon e assinar uma meta. Para obter mais informações, consulte Coleta de dados de telemetria de integridade do sistema dos AWS IoT Greengrass principais dispositivos.
- Uma API HTTP local retorna um instantâneo do estado atual dos processos de trabalho locais iniciados por AWS IoT Greengrass. Para obter mais informações, consulte Chamada da API de verificação de integridade local.
- Um gerenciador de streams exporta dados automaticamente para o Amazon S3 e. AWS IoT SiteWise

  Os novos parâmetros do gerenciador de fluxo permitem que você atualize os fluxos existentes e pause ou retome a exportação de dados.
- Suporte para execução Python 3.8.x core das funções do Lambda no núcleo.
- Uma nova propriedade `ggDaemonPort` no `config.json` usada para configurar o número da porta do IPC do núcleo do Greengrass. O número da porta padrão é 8000.

  Uma nova propriedade `systemComponentAuthTimeout` no `config.json` que você usa para configurar o tempo limite para a autenticação do IPC do núcleo do Greengrass. O padrão de tempo de saída é 5000 milissegundos.
- Aumentou o número máximo de AWS IoT dispositivos por AWS IoT Greengrass grupo de 200 para 2500.

Aumento do número máximo de assinaturas por grupo, de 1.000 para 10.000.

Para saber mais, consulte Endpoints e cotas do AWS IoT Greengrass.

Correções de bugs e melhorias:

- Otimização geral que pode reduzir a utilização da memória dos processos de serviço do Greengrass.
- Um novo parâmetro de configuração de runtime (`mountAllBlockDevices`) permite que o Greengrass use montagens bind para montar todos os dispositivos de blocos em um contêiner após configurar o OverlayFS. Esse atributo resolveu um problema que causava a falha na implantação do Greengrass se `/usr` não estivesse sob a hierarquia `/`.
- Corrigido um problema que causava falha no AWS IoT Greengrass núcleo `/tmp` se fosse um link simbólico.
- Correção de um problema que permitia que o atendente de implantação do Greengrass removesse artefatos do modelo de machine learning não utilizados da pasta `mlmodel_public`.
- Melhorias na performance geral e correções de erros.

Para instalar o software AWS IoT Greengrass Core em seu dispositivo principal, baixe o pacote para sua arquitetura e sistema operacional (SO) e siga as etapas no Guia de introdução.

> **Tip**
>
> AWS IoT Greengrass também fornece outras opções para instalar o software AWS IoT Greengrass Core. Por exemplo, você pode usar a configuração do dispositivo Greengrass para configurar seu ambiente e instalar a versão mais recente do software AWS IoT Greengrass Core. Ou, em plataformas Debian suportadas, você pode usar o gerenciador de pacotes APT para instalar ou atualizar o software AWS IoT Greengrass Core. Para obter mais informações, consulte the section called “Instale o AWS IoT Greengrass Software principal”.

Arquitetura	Sistema operacional	Link
Armv8 (AArch64)	Linux	Baixar

Arquitetura	Sistema operacional	Link
Armv8 (AArch64)	Linux (OpenWrt)	Baixar
Armv7l	Linux	Baixar
Armv7l	Linux (OpenWrt)	Baixar
Armv6l	Linux	Baixar
x86_64	Linux	Baixar

Extended life versions

1,10.5

Novos atributos na versão 1.10:

- Um gerenciador de fluxo que processa fluxos de dados localmente e os exporta para a Nuvem AWS automaticamente. Esse atributo requer Java 8 no dispositivo de núcleo do Greengrass. Para obter mais informações, consulte Gerenciamento de fluxos de dados.
- Um novo conector de implantação do aplicativo Docker do Greengrass que executa um aplicativo Docker em um dispositivo de núcleo. Para obter mais informações, consulte the section called “Implantação de aplicativo do Docker”.
- Um novo SiteWise conector de IoT que envia dados de dispositivos industriais de OPC-UA servidores para propriedades de ativos em AWS IoT SiteWise. Para obter mais informações, consulte the section called “IoT SiteWise”.
- Funções do Lambda executadas sem conteinerização podem acessar recursos de machine learning no grupo do Greengrass. Para obter mais informações, consulte the section called “Acesse os recursos de machine learning”.
- Suporte para sessões persistentes do MQTT com AWS IoT. Para obter mais informações, consulte the section called “Sessões persistentes do MQTT com AWS IoT Core”.
- O tráfego MQTT local pode passar por uma porta diferente da padrão 8883. Para obter mais informações, consulte the section called “Porta MQTT para mensagens locais”.
- Novas opções da `queueFullPolicy` no SDK do AWS IoT Greengrass Core. para publicação confiável de mensagens a partir de funções do Lambda
- Suporte para executar funções do Lambda Node.js 12.x no núcleo.

Correções de bugs e melhorias:

- Over-the-air As atualizações (OTA) com integração de segurança de hardware podem ser configuradas com o OpenSSL 1.1.
- O gerenciador de fluxos é mais resiliente a danos nos dados dos arquivos.
- Correção de um problema que causa uma falha de montagem sysfs em dispositivos que usam o kernel Linux 5.1 e posterior.
- Uma nova `mqttOperationTimeout` propriedade em config.json que você usa para definir o tempo limite para operações de publicação, assinatura e cancelamento de assinatura em conexões MQTT com. AWS IoT Core
- Correção de um problema que causava um aumento na utilização da memória do gerenciador de fluxo.
- Uma nova propriedade `systemComponentAuthTimeout` no `config.json` que você usa para configurar o tempo limite para a autenticação do IPC do núcleo do Greengrass. O padrão de tempo de saída é 5000 milissegundos.
- Correção de um problema que fazia com que o atendente de atualização OTA ignorasse o período `KeepAlive` do MQTT especificado na propriedade `keepAlive` no `config.json`.
- Correção de um problema que fazia com que as mensagens MQTT não fossem enviadas corretamente quando o valor `maxWorkItemCount` era definido como maior que `1024`.
- Correção de um problema que causava atrasos na entrega de mensagens MQTT para funções do Lambda de longa duração.
- Foi corrigido um problema que fazia com que o software AWS IoT Greengrass Core executado em um piscar de olhos em um dispositivo Ubuntu parasse de responder após uma perda repentina de energia no dispositivo.
- Melhorias na performance geral e correções de erros.

Para instalar o software AWS IoT Greengrass Core em seu dispositivo principal, baixe o pacote para sua arquitetura e sistema operacional (SO) e siga as etapas no Guia de introdução.

Arquitetura	Sistema operacional	Link
Armv8 (AArch64)	Linux	Baixar
Armv8 (AArch64)	Linux (OpenWrt)	Baixar

Arquitetura	Sistema operacional	Link
Armv7l	Linux	Baixar
Armv7l	Linux (OpenWrt)	Baixar
Armv6l	Linux	Baixar
x86_64	Linux	Baixar

1.9.4

Novos atributos na versão 1.9:

- Suporte para os tempos de execução Python 3.7 e Node.js 8.10 Lambda. As funções do Lambda que usam os tempos de execução do Python 3.7 e Node.js 8.10 agora podem ser executadas em um núcleo. AWS IoT Greengrass (AWS IoT Greengrass continua a oferecer suporte aos tempos de execução do Python 2.7 e Node.js 6.10.)
- Conexões MQTT otimizadas. O núcleo do Greengrass estabelece um número menor de conexões com o AWS IoT Core. Essa alteração pode reduzir os custos operacionais para cobranças com base no número de conexões.
- Chave de curva elíptica (EC) para o servidor MQTT local. O servidor MQTT local oferece suporte a chaves EC, além de chaves RSA. (O certificado do servidor MQTT tem uma assinatura SHA-256 RSA, independentemente do tipo de chave.) Para obter mais informações, consulte the section called “Entidades principais de segurança”.
- Suporte para OpenWrt. AWS IoT Greengrass O software principal v1.9.2 ou posterior pode ser instalado em OpenWrt distribuições com as arquiteturas Armv8 (AArch64) e ARMv7l. Atualmente, OpenWrt não oferece suporte à inferência de ML.
- Suporte para ARMv6L. AWS IoT Greengrass O software principal v1.9.3 ou posterior pode ser instalado em distribuições Raspbian em arquiteturas ARMv6l (por exemplo, em dispositivos Raspberry Pi Zero).
- Atualizações OTA na porta 443 com ALPN. Os núcleos do Greengrass que usam a porta 443 para tráfego MQTT agora oferecem suporte a atualizações de software over-the-air (OTA). AWS IoT Greengrass usa a extensão TLS da Application Layer Protocol Network (ALPN) para habilitar essas conexões. Para obter mais informações, consulte Atualizações OTA do software AWS IoT Greengrass Core e the section called “Conectar-se à porta 443 ou por meio de um proxy de rede”.

Para instalar o software AWS IoT Greengrass Core em seu dispositivo principal, baixe o pacote para sua arquitetura e sistema operacional (SO) e siga as etapas no Guia de introdução.

Arquitetura	Sistema operacional	Link
Armv8 (AArch64)	Linux	Baixar
Armv8 (AArch64)	Linux (OpenWrt)	Baixar
Armv7l	Linux	Baixar
Armv7l	Linux (OpenWrt)	Baixar
Armv6l	Linux	Baixar
x86_64	Linux	Baixar

1.8.4

- Novos atributos:
  - Identidade de acesso padrão configurável para as funções do Lambda no grupo. Essa configuração no nível de grupo determina as permissões padrão que são usadas para executar as funções do Lambda. Você pode definir o ID de usuário, ID de grupo, ou ambos. As funções do Lambda individuais podem substituir a identidade de acesso padrão de seu grupo. Para obter mais informações, consulte the section called “Definir a identidade de acesso padrão para as funções do Lambda em um grupo”.
  - Tráfego HTTPS pela porta 443. A comunicação HTTPS pode ser configurada para viajar pela porta 443, em vez da porta padrão 8443. Isso complementa o AWS IoT Greengrass suporte à extensão TLS da Application Layer Protocol Network (ALPN) e permite que todo o tráfego de mensagens do Greengrass — tanto MQTT quanto HTTPS — use a porta 443. Para obter mais informações, consulte the section called “Conectar-se à porta 443 ou por meio de um proxy de rede”.
  - IDs de cliente com nomes previsíveis para AWS IoT conexões. Essa alteração permite oferecer suporte para AWS IoT Device Defender e Eventos de ciclo de vida do AWS IoT, para que você possa receber notificações para se conectar, desconectar, assinar e cancelar a assinatura de eventos. A nomenclatura previsível também facilita a criação lógica em torno de IDs de conexão (por exemplo, para criar modelos de política de

assinatura com base em atributos de certificado). Para obter mais informações, consulte the section called “IDs de cliente para conexões MQTT com AWS IoT”.

Correções de bugs e melhorias:

- Corrigido um problema com sincronização de shadow e reconexão do gerenciador de certificados de dispositivo.
- Melhorias na performance geral e correções de erros.

Para instalar o software AWS IoT Greengrass Core em seu dispositivo principal, baixe o pacote para sua arquitetura e sistema operacional (SO) e siga as etapas no Guia de introdução.

Arquitetura	Sistema operacional	Link
Armv8 (AArch64)	Linux	Baixar
Armv7l	Linux	Baixar
x86_64	Linux	Baixar

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

Para obter informações sobre outras opções para instalar o software AWS IoT Greengrass Core em seu dispositivo, consultethe section called “Instale o AWS IoT Greengrass Software principal”.

AWS IoT Greengrass software instantâneo

AWS IoT Greengrass O snap 1.11.x permite que você execute uma versão limitada AWS IoT Greengrass por meio de pacotes de software convenientes, junto com todas as dependências necessárias, em um ambiente em contêiner.

> **Note**
>
> O AWS IoT Greengrass snap está disponível para o software AWS IoT Greengrass Core v1.11.x. AWS IoT Greengrass não fornece um snap para a v1.10.x. Versões não compatíveis não recebem correções de bugs ou atualizações.

O AWS IoT Greengrass snap não suporta conectores e inferência de aprendizado de máquina (ML).

Para obter mais informações, consulte the section called “Corra AWS IoT Greengrass em um piscar de olhos”.

AWS IoT Greengrass Software Docker

AWS fornece imagens do Dockerfile e do Docker que facilitam a execução AWS IoT Greengrass em um contêiner do Docker.

Dockerfile

Os Dockerfiles contêm código-fonte para criar imagens de AWS IoT Greengrass contêiner personalizadas. As imagens podem ser modificadas para funcionar com diferentes arquiteturas de plataforma ou para reduzir o tamanho da imagem. Para obter instruções, consulte o arquivo README.

Faça o download da versão AWS IoT Greengrass do software Core de destino.

v1.11

- Dockerfile para AWS IoT Greengrass v1.11.6.

Extended life versions

v1.10

Dockerfile para AWS IoT Greengrass v1.10.5.

v1.9

Dockerfile para AWS IoT Greengrass v1.9.4.

v1.8

Dockerfile para AWS IoT Greengrass v1.8.1.

Docker image (Imagem do Docker)

As imagens do Docker têm o software AWS IoT Greengrass Core e as dependências instaladas nas imagens base do Amazon Linux 2 (x86_64) e do Alpine Linux (x86_64, ARMv7l ou AArch64). É possível usar imagens pré-compiladas para começar a testar o AWS IoT Greengrass.

> **Important**
>
> Em 30 de junho de 2022, AWS IoT Greengrass encerrou a manutenção das imagens Docker v1.x do software AWS IoT Greengrass Core que são publicadas no Amazon Elastic Container Registry (Amazon ECR) e no Docker Hub. Você pode continuar baixando essas imagens do Docker do Amazon ECR e do Docker Hub até 30 de junho de 2023, ou seja, um ano após o término da manutenção. No entanto, as imagens do Docker do software AWS IoT Greengrass Core v1.x não recebem mais patches de segurança ou correções de erros após o término da manutenção em 30 de junho de 2022. Se você executar uma carga de trabalho de produção que depende dessas imagens do Docker, recomendamos que você crie suas próprias imagens do Docker usando os Dockerfiles fornecidos. AWS IoT Greengrass Para obter mais informações, consulte AWS IoT Greengrass Version 1 política de manutenção.

Faça download de uma imagem pré-criada do Docker Hub ou do Amazon Elastic Container Registry (Amazon ECR).

- Para o Docker Hub, use a *`version`* tag para baixar uma versão específica da imagem Docker do Greengrass. Para encontrar tags para todas as imagens disponíveis, verifique a página Tags no Hub do Docker.
- No Amazon ECR, use a tag `latest` para baixar a versão mais recente disponível da imagem do Docker do Greengrass. Para obter mais informações sobre como listar versões de imagens disponíveis e baixar imagens do Amazon ECR, consulte Em execução AWS IoT Greengrass em um contêiner Docker.

> **Warning**
>
> A partir da versão 1.11.6 do software AWS IoT Greengrass Core, as imagens do Greengrass Docker não incluem mais o Python 2.7, porque o Python 2.7 atingiu o fim da vida útil em 2020 e não recebe mais atualizações de segurança. Se você optar por atualizar essas imagens do Docker, recomendamos verificar se seus aplicativos

funcionam com as novas imagens do Docker antes de implantar as atualizações nos dispositivos de produção. Se você precisar do Python 2.7 para seu aplicativo que usa uma imagem do Docker do Greengrass, poderá modificar o Dockerfile do Greengrass para incluir o Python 2.7 em seu aplicativo.

AWS IoT Greengrass não fornece imagens Docker para o software AWS IoT Greengrass Core v1.11.1.

> **Note**
>
> Por padrão, `alpine-aarch64` as `alpine-armv7l` imagens só podem ser executadas em Arm-based hosts. Para executar essas imagens em um host x86, é possível instalar QEMU e montar as bibliotecas QEMU no host. Por exemplo:
>
> ```
> docker run --rm --privileged multiarch/qemu-user-static --reset -p yes
> ```

AWS IoT Greengrass SDK principal

As funções do Lambda usam o AWS IoT Greengrass Core SDK para interagir com o AWS IoT Greengrass núcleo localmente. Isso permite que as funções do Lambda implantadas:

- Troque mensagens MQTT com AWS IoT Core.
- Troque mensagens MQTT com conectores, dispositivos cliente e outras funções do Lambda no grupo do Greengrass.
- Interaja com o serviço de shadow local.
- Invoque outras funções locais do Lambda.
- Acesse recursos secretos.
- Interaja com o gerenciador de fluxo.

Faça o download do AWS IoT Greengrass Core SDK para seu idioma ou plataforma em. GitHub

- AWS IoT Greengrass SDK principal para Java

- AWS IoT Greengrass SDK principal para Node.js
- AWS IoT Greengrass SDK principal para Python
- AWS IoT Greengrass SDK principal para C

Para obter mais informações, consulte AWS IoT Greengrass SDK principal.

Bibliotecas e tempos de execução de machine learning compatíveis

Para executar inferência em um núcleo do Greengrass, você deve instalar a biblioteca ou o tempo de execução de machine learning para o tipo de modelo de ML.

AWS IoT Greengrass oferece suporte aos seguintes tipos de modelo de ML. Use esses links para encontrar informações sobre como instalar o tempo de execução ou a biblioteca para seu tipo de modelo e plataforma de dispositivo.

- Deep Learning Runtime (DLR)
- MXNet
- TensorFlow

Exemplos de machine learning

AWS IoT Greengrass fornece exemplos que você pode usar com bibliotecas e tempos de execução de ML compatíveis. Esses exemplos são liberados de acordo com o Contrato de licença de software do Greengrass Core.

Deep learning runtime (DLR)

Faça download do exemplo para a plataforma do dispositivo:

- Exemplo de DLR para Raspberry Pi
- Exemplo de DLR para NVIDIA Jetson TX2
- Exemplo de DLR para Intel Atom

Para obter um tutorial que usa o exemplo de DLR, consulte the section called “Como configurar a inferência otimizada de Machine Learning”.

MXNet

Faça download do exemplo para a plataforma do dispositivo:

- Exemplo de MXNet para Raspberry Pi
- Exemplo de MXNet para NVIDIA Jetson TX2
- Exemplo de MXNet para Intel Atom

Para obter um tutorial que usa o exemplo de MXNet, consulte the section called “Como configurar a inferência de machine learning”.

TensorFlow

Faça download do Exemplo de Tensorflow para a plataforma do seu dispositivo. Este exemplo funciona com Raspberry Pi, NVIDIA Jetson TX2 e Intel Atom.

AWS IoT Greengrass Software ML SDK

O AWS IoT Greengrass SDK de Machine Learning permite que as funções que você criar consumam um modelo de machine learning local e enviem dados ao conector ML Feedback para upload e publicação.

v1.1.0

- Python 3.7.

v1.0.0

- Python 2.7.

Deixe seu comentário

Os seus comentários são bem-vindos. Para entrar em contato conosco, acesse AWS ref: Post e use a tag AWS IoT Greengrass.

Instale o AWS IoT Greengrass Software principal

O software AWS IoT Greengrass Core estende a AWS funcionalidade a um dispositivo AWS IoT Greengrass principal, possibilitando que dispositivos locais atuem localmente nos dados que geram.

AWS IoT Greengrass fornece várias opções para instalar o software AWS IoT Greengrass Core:

- Faça download de um arquivo tar.gz e instale-o.
- Execute o script de configuração do dispositivo do Greengrass.
- Instale de um repositório do APT.

AWS IoT Greengrass também fornece ambientes em contêineres que executam o software AWS IoT Greengrass Core.

- Execute AWS IoT Greengrass em um contêiner Docker.
- Corra AWS IoT Greengrass em um piscar de olhos.

Baixe e extraia o AWS IoT Greengrass Pacote de software principal

Escolha o software AWS IoT Greengrass Core para sua plataforma para baixar como um arquivo tar.gz e extrair em seu dispositivo. É possível fazer download de versões recentes do software. Para obter mais informações, consulte the section called “AWS IoT Greengrass Software principal”.

Executar o script de configuração do dispositivo do Greengrass

Execute a configuração do dispositivo Greengrass para configurar seu dispositivo, instalar a versão mais recente do software AWS IoT Greengrass Core e implantar uma função Hello World Lambda em minutos. Para obter mais informações, consulte the section called “Início rápido: Configuração do dispositivo do Greengrass”.

Instale o AWS IoT Greengrass Software principal de um repositório APT

> **Important**
>
> A partir de 11 de fevereiro de 2022, você não poderá mais instalar ou atualizar o software AWS IoT Greengrass Core a partir de um repositório APT. Nos dispositivos em que você adicionou o AWS IoT Greengrass repositório, você deve remover o repositório da lista de fontes. Os dispositivos que executam o software a partir do repositório APT continuarão operando normalmente. Recomendamos que você atualize o software AWS IoT Greengrass Core usando arquivos tar.

O repositório APT fornecido pelo AWS IoT Greengrass inclui os seguintes pacotes:

- `aws-iot-greengrass-core`. Instala o software AWS IoT Greengrass Core.
- `aws-iot-greengrass-keyring`. Instala as chaves GnuPG (GPG) usadas para assinar o repositório de pacotes. AWS IoT Greengrass

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

Tópicos

- Usar scripts systemd para gerenciar o ciclo de vida do daemon do Greengrass
- Desinstale o AWS IoT Greengrass software principal usando o repositório APT
- Remova o AWS IoT Greengrass fontes principais do repositório de software

Usar scripts systemd para gerenciar o ciclo de vida do daemon do Greengrass

O `aws-iot-greengrass-core` pacote também instala `systemd` scripts que você pode usar para gerenciar o ciclo de vida do software AWS IoT Greengrass Core (daemon).

- Como iniciar o daemon do Greengrass durante a inicialização:

```
systemctl enable greengrass.service
```

- Como iniciar o daemon do Greengrass:

```
systemctl start greengrass.service
```

- Como interromper o daemon do Greengrass:

```
systemctl stop greengrass.service
```

- Como verificar o status do daemon do Greengrass:

```
systemctl status greengrass.service
```

Desinstale o AWS IoT Greengrass software principal usando o repositório APT

Ao desinstalar o software AWS IoT Greengrass principal, você pode escolher se deseja preservar ou remover as informações de configuração do software AWS IoT Greengrass principal, como certificados do dispositivo, informações do grupo e arquivos de log.

Para desinstalar o AWS IoT Greengrass software principal e preserva as informações de configuração

- Execute o comando a seguir para remover os AWS IoT Greengrass principais pacotes de software e preservar as informações de configuração na `/greengrass` pasta.

  ```
  sudo apt remove aws-iot-greengrass-core aws-iot-greengrass-keyring
  ```

Para desinstalar o AWS IoT Greengrass software principal e remova as informações de configuração

1. Execute o comando a seguir para remover os AWS IoT Greengrass principais pacotes de software e remover as informações de configuração do`/greengrass folder`.

   ```
   sudo apt purge aws-iot-greengrass-core aws-iot-greengrass-keyring
   ```

2. Remova o repositório de software AWS IoT Greengrass principal da sua lista de fontes. Para obter mais informações, consulte Remova o AWS IoT Greengrass fontes principais do repositório de software.

Remova o AWS IoT Greengrass fontes principais do repositório de software

Você pode remover as fontes do repositório AWS IoT Greengrass principal do software quando não precisar mais instalar ou atualizar o software AWS IoT Greengrass principal do repositório APT. Depois de 11 de fevereiro de 2022, você deve remover o repositório da sua lista de fontes para evitar um erro durante a execução do `apt update`.

Para remover o repositório APT da lista de fontes

- Execute os comandos a seguir para remover o repositório de software AWS IoT Greengrass principal da lista de fontes.

```
sudo rm /etc/apt/sources.list.d/greengrass.list
sudo apt update
```

Executar AWS IoT Greengrass em um contêiner Docker

AWS IoT Greengrass fornece imagens do Dockerfile e do Docker que facilitam a execução do software AWS IoT Greengrass Core em um contêiner Docker. Para obter mais informações, consulte the section called “AWS IoT Greengrass Software Docker”.

> **Note**
>
> Também é possível executar um aplicativo do Docker em um dispositivo de núcleo do Greengrass. Para isso, use o conector de implantação do aplicativo Docker do Greengrass.

Executar AWS IoT Greengrass em um piscar de olhos

AWS IoT Greengrass O snap 1.11.x permite que você execute uma versão limitada AWS IoT Greengrass por meio de pacotes de software convenientes, junto com todas as dependências necessárias, em um ambiente em contêiner.

Em 31 de dezembro de 2023, AWS IoT Greengrass encerrará a manutenção da versão 1.11.x do software AWS IoT Greengrass principal do Snap, publicada no snapcraft.io. https://snapcraft.io/aws-iot-greengrass Os dispositivos que atualmente executam o Snap continuarão funcionando até novo

aviso. No entanto, o Snap AWS IoT Greengrass principal não receberá mais patches de segurança ou correções de erros após o término da manutenção.

Conceitos do snap

A seguir estão os conceitos essenciais do snap para ajudar você a entender como usar o snap do AWS IoT Greengrass :

Channel (Canal)

Um componente do snap que define qual versão de um snap é instalada e rastreada para atualizações. Os snaps são atualizados automaticamente para a versão mais recente do canal atual.

Interface

Um componente do snap que concede acesso a recursos, como redes e arquivos do usuário.

Para executar o AWS IoT Greengrass snap, as seguintes interfaces devem estar conectadas. Observe que `greengrass-support-no-container` deve ser conectado primeiro e nunca desconectado.

```
- greengrass-support-no-container
- hardware-observe
- home-for-hooks
- hugepages-control
- log-observe
- mount-observe
- network
- network-bind
- network-control
- process-control
- system-observe
```

As outras interfaces são opcionais. Se suas funções do Lambda exigirem acesso a recursos específicos, talvez você precise se conectar às interfaces apropriadas.

Atualizar

Os snaps são atualizados automaticamente. O daemon do `snapd` é o gerenciador de pacotes snap que, por padrão, verifica as atualizações quatro vezes ao dia. Cada verificação de atualização é chamada de atualização. Quando ocorre uma atualização, o daemon é interrompido, o snap é atualizado e, em seguida, o daemon é reiniciado.

Para obter mais informações, consulte o site do Snapcraft.

O que há de novo com AWS IoT Greengrass snap v1.11.x

A seguir, descrevemos o que há de novo e o que mudou com a versão 1.11.x do snap do AWS IoT Greengrass .

- Essa versão só oferece suporte ao usuário `snap_daemon`, exposto com o ID de usuário (UID) e do grupo (GID). `584788`
- Essa versão só oferece suporte às funções do Lambda não conteinerizadas.

> **Important**
>
> Como as funções do Lambda não conteinerizadas devem compartilhar o mesmo usuário (`snap_daemon`), as funções do Lambda não têm isolamento umas das outras. Para obter mais informações, consulte Controlando a execução de funções do Lambda do Greengrass usando a configuração específica ao grupo.

- Esta versão oferece suporte aos tempos de execução C, C++, Java 8, Node.js 12.x, Python 2.7, Python 3.7 e Python 3.8.

> **Note**
>
> Para evitar runtimes redundantes no Python, as funções do Lambda no Python 3.7 realmente executam o runtime do Python 3.8.

Conceitos básicos de AWS IoT Greengrass estalar

O procedimento a seguir ajuda você a instalar e configurar o AWS IoT Greengrass snap no seu dispositivo.

Requisitos

Para executar o AWS IoT Greengrass snap, você deve fazer o seguinte:

- Execute o AWS IoT Greengrass snap em uma distribuição Linux compatível, como Ubuntu, Linux Mint, Debian e Fedora.
- Instale o daemon do `snapd` no seu dispositivo. O daemon do `snapd`, incluindo a ferramenta `snap`, gerencia o ambiente de snap em seu dispositivo.

Para ver a lista de distribuições compatíveis do Linux e as instruções de instalação, consulte Instalando o snapd na Documentação do Snap.

Instale e configure o AWS IoT Greengrass estalar

O tutorial a seguir mostra como instalar e configurar o AWS IoT Greengrass snap no seu dispositivo.

> **Note**
>
> - Embora este tutorial use uma instância do Amazon EC2 (x86 t2.micro Ubuntu 20.04), você pode executar o AWS IoT Greengrass snap com hardware físico, como um Raspberry Pi.
> - O daemon do `snapd` está pré-instalado no Ubuntu.

1. Instale o snap do `core18` executando o seguinte comando no terminal do dispositivo:

   ```
   sudo snap install core18
   ```

   O snap do `core18` é um snap básico que fornece um ambiente de runtime com bibliotecas comumente usadas. Esse snap foi compilado a partir do Ubuntu 18.04 LTS.
2. Atualize o `snapd` executando o seguinte comando.

   ```
   sudo snap install --channel=edge snapd; sudo snap refresh --channel=edge snapd
   ```
3. Execute o `snap list` comando para verificar se você tem o AWS IoT Greengrass snap instalado.

   O exemplo de resposta a seguir mostra que o `snapd` está instalado, mas o `aws-iot-greengrass` não está.

   ```
   Name              Version                 Rev    Tracking         Publisher   Notes
   amazon-ssm-agent  3.0.161.0               2996   latest/stable/…  aws#        classic
   core              16-2.48                 10444  latest/stable    canonical#  core
   core18            20200929                1932   latest/stable    canonical#  base
   lxd               4.0.4                   18150  4.0/stable/…     canonical#  -
   snapd             2.48+git548.g929ccfb    10526  latest/edge      canonical#  snapd
   ```
4. Escolha uma das seguintes opções para instalar o AWS IoT Greengrass snap 1.11.x.

   - Para instalar o AWS IoT Greengrass snap, execute o seguinte comando:

```
sudo snap install aws-iot-greengrass
```

Exemplo de resposta:

```
aws-iot-greengrass 1.11.5 from Amazon Web Services (aws) installed
```

- Para migrar de uma versão anterior para a v1.11.x ou atualizar para a versão de patch mais recente disponível, execute o seguinte comando:

```
sudo snap refresh --channel=1.11.x aws-iot-greengrass
```

Como outros snaps, o AWS IoT Greengrass snap usa canais para gerenciar versões secundárias. Os snaps são atualizados automaticamente para a versão mais recente disponível do canal atual. Por exemplo, se você especificar`--channel=1.11.x`, seu AWS IoT Greengrass snap será atualizado para a versão v1.11.5.

Você pode executar o `snap info aws-iot-greengrass` comando para obter a lista de canais disponíveis para AWS IoT Greengrass.

Exemplo de resposta:

```
name:      aws-iot-greengrass
summary:   AWS supported software that extends cloud capabilities to local devices.
publisher: Amazon Web Services (aws#)
store-url: https://snapcraft.io/aws-iot-greengrass
contact:   https://repost.aws/tags/TA4ckIed1sR4enZBey29rKTg/aws-io-t-greengrass
license:   Proprietary
description: |
  AWS IoT Greengrass seamlessly extends AWS onto edge devices so they can act
 locally on the data
  they generate, while still using the cloud for management, analytics, and durable
 storage.
  AWS IoT Greenrgrass snap v1.11.0 enables you to run a limited version of AWS IoT
 Greengrass with
  all necessary dependencies in a containerized environment.
  The AWS IoT Greengrass snap doesn't support connectors and machine learning (ML)
 inference.
  By downloading this software you agree to the Greengrass Core Software License
 Agreement
```

```
  (https://s3-us-west-2.amazonaws.com/greengrass-release-license/greengrass-
license-v1.pdf).
  For more information, see Run AWS IoT Greengrass in a snap
  (https://docs.aws.amazon.com/greengrass/latest/developerguide/install-
ggc.html#gg-snap-support) in
  the AWS IoT Greengrass Developer.
  If you need help, try the AWS IoT Greengrass tag on AWS re:Post
  (https://repost.aws/tags/TA4ckIed1sR4enZBey29rKTg/aws-io-t-greengrass).
snap-id: SRDuhPJGj4XPxFNNZQKOTvURAp0wxKnd
channels:
  latest/stable:    1.11.3 2021-06-15 (59) 111MB -
  latest/candidate: 1.11.3 2021-06-14 (59) 111MB -
  latest/beta:      1.11.3 2021-06-14 (59) 111MB -
  latest/edge:      1.11.3 2021-06-14 (59) 111MB -
  1.11.x/stable:    1.11.3 2021-06-15 (59) 111MB -
  1.11.x/candidate: 1.11.3 2021-06-15 (59) 111MB -
  1.11.x/beta:      1.11.3 2021-06-15 (59) 111MB -
  1.11.x/edge:      1.11.3 2021-06-15 (59) 111MB -
```

5. Para acessar recursos específicos os quais suas funções do Lambda precisam, você pode se conectar a outras interfaces.

   Execute o comando a seguir para obter a lista de interfaces suportadas pelo AWS IoT Greengrass snap:

   ```
   snap connections aws-iot-greengrass
   ```

   Exemplo de resposta:

   ```
   Interface               Plug                                              Slot
                 Notes
   camera                  aws-iot-greengrass:camera                         -
                 -
   dvb                     aws-iot-greengrass:dvb                            -
                 -
   gpio                    aws-iot-greengrass:gpio                           -
                 -
   gpio-memory-control     aws-iot-greengrass:gpio-memory-control            -
                 -
   greengrass-support      aws-iot-greengrass:greengrass-support-no-container
     :greengrass-support  -
   ```

```
hardware-observe         aws-iot-greengrass:hardware-observe
  :hardware-observe    manual
hardware-random-control  aws-iot-greengrass:hardware-random-control          -
               -
home                     aws-iot-greengrass:home-for-greengrassd             -
               -
home                     aws-iot-greengrass:home-for-hooks                   :home
               manual
hugepages-control        aws-iot-greengrass:hugepages-control
  :hugepages-control   manual
i2c                      aws-iot-greengrass:i2c                              -
               -
iio                      aws-iot-greengrass:iio                              -
               -
joystick                 aws-iot-greengrass:joystick                         -
               -
log-observe              aws-iot-greengrass:log-observe                      :log-
observe         manual
mount-observe            aws-iot-greengrass:mount-observe
  :mount-observe       manual
network                  aws-iot-greengrass:network
  :network             -
network-bind             aws-iot-greengrass:network-bind
  :network-bind        -
network-control          aws-iot-greengrass:network-control
  :network-control     -
opengl                   aws-iot-greengrass:opengl
  :opengl              -
optical-drive            aws-iot-greengrass:optical-drive
  :optical-drive       -
process-control          aws-iot-greengrass:process-control
  :process-control     -
raw-usb                  aws-iot-greengrass:raw-usb                          -
               -
removable-media          aws-iot-greengrass:removable-media                  -
               -
serial-port              aws-iot-greengrass:serial-port                      -
               -
spi                      aws-iot-greengrass:spi                              -
               -
system-observe           aws-iot-greengrass:system-observe
  :system-observe      -
```

Se houver um hífen (-) na coluna Slot, a interface correspondente não está conectada.

6. Siga Instalando o software AWS IoT Greengrass Core para criar AWS IoT algo, um grupo do Greengrass, recursos de segurança que permitem comunicações seguras com AWS IoT e o arquivo de configuração do software AWS IoT Greengrass Core. O arquivo de configuração,`config.json`, contém configurações específicas para seu núcleo do Greengrass, como a localização dos arquivos de certificado e o endpoint de dados do AWS IoT dispositivo.

   > **Note**
   >
   > Se você baixou o arquivo em um dispositivo diferente, siga esta etapa para transferir os arquivos para o dispositivo AWS IoT Greengrass principal.

7. Para o AWS IoT Greengrass snap, certifique-se de atualizar o arquivo config.json, conforme mostrado a seguir:

   - Substitua cada instância *`certificateId`* de pelo ID do certificado no nome do certificado e dos arquivos de chave.
   - Se você baixou um certificado de CA raiz da Amazon diferente do Amazon Root CA 1, substitua cada instância do *`AmazonRootCA1.pem`* pelo nome do arquivo da CA raiz da Amazon.

```
{
  ...
  "crypto" : {
    "principals" : {
      "SecretsManager" : {
        "privateKeyPath" : "file:///snap/aws-iot-greengrass/current/greengrass/
certs/certificateId-private.pem.keyy"
      },
      "IoTCertificate" : {
        "privateKeyPath" : "file:///snap/aws-iot-greengrass/current/greengrass/
certs/certificateId-private.pem.key",
        "certificatePath" : "file:///snap/aws-iot-greengrass/current/greengrass/
certs/certificateId-certificate.pem.crt"
      }
    },
    "caPath" : "file:///snap/aws-iot-greengrass/current/greengrass/
certs/AmazonRootCA1.pem"
```

```
  },
  "writeDirectory": "/var/snap/aws-iot-greengrass/current/ggc-write-directory",
  "pidFileDirectory": "/var/snap/aws-iot-greengrass/current/pidFileDirectory"
}
```

8. Execute o comando a seguir para adicionar seu AWS IoT Greengrass certificado e arquivos de configuração:

```
sudo snap set aws-iot-greengrass gg-certs=/home/ubuntu/my-certs
```

Implantar uma função do Lambda

Esta seção mostra como implantar uma função do Lambda gerenciada pelo cliente no snap do AWS IoT Greengrass .

> **Important**
>
> AWS IoT Greengrass O snap v1.11 só oferece suporte a funções Lambda não conteinerizadas.

1. Execute o seguinte comando para iniciar o AWS IoT Greengrass daemon:

```
sudo snap start aws-iot-greengrass
```

Exemplo de resposta:

```
Started.
```

> **Note**
>
> Se você receber um erro, poderá usar o comando do `snap run` para obter uma mensagem de erro detalhada. Para obter mais solução de problemas, consulte erro: não é possível executar as seguintes tarefas: - Executar o comando de serviço "start" para serviços ["greengrassd"] do snap "aws-iot-greengrass" ([start snap.aws-iot-greengrass.greengrassd.service] falhou com o status de saída 1: o trabalho do snap.aws-iot-greengrass.greengrassd.service falhou porque o processo de

controle foi encerrado com o código de erro. Consulte "systemctl status snap.aws-iot-greengrass.greengrassd.service" e "journalctl -xe" para obter detalhes.).

2. Execute o comando a seguir para confirmar se o daemon está em execução:

```
snap services aws-iot-greengrass.greengrassd
```

Exemplo de resposta:

```
Service                          Startup   Current  Notes
aws-iot-greengrass.greengrassd   disabled  active   -
```

3. Siga o Módulo 3 (parte 1): a função do Lambda no AWS IoT Greengrass para criar e implantar uma função do Lambda Hello World . No entanto, antes de implantar a função do Lambda, conclua a próxima etapa.
4. Certifique-se de que sua função do Lambda seja executada como usuário do `snap_daemon` e no modo sem contêiner. Para atualizar as configurações do seu grupo do Greengrass, faça o seguinte no AWS IoT Greengrass console:
   a. Faça login no AWS IoT Greengrass console.
   b. No painel de navegação do AWS IoT console, em Gerenciar, expanda dispositivos Greengrass e escolha Grupos (V1).
   c. Em Grupos do Greengrass, selecione o grupo-alvo.
   d. Na página de configuração do grupo, no painel de navegação, selecione a guia Funções do Lambda.
   e. Em Ambiente de runtime da função do Lambda padrão, selecione Editar e faça o seguinte:
      i. Em Usuário e grupo padrão do sistema, escolha Outra ID/group ID de usuário e insira **584788** a ID do usuário do sistema (número) e a ID do grupo do sistema (número).
      ii. Em Ambiente de runtime da função do Lambda padrão, selecione Editar e faça o seguinte:
      iii. Escolha Salvar.

Parando o AWS IoT Greengrass daemon

Você pode usar o comando `snap stop` para interromper um serviço.

Para parar o AWS IoT Greengrass daemon, execute o seguinte comando:

```
sudo snap stop aws-iot-greengrass
```

O comando deve retornar um `Stopped.`.

Para verificar se você interrompeu o snap com êxito, execute o comando a seguir :

```
snap services aws-iot-greengrass.greengrassd
```

Exemplo de resposta:

```
Service                         Startup   Current   Notes
aws-iot-greengrass.greengrassd  disabled  inactive  -
```

Desinstalando o AWS IoT Greengrass estalar

Para desinstalar o AWS IoT Greengrass snap, execute o seguinte comando:

```
sudo snap remove aws-iot-greengrass
```

Exemplo de resposta:

```
aws-iot-greengrass removed
```

Solução de problemas do AWS IoT Greengrass estalar

Use as informações a seguir para ajudar a solucionar problemas com o AWS IoT Greengrass snap.

Obteve erros de permissão negada

Solução: os erros de permissão negada geralmente ocorrem devido à falta de interfaces. Para obter a lista de interfaces ausentes e informações detalhadas sobre a solução de problemas, você pode usar a ferramenta `snappy-debug`.

Execute o comando a seguir para instalar a ferramenta.

```
sudo snap install snappy-debug
```

Exemplo de resposta:

```
snappy-debug 0.36-snapd2.45.1 from Canonical# installed
```

Execute o comando `sudo snappy-debug` em uma sessão separada do terminal. A operação continua até que ocorra um erro de permissão negada.

Por exemplo, se sua função do Lambda tentar ler um arquivo no diretório `$HOME`, você poderá obter a seguinte resposta:

```
INFO: Following '/var/log/syslog'. If have dropped messages, use:
INFO: $ sudo journalctl --output=short --follow --all | sudo snappy-debug
kernel.printk_ratelimit = 0
= AppArmor =
Time: Dec  6 04:48:26
Log: apparmor="DENIED" operation="mknod" profile="snap.aws-iot-greengrass.greengrassd"
 name="/home/ubuntu/my-file.txt" pid=12345 comm="touch" requested_mask="c"
 denied_mask="c" fsuid=0 ouid=0
File: /home/ubuntu/my-file.txt (write)
Suggestion:
* add 'home' to 'plugs'
```

Este exemplo mostra que a criação do arquivo `/home/ubuntu/my-file.txt` causou o erro de permissão. Também sugere que você adicione home a `plugs`. No entanto, essa sugestão não é aplicável. Os plugues `home-for-greengrassd` e `home-for-hooks` só recebem apenas acesso para leitura.

Para obter mais informações, consulte O snap snappy-debug na Documentação do Snap.

erro: não é possível executar as seguintes tarefas: - Executar o comando de serviço "start" para serviços ["greengrassd"] do snap "aws-iot-greengrass" ([start snap.aws-iot-greengrass.greengrassd.service] falhou com o status de saída 1: o trabalho do snap.aws-iot-greengrass.greengrassd.service falhou porque o processo de controle foi encerrado com o código de erro. Consulte "systemctl status snap.aws-iot-greengrass.greengrassd.service" e "journalctl -xe" para obter detalhes.)

Solução: Você pode ver esse erro quando o `snap start aws-iot-greengrass` comando falha ao iniciar o software AWS IoT Greengrass Core.

Para obter mais informações sobre a solução de problemas, execute o comando a seguir:

```
sudo snap run aws-iot-greengrass.greengrassd
```

Exemplo de resposta:

```
Couldn't find /snap/aws-iot-greengrass/44/greengrass/config/config.json.
```

Esses exemplos mostram que AWS IoT Greengrass não foi possível encontrar o `config.json` arquivo. Você pode verificar os arquivos de configuração e de certificado.

/var/snapgreengrass/current/aws-iot-/ggc-write- /1.11. directory/packages 5/rootfs/merged não é um caminho absoluto nem é um link simbólico.

Solução: o AWS IoT Greengrass snap é compatível somente com funções do Lambda não conteinerizadas. Certifique-se de que sua função do Lambda seja executada como usuário e no modo sem contêiner. Para obter mais informações, consulte Considerações ao escolher a conteinerização da função do Lambda no Guia do desenvolvedor do AWS IoT Greengrass Version 1 .

O daemon snapd falhou ao reiniciar depois que você executou o comando sudo snap refresh snapd command.

Solução: Siga as etapas de 6 a 8 Instale e configure o AWS IoT Greengrass estalar para adicionar o AWS IoT Greengrass certificado e os arquivos de configuração ao AWS IoT Greengrass snap.

Arquivar um AWS IoT Greengrass Instalação do software principal

Ao atualizar para uma nova versão do software AWS IoT Greengrass Core, você pode arquivar a versão atualmente instalada. Isso preserva o ambiente da sua instalação atual para que você possa testar uma nova versão de software no mesmo hardware. Isso também facilita a reversão para a versão arquivada por qualquer motivo.

Para arquivar a instalação atual e instalar uma nova versão

1. Faça download do pacote de instalação do software do AWS IoT Greengrass Core para o qual você deseja atualizar.
2. Copie o pacote para o dispositivo de núcleo de destino. Para obter instruções que mostram como transferir arquivos, consulte esta etapa.

> **Note**
>
> Posteriormente, copie seus certificados atuais, chaves e o arquivo de configuração para a nova instalação.

Execute os comandos de acordo com as etapas a seguir no terminal de dispositivo de núcleo.

3. Verifique se o daemon do Greengrass está parado no dispositivo de núcleo.

   a. Para verificar se o daemon está em execução:

   ```
   ps aux | grep -E 'greengrass.*daemon'
   ```

   Se a saída contém uma entrada `root` para `/greengrass/ggc/packages/`*`ggc-version`*`/bin/daemon`, o daemon está em execução.

   > **Note**
   >
   > Esse procedimento é escrito com a suposição de que o software AWS IoT Greengrass Core está instalado no `/greengrass` diretório.

   b. Para interromper o daemon do :

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

4. Mova o diretório raiz do Greengrass atual para um diretório diferente.

   ```
   sudo mv /greengrass /greengrass_backup
   ```

5. Descompacte o novo software no dispositivo de núcleo. Substitua os *`version`* espaços reservados *`os-architecture`* e no comando.

   ```
   sudo tar -zxvf greengrass-os-architecture-version.tar.gz -C /
   ```

6. Copie os certificados arquivados, as chaves e o arquivo de configuração para a nova instalação.

   ```
   sudo cp /greengrass_backup/certs/* /greengrass/certs
   ```

```
sudo cp /greengrass_backup/config/* /greengrass/config
```

7. Inicie o daemon:

```
cd /greengrass/ggc/core/
sudo ./greengrassd start
```

Agora, você pode fazer uma implantação do grupo para testar a nova instalação. Se algo falhar, você poderá restaurar a instalação arquivada.

Para restaurar a instalação arquivada

1. Pare o daemon.
2. Exclua o novo diretório `/greengrass`.
3. Mova o diretório `/greengrass_backup` de volta para `/greengrass`.
4. Inicie o daemon.

Configure o AWS IoT Greengrass core

Um AWS IoT Greengrass núcleo é AWS IoT algo (dispositivo) que atua como um hub ou gateway em ambientes periféricos. Como outros AWS IoT dispositivos, existe um núcleo no registro, tem uma sombra de dispositivo e usa um certificado de dispositivo para se autenticar com AWS IoT Core e. AWS IoT Greengrass O dispositivo de núcleo executa o software de núcleo do AWS IoT Greengrass , que o permite gerenciar processos locais para grupos do Greengrass, como comunicação, sincronização de shadow e troca de tokens.

O software AWS IoT Greengrass Core fornece as seguintes funcionalidades:

- Implantação e execução local de conectores e funções do Lambda.
- Processe fluxos de dados localmente com exportações automáticas para o. Nuvem AWS
- Mensagens MQTT pela rede local entre dispositivos, conectores e funções do Lambda usando assinaturas gerenciadas.
- Mensagens MQTT entre AWS IoT dispositivos, conectores e funções do Lambda usando assinaturas gerenciadas.
- Conexões seguras entre dispositivos e o Nuvem AWS uso da autenticação e autorização do dispositivo.

- Sincronização da shadow local de dispositivos. As shadows podem ser configuradas para serem sincronizadas com a Nuvem AWS.
- Acesso controlado ao dispositivo local e recursos de volume.
- Implantação de modelos de machine learning treinados em nuvem para executar inferência local.
- Detecção automática de endereço IP que permite aos dispositivos descobrirem o dispositivo de núcleo do Greengrass.
- Implantação central de configuração de grupo nova ou atualizada. Depois que os dados de configuração forem obtidos por download, o dispositivo de núcleo será reiniciado automaticamente.
- Atualizações seguras de software por rede sem fio (OTA) das funções do Lambda definidas pelo usuário.
- Armazenamento criptografado e seguro de segredos locais e acesso controlado por conectores e funções do Lambda.

AWS IoT Greengrass arquivo de configuração principal

O arquivo de configuração do software AWS IoT Greengrass Core é `config.json`. É possível encontrá-lo no diretório `/greengrass-root/config`.

> **Note**
>
> *greengrass-root* representa o caminho em que o software AWS IoT Greengrass Core está instalado em seu dispositivo. Normalmente, esse é o diretório `/greengrass`.
> Se você usar a opção de criação de grupo padrão no AWS IoT Greengrass console, o `config.json` arquivo será implantado no dispositivo principal em um estado de funcionamento.

Você pode examinar o conteúdo desse arquivo executando o seguinte comando:

```
cat /greengrass-root/config/config.json
```

Veja a seguir um exemplo de arquivo `config.json`. Essa é a versão gerada quando você cria o núcleo a partir do AWS IoT Greengrass console.

GGC v1.11

```
{
    "coreThing": {
        "caPath": "root.ca.pem",
        "certPath": "hash.cert.pem",
        "keyPath": "hash.private.key",
        "thingArn": "arn:partition:iot:region:account-id:thing/core-thing-name",
        "iotHost": "host-prefix-ats.iot.region.amazonaws.com",
        "ggHost": "greengrass-ats.iot.region.amazonaws.com",
        "keepAlive": 600,
        "ggDaemonPort": 8000,
        "systemComponentAuthTimeout": 5000
    },
    "runtime": {
        "maxWorkItemCount": 1024,
        "maxConcurrentLimit": 25,
        "lruSize": 25,
        "mountAllBlockDevices": "no",
        "cgroup": {
            "useSystemd": "yes"
        }
    },
    "managedRespawn": false,
    "crypto": {
        "principals": {
            "SecretsManager": {
                "privateKeyPath": "file:///greengrass/certs/hash.private.key"
            },
            "IoTCertificate": {
                "privateKeyPath": "file:///greengrass/certs/hash.private.key",
                "certificatePath": "file:///greengrass/certs/hash.cert.pem"
            }
        },
        "caPath": "file:///greengrass/certs/root.ca.pem"
    },
    "writeDirectory": "/var/snap/aws-iot-greengrass/current/ggc-write-directory",
    "pidFileDirectory": "/var/snap/aws-iot-greengrass/current/pidFileDirectory"
}
```

O arquivo `config.json` é compatível com as seguintes propriedades:

coreThing

Campo	Description	Observações
`caPath`	O caminho para a CA AWS IoT raiz em relação ao *`/greengrass-root`* `/certs` diretório.	Para compatibilidade retroativa com versões anteriores à 1.7.0. Essa propriedade é ignorada quando o objeto `crypto` está presente. **Note** Certifique-se de que os endpoints correspondem ao seu tipo de certificado.
`certPath`	O caminho para o certifica do de dispositivo de núcleo relativo ao diretório *`/greengrass-root`* `/certs`.	Para compatibilidade retroativa com versões anteriores à 1.7.0. Essa propriedade é ignorada quando o objeto `crypto` está presente.
`keyPath`	O caminho para a chave privada do núcleo relativo ao diretório *`/greengrass-root`*`/certs`.	Para compatibilidade retroativa com versões anteriores à 1.7.0. Essa propriedade é ignorada quando o objeto `crypto` está presente.
`thingArn`	O nome de recurso da Amazon (ARN) da AWS IoT coisa que represent a o dispositivo AWS IoT Greengrass principal.	Encontre o ARN do seu núcleo no AWS IoT Greengrass console em Núcleos ou executando o comando `aws greengras`

Campo	Description	Observações
		`s get-core-definitio n-version` CLI.
`iotHost`	Seu AWS IoT endpoint.	Encontre o endpoint no AWS IoT console em Configurações ou executando o comando `aws iot describe-endpoint --endpoint-type iot:Data-ATS` CLI. Este comando retorna o endpoint do Amazon Trust Services (ATS). Para obter mais informações, consulte a documentação de Autenticação do servidor. **Note** Certifique-se de que os endpoints correspondem ao seu tipo de certificado. Verifique se os endpoints correspondem à sua Região da AWS.

Campo	Description	Observações
`ggHost`	Seu AWS IoT Greengrass endpoint.	Este é o seu endpoint `iotHost` com o prefixo do host substituído pelo greengrass (por exemplo, `greengrass-ats.iot.`*`region`*`.amazonaws.com`). Use o mesmo Região da AWS que`iotHost`. **Note** Certifique-se de que os endpoints correspondem ao seu tipo de certificado. Verifique se os endpoints correspondem à sua Região da AWS.
`iotMqttPort`	Opcional. O número da porta a ser usada para comunicação MQTT com AWS IoT.	Os valores válidos são `8883` ou `443`. O valor padrão é `8883`. Para obter mais informações, consulte Conectar-se à porta 443 ou por meio de um proxy de rede.

Campo	Description	Observações
`iotHttpPort`	Opcional. O número da porta usada para criar as conexões HTTPS para AWS IoT.	Os valores válidos são `8443` ou `443`. O valor padrão é `8443`. Para obter mais informações, consulte Conectar-se à porta 443 ou por meio de um proxy de rede.
`ggMqttPort`	Opcional. O número da porta a ser usada para comunicação MQTT pela rede local.	Os valores válidos vão de `1024` a `65535`. O valor padrão é `8883`. Para obter mais informações, consulte the section called “Porta MQTT para mensagens locais”.
`ggHttpPort`	Opcional. O número da porta usada para criar as conexões HTTPS para o serviço AWS IoT Greengrass .	Os valores válidos são `8443` ou `443`. O valor padrão é `8443`. Para obter mais informações, consulte Conectar-se à porta 443 ou por meio de um proxy de rede.
`keepAlive`	Opcional. O período `KeepAlive` do MQTT, em segundos.	O intervalo válido é entre 30 e 1200. O valor padrão é `600`.
`networkProxy`	Opcional. Um objeto que define um servidor de proxy para se conectar.	O servidor proxy pode ser HTTP ou HTTPS. Para obter mais informações, consulte Conectar-se à porta 443 ou por meio de um proxy de rede.

Campo	Description	Observações
`mqttOperationTimeout`	Opcional. A quantidade de tempo (em segundos) para permitir que o núcleo do Greengrass conclua uma operação de publicação, assinatura ou cancelamento de assinatura em conexões MQTT com o AWS IoT Core.	O valor padrão é 5. O valor mínimo é 5.
`ggDaemonPort`	Opcional. O número da porta IPC do núcleo do Greengras s.	Essa propriedade está disponível na AWS IoT Greengrass versão v1.11.0 ou posterior. Os valores válidos estão entre 1.024 e 65.535. O valor padrão é 8000.
`systemComponentAut hTimeout`	Opcional. O tempo (em milissegundos) para permitir que o IPC do núcleo do Greengrass conclua a autenticação.	Essa propriedade está disponível na AWS IoT Greengrass versão v1.11.0 ou posterior. Os valores válidos estão entre 500 e 5000. O valor padrão é 5000.

runtime

Campo	Description	Observações
`maxWorkItemCount`	Opcional. O número máximo de itens de trabalho que o daemon do Greengrass pode processar por vez. As	O valor padrão é 1024. O valor máximo é limitado pelo hardware do dispositivo.

Campo	Description	Observações
	solicitações que excederem esse limite serão ignoradas. A fila de itens de trabalho é compartilhada por component es do sistema, funções do Lambda definidas pelo usuário, e conectores.	Aumentar esse valor aumenta a memória que AWS IoT Greengrass usa. Você pode aumentar esse valor caso espere que seu núcleo receba tráfego pesado de mensagens MQTT.
`maxConcurrentLimit`	Opcional. O número máximo de trabalhadores do Lambda não fixados simultâneos que o daemon do Greengras s pode ter. Você pode especificar um número inteiro diferente para substituir esse parâmetro.	O valor padrão é 25. O valor mínimo é definido por `lruSize`.
`lruSize`	Opcional. Define o valor mínimo para`maxConcur rentLimit` .	O valor padrão é 25.
`mountAllBlockDevices`	Opcional. Permite AWS IoT Greengrass usar montagens de ligação para montar todos os dispositivos de bloco em um contêiner após configurar o OverlayFS.	Essa propriedade está disponível na AWS IoT Greengrass versão v1.11.0 ou posterior. Os valores válidos são `yes` e `no`. O valor padrão é `no`. Defina esse valor como `yes` se seu diretório `/usr` não estiver sob a hierarquia `/`.

Campo	Description	Observações
`postStartHealthCheckTimeout`	Opcional. O tempo (em milissegundos) que o daemon Greengrass aguarda a conclusão da verificação de integridade depois de iniciar.	O tempo limite padrão é de 30 segundos (30000 ms).
`cgroup`		
`useSystemd`	Indica se o dispositivo usa `systemd`.	Os valores válidos são `yes` ou `no`. Execute o script `check_ggc_dependencies` no Módulo 1 para ver se o dispositivo usa `systemd`.

crypto

A `crypto` apresenta propriedades que oferecem suporte ao armazenamento da chave privada em um módulo de segurança de hardware (HSM) por meio de PKCS#11 e armazenamento local secreto. Para obter mais informações, consulte the section called “Entidades principais de segurança”, the section called “Integração de segurança de hardware” e Implantar segredos no núcleo. Configurações para o armazenamento da chave privada em HSMs ou no sistema de arquivo são compatíveis.

Campo	Description	Observações
`caPath`	O caminho absoluto para a CA AWS IoT raiz.	Deve ser um URI de arquivo no formato: `file:///absolute/path/to/file` . **Note** Certifique-se de que os endpoints

Campo	Description	Observações
		correspondem ao seu tipo de certificado.
`PKCS11`		
`OpenSSLEngine`	Opcional. O caminho absoluto para o arquivo `.so` do mecanismo OpenSSL para habilitar o suporte PKCS#11 no OpenSSL.	Deve ser um caminho para o arquivo no sistema de arquivos. Esta propriedade será necessária se você estiver usando o atendente de atualizações OTA do Greengrass com segurança de hardware. Para obter mais informações, consulte the section called “Configurar as atualizações OTA”.
`P11Provider`	O caminho absoluto para a biblioteca carregável libdl da implementação PKCS#11.	Deve ser um caminho para o arquivo no sistema de arquivos.
`slotLabel`	O rótulo de slot usado para identificar o módulo de hardware.	Deve estar em conformidade com as especificações de rótulo PKCS#11.
`slotUserPin`	O PIN do usuário usado para autenticar o núcleo do Greengrass no módulo.	É necessário ter permissões suficientes para executar C_Sign com as chaves privadas configuradas.
`principals`		
`IoTCertificate`	O certificado e a chave privada que o núcleo usa para fazer solicitações à AWS IoT.	

Campo	Description	Observações
`IoTCertificate.privateKeyPath`	O caminho para a chave privada do núcleo.	Para o armazenamento do sistema de arquivos, deve ser um URI de arquivo no formato: `file:///`*`absolute/path/to/file`* . Para o armazenamento HSM, deve ser um caminho RFC 7512 PKCS#11 que especifica o rótulo do objeto.
`IoTCertificate.certificatePath`	O caminho absoluto para o certificado do dispositivo do núcleo.	Deve ser um URI de arquivo no formato: `file:///`*`absolute/path/to/file`* .
`MQTTServerCertificate`	Opcional. A chave privada que o núcleo usa em combinação com o certificado para atuar como um gateway ou servidor MQTT.	

Campo	Description	Observações
`MQTTServerCertificate.privateKeyPath`	O caminho para a chave privada do servidor MQTT local.	Use esse valor para especificar sua própria chave privada para o servidor MQTT local. Para o armazenamento do sistema de arquivos, deve ser um URI de arquivo no formato: `file:///`*`absolute/path/to/file`*. Para o armazenamento HSM, deve ser um caminho RFC 7512 PKCS#11 que especifica o rótulo do objeto. Se essa propriedade for omitida, AWS IoT Greengrass gira a chave com base nas configurações de rotação. Se especificado, o cliente será responsável por girar a chave.
`SecretsManager`	A chave privada que protege a chave de dados usada para criptografia. Para obter mais informações, consulte Implantar segredos no núcleo.	

Campo	Description	Observações
`SecretsManager .privateKeyPath`	O caminho para a chave privada do secrets manager local.	Somente uma chave RSA é compatível. Para o armazenamento do sistema de arquivos, deve ser um URI de arquivo no formato: `file:///`*`absolute/path/to/file`* . Para o armazenamento HSM, deve ser um caminho RFC 7512 PKCS#11 que especific a o rótulo do objeto. A chave privada deve ser gerada usando o mecanismo de preenchimento PKCS#1 v1.5.

As seguintes propriedades de configuração também são compatíveis:

Campo	Description	Observações
`mqttMaxConnectionR etryInterval`	Opcional. O intervalo máximo (em segundos) entre novas tentativas de conexão MQTT caso a conexão caia.	Especifique esse valor como um inteiro não assinado. O padrão é `60`.
`managedRespawn`	Opcional. Indica que o atendente OTA precisa executar o código personali zado antes de uma atualizaç ão.	Os valores válidos são `true` ou `false`. Para obter mais informações, consulte Atualizações OTA de AWS IoT Greengrass Software principal.

Campo	Description	Observações
`writeDirectory`	Opcional. O diretório de gravação em que AWS IoT Greengrass cria todos os read/write recursos.	Para obter mais informaçõ es, consulte Configurar um diretório de gravação para AWS IoT Greengrass.
pid FileDirectory	Opcional. AWS IoT Greengrass armazena seu ID de processo (PID) nesse diretório.	O valor padrão é `/var/run`.

Extended life versions

As seguintes versões do software AWS IoT Greengrass Core estão na fase de vida útil estendida. Essas informações estão incluídas apenas para fins de referência.

GGC v1.10

```
{
  "coreThing" : {
    "caPath" : "root.ca.pem",
    "certPath" : "hash.cert.pem",
    "keyPath" : "hash.private.key",
    "thingArn" : "arn:partition:iot:region:account-id:thing/core-thing-name",
    "iotHost" : "host-prefix-ats.iot.region.amazonaws.com",
    "ggHost" : "greengrass-ats.iot.region.amazonaws.com",
    "keepAlive" : 600,
    "systemComponentAuthTimeout": 5000
  },
  "runtime" : {
    "maxWorkItemCount" : 1024,
    "maxConcurrentLimit" : 25,
    "lruSize": 25,
    "cgroup" : {
      "useSystemd" : "yes"
    }
  },
  "managedRespawn" : false,
  "crypto" : {
    "principals" : {
```

```
      "SecretsManager" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key"
      },
      "IoTCertificate" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key",
        "certificatePath" : "file:///greengrass/certs/hash.cert.pem"
      }
    },
    "caPath" : "file:///greengrass/certs/root.ca.pem"
  }
}
```

O arquivo `config.json` é compatível com as seguintes propriedades:

coreThing

Campo	Description	Observações
`caPath`	O caminho para a CA AWS IoT raiz em relação ao */greengrass-root* / `certs` diretório.	Para compatibilidade retroativa com versões anteriores à 1.7.0. Essa propriedade é ignorada quando o objeto `crypto` está presente. **Note** Certifique-se de que os endpoints correspondem ao seu tipo de certificado.
`certPath`	O caminho para o certificado de dispositivo de núcleo relativo ao diretório */greengrass-root* / `certs`.	Para compatibilidade retroativa com versões anteriores à 1.7.0. Essa propriedade é ignorada quando o objeto `crypto` está presente.

Campo	Description	Observações
`keyPath`	O caminho para a chave privada do núcleo relativo ao diretório */greengrass-root* `/certs`.	Para compatibilidade retroativa com versões anteriores à 1.7.0. Essa propriedade é ignorada quando o objeto `crypto` está presente.
`thingArn`	O nome de recurso da Amazon (ARN) da AWS IoT coisa que represent a o dispositivo AWS IoT Greengrass principal.	Encontre o ARN do seu núcleo no AWS IoT Greengrass console em Núcleos ou executand o o comando `aws greengrass get-core-definition-version` CLI.

Campo	Description	Observações
`iotHost`	Seu AWS IoT endpoint.	Encontre o endpoint no AWS IoT console em Configurações ou executando o comando `aws iot describe-endpoint --endpoint-type iot:Data-ATS` CLI. Este comando retorna o endpoint do Amazon Trust Services (ATS). Para obter mais informações, consulte a documentação de Autenticação do servidor. **Note** Certifique-se de que os endpoints correspondem ao seu tipo de certificado. Verifique se os endpoints correspondem à sua Região da AWS.

Campo	Description	Observações
`ggHost`	Seu AWS IoT Greengrass endpoint.	Este é o seu endpoint `iotHost` com o prefixo do host substituído pelo greengrass (por exemplo, `greengrass-ats.iot.`*`region`*`.amazonaws.com`). Use o mesmo Região da AWS que`iotHost`. **Note** Certifique-se de que os endpoints correspondem ao seu tipo de certificado. Verifique se os endpoints correspondem à sua Região da AWS.
`iotMqttPort`	Opcional. O número da porta a ser usada para comunicação MQTT com AWS IoT.	Os valores válidos são `8883` ou `443`. O valor padrão é `8883`. Para obter mais informações, consulte Conectar-se à porta 443 ou por meio de um proxy de rede.

Campo	Description	Observações
`iotHttpPort`	Opcional. O número da porta usada para criar as conexões HTTPS para AWS IoT.	Os valores válidos são `8443` ou `443`. O valor padrão é `8443`. Para obter mais informações, consulte Conectar-se à porta 443 ou por meio de um proxy de rede.
`ggMqttPort`	Opcional. O número da porta a ser usada para comunicação MQTT pela rede local.	Os valores válidos vão de `1024` a `65535`. O valor padrão é `8883`. Para obter mais informações, consulte the section called “Porta MQTT para mensagens locais”.
`ggHttpPort`	Opcional. O número da porta usada para criar as conexões HTTPS para o serviço AWS IoT Greengrass .	Os valores válidos são `8443` ou `443`. O valor padrão é `8443`. Para obter mais informações, consulte Conectar-se à porta 443 ou por meio de um proxy de rede.
`keepAlive`	Opcional. O período `KeepAlive` do MQTT, em segundos.	O intervalo válido é entre 30 e 1200. O valor padrão é `600`.
`networkProxy`	Opcional. Um objeto que define um servidor de proxy para se conectar.	O servidor proxy pode ser HTTP ou HTTPS. Para obter mais informações, consulte Conectar-se à porta 443 ou por meio de um proxy de rede.

Campo	Description	Observações
`mqttOperationTimeout`	Opcional. A quantidade de tempo (em segundos) para permitir que o núcleo do Greengrass conclua uma operação de publicação, assinatura ou cancelamento de assinatura em conexões MQTT com o AWS IoT Core.	Essa propriedade está disponível a partir da AWS IoT Greengrass v1.10.2. O valor padrão é 5. O valor mínimo é 5.

runtime

Campo	Description	Observações
`maxWorkItemCount`	Opcional. O número máximo de itens de trabalho que o daemon do Greengrass pode processar por vez. As solicitações que excederem esse limite serão ignoradas. A fila de itens de trabalho é compartilhada por componentes do sistema, funções do Lambda definidas pelo usuário, e conectores.	O valor padrão é 1024. O valor máximo é limitado pelo hardware do dispositivo. Aumentar esse valor aumenta a memória que AWS IoT Greengrass usa. Você pode aumentar esse valor caso espere que seu núcleo receba tráfego pesado de mensagens MQTT.
`maxConcurrentLimit`	Opcional. O número máximo de trabalhadores do Lambda não fixados simultâneos que o daemon do Greengrass pode ter. Você pode especificar um número inteiro	O valor padrão é 25. O valor mínimo é definido por `lruSize`.

Campo	Description	Observações
	diferente para substituir esse parâmetro.	
`lruSize`	Opcional. Define o valor mínimo para `maxConcurrentLimit` .	O valor padrão é 25.
`postStartHealthCheckTimeout`	Opcional. O tempo (em milissegundos) que o daemon Greengrass aguarda a conclusão da verificação de integridade depois de iniciar.	O tempo limite padrão é de 30 segundos (30000 ms).
`cgroup`		
`useSystemd`	Indica se o dispositivo usa `systemd`.	Os valores válidos são `yes` ou no. Execute o script `check_ggc_dependencies` no Módulo 1 para ver se o dispositivo usa `systemd`.

crypto

A `crypto` apresenta propriedades que oferecem suporte ao armazenamento da chave privada em um módulo de segurança de hardware (HSM) por meio de PKCS#11 e armazenamento local secreto. Para obter mais informações, consulte the section called “Entidades principais de segurança”, the section called “Integração de segurança de hardware” e Implantar segredos no núcleo. Configurações para o armazenamento da chave privada em HSMs ou no sistema de arquivo são compatíveis.

Campo	Description	Observações
`caPath`	O caminho absoluto para a CA AWS IoT raiz.	Deve ser um URI de arquivo no formato:

Campo	Description	Observações
		`file:///absolute/path/to/file` . **Note** Certifique-se de que os endpoints correspondem ao seu tipo de certificado.
`PKCS11`		
`OpenSSLEngine`	Opcional. O caminho absoluto para o arquivo `.so` do mecanismo OpenSSL para habilitar o suporte PKCS#11 no OpenSSL.	Deve ser um caminho para o arquivo no sistema de arquivos. Esta propriedade será necessária se você estiver usando o atendente de atualizações OTA do Greengrass com segurança de hardware. Para obter mais informações, consulte the section called “Configurar as atualizações OTA”.
`P11Provider`	O caminho absoluto para a biblioteca carregável libdl da implementação PKCS#11.	Deve ser um caminho para o arquivo no sistema de arquivos.
`slotLabel`	O rótulo de slot usado para identificar o módulo de hardware.	Deve estar em conformidade com as especificações de rótulo PKCS#11.

Campo	Description	Observações
`slotUserPin`	O PIN do usuário usado para autenticar o núcleo do Greengrass no módulo.	É necessário ter permissões suficientes para executar C_Sign com as chaves privadas configuradas.
`principals`		
`IoTCertificate`	O certificado e a chave privada que o núcleo usa para fazer solicitações à AWS IoT.	
`IoTCertificate .privateKeyPath`	O caminho para a chave privada do núcleo.	Para o armazenamento do sistema de arquivos, deve ser um URI de arquivo no formato: `file:///`*`absolute/path/to/file`* . Para o armazenamento HSM, deve ser um caminho RFC 7512 PKCS#11 que especifica o rótulo do objeto.
`IoTCertificate .certificatePath`	O caminho absoluto para o certificado do dispositivo do núcleo.	Deve ser um URI de arquivo no formato: `file:///`*`absolute/path/to/file`* .
`MQTTServerCertificate`	Opcional. A chave privada que o núcleo usa em combinação com o certificado para atuar como um gateway ou servidor MQTT.	

Campo	Description	Observações
`MQTTServerCertificate .privateKeyPath`	O caminho para a chave privada do servidor MQTT local.	Use esse valor para especificar sua própria chave privada para o servidor MQTT local. Para o armazenamento do sistema de arquivos, deve ser um URI de arquivo no formato: `file:///`*`absolute/path/to/file`* . Para o armazenamento HSM, deve ser um caminho RFC 7512 PKCS#11 que especifica o rótulo do objeto. Se essa propriedade for omitida, AWS IoT Greengrass gira a chave com base nas configurações de rotação. Se especificado, o cliente será responsável por girar a chave.
`SecretsManager`	A chave privada que protege a chave de dados usada para criptografia. Para obter mais informações, consulte Implantar segredos no núcleo.	

Campo	Description	Observações
`SecretsManager .privateKeyPath`	O caminho para a chave privada do secrets manager local.	Somente uma chave RSA é compatível. Para o armazenamento do sistema de arquivos, deve ser um URI de arquivo no formato: `file:///`*`absolute/path/to/file`* . Para o armazenamento HSM, deve ser um caminho RFC 7512 PKCS#11 que especifica o rótulo do objeto. A chave privada deve ser gerada usando o mecanismo de preenchim ento PKCS#1 v1.5.

As seguintes propriedades de configuração também são compatíveis:

Campo	Description	Observações
`mqttMaxConnectionR etryInterval`	Opcional. O intervalo máximo (em segundos) entre novas tentativas de conexão MQTT caso a conexão caia.	Especifique esse valor como um inteiro não assinado. O padrão é `60`.
`managedRespawn`	Opcional. Indica que o atendente OTA precisa executar o código personali zado antes de uma atualização.	Os valores válidos são `true` ou `false`. Para obter mais informações, consulte Atualizações OTA de AWS IoT Greengrass Software principal.

Campo	Description	Observações
`writeDirectory`	Opcional. O diretório de gravação em que AWS IoT Greengrass cria todos os read/write recursos.	Para obter mais informaçõ es, consulte Configurar um diretório de gravação para AWS IoT Greengrass.

GGC v1.9

```
{
  "coreThing" : {
    "caPath" : "root.ca.pem",
    "certPath" : "hash.cert.pem",
    "keyPath" : "hash.private.key",
    "thingArn" : "arn:partition:iot:region:account-id:thing/core-thing-name",
    "iotHost" : "host-prefix-ats.iot.region.amazonaws.com",
    "ggHost" : "greengrass-ats.iot.region.amazonaws.com",
    "keepAlive" : 600
  },
  "runtime" : {
    "cgroup" : {
      "useSystemd" : "yes"
    }
  },
  "managedRespawn" : false,
  "crypto" : {
    "principals" : {
      "SecretsManager" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key"
      },
      "IoTCertificate" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key",
        "certificatePath" : "file:///greengrass/certs/hash.cert.pem"
      }
    },
    "caPath" : "file:///greengrass/certs/root.ca.pem"
  }
}
```

O arquivo `config.json` é compatível com as seguintes propriedades:

coreThing

Campo	Description	Observações
`caPath`	O caminho para a CA AWS IoT raiz em relação ao `/greengrass-root /certs` diretório.	Para compatibilidade retroativa com versões anteriores à 1.7.0. Essa propriedade é ignorada quando o objeto `crypto` está presente. **Note** Certifique-se de que os endpoints correspondem ao seu tipo de certificado.
`certPath`	O caminho para o certificado de dispositivo de núcleo relativo ao diretório `/greengrass-root /certs`.	Para compatibilidade retroativa com versões anteriores à 1.7.0. Essa propriedade é ignorada quando o objeto `crypto` está presente.
`keyPath`	O caminho para a chave privada do núcleo relativo ao diretório `/greengrass-root /certs`.	Para compatibilidade retroativa com versões anteriores à 1.7.0. Essa propriedade é ignorada quando o objeto `crypto` está presente.
`thingArn`	O nome de recurso da Amazon (ARN) da AWS IoT coisa que represent	Encontre o ARN do seu núcleo no AWS IoT Greengrass console em Núcleos ou executand

Campo	Description	Observações
	a o dispositivo AWS IoT Greengrass principal.	o o comando `aws greengrass get-core-definition-version` CLI.
`iotHost`	Seu AWS IoT endpoint.	Encontre o endpoint no AWS IoT console em Configurações ou executando o comando `aws iot describe-endpoint --endpoint-type iot:Data-ATS` CLI. Este comando retorna o endpoint do Amazon Trust Services (ATS). Para obter mais informações, consulte a documentação de Autenticação do servidor. **Note** Certifique-se de que os endpoints correspondem ao seu tipo de certificado. Verifique se os endpoints correspondem à sua Região da AWS.

Campo	Description	Observações
`ggHost`	Seu AWS IoT Greengrass endpoint.	Este é o seu endpoint `iotHost` com o prefixo do host substituído pelo greengrass (por exemplo, `greengrass-ats.iot.` *`region`*`.amazonaws.com`). Use o mesmo Região da AWS que`iotHost`. **Note** Certifique-se de que os endpoints correspondem ao seu tipo de certificado. Verifique se os endpoints correspondem à sua Região da AWS.
`iotMqttPort`	Opcional. O número da porta a ser usada para comunicação MQTT com AWS IoT.	Os valores válidos são `8883` ou `443`. O valor padrão é `8883`. Para obter mais informações, consulte Conectar-se à porta 443 ou por meio de um proxy de rede.

Campo	Description	Observações
`iotHttpPort`	Opcional. O número da porta usada para criar as conexões HTTPS para AWS IoT.	Os valores válidos são `8443` ou `443`. O valor padrão é `8443`. Para obter mais informações, consulte Conectar-se à porta 443 ou por meio de um proxy de rede.
`ggHttpPort`	Opcional. O número da porta usada para criar as conexões HTTPS para o serviço AWS IoT Greengrass .	Os valores válidos são `8443` ou `443`. O valor padrão é `8443`. Para obter mais informações, consulte Conectar-se à porta 443 ou por meio de um proxy de rede.
`keepAlive`	Opcional. O período `KeepAlive` do MQTT, em segundos.	O intervalo válido é entre 30 e 1200. O valor padrão é `600`.
`networkProxy`	Opcional. Um objeto que define um servidor de proxy para se conectar.	O servidor proxy pode ser HTTP ou HTTPS. Para obter mais informações, consulte Conectar-se à porta 443 ou por meio de um proxy de rede.

runtime

Campo	Description	Observações
`maxConcurrentLimit`	Opcional. O número máximo de trabalhadores do Lambda não fixados simultâneos que o daemon	O valor padrão é 25. O valor mínimo é definido por `lruSize`.

Campo	Description	Observações
	do Greengrass pode ter. Você pode especificar um número inteiro diferente para substituir esse parâmetro.	
`lruSize`	Opcional. Define o valor mínimo para `maxConcurrentLimit` .	O valor padrão é 25.
`postStartHealthCheckTimeout`	Opcional. O tempo (em milissegundos) que o daemon Greengrass aguarda a conclusão da verificação de integridade depois de iniciar.	O tempo limite padrão é de 30 segundos (30000 ms).
`cgroup`		
`useSystemd`	Indica se o dispositivo usa `systemd`.	Os valores válidos são `yes` ou `no`. Execute o script `check_ggc_dependencies` no Módulo 1 para ver se o dispositivo usa `systemd`.

crypto

O objeto `crypto` é adicionado em v1.7.0. Ele apresenta propriedades que oferecem suporte ao armazenamento da chave privada em um módulo de segurança de hardware (HSM) por meio de PKCS#11 e armazenamento local secreto. Para obter mais informações, consulte the section called “Entidades principais de segurança”, the section called “Integração de segurança de hardware” e Implantar segredos no núcleo. Configurações para o armazenamento da chave privada em HSMs ou no sistema de arquivo são compatíveis.

Campo	Description	Observações
`caPath`	O caminho absoluto para a CA AWS IoT raiz.	Deve ser um URI de arquivo no formato: `file:///`*`absolute/path/to/file`* . **Note** Certifique-se de que os endpoints correspondem ao seu tipo de certificado.
`PKCS11`		
`OpenSSLEngine`	Opcional. O caminho absoluto para o arquivo `.so` do mecanismo OpenSSL para habilitar o suporte PKCS#11 no OpenSSL.	Deve ser um caminho para o arquivo no sistema de arquivos. Esta propriedade será necessária se você estiver usando o atendente de atualizações OTA do Greengrass com segurança de hardware. Para obter mais informações, consulte the section called “Configurar as atualizações OTA”.
`P11Provider`	O caminho absoluto para a biblioteca carregável libdl da implementação PKCS#11.	Deve ser um caminho para o arquivo no sistema de arquivos.

Campo	Description	Observações
`slotLabel`	O rótulo de slot usado para identificar o módulo de hardware.	Deve estar em conformidade com as especificações de rótulo PKCS#11.
`slotUserPin`	O PIN do usuário usado para autenticar o núcleo do Greengrass no módulo.	É necessário ter permissões suficientes para executar C_Sign com as chaves privadas configuradas.
`principals`		
`IoTCertificate`	O certificado e a chave privada que o núcleo usa para fazer solicitações à AWS IoT.	
`IoTCertificate.privateKeyPath`	O caminho para a chave privada do núcleo.	Para o armazenamento do sistema de arquivos, deve ser um URI de arquivo no formato: `file:///`*`absolute/path/to/file`* . Para o armazenamento HSM, deve ser um caminho RFC 7512 PKCS#11 que especifica o rótulo do objeto.
`IoTCertificate.certificatePath`	O caminho absoluto para o certificado do dispositivo do núcleo.	Deve ser um URI de arquivo no formato: `file:///`*`absolute/path/to/file`* .
`MQTTServerCertificate`	Opcional. A chave privada que o núcleo usa em combinação com o certificado para atuar como um gateway ou servidor MQTT.	

Campo	Description	Observações
`MQTTServerCertificate .privateKeyPath`	O caminho para a chave privada do servidor MQTT local.	Use esse valor para especificar sua própria chave privada para o servidor MQTT local. Para o armazenamento do sistema de arquivos, deve ser um URI de arquivo no formato: `file:///`*`absolute/path/to/file`* . Para o armazenamento HSM, deve ser um caminho RFC 7512 PKCS#11 que especifica o rótulo do objeto. Se essa propriedade for omitida, AWS IoT Greengrass gira a chave com base nas configurações de rotação. Se especificado, o cliente será responsável por girar a chave.
`SecretsManager`	A chave privada que protege a chave de dados usada para criptografia. Para obter mais informações, consulte Implantar segredos no núcleo.	

Campo	Description	Observações
`SecretsManager .privateKeyPath`	O caminho para a chave privada do secrets manager local.	Somente uma chave RSA é compatível. Para o armazenamento do sistema de arquivos, deve ser um URI de arquivo no formato: `file:///`*`absolute/path/to/file`* . Para o armazenamento HSM, deve ser um caminho RFC 7512 PKCS#11 que especifica o rótulo do objeto. A chave privada deve ser gerada usando o mecanismo de preenchim ento PKCS#1 v1.5.

As seguintes propriedades de configuração também são compatíveis:

Campo	Description	Observações
`mqttMaxConnectionR etryInterval`	Opcional. O intervalo máximo (em segundos) entre novas tentativas de conexão MQTT caso a conexão caia.	Especifique esse valor como um inteiro não assinado. O padrão é `60`.
`managedRespawn`	Opcional. Indica que o atendente OTA precisa executar o código personali zado antes de uma atualização.	Os valores válidos são `true` ou `false`. Para obter mais informações, consulte Atualizações OTA de AWS IoT Greengrass Software principal.

Campo	Description	Observações
`writeDirectory`	Opcional. O diretório de gravação em que AWS IoT Greengrass cria todos os read/write recursos.	Para obter mais informaçõ es, consulte Configurar um diretório de gravação para AWS IoT Greengrass.

GGC v1.8

```
{
  "coreThing" : {
    "caPath" : "root.ca.pem",
    "certPath" : "hash.cert.pem",
    "keyPath" : "hash.private.key",
    "thingArn" : "arn:aws:iot:region:account-id:thing/core-thing-name",
    "iotHost" : "host-prefix-ats.iot.region.amazonaws.com",
    "ggHost" : "greengrass-ats.iot.region.amazonaws.com",
    "keepAlive" : 600
  },
  "runtime" : {
    "cgroup" : {
      "useSystemd" : "yes"
    }
  },
  "managedRespawn" : false,
  "crypto" : {
    "principals" : {
      "SecretsManager" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key"
      },
      "IoTCertificate" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key",
        "certificatePath" : "file:///greengrass/certs/hash.cert.pem"
      }
    },
    "caPath" : "file:///greengrass/certs/root.ca.pem"
  }
}
```

O arquivo `config.json` é compatível com as seguintes propriedades.

coreThing

Campo	Description	Observações
`caPath`	O caminho para a CA AWS IoT raiz em relação ao `/greengrass-root /certs` diretório.	Para compatibilidade retroativa com versões anteriores à 1.7.0. Essa propriedade é ignorada quando o objeto `crypto` está presente. **Note** Certifique-se de que os endpoints correspondem ao seu tipo de certificado.
`certPath`	O caminho para o certificado de dispositivo de núcleo relativo ao diretório `/greengrass-root /certs`.	Para compatibilidade retroativa com versões anteriores à 1.7.0. Essa propriedade é ignorada quando o objeto `crypto` está presente.
`keyPath`	O caminho para a chave privada do núcleo relativo ao diretório `/greengrass-root /certs`.	Para compatibilidade retroativa com versões anteriores à 1.7.0. Essa propriedade é ignorada quando o objeto `crypto` está presente.
`thingArn`	O nome de recurso da Amazon (ARN) da AWS IoT coisa que represent	Encontre o ARN do seu núcleo no AWS IoT Greengrass console em Núcleos ou executand

Campo	Description	Observações
	a o dispositivo AWS IoT Greengrass principal.	o o comando `aws greengrass get-core-definition-version` CLI.
`iotHost`	Seu AWS IoT endpoint.	Encontre o endpoint no AWS IoT console em Configurações ou executando o comando `aws iot describe-endpoint --endpoint-type iot:Data-ATS` CLI. Este comando retorna o endpoint do Amazon Trust Services (ATS). Para obter mais informações, consulte a documentação de Autenticação do servidor. **Note** Certifique-se de que os endpoints correspondem ao seu tipo de certificado. Certifique-se de que seus endpoints correspondam à sua Região da AWS.

Campo	Description	Observações
`ggHost`	Seu AWS IoT Greengrass endpoint.	Este é o seu endpoint `iotHost` com o prefixo do host substituído pelo greengrass (por exemplo, `greengrass-ats.iot.`*`region`*`.amazonaws.com`). Use o mesmo Região da AWS que`iotHost`. **Note** Certifique-se de que os endpoints correspondem ao seu tipo de certificado. Certifique-se de que seus endpoints correspondam à sua Região da AWS.
`iotMqttPort`	Opcional. O número da porta a ser usada para comunicação MQTT com AWS IoT.	Os valores válidos são `8883` ou `443`. O valor padrão é `8883`. Para obter mais informações, consulte Conectar-se à porta 443 ou por meio de um proxy de rede.

Campo	Description	Observações
`iotHttpPort`	Opcional. O número da porta usada para criar as conexões HTTPS para AWS IoT.	Os valores válidos são `8443` ou `443`. O valor padrão é `8443`. Para obter mais informações, consulte Conectar-se à porta 443 ou por meio de um proxy de rede.
`ggHttpPort`	Opcional. O número da porta usada para criar as conexões HTTPS para o serviço AWS IoT Greengrass .	Os valores válidos são `8443` ou `443`. O valor padrão é `8443`. Para obter mais informações, consulte Conectar-se à porta 443 ou por meio de um proxy de rede.
`keepAlive`	Opcional. O período `KeepAlive` do MQTT, em segundos.	O intervalo válido é entre 30 e 1200. O valor padrão é `600`.
`networkProxy`	Opcional. Um objeto que define um servidor de proxy para se conectar.	O servidor proxy pode ser HTTP ou HTTPS. Para obter mais informações, consulte Conectar-se à porta 443 ou por meio de um proxy de rede.

runtime

Campo	Description	Observações
`cgroup`		
`useSystemd`	Indica se o dispositivo usa `systemd`.	Os valores válidos são `yes` ou no. Execute o script

Campo	Description	Observações
		`check_ggc_dependencies` no Módulo 1 para ver se o dispositivo usa `systemd`.

crypto

O objeto `crypto` é adicionado em v1.7.0. Ele apresenta propriedades que oferecem suporte ao armazenamento da chave privada em um módulo de segurança de hardware (HSM) por meio de PKCS#11 e armazenamento local secreto. Para obter mais informações, consulte the section called “Entidades principais de segurança”, the section called “Integração de segurança de hardware” e Implantar segredos no núcleo. Configurações para o armazenamento da chave privada em HSMs ou no sistema de arquivo são compatíveis.

Campo	Description	Observações
`caPath`	O caminho absoluto para a CA AWS IoT raiz.	Deve ser um URI de arquivo no formato: `file:///absolute/path/to/file` . **Note** Certifique-se de que os endpoints correspondem ao seu tipo de certificado.
`PKCS11`		
`OpenSSLEngine`	Opcional. O caminho absoluto para o arquivo `.so` do mecanismo OpenSSL	Deve ser um caminho para o arquivo no sistema de arquivos.

Campo	Description	Observações
	para habilitar o suporte PKCS#11 no OpenSSL.	Esta propriedade será necessária se você estiver usando o atendente de atualizações OTA do Greengrass com segurança de hardware. Para obter mais informações, consulte the section called “Configurar as atualizações OTA”.
`P11Provider`	O caminho absoluto para a biblioteca carregável libdl da implementação PKCS#11.	Deve ser um caminho para o arquivo no sistema de arquivos.
`slotLabel`	O rótulo de slot usado para identificar o módulo de hardware.	Deve estar em conformidade com as especificações de rótulo PKCS#11.
`slotUserPin`	O PIN do usuário usado para autenticar o núcleo do Greengrass no módulo.	É necessário ter permissões suficientes para executar C_Sign com as chaves privadas configuradas.
`principals`		
`IoTCertificate`	O certificado e a chave privada que o núcleo usa para fazer solicitações à AWS IoT.	

Campo	Description	Observações
`IoTCertificate .privateKeyPath`	O caminho para a chave privada do núcleo.	Para o armazenamento do sistema de arquivos, deve ser um URI de arquivo no formato: `file:///`*`absolute/path/to/file`* . Para o armazenamento HSM, deve ser um caminho RFC 7512 PKCS#11 que especifica o rótulo do objeto.
`IoTCertificate .certificatePath`	O caminho absoluto para o certificado do dispositivo do núcleo.	Deve ser um URI de arquivo no formato: `file:///`*`absolute/path/to/file`* .
`MQTTServerCertificate`	Opcional. A chave privada que o núcleo usa em combinação com o certificado para atuar como um gateway ou servidor MQTT.	

Campo	Description	Observações
`MQTTServerCertificate.privateKeyPath`	O caminho para a chave privada do servidor MQTT local.	Use esse valor para especificar sua própria chave privada para o servidor MQTT local. Para o armazenamento do sistema de arquivos, deve ser um URI de arquivo no formato: `file:///`*`absolute/path/to/file`* . Para o armazenamento HSM, deve ser um caminho RFC 7512 PKCS#11 que especifica o rótulo do objeto. Se essa propriedade for omitida, AWS IoT Greengrass gira a chave com base nas configurações de rotação. Se especificado, o cliente será responsável por girar a chave.
`SecretsManager`	A chave privada que protege a chave de dados usada para criptografia. Para obter mais informações, consulte Implantar segredos no núcleo.	

Campo	Description	Observações
`SecretsManager .privateKeyPath`	O caminho para a chave privada do secrets manager local.	Somente uma chave RSA é compatível. Para o armazenamento do sistema de arquivos, deve ser um URI de arquivo no formato: `file:///`*`absolute/path/to/file`* . Para o armazenamento HSM, deve ser um caminho RFC 7512 PKCS#11 que especifica o rótulo do objeto. A chave privada deve ser gerada usando o mecanismo de preenchim ento PKCS#1 v1.5.

As seguintes propriedades de configuração também são compatíveis:

Campo	Description	Observações
`mqttMaxConnectionR etryInterval`	Opcional. O intervalo máximo (em segundos) entre novas tentativas de conexão MQTT caso a conexão caia.	Especifique esse valor como um inteiro não assinado. O padrão é `60`.
`managedRespawn`	Opcional. Indica que o atendente OTA precisa executar o código personali zado antes de uma atualização.	Os valores válidos são `true` ou `false`. Para obter mais informações, consulte Atualizações OTA de AWS IoT Greengrass Software principal.

Campo	Description	Observações
`writeDirectory`	Opcional. O diretório de gravação em que AWS IoT Greengrass cria todos os read/write recursos.	Para obter mais informaçõ es, consulte Configurar um diretório de gravação para AWS IoT Greengrass.

GGC v1.7

```
{
  "coreThing" : {
    "caPath" : "root.ca.pem",
    "certPath" : "hash.cert.pem",
    "keyPath" : "hash.private.key",
    "thingArn" : "arn:aws:iot:region:account-id:thing/core-thing-name",
    "iotHost" : "host-prefix-ats.iot.region.amazonaws.com",
    "ggHost" : "greengrass-ats.iot.region.amazonaws.com",
    "keepAlive" : 600
  },
  "runtime" : {
    "cgroup" : {
      "useSystemd" : "yes"
    }
  },
  "managedRespawn" : false,
  "crypto" : {
    "principals" : {
      "SecretsManager" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key"
      },
      "IoTCertificate" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key",
        "certificatePath" : "file:///greengrass/certs/hash.cert.pem"
      }
    },
    "caPath" : "file:///greengrass/certs/root.ca.pem"
  }
}
```

O arquivo `config.json` é compatível com as seguintes propriedades:

coreThing

Campo	Description	Observações
`caPath`	O caminho para a CA AWS IoT raiz em relação ao `/greengrass-root /certs` diretório.	Para compatibilidade retroativa com versões anteriores à 1.7.0. Essa propriedade é ignorada quando o objeto `crypto` está presente. **Note** Certifique-se de que os endpoints correspondem ao seu tipo de certificado.
`certPath`	O caminho para o certificado de dispositivo de núcleo relativo ao diretório `/greengrass-root /certs`.	Para compatibilidade retroativa com versões anteriores à 1.7.0. Essa propriedade é ignorada quando o objeto `crypto` está presente.
`keyPath`	O caminho para a chave privada do núcleo relativo ao diretório `/greengrass-root /certs`.	Para compatibilidade retroativa com versões anteriores à 1.7.0. Essa propriedade é ignorada quando o objeto `crypto` está presente.
`thingArn`	O nome de recurso da Amazon (ARN) da AWS IoT coisa que represent	Encontre o ARN do seu núcleo no AWS IoT Greengrass console em Núcleos ou executand

Campo	Description	Observações
	a o dispositivo AWS IoT Greengrass principal.	o o comando `aws greengrass get-core-definition-version` CLI.
`iotHost`	Seu AWS IoT endpoint.	Encontre o endpoint no AWS IoT console em Configurações ou executando o comando `aws iot describe-endpoint --endpoint-type iot:Data-ATS` CLI. Este comando retorna o endpoint do Amazon Trust Services (ATS). Para obter mais informações, consulte a documentação de Autenticação do servidor. **Note** Certifique-se de que os endpoints correspondem ao seu tipo de certificado. Certifique-se de que seus endpoints correspondam à sua Região da AWS.

Campo	Description	Observações
`ggHost`	Seu AWS IoT Greengrass endpoint.	Este é o seu endpoint `iotHost` com o prefixo do host substituído pelo greengrass (por exemplo, `greengrass-ats.iot.` *`region`*`.amazonaws.com`). Use o mesmo Região da AWS que`iotHost`. **Note** Certifique-se de que os endpoints correspondem ao seu tipo de certificado. Certifique-se de que seus endpoints correspondam à sua Região da AWS.
`iotMqttPort`	Opcional. O número da porta a ser usada para comunicação MQTT com AWS IoT.	Os valores válidos são `8883` ou `443`. O valor padrão é `8883`. Para obter mais informações, consulte Conectar-se à porta 443 ou por meio de um proxy de rede.
`keepAlive`	Opcional. O período `KeepAlive` do MQTT, em segundos.	O intervalo válido é entre 30 e 1200. O valor padrão é `600`.

Campo	Description	Observações
`networkProxy`	Opcional. Um objeto que define um servidor de proxy para se conectar.	O servidor proxy pode ser HTTP ou HTTPS. Para obter mais informações, consulte Conectar-se à porta 443 ou por meio de um proxy de rede.

runtime

Campo	Description	Observações
`cgroup`		
`useSystemd`	Indica se o dispositivo usa `systemd`.	Os valores válidos são `yes` ou no. Execute o script `check_ggc_dependencies` no Módulo 1 para ver se o dispositivo usa `systemd`.

crypto

O objeto `crypto`, adicionado na v1.7.0, apresenta propriedades que oferecem suporte ao armazenamento da chave privada em um módulo de segurança de hardware (HSM) por meio de PKCS#11 e armazenamento local secreto. Para obter mais informações, consulte the section called “Integração de segurança de hardware” e Implantar segredos no núcleo. Configurações para o armazenamento da chave privada em HSMs ou no sistema de arquivo são compatíveis.

Campo	Description	Observações
`caPath`	O caminho absoluto para a CA AWS IoT raiz.	Deve ser um URI de arquivo no formato:

Campo	Description	Observações
		`file:///absolute/path/to/file` . **Note** Certifique-se de que os endpoints correspondem ao seu tipo de certificado.
`PKCS11`		
`OpenSSLEngine`	Opcional. O caminho absoluto para o arquivo `.so` do mecanismo OpenSSL para habilitar o suporte PKCS#11 no OpenSSL.	Deve ser um caminho para o arquivo no sistema de arquivos. Esta propriedade será necessária se você estiver usando o atendente de atualizações OTA do Greengrass com segurança de hardware. Para obter mais informações, consulte the section called “Configurar as atualizações OTA”.
`P11Provider`	O caminho absoluto para a biblioteca carregável libdl da implementação PKCS#11.	Deve ser um caminho para o arquivo no sistema de arquivos.
`slotLabel`	O rótulo de slot usado para identificar o módulo de hardware.	Deve estar em conformidade com as especificações de rótulo PKCS#11.

Campo	Description	Observações
`slotUserPin`	O PIN do usuário usado para autenticar o núcleo do Greengrass no módulo.	É necessário ter permissões suficientes para executar C_Sign com as chaves privadas configuradas.
`principals`		
`IoTCertificate`	O certificado e a chave privada que o núcleo usa para fazer solicitações à AWS IoT.	
`IoTCertificate .privateKeyPath`	O caminho para a chave privada do núcleo.	Para o armazenamento do sistema de arquivos, deve ser um URI de arquivo no formato: `file:///`*`absolute/path/to/file`* . Para o armazenamento HSM, deve ser um caminho RFC 7512 PKCS#11 que especifica o rótulo do objeto.
`IoTCertificate .certificatePath`	O caminho absoluto para o certificado do dispositivo do núcleo.	Deve ser um URI de arquivo no formato: `file:///`*`absolute/path/to/file`* .
`MQTTServerCertificate`	Opcional. A chave privada que o núcleo usa em combinação com o certificado para atuar como um gateway ou servidor MQTT.	

Campo	Description	Observações
`MQTTServerCertificate.privateKeyPath`	O caminho para a chave privada do servidor MQTT local.	Use esse valor para especificar sua própria chave privada para o servidor MQTT local. Para o armazenamento do sistema de arquivos, deve ser um URI de arquivo no formato: `file:///`*`absolute/path/to/file`* . Para o armazenamento HSM, deve ser um caminho RFC 7512 PKCS#11 que especifica o rótulo do objeto. Se essa propriedade for omitida, AWS IoT Greengrass gira a chave com base nas configurações de rotação. Se especificado, o cliente será responsável por girar a chave.
`SecretsManager`	A chave privada que protege a chave de dados usada para criptografia. Para obter mais informações, consulte Implantar segredos no núcleo.	

Campo	Description	Observações
`SecretsManager .privateKeyPath`	O caminho para a chave privada do secrets manager local.	Somente uma chave RSA é compatível. Para o armazenamento do sistema de arquivos, deve ser um URI de arquivo no formato: `file:///`*`absolute/path/to/file`* . Para o armazenamento HSM, deve ser um caminho RFC 7512 PKCS#11 que especifica o rótulo do objeto. A chave privada deve ser gerada usando o mecanismo de preenchim ento PKCS#1 v1.5.

As seguintes propriedades de configuração também são compatíveis:

Campo	Description	Observações
`mqttMaxConnectionR etryInterval`	Opcional. O intervalo máximo (em segundos) entre novas tentativas de conexão MQTT caso a conexão caia.	Especifique esse valor como um inteiro não assinado. O padrão é `60`.
`managedRespawn`	Opcional. Indica que o atendente OTA precisa executar o código personali zado antes de uma atualização.	Os valores válidos são `true` ou `false`. Para obter mais informações, consulte Atualizações OTA de AWS IoT Greengrass Software principal.

Campo	Description	Observações
`writeDirectory`	Opcional. O diretório de gravação em que AWS IoT Greengrass cria todos os read/write recursos.	Para obter mais informaçõ es, consulte Configurar um diretório de gravação para AWS IoT Greengrass.

GGC v1.6

```
{
   "coreThing": {
       "caPath": "root-ca-pem",
       "certPath": "cloud-pem-crt",
       "keyPath": "cloud-pem-key",
       "thingArn": "arn:aws:iot:region:account-id:thing/core-thing-name",
       "iotHost": "host-prefix.iot.region.amazonaws.com",
       "ggHost": "greengrass.iot.region.amazonaws.com",
       "keepAlive": 600,
       "mqttMaxConnectionRetryInterval": 60
   },
   "runtime": {
       "cgroup": {
           "useSystemd": "yes|no"
       }
   },
   "managedRespawn": true,
   "writeDirectory": "/write-directory"
}
```

> **Note**
>
> Se você usar a opção de criação de grupo padrão no AWS IoT Greengrass console, o `config.json` arquivo será implantado no dispositivo principal em um estado de funcionamento que especifica a configuração padrão.

O arquivo `config.json` é compatível com as seguintes propriedades:

Campo	Description	Observações
`caPath`	O caminho para o CA raiz da AWS IoT relativo ao diretório */greengrass-root*`/certs`.	Salve o arquivo em */greengrass-root* `/certs`.
`certPath`	O caminho para o certifica do AWS IoT Greengras s principal em relação ao */greengrass-root* `/certs` diretório.	Salve o arquivo em */greengrass-root* `/certs`.
`keyPath`	O caminho para a chave privada AWS IoT Greengras s principal em relação ao */greengrass-root* `/certs` diretório.	Salve o arquivo em */greengrass-root* `/certs`.
`thingArn`	O nome de recurso da Amazon (ARN) da AWS IoT coisa que represent a o dispositivo AWS IoT Greengrass principal.	Encontre o ARN do seu núcleo no AWS IoT Greengrass console em Núcleos ou executand o o comando `aws greengrass get-core-definition-version` CLI.
`iotHost`	Seu AWS IoT endpoint.	Encontre isso no AWS IoT console em Configurações ou executando o comando `aws iot describe-endpoint` CLI.
`ggHost`	Seu AWS IoT Greengrass endpoint.	Esse valor usa o formato `greengras s.iot.` *region*`.amazonaw`

Campo	Description	Observações
		`s.com` . Use a mesma região de `iotHost`.
`keepAlive`	O período `KeepAlive` do MQTT, em segundos.	Esse é um valor opcional. O padrão é `600`.
`mqttMaxConnectionRetryInterval`	O intervalo máximo (em segundos) entre novas tentativas de conexão MQTT caso a conexão caia.	Especifique esse valor como um inteiro não assinado. Esse é um valor opcional. O padrão é `60`.
`useSystemd`	Indica se o dispositivo usa `systemd`.	Os valores válidos são `yes` ou no. Execute o script `check_ggc_dependencies` no Módulo 1 para ver se o dispositivo usa `systemd`.
`managedRespawn`	Um atributo opcional de atualizações over-the-air (OTA), isso indica que o atendente OTA precisa executar o código personalizado antes de uma atualização.	Os valores válidos são `true` ou `false`. Para obter mais informações, consulte Atualizações OTA de AWS IoT Greengrass Software principal.
`writeDirectory`	O diretório de gravação em que AWS IoT Greengrass cria todos os read/write recursos.	Esse é um valor opcional. Para obter mais informações, consulte Configurar um diretório de gravação para AWS IoT Greengrass.

GGC v1.5

```
{
   "coreThing": {
```

```
        "caPath": "root-ca-pem",
        "certPath": "cloud-pem-crt",
        "keyPath": "cloud-pem-key",
        "thingArn": "arn:aws:iot:region:account-id:thing/core-thing-name",
        "iotHost": "host-prefix.iot.region.amazonaws.com",
        "ggHost": "greengrass.iot.region.amazonaws.com",
        "keepAlive": 600
    },
    "runtime": {
        "cgroup": {
            "useSystemd": "yes|no"
        }
    },
    "managedRespawn": true
}
```

O arquivo `config.json` está presente em */greengrass-root*`/config` e contém os seguintes parâmetros:

Campo	Description	Observações
`caPath`	Caminho para o CA raiz da AWS IoT relativo à pasta */greengrass-root* `/ certs`.	Salve o arquivo na pasta */greengrass-root* `/ certs`.
`certPath`	O caminho para o certifica do AWS IoT Greengras s principal em relação à */greengrass-root* `/ certs` pasta.	Salve o arquivo na pasta */greengrass-root* `/ certs`.
`keyPath`	O caminho para a chave privada AWS IoT Greengras s principal em relação à */greengrass-root* `/ certs` pasta.	Salve o arquivo na pasta */greengrass-root* `/ certs`.
`thingArn`	O nome de recurso da Amazon (ARN) da AWS	Encontre o ARN do seu núcleo no AWS IoT

Campo	Description	Observações
	IoT coisa que represent a o dispositivo AWS IoT Greengrass principal.	Greengrass console em Núcleos ou executand o o comando `aws greengrass get-core-definition-version` CLI.
`iotHost`	Seu AWS IoT endpoint.	Encontre isso no AWS IoT console, em Configurações, ou executando o `aws iot describe-endpoint` comando.
`ggHost`	Seu AWS IoT Greengrass endpoint.	Esse valor usa o formato `greengras s.iot.` *`region`*`.amazonaw s.com` . Use a mesma região de `iotHost`.
`keepAlive`	O período `KeepAlive` do MQTT, em segundos.	Esse é um valor opcional. O valor padrão é 600 segundos.
`useSystemd`	Indica se o dispositivo usa `systemd`.	Os valores válidos são `yes` ou `no`. Execute o script `check_ggc_dependen cies` no Módulo 1 para ver se o dispositivo usa `systemd`.

Campo	Description	Observações
`managedRespawn`	Um atributo opcional de atualizações over-the-air (OTA), isso indica que o atendente OTA precisa executar o código personalizado antes de uma atualização.	Para obter mais informaçõ es, consulte Atualizaç ões OTA de AWS IoT Greengrass Software principal.

GGC v1.3

```
{
   "coreThing": {
       "caPath": "root-ca-pem",
       "certPath": "cloud-pem-crt",
       "keyPath": "cloud-pem-key",
       "thingArn": "arn:aws:iot:region:account-id:thing/core-thing-name",
       "iotHost": "host-prefix.iot.region.amazonaws.com",
       "ggHost": "greengrass.iot.region.amazonaws.com",
       "keepAlive": 600
   },
   "runtime": {
       "cgroup": {
           "useSystemd": "yes|no"
       }
   },
   "managedRespawn": true
}
```

O arquivo `config.json` está presente em */greengrass-root*`/config` e contém os seguintes parâmetros:

Campo	Description	Observações
`caPath`	Caminho para o CA raiz da AWS IoT relativo à pasta	Salve o arquivo na pasta */greengrass-root* `/ certs`.

Campo	Description	Observações
	/greengrass-root / certs.	
certPath	O caminho para o certifica do AWS IoT Greengras s principal em relação à */greengrass-root* / certs pasta.	Salve o arquivo na pasta */greengrass-root* / certs.
keyPath	O caminho para a chave privada AWS IoT Greengras s principal em relação à */greengrass-root* / certs pasta.	Salve o arquivo na pasta */greengrass-root* / certs.
thingArn	O nome de recurso da Amazon (ARN) da AWS IoT coisa que represent a o AWS IoT Greengrass núcleo.	Você pode encontrar esse valor no AWS IoT Greengrass console sob a definição de sua AWS IoT coisa.
iotHost	Seu AWS IoT endpoint.	Você pode encontrar esse valor no AWS IoT console em Configurações.
ggHost	Seu AWS IoT Greengrass endpoint.	Você pode encontrar esse valor no AWS IoT console em Configurações com greengrass. prefixo.
keepAlive	O período KeepAlive do MQTT, em segundos.	Esse é um valor opcional. O valor padrão é 600 segundos.

Campo	Description	Observações
`useSystemd`	Um sinalizador binário, se seu dispositivo usar `systemd`.	Os valores são `yes` ou `no`. Use o script de dependência no Módulo 1 para ver se seu dispositivo usa `systemd`.
`managedRespawn`	Um atributo opcional de atualizações over-the-air (OTA), isso indica que o atendente OTA precisa executar o código personalizado antes de uma atualização.	Para obter mais informaçõ es, consulte Atualizaç ões OTA de AWS IoT Greengrass Software principal.

GGC v1.1

```
{
   "coreThing": {
       "caPath": "root-ca-pem",
       "certPath": "cloud-pem-crt",
       "keyPath": "cloud-pem-key",
       "thingArn": "arn:aws:iot:region:account-id:thing/core-thing-name",
       "iotHost": "host-prefix.iot.region.amazonaws.com",
       "ggHost": "greengrass.iot.region.amazonaws.com",
       "keepAlive": 600
   },
   "runtime": {
       "cgroup": {
           "useSystemd": "yes|no"
       }
   }
}
```

O arquivo `config.json` está presente em */greengrass-root*`/config` e contém os seguintes parâmetros:

Campo	Description	Observações
`caPath`	Caminho para o CA raiz da AWS IoT relativo à pasta */greengrass-root /certs*.	Salve o arquivo na pasta */greengrass-root /certs*.
`certPath`	O caminho para o certifica do AWS IoT Greengras s principal em relação à */greengrass-root /certs* pasta.	Salve o arquivo na pasta */greengrass-root /certs*.
`keyPath`	O caminho para a chave privada AWS IoT Greengras s principal em relação à */greengrass-root /certs* pasta.	Salve o arquivo na pasta */greengrass-root /certs*.
`thingArn`	O nome de recurso da Amazon (ARN) da AWS IoT coisa que represent a o AWS IoT Greengrass núcleo.	Você pode encontrar esse valor no AWS IoT Greengrass console sob a definição de sua AWS IoT coisa.
`iotHost`	Seu AWS IoT endpoint.	Você pode encontrar esse valor no AWS IoT console em Configurações.
`ggHost`	Seu AWS IoT Greengrass endpoint.	Você pode encontrar esse valor no AWS IoT console em Configurações com `greengrass.` prefixo.
`keepAlive`	O período `KeepAlive` do MQTT, em segundos.	Esse é um valor opcional. O valor padrão é 600 segundos.

Campo	Description	Observações
`useSystemd`	Um sinalizador binário, se seu dispositivo usar `systemd`.	Os valores são `yes` ou `no`. Use o script de dependência no Módulo 1 para ver se seu dispositivo usa `systemd`.

GGC v1.0

No AWS IoT Greengrass Core v1.0, `config.json` é implantado em. *`greengrass-root`*`/configuration`

```
{
   "coreThing": {
       "caPath": "root-ca-pem",
       "certPath": "cloud-pem-crt",
       "keyPath": "cloud-pem-key",
       "thingArn": "arn:aws:iot:region:account-id:thing/core-thing-name",
       "iotHost": "host-prefix.iot.region.amazonaws.com",
       "ggHost": "greengrass.iot.region.amazonaws.com",
       "keepAlive": 600
   },
   "runtime": {
       "cgroup": {
           "useSystemd": "yes|no"
       }
   }
}
```

O arquivo `config.json` está presente em */greengrass-root*`/configuration` e contém os seguintes parâmetros:

Campo	Description	Observações
`caPath`	Caminho para o CA raiz da AWS IoT relativo à pasta */greengrass-root* `/configuration/certs` .	Salve o arquivo na pasta */greengrass-root* `/configuration/certs` .

Campo	Description	Observações
`certPath`	O caminho para o certifica do AWS IoT Greengras s principal em relação à */greengrass-root* / `configuration/certs` pasta.	Salve o arquivo na pasta */greengrass-root* / `configuration/cer ts` .
`keyPath`	O caminho para a chave privada AWS IoT Greengras s principal em relação à */greengrass-root* / `configuration/certs` pasta.	Salve o arquivo na pasta */greengrass-root* / `configuration/cer ts` .
`thingArn`	O nome de recurso da Amazon (ARN) da AWS IoT coisa que represent a o AWS IoT Greengrass núcleo.	Você pode encontrar esse valor no AWS IoT Greengrass console abaixo da definição de sua AWS IoT coisa.
`iotHost`	Seu AWS IoT endpoint.	Você pode encontrar esse valor no AWS IoT console em Configurações.
`ggHost`	Seu AWS IoT Greengrass endpoint.	Você pode encontrar esse valor no AWS IoT console em Configurações com `greengrass.` prefixo.
`keepAlive`	O período `KeepAlive` do MQTT, em segundos.	Esse é um valor opcional. O valor padrão é 600 segundos.

Campo	Description	Observações
`useSystemd`	Um sinalizador binário se seu dispositivo usar `systemd`.	Os valores são `yes` ou `no`. Use o script de dependência no Módulo 1 para ver se seu dispositivo usa `systemd`.

Os endpoints do serviço devem corresponder ao tipo de certificado da CA raiz

Seus AWS IoT Greengrass endpoints AWS IoT Core e endpoints devem corresponder ao tipo de certificado do certificado CA raiz em seu dispositivo. Se os endpoints e o tipo de certificado não corresponderem, as tentativas de autenticação falharão entre o dispositivo e AWS IoT Core ou AWS IoT Greengrass. Para obter mais informações, consulte Autenticação do servidor no Guia do desenvolvedor do AWS IoT .

Se seu dispositivo usar um certificado CA raiz do Amazon Trust Services (ATS), que é o método preferencial, ele também deverá usar endpoints do ATS para gerenciamento do dispositivo e operações de plano de dados de descoberta. Os endpoints ATS incluem o `ats` segmento, conforme mostrado na sintaxe a seguir para o AWS IoT Core endpoint.

```
prefix-ats.iot.region.amazonaws.com
```

> **Note**
>
> Para compatibilidade com versões anteriores, AWS IoT Greengrass atualmente oferece suporte a certificados e endpoints de CA VeriSign raiz legados em alguns s. Região da AWS Se você estiver usando um certificado de CA VeriSign raiz legado, recomendamos que você crie um endpoint ATS e, em vez disso, use um certificado de CA raiz ATS. Caso contrário, certifique-se de usar os endpoints legados correspondentes. Para obter mais informações, consulte Endpoints legados compatíveis no Referência geral da Amazon Web Services.

Endpoints no config.json

Em um dispositivo de núcleo do Greengrass, os endpoints são especificados no objeto `coreThing` no arquivo `config.json`. A `iotHost` propriedade representa o AWS IoT Core endpoint. A `ggHost`

propriedade representa o AWS IoT Greengrass endpoint. No exemplo de trecho a seguir, essas propriedades especificam endpoints do ATS.

```
{
  "coreThing" : {
    ...
    "iotHost" : "abcde1234uwxyz-ats.iot.us-west-2.amazonaws.com",
    "ggHost" : "greengrass-ats.iot.us-west-2.amazonaws.com",
    ...
  },
```

AWS IoT Core endpoint

Você pode obter seu AWS IoT Core endpoint executando o comando aws iot describe-endpoint CLI com o parâmetro apropriado`--endpoint-type`.

- Para retornar um endpoint assinado pelo ATS, execute:

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

- Para retornar um endpoint herdado VeriSign assinado, execute:

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

AWS IoT Greengrass endpoint

Seu AWS IoT Greengrass endpoint é seu `iotHost` endpoint com o prefixo host substituído por greengrass. Por exemplo, o endpoint assinado pelo ATS é `greengrass-ats.iot.`*`region`*`.amazonaws.com`. Isso usa a mesma região do seu AWS IoT Core endpoint.

Conectar-se à porta 443 ou por meio de um proxy de rede

Esse recurso está disponível para AWS IoT Greengrass Core v1.7 e versões posteriores.

Os núcleos do Greengrass 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 HTTP antes de se conectar à Internet.

Para habilitar a comunicação nesses cenários, AWS IoT Greengrass permite as seguintes configurações:

- MQTT com autenticação de cliente TLS na porta 443. Se a rede permitir conexões com a porta 443, você poderá configurar o núcleo para usar a porta 443 para tráfego MQTT em vez da porta 8883 padrão. Isso pode ser uma conexão direta com a porta 443 ou uma conexão por meio de um servidor de proxy de rede.

  AWS IoT Greengrass usa a extensão TLS da Application Layer Protocol Network (ALPN) para habilitar essa conexão. Assim como ocorre com a configuração padrão, o MQTT sobre TLS na porta 443 usa autenticação de cliente baseada em certificado.

  Quando configurado para usar uma conexão direta com a porta 443, o núcleo oferece suporte a atualizações OTA (over-the-air) para software do AWS IoT Greengrass . Esse suporte requer o AWS IoT Greengrass Core v1.9.3 ou posterior.
- Comunicação HTTPS pela porta 443. AWS IoT Greengrass envia tráfego HTTPS pela porta 8443 por padrão, mas você pode configurá-lo para usar a porta 443.
- Conexão por meio de um proxy de rede. É possível configurar um servidor de proxy de rede para atuar como intermediário para se conectar ao núcleo do Greengrass. Somente a autenticação básica e os proxies HTTP e HTTPS são compatíveis.

  A configuração do proxy é passada para funções do Lambda definidas pelo usuário por meio das variáveis de `no_proxy` ambiente `http_proxy``https_proxy`, e. User-defined As funções do Lambda devem usar essas configurações passadas para se conectar por meio do proxy. As bibliotecas comuns usadas por funções do Lambda para fazer conexões (como boto3 ou cURL e pacotes `requests` python) normalmente usam essas variáveis de ambiente por padrão. Se uma função do Lambda também especificar essas mesmas variáveis de ambiente, AWS IoT Greengrass não as substituirá.

  > **Important**
  >
  > Os núcleos do Greengrass que são configurados para usar um proxy de rede não oferecem suporte a atualizações OTA (over-the-air).

Para configurar o MQTT na porta 443

Esse recurso requer o AWS IoT Greengrass Core v1.7 ou posterior.

Este procedimento permite que o núcleo do Greengrass use a porta 443 para mensagens MQTT com o AWS IoT Core.

1. Execute o comando a seguir para interromper o daemon do Greengrass:

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

2. Abra *greengrass-root*`/config/config.json` para editar como o usuário su.
3. No objeto `coreThing`, adicione a propriedade `iotMqttPort` e defina o valor **`443`**, conforme exibido no seguinte exemplo.

```
{
    "coreThing" : {
        "caPath" : "root.ca.pem",
        "certPath" : "12345abcde.cert.pem",
        "keyPath" : "12345abcde.private.key",
        "thingArn" : "arn:aws:iot:us-west-2:123456789012:thing/core-thing-name",
        "iotHost" : "abcd123456wxyz-ats.iot.us-west-2.amazonaws.com",
        "iotMqttPort" : 443,
        "ggHost" : "greengrass-ats.iot.us-west-2.amazonaws.com",
        "keepAlive" : 600
    },
    ...
}
```

4. Inicie o daemon.

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

Para configurar o HTTPS na porta 443

Esse recurso requer o AWS IoT Greengrass Core v1.8 ou posterior.

Esse procedimento configura o núcleo para usar a porta 443 para comunicação HTTPS.

1. Execute o comando a seguir para interromper o daemon do Greengrass:

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

2. Abra *greengrass-root*/config/config.json para editar como o usuário su.
3. No objeto coreThing, adicione as propriedades iotHttpPort e ggHttpPort, conforme exibido no exemplo a seguir.

```
{
    "coreThing" : {
        "caPath" : "root.ca.pem",
        "certPath" : "12345abcde.cert.pem",
        "keyPath" : "12345abcde.private.key",
        "thingArn" : "arn:aws:iot:us-west-2:123456789012:thing/core-thing-name",
        "iotHost" : "abcd123456wxyz-ats.iot.us-west-2.amazonaws.com",
        "iotHttpPort" : 443,
        "ggHost" : "greengrass-ats.iot.us-west-2.amazonaws.com",
        "ggHttpPort" : 443,
        "keepAlive" : 600
    },
    ...
}
```

4. Inicie o daemon.

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

Para configurar um proxy de rede

Esse recurso requer o AWS IoT Greengrass Core v1.7 ou posterior.

Este procedimento permite AWS IoT Greengrass conectar-se à Internet por meio de um proxy de rede HTTP ou HTTPS.

1. Execute o comando a seguir para interromper o daemon do Greengrass:

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

2. Abra *greengrass-root*/config/config.json para editar como o usuário su.
3. No objeto coreThing, adicione o objeto networkProxy, conforme mostrado no exemplo a seguir.

```
{
    "coreThing" : {
        "caPath" : "root.ca.pem",
        "certPath" : "12345abcde.cert.pem",
        "keyPath" : "12345abcde.private.key",
        "thingArn" : "arn:aws:iot:us-west-2:123456789012:thing/core-thing-name",
        "iotHost" : "abcd123456wxyz-ats.iot.us-west-2.amazonaws.com",
        "ggHost" : "greengrass-ats.iot.us-west-2.amazonaws.com",
        "keepAlive" : 600,
        "networkProxy": {
            "noProxyAddresses" : "http://128.12.34.56,www.mywebsite.com",
            "proxy" : {
                "url" : "https://my-proxy-server:1100",
                "username" : "Mary_Major",
                "password" : "pass@word1357"
            }
        }
    },
    ...
}
```

4. Inicie o daemon.

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

Objeto networkProxy

Use o objeto networkProxy para especificar informações sobre o proxy de rede. Esse objeto tem as seguintes propriedades.

Campo	Description
noProxyAddresses	Opcional. Uma lista separada por vírgulas de endereços IP ou nomes de host isentos do proxy.

Campo	Description

`proxy`	O proxy para se conectar. Um proxy tem as seguintes propriedades.
	• `url`. O URL do servidor de proxy, no formato `scheme://userinfo@host:port` .
	  • `scheme`. O esquema. Precisa ser `http` ou `https`.
	  • `userinfo`. Opcional. As informações de nome de usuário e senha. Se especific ados, os campos `password` e `username` são ignorados.
	  • `host`. O nome do host ou endereço IP do servidor de proxy.
	  • `port`. Opcional. O número da porta. Se não for especificado, os seguintes valores padrão serão usados:
	  • `http`: 80
	  • `https`: 443
	• `username`. Opcional. O nome do usuário a usar para a autenticação no servidor de proxy.
	• `password`. Opcional. A senha a usar para a autenticação no servidor de proxy.

Permitir endpoints

A comunicação entre dispositivos Greengrass AWS IoT Core e/ou AWS IoT Greengrass deve ser autenticada. Essa autenticação é baseada em certificados de X.509 dispositivos registrados e chaves criptográficas. Para permitir que as solicitações autenticadas passem pelos proxies sem criptografia adicional, aprove os endpoints a seguir.

Endpoint	Porta	Description
`greengrass. region.amazonaws.com`	443	Usado para operações de ambiente de gerenciamento para gerenciamento de grupos.

`prefix-ats.iot. region.amazonaws.com`
 	
ou
 	
`prefix.iot.region.amazonaws.com`
	MQTT: 8883 ou 443
	
	HTTPS: 8443 ou 443
		Usado para operações de plano de dados para o gerenciamento de dispositivos, como sincronização de sombra.
		
		Permite o uso de um ou ambos endpoints, dependendo se os dispositivos cliente e o núcleo usam os certificados CA raiz Amazon Trust Services (preferid

Endpoint	Porta	Description
		o), certificados CA raiz legados, ou ambos. Para obter mais informações, consulte the section called “Os endpoints do serviço devem corresponder ao tipo de certificado”.

Endpoint	Porta	Description

`greengrass-ats.iot. region.amazonaws.com`
 	
ou
 	
`greengrass.iot. region.amazonaws.com`
	8443 ou 443	Usado para operações de descoberta de dispositivo.
		
		Permite o uso de um ou ambos endpoints, dependendo se os dispositivos cliente e o núcleo usam os certificados CA raiz Amazon Trust Services (preferido), certificados CA raiz legados, ou ambos. Para obter mais informações, consulte the section called “Os endpoints do serviço devem correspon

Endpoint	Porta	Description
		der ao tipo de certificado”. **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

Endpoint	Porta	Description
		`ProtocolName` no `ProtocolNameList`. Para obter mais informações, consulte Protocolos no Guia do desenvolvedor do AWS IoT.

Endpoint	Porta	Description
`*.s3.amazonaws.com`	443	Usado para operações de implantação e atualizações over-the-air (OTA). Este formato inclui o caractere *, porque os prefixos de endpoint são controlados internamente e podem mudar a qualquer momento.
`logs.region.amazonaws.com`	443	Obrigatório se o grupo do Greengrass estiver configurado para gravar logs no CloudWatch.

Configurar um diretório de gravação para AWS IoT Greengrass

Esse recurso está disponível para AWS IoT Greengrass Core v1.6 e versões posteriores.

Por padrão, o software AWS IoT Greengrass Core é implantado em um único diretório raiz, onde AWS IoT Greengrass executa todas as operações de leitura e gravação. No entanto, você pode configurar o AWS IoT Greengrass uso de um diretório separado para todas as operações de

gravação, incluindo a criação de diretórios e arquivos. Nesse caso, AWS IoT Greengrass usa dois diretórios de nível superior:

- O diretório *greengrass-root*, que você pode deixar como leitura/gravação ou, como opção, somente leitura. Ele contém o software AWS IoT Greengrass Core e outros componentes essenciais que devem permanecer imutáveis durante o tempo de execução, como certificados e. `config.json`
- O diretório de gravação especificado. Isso contém conteúdo gravável, como logs, informações de estado e funções do Lambda definidas pelo usuário implantadas.

Essa configuração resulta na estrutura de diretórios a seguir.

Diretório raiz do Greengrass

```
greengrass-root/
|-- certs/
|   |-- root.ca.pem
|   |-- hash.cert.pem
|   |-- hash.private.key
|   |-- hash.public.key
|-- config/
|   |-- config.json
|-- ggc/
|   |-- packages/
|       |-- package-version/
|           |-- bin/
|               |-- daemon
|           |-- greengrassd
|           |-- lambda/
|           |-- LICENSE/
|           |-- release_notes_package-version.html
|               |-- runtime/
|                   |-- java8/
|                   |-- nodejs8.10/
|                   |-- python3.8/
|   |-- core/
```

Diretório de gravação

```
write-directory/
|-- packages/
```

```
|   |-- package-version/
|       |-- ggc_root/
|       |-- rootfs_nosys/
|       |-- rootfs_sys/
|       |-- var/
|-- deployment/
|   |-- group/
|       |-- group.json
|   |-- lambda/
|   |-- mlmodel/
|-- var/
|   |-- log/
|   |-- state/
```

Para configurar um diretório de gravação

1. Execute o seguinte comando para parar o AWS IoT Greengrass daemon:

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

2. Abra *greengrass-root*`/config/config.json` para editar como o usuário su.

3. Adicione `writeDirectory` como parâmetro e especifique o caminho para o diretório de destino, conforme mostrado no exemplo a seguir.

   ```
   {
       "coreThing": {
           "caPath": "root-CA.pem",
           "certPath": "hash.pem.crt",
           ...
       },
       ...
       "writeDirectory" : "/write-directory"
   }
   ```

> **Note**
>
> Você poderá atualizar a configuração `writeDirectory` sempre que quiser. Depois que a configuração for atualizada, AWS IoT Greengrass usa o diretório de gravação recém-especificado na próxima inicialização, mas não migra o conteúdo do diretório de gravação anterior.

4. Agora que o diretório de gravação está configurado, você pode tornar o diretório *greengrass-root* somente leitura. Para obter instruções, consulte Para criar o diretório Read-Only raiz do Greengrass.

   Caso contrário, inicie o AWS IoT Greengrass daemon:

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

Para tornar o diretório raiz do Greengrass somente leitura

Siga essas etapas somente se você quiser tornar o diretório raiz do Greengrass somente leitura. O diretório de gravação deve estar configurado antes de você começar.

1. Conceder permissões de acesso a diretórios necessários:

   a. Dê permissões de leitura e gravação ao proprietário `config.json`.

      ```
      sudo chmod 0600 /greengrass-root/config/config.json
      ```

   b. Torne ggc_user o proprietário das certificações e dos diretórios Lambda do sistema.

      ```
      sudo chown -R ggc_user:ggc_group /greengrass-root/certs/
      sudo chown -R ggc_user:ggc_group /greengrass-root/ggc/packages/1.11.6/lambda/
      ```

      > **Note**
      >
      > As contas ggc_user e ggc_group são usadas por padrão para executar funções do Lambda do sistema. Se você configurou a identidade de acesso padrão no nível do

grupo para usar contas diferentes, em vez disso, deve conceder permissões a esse usuário (UID) e grupo (GID).

2. Torne o diretório *greengrass-root* somente leitura usando seu mecanismo preferido.

> **Note**
>
> Uma maneira de tornar o diretório *greengrass-root* somente leitura é montar o diretório como somente leitura. No entanto, para aplicar atualizações Over-The-Air (OTA – Pelo ar) ao software do núcleo do AWS IoT Greengrass em um diretório montado, o diretório deve ser inicialmente desmontado e, em seguida, remontado após a atualização. Você pode adicionar essas operações `umount` e `mount` aos scripts `ota_pre_update` e `ota_post_update`. Para obter mais informações sobre atualizações OTA, consulte the section called “Agente de atualizações OTA do Greengrass” e the section called “Respawn gerenciado com atualizações OTA”.

3. Inicie o daemon.

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

Se as permissões da etapa 1 não forem definidas corretamente, o daemon não será iniciado.

Definir configurações MQTT

No AWS IoT Greengrass ambiente, dispositivos clientes locais, funções do Lambda, conectores e componentes do sistema podem se comunicar entre si e com AWS IoT Core. Toda a comunicação passa pelo núcleo, que gerencia as assinaturas que autorizam a comunicação MQTT entre entidades.

Para obter informações sobre as configurações de MQTT que você pode definir AWS IoT Greengrass, consulte as seções a seguir:

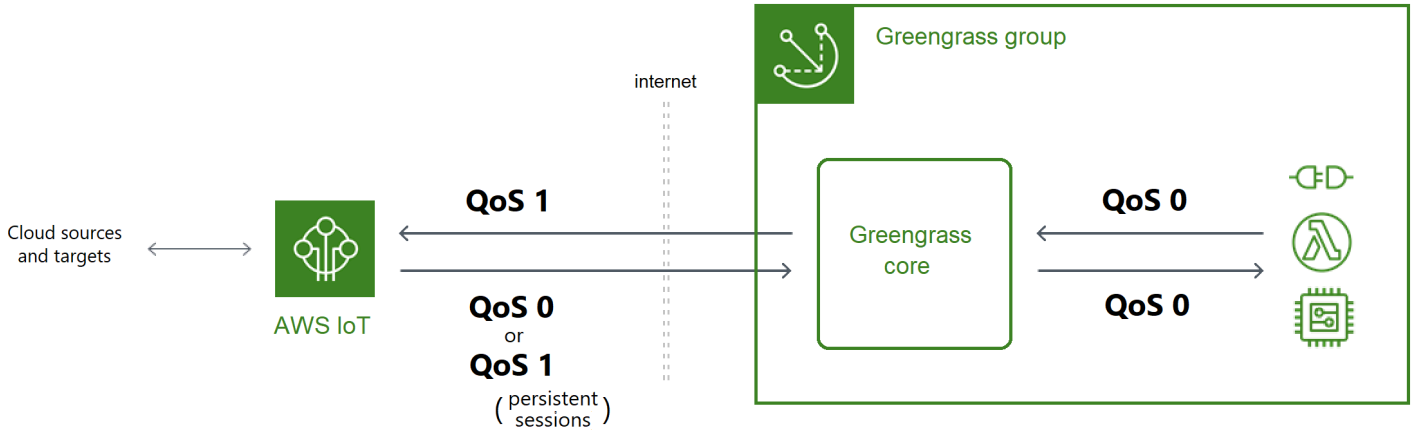
- the section called “Enviar mensagem sobre a qualidade de serviço”
- the section called “Fila de mensagens MQTT”
- the section called “Sessões persistentes do MQTT com AWS IoT Core”
- the section called “IDs de cliente para conexões MQTT com AWS IoT”

- Porta MQTT para mensagens locais
- the section called “Tempo limite para operações de publicação, assinatura e cancelamento de assinatura em conexões MQTT com o Nuvem AWS”

> **Note**
>
> OPC-UA é um padrão de troca de informações para comunicação industrial. Para implementar o suporte para OPC-UA no núcleo do Greengrass, você pode usar o conector IoT SiteWise . O conector envia dados de dispositivos industriais de OPC-UA servidores para propriedades de ativos em AWS IoT SiteWise.

Enviar mensagem sobre a qualidade de serviço

AWS IoT Greengrass suporta os níveis 0 ou 1 de qualidade de serviço (QoS), dependendo da sua configuração e do destino e direção da comunicação. O núcleo do Greengrass atua como um cliente para comunicação AWS IoT Core e um agente de mensagens para comunicação na rede local.



Para obter mais informações sobre o MQTT e QoS, consulte Introdução no site do MQTT.

Comunicação com o Nuvem AWS

- Mensagens de saída usam QoS 1

  O núcleo envia mensagens destinadas a Nuvem AWS alvos usando QoS 1. AWS IoT Greengrass usa uma fila de mensagens MQTT para processar essas mensagens. Se a entrega da mensagem não for confirmada pelo AWS IoT, a mensagem será colocada em spool para ser repetida mais tarde. A mensagem não pode ser repetida se a fila estiver cheia. A confirmação

de entrega de mensagens pode ajudar a minimizar a perda de dados devido à conectividade intermitente.

Como as mensagens de saída AWS IoT usam QoS 1, a taxa máxima na qual o núcleo do Greengrass pode enviar mensagens depende da latência entre o núcleo e. AWS IoT Cada vez que o núcleo envia uma mensagem, ele espera até AWS IoT confirmar a mensagem antes de enviar a próxima mensagem. Por exemplo, se o tempo de ida e volta entre o núcleo e o sistema Região da AWS for de 50 milissegundos, o núcleo poderá enviar até 20 mensagens por segundo. Considere esse comportamento ao escolher Região da AWS onde seu núcleo se conecta. Para ingerir dados de IoT de alto volume para o Nuvem AWS, você pode usar o gerenciador de fluxo.

Para obter mais informações sobre a fila de mensagens do MQTT, incluindo como configurar um cache de armazenamento local que possa persistir nas mensagens destinadas Nuvem AWS aos destinos, consulte. the section called “Fila de mensagens MQTT”

- As mensagens de entrada usam QoS 0 (padrão) ou QoS 1

  Por padrão, o núcleo se inscreve com QoS 0 para mensagens de fontes da Nuvem AWS . Se você habilitar sessões persistentes, o núcleo se inscreverá com QoS 1. Isso pode ajudar a minimizar a perda de dados devido à conectividade intermitente. Para gerenciar a QoS para essas assinaturas, defina as configurações de persistência no componente de sistema do spooler local.

  Para obter mais informações, incluindo como permitir que o núcleo estabeleça uma sessão persistente com Nuvem AWS metas, consultethe section called “Sessões persistentes do MQTT com AWS IoT Core”.

Comunicação com os destinos locais

Toda comunicação local usa QoS 0. O núcleo tenta enviar uma mensagem para um destino local, que pode ser uma função do Lambda do Greengrass, um conector ou um dispositivo cliente. O núcleo não armazena mensagens nem confirma a entrega. As mensagens podem ser descartadas em qualquer lugar entre os componentes.

> **Note**
>
> Embora a comunicação direta entre funções do Lambda não use mensagens MQTT, o comportamento é o mesmo.

Fila de mensagens MQTT para destinos de nuvem

As mensagens MQTT destinadas aos Nuvem AWS alvos são enfileiradas para aguardar o processamento. As mensagens enfileiradas são processadas na ordem First In First Out (FIFO – Primeiro a entrar, primeiro a sair). Depois que uma mensagem é processada e publicada no AWS IoT Core, ela é removida da fila.

Por padrão, o núcleo do Greengrass armazena mensagens não processadas na memória para destinos da Nuvem AWS . Em vez disso, você pode configurar o núcleo para armazenar mensagens não processadas em um cache de armazenamento local. Diferentemente do armazenamento na memória, o cache de armazenamento local tem a capacidade de persistir nas reinicializações principais (por exemplo, após uma implantação em grupo ou uma reinicialização do dispositivo), portanto, AWS IoT Greengrass pode continuar processando as mensagens. Você também pode configurar o tamanho do armazenamento.

> **Warning**
>
> O núcleo do Greengrass pode enfileirar mensagens MQTT duplicadas quando perde a conexão, porque ele tenta novamente uma operação de publicação antes que o cliente MQTT detecte que está off-line. Para evitar mensagens MQTT duplicadas para destinos na nuvem, configure o valor `keepAlive` do núcleo para menos da metade de seu valor `mqttOperationTimeout`. Para obter mais informações, consulte AWS IoT Greengrass arquivo de configuração principal.

AWS IoT Greengrass usa o componente do sistema spooler (a função `GGCloudSpooler` Lambda) para gerenciar a fila de mensagens. Você pode usar as seguintes variáveis de ambiente `GGCloudSpooler` para definir as configurações de armazenamento.

- GG_CONFIG_STORAGE_TYPE. O local da fila de mensagens. Estes são valores válidos:
  - `FileSystem`. Armazena mensagens não processadas no cache de armazenamento local no disco do dispositivo de núcleo físico. Quando o núcleo é reiniciado, as mensagens são mantidas na fila para processamento. As mensagens serão removidas depois de serem processadas.
  - `Memory` (padrão). Armazene mensagens não processadas na memória. Quando o núcleo é reiniciado, as mensagens enfileiradas são perdidas.

Essa opção é otimizado para dispositivos com recursos de hardware restritos. Ao usar essa configuração, recomendamos que você implante grupos ou reinicie o dispositivo quando a interrupção do serviço é a mais baixa.

- GG_CONFIG_MAX_SIZE_BYTES. O tamanho de armazenamento, em bytes. Esse valor pode ser qualquer número inteiro não negativo maior ou igual a 262144 (256 KB); um tamanho menor impede que o software AWS IoT Greengrass Core seja iniciado. O tamanho padrão é 2,5 MB. Quando o limite de tamanho é atingido, as mensagens enfileiradas anteriores são substituídas por novas mensagens.

> **Note**
>
> Esse recurso está disponível para AWS IoT Greengrass Core v1.6 e versões posteriores. As versões anteriores usam o armazenamento na memória com um tamanho de fila de 2,5 MB. Você não pode definir configurações de armazenamento para versões anteriores.

Como armazenar mensagens em cache no armazenamento local

Você pode configurar AWS IoT Greengrass para armazenar mensagens em cache no sistema de arquivos para que elas persistam nas reinicializações principais. Para fazer isso, implante uma versão de definição de função em que a função `GGCloudSpooler` defina o tipo de armazenamento como `FileSystem`. Você deve usar a AWS IoT Greengrass API para configurar o cache de armazenamento local. Você pode fazer isso no console.

O procedimento a seguir usa o comando da CLI `create-function-definition-version` para configurar o spooler a fim de salvar mensagens enfileiradas no sistema de arquivos. Ele também configura um tamanho de fila de 2,6 MB.

1. Obtenha os IDs do grupo do Greengrass de destino e a versão do grupo. Esse procedimento pressupõe que esse seja o grupo e a versão mais recente do grupo. A consulta a seguir retorna o grupo criado mais recentemente.

   ```
   aws greengrass list-groups --query "reverse(sort_by(Groups, &CreationTimestamp))
   [0]"
   ```

Ou é possível consultar por nome. Os nomes de grupo não precisam ser exclusivos, portanto, vários grupos podem ser retornados.

```
aws greengrass list-groups --query "Groups[?Name=='MyGroup']"
```

> **Note**
>
> Você também pode encontrar esses valores no AWS IoT console. O ID do grupo é exibido na página Settings (Configurações) do grupo. Os IDs de versão do grupo são exibidos na guia Implantações do grupo.

2. Copie os valores `Id` e `LatestVersion` do grupo de destino na saída.
3. Obtenha a versão do grupo mais recente.

   - Substitua *group-id* pelo `Id` que você copiou.
   - Substitua *latest-group-version-id* pelo `LatestVersion` que você copiou.

```
aws greengrass get-group-version \
--group-id group-id \
--group-version-id latest-group-version-id
```

4. No objeto `Definition` da saída, copie `CoreDefinitionVersionArn` e os ARNs de todos os outros componentes do grupo, exceto `FunctionDefinitionVersionArn`. Você usa esses valores ao criar uma nova versão do grupo.
5. No `FunctionDefinitionVersionArn` na saída, copie o ID da definição de função. O ID é o GUID que segue o segmento `functions` no ARN, conforme mostrado no exemplo a seguir.

```
arn:aws:greengrass:us-west-2:123456789012:/greengrass/
definition/functions/bcfc6b49-beb0-4396-b703-6dEXAMPLEcu5/
versions/0f7337b4-922b-45c5-856f-1aEXAMPLEsf6
```

> **Note**
>
> Ou você pode criar uma definição de função executando o comando `create-function-definition` e copiar o ID da saída.

6. Adicione uma versão de definição de função à definição da função.

   - *function-definition-id*Substitua pelo `Id` que você copiou para a definição da função.
   - *arbitrary-function-id*Substitua por um nome para a função, como**spooler-function**.
   - Adicione todas as funções do Lambda que você deseja incluir nesta versão para a matriz `functions`. Você pode usar o comando `get-function-definition-version` para obter as funções do Lambda do Greengrass de uma versão de definição da função existente.

   > **Warning**
   >
   > Não se esqueça de especificar um valor para `GG_CONFIG_MAX_SIZE_BYTES` que seja maior que ou igual a 262.144. Um tamanho menor impede que o software AWS IoT Greengrass Core seja iniciado.

   ```
   aws greengrass create-function-definition-version \
   --function-definition-id function-definition-id \
   --functions '[{"FunctionArn":
    "arn:aws:lambda:::function:GGCloudSpooler:1","FunctionConfiguration":
    {"Environment": {"Variables":
   {"GG_CONFIG_MAX_SIZE_BYTES":"2621440","GG_CONFIG_STORAGE_TYPE":"FileSystem"}},"Executable":
    "spooler","MemorySize": 32768,"Pinned": true,"Timeout": 3},"Id": "arbitrary-
   function-id"}]'
   ```

   > **Note**
   >
   > Se você definiu anteriormente a variável de `GG_CONFIG_SUBSCRIPTION_QUALITY` ambiente para oferecer suporte a sessões persistentes AWS IoT Core, inclua-a nessa instância de função.

7. Copie o `Arn` da versão da definição de função da saída.
8. Crie uma versão do grupo que contém a função do Lambda definida pelo sistema.

   - *group-id*Substitua `Id` pelo para o grupo.
   - *core-definition-version-arn*Substitua pelo `CoreDefinitionVersionArn` que você copiou da versão mais recente do grupo.

- *function-definition-version-arn*Substitua pelo `Arn` que você copiou para a nova versão de definição de função.
- Substitua os ARNs de outros componentes do grupo (por exemplo, `SubscriptionDefinitionVersionArn` ou `DeviceDefinitionVersionArn`) que você copiou na versão do grupo mais recente.
- Remova todos os parâmetros inutilizados. Por exemplo, remova a versão `--resource-definition-version-arn` caso a versão do grupo não contenha recursos.

```
aws greengrass create-group-version \
--group-id group-id \
--core-definition-version-arn core-definition-version-arn \
--function-definition-version-arn function-definition-version-arn \
--device-definition-version-arn device-definition-version-arn \
--logger-definition-version-arn logger-definition-version-arn \
--resource-definition-version-arn resource-definition-version-arn \
--subscription-definition-version-arn subscription-definition-version-arn
```

9. Copie a `Version` da saída. Este é o ID da nova versão do grupo.
10. Implante o grupo com a nova versão do grupo.

    - *group-id*Substitua pelo `Id` que você copiou para o grupo.
    - *group-version-id*Substitua pelo `Version` que você copiou para a nova versão do grupo.

```
aws greengrass create-deployment \
--group-id group-id \
--group-version-id group-version-id \
--deployment-type NewDeployment
```

Para atualizar as configurações de armazenamento, você usa a AWS IoT Greengrass API para criar uma nova versão de definição de função que contém a `GGCloudSpooler` função com a configuração atualizada. Em seguida, adicione a versão de definição da função a uma nova versão do grupo (com os outros componentes do grupo) e implantar a versão do grupo. Se quiser restaurar a configuração padrão, você poderá implantar uma versão de definição da função que não inclua a função `GGCloudSpooler`.

Essa função do Lambda do sistema não está visível no console. No entanto, depois que a função for adicionada à versão do grupo mais recente, ela será incluída em implantações que você fizer no console, a menos que você use a API para substituir ou removê-la.

Sessões persistentes do MQTT com AWS IoT Core

Esse recurso está disponível para AWS IoT Greengrass Core v1.10 e versões posteriores.

Um núcleo do Greengrass pode estabelecer uma sessão persistente com o agente de AWS IoT mensagens. Uma sessão persistente é uma conexão contínua que permite que o núcleo receba mensagens enviadas enquanto ele está off-line. O núcleo é o cliente na conexão.

Em uma sessão persistente, o agente de AWS IoT mensagens salva todas as assinaturas que o núcleo faz durante a conexão. Se o núcleo se desconectar, o agente de AWS IoT mensagens armazenará mensagens novas e não confirmadas publicadas como QoS 1 e destinadas a destinos locais, como funções do Lambda e dispositivos clientes. ??? Quando o núcleo se reconecta, a sessão persistente é retomada e o agente de AWS IoT mensagens envia mensagens armazenadas para o núcleo a uma taxa máxima de 10 mensagens por segundo. As sessões persistentes têm um período de expiração padrão de 1 hora, que tem início quando o atendente de mensagens detecta que o núcleo se desconectou. Para obter mais informações, consulte Sessões Persistentes do MQTT no Guia do desenvolvedor do AWS IoT .

AWS IoT Greengrass usa o componente do sistema spooler (a função `GGCloudSpooler` Lambda) para criar assinaturas que tenham como fonte. AWS IoT Você pode usar a seguinte variável de ambiente `GGCloudSpooler` para configurar sessões persistentes.

- GG_CONFIG_SUBSCRIPTION_QUALITY. A qualidade das assinaturas que têm AWS IoT como fonte. Estes são valores válidos:
  - `AtMostOnce` (padrão). Desabilita sessões persistentes. As assinaturas usam QoS 0.
  - `AtLeastOncePersistent`. Habilita sessões persistentes. Define o sinalizador `cleanSession` como `0` em mensagens `CONNECT` e se inscreve com QoS 1.

    As mensagens publicadas com QoS 1 recebidas pelo núcleo têm a garantia de alcançar a fila de trabalho na memória do daemon do Greengrass. O núcleo reconhece a mensagem depois que ela é adicionada à fila. A comunicação subsequente da fila para o destino local (por exemplo, função, conector ou dispositivo do Lambda do Greengrass) é enviada como QoS 0. O AWS IoT Greengrass não garante a entrega para destinos locais.

> **Note**
>
> Você pode usar a propriedade de WorkItemCount configuração máxima para controlar o tamanho da fila de itens de trabalho. Por exemplo, você pode aumentar o tamanho da fila se sua carga de trabalho exigir tráfego MQTT pesado.

Quando as sessões persistentes são habilitadas, o núcleo abre pelo menos uma conexão adicional para troca de mensagens MQTT com AWS IoT. Para obter mais informações, consulte the section called “IDs de cliente para conexões MQTT com AWS IoT”.

Como configurar sessões persistentes do MQTT

Você pode configurar AWS IoT Greengrass para usar sessões persistentes com AWS IoT Core. Para fazer isso, implante uma versão de definição de função em que a função `GGCloudSpooler` defina a qualidade da assinatura como `AtLeastOncePersistent`. Essa definição se aplica a todas as assinaturas que tenham o AWS IoT Core (`cloud`) como origem. Você deve usar a AWS IoT Greengrass API para configurar sessões persistentes. Você pode fazer isso no console.

O procedimento a seguir usa o comando da CLI `create-function-definition-version` para configurar o spooler de modo a usar sessões persistentes. Nesse procedimento, estamos supondo que você esteja atualizando a configuração da versão do grupo mais recente de um grupo existente.

1. Obtenha os IDs do grupo do Greengrass de destino e a versão do grupo. Esse procedimento pressupõe que esse seja o grupo e a versão mais recente do grupo. A consulta a seguir retorna o grupo criado mais recentemente.

   ```
   aws greengrass list-groups --query "reverse(sort_by(Groups, &CreationTimestamp))
   [0]"
   ```

   Ou é possível consultar por nome. Os nomes de grupo não precisam ser exclusivos, portanto, vários grupos podem ser retornados.

   ```
   aws greengrass list-groups --query "Groups[?Name=='MyGroup']"
   ```

> **Note**
>
> Você também pode encontrar esses valores no AWS IoT console. O ID do grupo é exibido na página Settings (Configurações) do grupo. Os IDs de versão do grupo são exibidos na guia Implantações do grupo.

2. Copie os valores `Id` e `LatestVersion` do grupo de destino na saída.
3. Obtenha a versão do grupo mais recente.

   - Substitua *`group-id`* pelo `Id` que você copiou.
   - Substitua *`latest-group-version-id`* pelo `LatestVersion` que você copiou.

```
aws greengrass get-group-version \
--group-id group-id \
--group-version-id latest-group-version-id
```

4. No objeto `Definition` da saída, copie `CoreDefinitionVersionArn` e os ARNs de todos os outros componentes do grupo, exceto `FunctionDefinitionVersionArn`. Você usa esses valores ao criar uma nova versão do grupo.
5. No `FunctionDefinitionVersionArn` na saída, copie o ID da definição de função. O ID é o GUID que segue o segmento `functions` no ARN, conforme mostrado no exemplo a seguir.

```
arn:aws:greengrass:us-west-2:123456789012:/greengrass/
definition/functions/bcfc6b49-beb0-4396-b703-6dEXAMPLEcu5/
versions/0f7337b4-922b-45c5-856f-1aEXAMPLEsf6
```

> **Note**
>
> Ou você pode criar uma definição de função executando o comando `create-function-definition` e copiar o ID da saída.

6. Adicione uma versão de definição de função à definição da função.

   - *`function-definition-id`*Substitua pelo `Id` que você copiou para a definição da função.
   - *`arbitrary-function-id`*Substitua por um nome para a função, como**`spooler-function`**.

- Adicione todas as funções do Lambda que você deseja incluir nesta versão para a matriz `functions`. Você pode usar o comando `get-function-definition-version` para obter as funções do Lambda do Greengrass de uma versão de definição da função existente.

```
aws greengrass create-function-definition-version \
--function-definition-id function-definition-id \
--functions '[{"FunctionArn":
 "arn:aws:lambda:::function:GGCloudSpooler:1","FunctionConfiguration":
 {"Environment": {"Variables":
{"GG_CONFIG_SUBSCRIPTION_QUALITY":"AtLeastOncePersistent"}},"Executable":
 "spooler","MemorySize": 32768,"Pinned": true,"Timeout": 3},"Id": "arbitrary-
function-id"}]'
```

> **Note**
>
> Se você definiu anteriormente as variáveis de ambiente `GG_CONFIG_STORAGE_TYPE` ou `GG_CONFIG_MAX_SIZE_BYTES` para definir configurações de armazenamento, inclua as varáveis nessa instância de função.

7. Copie o `Arn` da versão da definição de função da saída.
8. Crie uma versão do grupo que contém a função do Lambda definida pelo sistema.

   - *`group-id`*Substitua `Id` pelo para o grupo.
   - *`core-definition-version-arn`*Substitua pelo `CoreDefinitionVersionArn` que você copiou da versão mais recente do grupo.
   - *`function-definition-version-arn`*Substitua pelo `Arn` que você copiou para a nova versão de definição de função.
   - Substitua os ARNs de outros componentes do grupo (por exemplo, `SubscriptionDefinitionVersionArn` ou `DeviceDefinitionVersionArn`) que você copiou na versão do grupo mais recente.
   - Remova todos os parâmetros inutilizados. Por exemplo, remova a versão `--resource-definition-version-arn` caso a versão do grupo não contenha recursos.

```
aws greengrass create-group-version \
--group-id group-id \
--core-definition-version-arn core-definition-version-arn \
```

```
--function-definition-version-arn function-definition-version-arn \
--device-definition-version-arn device-definition-version-arn \
--logger-definition-version-arn logger-definition-version-arn \
--resource-definition-version-arn resource-definition-version-arn \
--subscription-definition-version-arn subscription-definition-version-arn
```

9. Copie a `Version` da saída. Este é o ID da nova versão do grupo.
10. Implante o grupo com a nova versão do grupo.

    - *group-id*Substitua pelo `Id` que você copiou para o grupo.
    - *group-version-id*Substitua pelo `Version` que você copiou para a nova versão do grupo.

    ```
    aws greengrass create-deployment \
    --group-id group-id \
    --group-version-id group-version-id \
    --deployment-type NewDeployment
    ```

11. (Opcional) Aumente a WorkItemCount propriedade máxima no arquivo de configuração principal. Isso pode ajudar o núcleo a lidar com o aumento do tráfego MQTT e a comunicação com destinos locais.

Para atualizar o núcleo com essas alterações de configuração, você usa a AWS IoT Greengrass API para criar uma nova versão de definição de função que contém a `GGCloudSpooler` função com a configuração atualizada. Em seguida, adicione a versão de definição da função a uma nova versão do grupo (com os outros componentes do grupo) e implantar a versão do grupo. Se quiser restaurar a configuração padrão, você poderá criar uma versão de definição da função que não inclua a função `GGCloudSpooler`.

Essa função do Lambda do sistema não está visível no console. No entanto, depois que a função for adicionada à versão do grupo mais recente, ela será incluída em implantações que você fizer no console, a menos que você use a API para substituir ou removê-la.

IDs de cliente para conexões MQTT com AWS IoT

Esse recurso está disponível para AWS IoT Greengrass Core v1.8 e versões posteriores.

O núcleo do Greengrass abre conexões MQTT com AWS IoT Core para operações como sincronização de sombra e gerenciamento de certificados. Para essas conexões, o núcleo gera IDs de cliente previsíveis com base no nome da coisa. IDs de cliente previsíveis podem ser usadas com

recursos de monitoramento, auditoria e preços, incluindo AWS IoT Device Defender eventos do AWS IoT ciclo de vida. Você também pode criar uma lógica em torno de IDs de cliente previsíveis (por exemplo, modelos de política de assinatura com base em atributos de certificado).

GGC v1.9 and later

Dois componentes do sistema Greengrass abrem conexões MQTT com. AWS IoT Core Esses componentes usam os padrões a seguir para gerar os IDs de cliente para as conexões.

Operation	Padrão de ID de cliente
Implantações	*core-thing-name* Exemplo: `MyCoreThing` Use esse ID de cliente para se conectar, desconectar, assinar e cancelar a assinatura de notificações de eventos de ciclo de vida.
Assinaturas	*core-thing-name* `-c`*n* Exemplo: `MyCoreThing-c01` *n* é um número inteiro que começa com 00 e é incrementado a cada nova conexão até um número máximo de 250. O número de conexões é determinado pelo número de dispositivos que sincronizam o estado de shadow com o AWS IoT Core (máximo de 2.500 por grupo) e o número de assinaturas com `cloud` como origem no grupo (máximo de 10.000 por grupo). O componente do sistema de spooler se conecta com o AWS IoT Core para trocar mensagens de assinaturas com uma nuvem de origem ou de destino. O spooler também atua como proxy para troca de mensagens

Operation	Padrão de ID de cliente
	entre o serviço paralelo local AWS IoT Core e o gerenciador de certificados de dispositivos.

Para calcular o número de conexões MQTT por grupo, use a seguinte fórmula:

```
number of MQTT connections per group = number of connections for
Deployment Agent + number of connections for Subscriptions
```

Onde,

- número de conexões para o atendente de implantação = 1.
- número de conexões para assinaturas = `(2 subscriptions for supporting certificate generation + number of MQTT topics in AWS IoT Core + number of device shadows synced) / 50`.
  - Onde, `50` = o número máximo de assinaturas por conexão que AWS IoT Core podem suportar.

> **Note**
>
> Se você habilitar sessões persistentes para assinatura AWS IoT Core, o núcleo abrirá pelo menos uma conexão adicional para usar em uma sessão persistente. Os componentes do sistema não são compatíveis com sessões persistentes, por isso não podem compartilhar essa conexão.

Para reduzir o número de conexões MQTT e ajudar a reduzir custos, você pode usar as funções locais do Lambda para agregar dados na borda. Em seguida, você envia os dados agregados para o Nuvem AWS. Como resultado, você usa menos tópicos do MQTT em AWS IoT Core. Para obter mais informações, consulte Preços do AWS IoT Greengrass.

GGC v1.8

Vários componentes do sistema do Greengrass abrem conexões MQTT com o AWS IoT Core. Esses componentes usam os padrões a seguir para gerar os IDs de cliente para as conexões.

Operation	Padrão de ID de cliente
Implantações	*core-thing-name* Exemplo: `MyCoreThing` Use esse ID de cliente para se conectar, desconectar, assinar e cancelar a assinatura de notificações de eventos de ciclo de vida.
Troca de mensagens MQTT com AWS IoT Core	*core-thing-name* `-spr` Exemplo: `MyCoreThing-spr`
Sincronização de sombra	*core-thing-name* `-s`*nn* Exemplo: `MyCoreThing-s01` *nn* é um número inteiro que começa em 00 e é incrementado a cada nova conexão até um máximo de 03. O número de conexões é determinado pelo número de dispositivos (máximo de 200 dispositivos por grupo) que sincronizam seu estado de sombra com AWS IoT Core (máximo de 50 assinaturas por conexão).
Gerenciamento de certificados do dispositivo	*core-thing-name* `-dcm` Exemplo: `MyCoreThing-dcm`

> **Note**
>
> Duplicar os IDs de cliente usados em conexões simultâneas pode causar um loop infinito de desconexão e conexão. Isso pode acontecer se outro dispositivo for codificado para usar o nome do dispositivo de núcleo como o ID do cliente nas conexões. Para obter mais informações, consulte esta etapa de solução de problemas.

Os Dispositivos Greengrass também são totalmente integrados ao serviço de indexação de frota do AWS IoT Device Management. Isso permite indexar e pesquisar dispositivos com base nos atributos do dispositivo, no estado da sombra e no estado da conexão na nuvem. Por exemplo, os Dispositivos Greengrass estabelecem pelo menos uma conexão que usa o nome da coisa como o ID do cliente. Assim, é possível usar a indexação de conectividade do dispositivo para descobrir quais Dispositivos Greengrass estão conectados ou desconectados do AWS IoT Core no momento. Para obter mais informações, consulte Serviço de indexação de frota no Guia do desenvolvedor do AWS IoT .

Configurar a porta MQTT para mensagens locais

Esse recurso requer o AWS IoT Greengrass Core v1.10 ou posterior.

O núcleo do Greengrass atua como atendente de mensagens local para mensagens MQTT entre funções locais do Lambda, conectores e dispositivos cliente. Por padrão, o núcleo usa a porta 8883 para o tráfego MQTT na rede local. Talvez você queira alterar a porta para evitar um conflito com outro software executado na porta 8883.

Para configurar o número da porta usada pelo núcleo para o tráfego MQTT local

1. Execute o comando a seguir para interromper o daemon do Greengrass:

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

2. Abra *greengrass-root*`/config/config.json` para editar como o usuário su.
3. No objeto `coreThing`, adicione a propriedade `ggMqttPort` e defina o valor para o número da porta que você deseja usar. Os valores válidos são de 1024 a 65535. O exemplo a seguir define o número da porta como `9000`.

   ```
   {
       "coreThing" : {
           "caPath" : "root.ca.pem",
           "certPath" : "12345abcde.cert.pem",
           "keyPath" : "12345abcde.private.key",
           "thingArn" : "arn:aws:iot:us-west-2:123456789012:thing/core-thing-name",
           "iotHost" : "abcd123456wxyz-ats.iot.us-west-2.amazonaws.com",
           "ggHost" : "greengrass-ats.iot.us-west-2.amazonaws.com",
           "ggMqttPort" : 9000,
           "keepAlive" : 600
       },
       ...
   ```

```
}
```

4. Inicie o daemon.

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

5. Se a detecção automática de IP estiver habilitada para o núcleo, a configuração será concluída.

   Se a detecção automática de IP não estiver habilitada, você deverá atualizar as informações de conectividade para o núcleo. Isso permite que os dispositivos cliente recebam o número de porta correto durante as operações de descoberta para adquirir informações de conectividade de núcleo. Você pode usar o AWS IoT console ou a AWS IoT Greengrass API para atualizar as principais informações de conectividade. Para esse procedimento, atualize apenas o número da porta. O endereço IP local do núcleo permanece o mesmo.

   Para atualizar as informações de conectividade para o núcleo (console)

   1. Na página de configuração do grupo, selecione o núcleo do Greengrass.
   2. Na página de detalhes do núcleo, selecione a guia de endpoints do atendente MQTT.
   3. Selecione Gerenciar endpoints e, em seguida, Add endpoint (Adicionar endpoint).
   4. Insira seu endereço IP local atual e o novo número da porta. O exemplo a seguir define o número da porta `9000` para o endereço IP `192.168.1.8`.
   5. Remova o endpoint obsoleto e selecione Update (Atualizar)

   Atualizar as informações de conectividade para o núcleo (API)

   - Use a ação UpdateConnectivityInfo. O exemplo a seguir usa `update-connectivity-info` na AWS CLI para definir o número da porta `9000` para o endereço IP `192.168.1.8`.

     ```
     aws greengrass update-connectivity-info \
         --thing-name "MyGroup_Core" \
         --connectivity-info "[{\"Metadata\":\"\",\"PortNumber\":9000,
     \"HostAddress\":\"192.168.1.8\",\"Id\":\"localIP_192.168.1.8\"},{\"Metadata
     \":\"\",\"PortNumber\":8883,\"HostAddress\":\"127.0.0.1\",\"Id\":
     \"localhost_127.0.0.1_0\"}]"
     ```

> **Note**
>
> Você também pode configurar a porta que o núcleo usa para enviar mensagens MQTT. AWS IoT Core Para obter mais informações, consulte the section called “Conectar-se à porta 443 ou por meio de um proxy de rede”.

Tempo limite para operações de publicação, assinatura e cancelamento de assinatura em conexões MQTT com o Nuvem AWS

Esse recurso está disponível na AWS IoT Greengrass versão v1.10.2 ou posterior.

É possível configurar a quantidade de tempo (em segundos) para permitir que o núcleo do Greengrass conclua uma operação de publicação, assinatura ou cancelamento de assinatura em conexões MQTT ao AWS IoT Core. Talvez você queira ajustar essa configuração se as operações atingirem o tempo limite devido a restrições de largura de banda ou à alta latência. Para definir essa configuração no arquivo config.json, adicione ou altere a propriedade `mqttOperationTimeout` no objeto `coreThing`. Por exemplo:

```
{
  "coreThing": {
    "mqttOperationTimeout": 10,
    "caPath": "root-ca.pem",
    "certPath": "hash.cert.pem",
    "keyPath": "hash.private.key",
    ...
  },
  ...
}
```

O tempo limite padrão é 5 segundos. O tempo limite mínimo é de 5 segundos.

Ativar detecção automática de IP

Você pode configurar AWS IoT Greengrass para permitir que dispositivos clientes em um grupo do Greengrass descubram automaticamente o núcleo do Greengrass. Quando ativado, o núcleo observa as alterações em seus endereços IP. Se um endereço mudar, o núcleo publica uma lista

atualizada de endereços. Esses endereços são disponibilizados para dispositivos cliente que estão no mesmo grupo do Greengrass que o núcleo.

> **Note**
>
> A AWS IoT política para dispositivos cliente deve conceder a `greengrass:Discover` permissão para permitir que os dispositivos recuperem as informações de conectividade do núcleo. Para obter mais informações sobre a declaração de política, consulte the section called “Autorização de descoberta”.

Para ativar esse recurso no AWS IoT Greengrass console, escolha Detecção automática ao implantar seu grupo do Greengrass pela primeira vez. Você também pode ativar ou desativar esse atributo na página de configuração do grupo escolhendo a guia Funções do Lambda e selecionando o Detector de IP. A detecção automática de IP é habilitada se Automatically detect and override MQTT broker endpoints (Detectar e substituir automaticamente os endpoints de atendente MQTT) estiver selecionado.

Para gerenciar a descoberta automática com a AWS IoT Greengrass API, você deve configurar a função Lambda `IPDetector` do sistema. O procedimento a seguir mostra como usar o comando create-function-definition-version da CLI para configurar a identificação automática do núcleo do Greengrass.

1. Obtenha os IDs do grupo do Greengrass de destino e a versão do grupo. Esse procedimento pressupõe que esse seja o grupo e a versão mais recente do grupo. A consulta a seguir retorna o grupo criado mais recentemente.

   ```
   aws greengrass list-groups --query "reverse(sort_by(Groups, &CreationTimestamp))
   [0]"
   ```

   Ou é possível consultar por nome. Os nomes de grupo não precisam ser exclusivos, portanto, vários grupos podem ser retornados.

   ```
   aws greengrass list-groups --query "Groups[?Name=='MyGroup']"
   ```

> **Note**
>
> Você também pode encontrar esses valores no AWS IoT console. O ID do grupo é exibido na página Settings (Configurações) do grupo. Os IDs de versão do grupo são exibidos na guia Implantações do grupo.

2. Copie os valores `Id` e `LatestVersion` do grupo de destino na saída.
3. Obtenha a versão do grupo mais recente.

   - Substitua *`group-id`* pelo `Id` que você copiou.
   - Substitua *`latest-group-version-id`* pelo `LatestVersion` que você copiou.

```
aws greengrass get-group-version \
--group-id group-id \
--group-version-id latest-group-version-id
```

4. No objeto `Definition` da saída, copie `CoreDefinitionVersionArn` e os ARNs de todos os outros componentes do grupo, exceto `FunctionDefinitionVersionArn`. Você usa esses valores ao criar uma nova versão do grupo.
5. No `FunctionDefinitionVersionArn` na saída, copie o ID da definição de função e a versão de definição de função.

```
arn:aws:greengrass:region:account-id:/greengrass/groups/function-definition-id/
versions/function-definition-version-id
```

> **Note**
>
> Você também pode criar uma definição de função executando o comando `create-function-definition` e copiar o ID da saída.

6. Use o comando `get-function-definition-version` para obter o estado da definição atual. Use o *`function-definition-id`* que você copiou para a definição da função. Por exemplo, *`.4d941bc7-92a1-4f45-8d64-EXAMPLEf76c3`*

```
aws greengrass get-function-definition-version
--function-definition-id function-definition-id
```

```
--function-definition-version-id function-definition-version-id
```

Anote os valores das configurações de função listados. Você precisará incluí-los ao criar uma nova versão de definição de função para evitar a perda de suas configurações de definição atuais.

7. Adicione uma versão de definição de função à definição da função.

   - *function-definition-id*Substitua pelo `Id` que você copiou para a definição da função. Por exemplo, *.4d941bc7-92a1-4f45-8d64-EXAMPLEf76c3*
   - *arbitrary-function-id*Substitua por um nome para a função, como**auto-detection-function**.
   - Adicione todas as funções do Lambda que você deseja incluir nesta versão à matriz `functions`, como as listadas na etapa anterior.

```
aws greengrass create-function-definition-version \
--function-definition-id function-definition-id \
--functions
 '[{"FunctionArn":"arn:aws:lambda:::function:GGIPDetector:1","Id":"arbitrary-
function-id","FunctionConfiguration":
{"Pinned":true,"MemorySize":32768,"Timeout":3}}]'\
--region us-west-2
```

8. Copie o `Arn` da versão da definição de função da saída.
9. Crie uma versão do grupo que contém a função do Lambda definida pelo sistema.

   - *group-id*Substitua `Id` pelo para o grupo.
   - *core-definition-version-arn*Substitua pelo `CoreDefinitionVersionArn` que você copiou da versão mais recente do grupo.
   - *function-definition-version-arn*Substitua pelo `Arn` que você copiou para a nova versão de definição de função.
   - Substitua os ARNs de outros componentes do grupo (por exemplo, `SubscriptionDefinitionVersionArn` ou `DeviceDefinitionVersionArn`) que você copiou na versão do grupo mais recente.
   - Remova todos os parâmetros inutilizados. Por exemplo, remova a versão `--resource-definition-version-arn` caso a versão do grupo não contenha recursos.

```
aws greengrass create-group-version \
--group-id group-id \
--core-definition-version-arn core-definition-version-arn \
--function-definition-version-arn function-definition-version-arn \
--device-definition-version-arn device-definition-version-arn \
--logger-definition-version-arn logger-definition-version-arn \
--resource-definition-version-arn resource-definition-version-arn \
--subscription-definition-version-arn subscription-definition-version-arn
```

10. Copie a `Version` da saída. Este é o ID da nova versão do grupo.
11. Implante o grupo com a nova versão do grupo.

    - *group-id*Substitua pelo `Id` que você copiou para o grupo.
    - *group-version-id*Substitua pelo `Version` que você copiou para a nova versão do grupo.

```
aws greengrass create-deployment \
--group-id group-id \
--group-version-id group-version-id \
--deployment-type NewDeployment
```

Se quiser incluir manualmente o endereço IP do núcleo do Greengrass, conclua este tutorial com uma definição de função diferente que não inclua a função `IPDetector`. Isso impedirá que a função de detecção localize e automaticamente e insira seu endereço IP do núcleo do Greengrass.

Essa função do Lambda do sistema não está visível no console do Lambda. Depois que a função for adicionada à versão do grupo mais recente, ela será incluída nas implantações que você fizer com o console (a menos que você use a API para substituí-la ou removê-la).

Configurar o sistema init para iniciar o daemon do Greengrass

Trata-se de uma prática recomendada configurar o sistema init para iniciar o daemon do Greengrass durante a inicialização, especialmente ao gerenciar grandes frotas de dispositivos.

> **Note**
>
> Se você costumava `apt` instalar o software AWS IoT Greengrass Core, pode usar os scripts systemd para ativar o início na inicialização. Para obter mais informações, consulte

the section called “Usar scripts systemd para gerenciar o ciclo de vida do daemon do Greengrass”.

Há tipos diferentes de sistema init, como initd, systemd e SystemV, e eles usam parâmetros de configuração semelhantes. O exemplo a seguir é um arquivo de serviço systemd. O parâmetro `Type` é definido como `forking` porque greengrassd (que é usado para iniciar o Greengrass) divide o processo de daemon do Greengrass, e o parâmetro `Restart` é definido como `on-failure` para direcionar systemd a fim de reiniciar o Greengrass caso o Greengrass entre em um estado de falha.

> **Note**
>
> Para ver se o dispositivo usa systemd, execute o script `check_ggc_dependencies` conforme descrito em Módulo 1. Em seguida, para usar systemd, verifique se o parâmetro `useSystemd` em `config.json` está definido como `yes`.

```
[Unit]
Description=Greengrass Daemon

[Service]
Type=forking
PIDFile=/var/run/greengrassd.pid
Restart=on-failure
ExecStart=/greengrass/ggc/core/greengrassd start
ExecReload=/greengrass/ggc/core/greengrassd restart
ExecStop=/greengrass/ggc/core/greengrassd stop

[Install]
WantedBy=multi-user.target
```

Consulte também

- O que é AWS IoT Greengrass?
- the section called “Plataformas compatíveis e requisitos”
- Começando com AWS IoT Greengrass
- the section called “Visão geral do modelo de objeto de grupo”
- the section called “Integração de segurança de hardware”

AWS IoT Greengrass Version 1 política de manutenção

Use essa política de AWS IoT Greengrass V1 manutenção para entender os diferentes níveis de manutenção e atualizações do AWS IoT Greengrass V1 serviço e do software AWS IoT Greengrass Core v1.x.

Tópicos

- AWS IoT Greengrass esquema de controle de versão
- Fases do ciclo de vida das principais versões do AWS IoT Greengrass Software principal
- Política de manutenção para AWS IoT Greengrass Software principal
- Programação de suspensão
- Política de suporte para AWS Lambda funções nos dispositivos principais do Greengrass
- Política de suporte para AWS IoT Testador de dispositivos para AWS IoT Greengrass V1
- Cronograma do fim da manutenção

AWS IoT Greengrass esquema de controle de versão

AWS IoT Greengrass usa versionamento semântico para o AWS IoT Greengrass software Core. As versões semânticas seguem um sistema de numeração principal.secundária.patch. A versão principal é destinada para alterações funcionais e de API incompatíveis com versões anteriores das versões principais. A versão secundária é destinada para versões que adicionam novas funcionalidades compatíveis com versões anteriores. A versão do patch é destinada para patches de segurança ou correções de erros. Desde sua primeira versão principal, a v1.0.0, AWS IoT Greengrass lançou 11 versões secundárias do software AWS IoT Greengrass Core v1.x, onde a v1.11.6 é a versão mais recente. Recomendamos que você atualize seu software AWS IoT Greengrass Core para a versão mais recente disponível para aproveitar os novos recursos, aprimoramentos e correções de erros.

Em dezembro de 2020, AWS IoT Greengrass lançou sua primeira atualização de versão principal. Essa atualização incluiu o AWS IoT Greengrass V2 serviço e a versão 2.0.3 do software AWS IoT Greengrass Core. Para novos aplicativos, é altamente recomendável que você use AWS IoT Greengrass Version 2 o software AWS IoT Greengrass Core v2.x. A versão 2 recebe novos atributos, inclui todos os principais atributos da V1 e oferece suporte a plataformas adicionais e implantações contínuas em grandes frotas de dispositivos. Para obter mais informações, consulte O que é AWS IoT Greengrass V2? .

Fases do ciclo de vida das principais versões do AWS IoT Greengrass Software principal

Cada versão principal do software AWS IoT Greengrass Core tem as seguintes três fases sequenciais do ciclo de vida. Cada fase do ciclo de vida fornece níveis diferentes de manutenção ao longo de um período após a data de lançamento inicial.

- Fase de lançamento — AWS IoT Greengrass pode lançar as seguintes atualizações:
  - Atualizações de versões secundárias que fornecem novos atributos ou aprimoramentos para atributos existentes
  - Atualizações de versões que fornecem patches de segurança e correções de erros
- Fase de manutenção — AWS IoT Greengrass pode lançar atualizações da versão do patch que fornecem patches de segurança e correções de bugs. AWS IoT Greengrass não lançará novos recursos ou aprimoramentos nos recursos existentes durante a fase de manutenção.
- Fase de vida útil estendida — AWS IoT Greengrass não lançará atualizações que forneçam recursos, aprimoramentos em recursos existentes, patches de segurança ou correções de erros. No entanto, os Nuvem AWS endpoints e as operações de API permanecerão disponíveis e operarão de acordo com o Acordo AWS IoT Greengrass de Nível de Serviço. Os dispositivos que executam o software AWS IoT Greengrass Core v1.x podem continuar se conectando Nuvem AWS e operando.

Após o término da fase de vida útil estendida de uma versão principal do AWS IoT Greengrass, os Nuvem AWS endpoints e as operações da API serão descontinuados e não estarão mais disponíveis. Os dispositivos que executam o software AWS IoT Greengrass Core v1.x não conseguirão se conectar aos Nuvem AWS serviços para operar.

Política de manutenção para AWS IoT Greengrass Software principal

O software AWS IoT Greengrass Core v1.x entrou na fase de vida útil estendida em 30 de junho de 2023. Após essa data, o software AWS IoT Greengrass Core v1.x permanecerá na fase de vida útil estendida até novo aviso.

O software AWS IoT Greengrass Core v2.x está atualmente na fase de lançamento e permanecerá na fase de lançamento até novo aviso. AWS IoT Greengrass continua adicionando novos recursos

e aprimoramentos ao software AWS IoT Greengrass Core v2.x. Por exemplo, AWS IoT Greengrass lançou o suporte ao Windows na versão 2.5.0 do software AWS IoT Greengrass Core. AWS IoT Greengrass lança patches de segurança e correções de erros para todas as versões secundárias do AWS IoT Greengrass Core v2.x por pelo menos 1 ano após a data de lançamento. Para obter mais informações, consulte Novidades no AWS IoT Greengrass V2.

Cronograma da fase de manutenção

Em 30 de junho de 2023, a fase de manutenção do software AWS IoT Greengrass Core v1.11.x terminou. Em 31 de março de 2022, a fase de manutenção do software AWS IoT Greengrass Core v1.10.x terminou. A fase de manutenção termina para determinados artefatos e recursos do software AWS IoT Greengrass Core v1.x antes dessas datas. Para obter mais informações, consulte Cronograma do fim da manutenção.

Se você tem um AWS Support plano, a fase de manutenção do software AWS IoT Greengrass Core v1.x não afeta seu AWS Support plano. Você pode continuar abrindo AWS Support tíquetes mesmo após o término da fase de manutenção. Se você tiver dúvidas ou preocupações, fale com seu contato do AWS Support ou faça uma pergunta em ref:Post da AWS usando a tag AWS IoT Greengrass.

Programação de suspensão

Atualmente, não há nenhum plano para interromper o suporte ao software AWS IoT Greengrass Core v1.x. Os endpoints do AWS IoT Greengrass V1 e as operações da API permanecerão disponíveis até novo aviso. O software AWS IoT Greengrass Core v1.11.6 entrou na fase de vida útil estendida em 30 de junho de 2023. Durante essa fase, os dispositivos que executam o software AWS IoT Greengrass Core v1.x podem continuar se conectando ao AWS IoT Greengrass V1 serviço para operar até novo aviso.

Se AWS IoT Greengrass V1 deixar de receber suporte no futuro, AWS IoT Greengrass forneceremos um aviso prévio de 12 meses antes que isso aconteça. Isso ajudará você a planejar a atualização dos aplicativos a serem usados AWS IoT Greengrass V2 e do software AWS IoT Greengrass Core v2.x. Para obter mais informações sobre como atualizar seus aplicativos para a V2, consulte Mudar de AWS IoT Greengrass V1 para a V2.

Política de suporte para AWS Lambda funções nos dispositivos principais do Greengrass

AWS IoT Greengrass permite que você execute AWS Lambda funções em dispositivos de IoT. AWS Lambda fornece uma política de suporte e cronogramas que determinam o suporte para tempos de execução do Lambda em. AWS IoT Greengrass Depois que um tempo de execução do Lambda chega ao final da fase de suporte, AWS IoT Greengrass também encerra o suporte para esse tempo de execução. Para obter mais informações, consulte Política de suporte do Runtime, no Guia do desenvolvedor do AWS Lambda .

Quando um runtime do Lambda chega ao fim do suporte, você não pode criar ou atualizar funções do Lambda que usem esse runtime. No entanto, você pode continuar a implantar essas funções do Lambda nos dispositivos principais do Greengrass e invocar as funções do Lambda implantadas. Essa política também se aplica ao AWS IoT Greengrass V2.

Política de suporte para AWS IoT Testador de dispositivos para AWS IoT Greengrass V1

AWS IoT O Device Tester (IDT) AWS IoT Greengrass V1 permite que você valide e qualifique seus AWS IoT Greengrass dispositivos para inclusão no Catálogo de dispositivos.AWS Partner A partir de 4 de abril de 2022, AWS IoT o Device Tester (IDT) AWS IoT Greengrass V1 não gera mais relatórios de qualificação assinados. Você não pode mais qualificar novos AWS IoT Greengrass V1 dispositivos para serem listados no Catálogo de AWS Partner Dispositivos por meio do Programa de Qualificação de AWS Dispositivos. Embora você não possa qualificar os dispositivos Greengrass V1, você pode continuar usando o IDT AWS IoT Greengrass V1 para testar seus dispositivos Greengrass V1. Recomendamos que você use o IDT para AWS IoT Greengrass V2 para qualificar e listar dispositivos Greengrass no AWS Partner Catálogo de dispositivos. Para obter mais informações, consulte Política de suporte para AWS IoT Testador de dispositivos para AWS IoT Greengrass V1.

Cronograma do fim da manutenção

A tabela a seguir lista as datas de fim da manutenção dos artefatos e recursos do AWS IoT Greengrass Core v1.x. Se você tiver dúvidas sobre o cronograma ou a política de manutenção, entre em contato com o Suporte da AWS.

Artefato ou atributo	Data de fim da manutenção
Instalação do repositório Greengrass APT	11 de fevereiro de 2022
Conector de classificação de imagem do ML	31 de março de 2022
Conector de detecção de objetos do ML	31 de março de 2022
Conector ML Feedback	31 de março de 2022
AWS IoT Analytics conector	31 de março de 2022
Conector de notificações Twilio	31 de março de 2022
Conector de integração Splunk	31 de março de 2022
Conector de fluxo serial	31 de março de 2022
ServiceNow MetricBase Conector de integração	31 de março de 2022
Conector Raspberry Pi GPIO	31 de março de 2022
AWS IoT Greengrass Software principal v1.10.x	31 de março de 2022
AWS IoT Greengrass Imagens do Docker v1.x do software principal	30 de junho de 2022
AWS IoT Greengrass Software principal v1.11.x	30 de junho de 2023
AWS IoT Greengrass Software principal v1.11.x Snap	31 de dezembro de 2023

Fim da manutenção para AWS IoT Greengrass Imagens do Docker v1.x do software principal

Em 30 de junho de 2022, AWS IoT Greengrass encerrou a manutenção das imagens Docker AWS IoT Greengrass do software Core v1.x que são publicadas no Amazon Elastic Container Registry (Amazon ECR) e no Docker Hub. Você pode continuar baixando essas imagens do Docker do Amazon ECR e do Docker Hub até 30 de junho de 2023, ou seja, um ano após o término da manutenção. No entanto, as imagens do Docker do software AWS IoT Greengrass Core v1.x não recebem mais patches de segurança ou correções de erros após o término da manutenção em 30 de junho de 2022. Se você executa uma carga de trabalho de produção que depende dessas imagens do Docker, recomendamos que você crie suas próprias imagens do Docker usando os Dockerfiles fornecidos. AWS IoT Greengrass Para obter mais informações, consulte AWS IoT Greengrass Software Docker.

Fim da manutenção para AWS IoT Greengrass Repositório APT v1.x do software principal

Em 11 de fevereiro de 2022, AWS IoT Greengrass encerrou a manutenção da opção de instalar o software AWS IoT Greengrass Core v1.x a partir de um repositório APT. O repositório APT foi removido nessa data, então você não pode mais usar o repositório APT para atualizar o software Core ou instalar o software AWS IoT Greengrass Core em novos AWS IoT Greengrass dispositivos. Nos dispositivos em que você adicionou o AWS IoT Greengrass repositório, você deve remover o repositório da lista de fontes. Recomendamos que você atualize o software AWS IoT Greengrass Core v1.x usando arquivos tar.

Fim da manutenção para AWS IoT Greengrass Software principal v1.11.x Snap

Em 31 de dezembro de 2023, AWS IoT Greengrass a manutenção da versão 1.11.x do software AWS IoT Greengrass principal do Snap, publicada no snapcraft.io, será encerrada. Os dispositivos que atualmente executam o Snap continuarão funcionando até novo aviso. No entanto, o Snap AWS IoT Greengrass principal não receberá mais patches de segurança ou correções de bugs após o término da manutenção.

Conceitos básicos de AWS IoT Greengrass

Este tutorial de introdução inclui vários módulos projetados para mostrar o AWS IoT Greengrass básico e ajudar você a começar a usar AWS IoT Greengrass. Este tutorial aborda conceitos fundamentais, como:

- Configurando AWS IoT Greengrass núcleos e grupos.
- O processo de implantação para executar AWS Lambda funções na borda.
- Conectando AWS IoT dispositivos, chamados de dispositivos clientes, ao AWS IoT Greengrass núcleo.
- Criação de assinaturas para permitir a comunicação MQTT entre funções locais do Lambda, dispositivos clientes e. AWS IoT

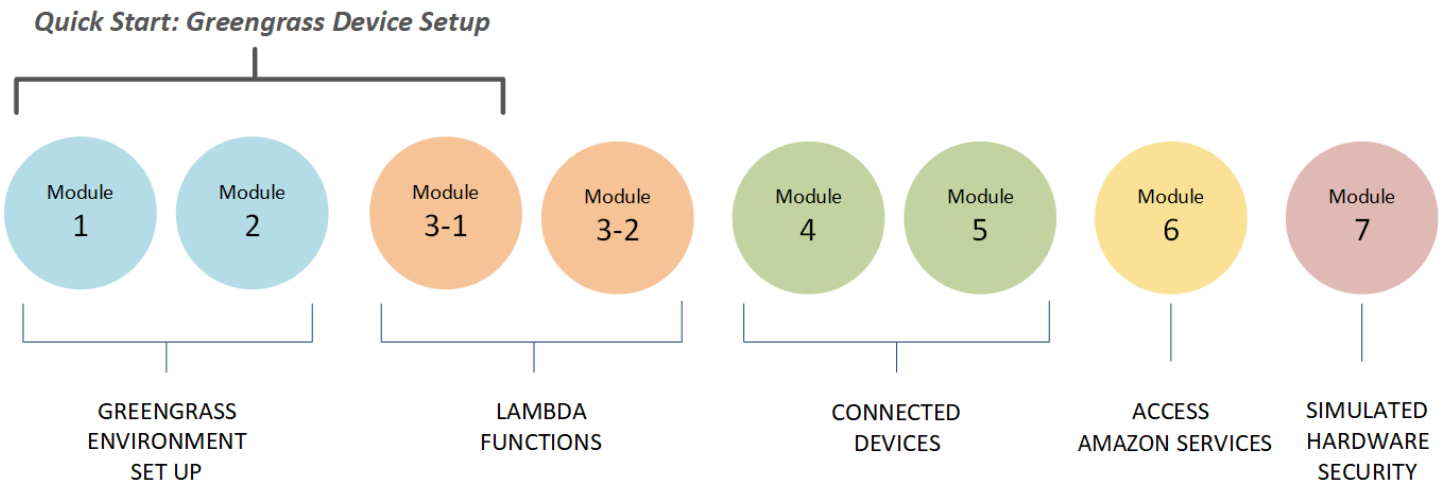
Escolha como começar a usar AWS IoT Greengrass

Você pode escolher como usar este tutorial para configurar seu dispositivo de núcleo:

- Execute a configuração do dispositivo Greengrass em seu dispositivo principal, que leva você da instalação de AWS IoT Greengrass dependências ao teste de uma função Hello World Lambda em minutos. Este script reproduz os passos do Módulo 1 até o Módulo 3-1.

  - ou -

- Siga as instruções do Módulo 1 até o Módulo 3-1 para entender os requisitos e processos do Greengrass mais detalhadamente. Estes passos ensinam a configurar seu dispositivo de núcleo, criar e configurar um grupo do Greengrass que contém uma função Hello World do Lambda e implantar seu grupo do Greengrass. Geralmente, isso leva uma ou duas horas para concluir.



Início rápido

A configuração do dispositivo do Greengrass faz os ajustes do seu dispositivo de núcleo e dos recursos do Greengrass. O script:

- Instala AWS IoT Greengrass dependências.
- Faz download do certificado raiz da CA e do certificado e chaves do dispositivo de núcleo.
- Faz o download, instala e configura o software AWS IoT Greengrass Core em seu dispositivo.
- Inicia o processo daemon do Greengrass no dispositivo de núcleo.
- Cria ou atualiza a função de serviço do Greengrass, se necessário.
- Cria um grupo do Greengrass e um núcleo do Greengrass.
- (Opcional) Cria uma função Hello World do Lambda, uma assinatura e uma configuração de registro em log local.
- (Opcional) Implanta o grupo do Greengrass.

Módulos 1 e 2

OMódulo 1 e o Módulo 2 descrevem como configurar seu ambiente. (Ou use a configuração do dispositivo do Greengrass para que execute esses módulos para você.)

- Configure seu dispositivo de núcleo para o Greengrass.
- Execute o script verificador de dependências
- Crie um grupo do Greengrass e um núcleo do Greengrass.
- Baixe e instale o software AWS IoT Greengrass Core mais recente a partir de um arquivo tar.gz.

- Inicie o processo daemon do Greengrass no núcleo.

> **Note**
>
> AWS IoT Greengrass também fornece outras opções para instalar o software AWS IoT Greengrass Core, incluindo `apt` instalações em plataformas Debian suportadas. Para obter mais informações, consulte the section called “Instale o AWS IoT Greengrass Software principal”.

Módulos 3-1 e 3-2

O Módulo 3-1 e o Módulo 3-2 descrevem como usar as funções locais do Lambda. (Ou use a configuração do dispositivo do Greengrass para que execute o Módulo 3-1 para você.)

- Crie funções Hello World Lambda em. AWS Lambda
- Adicione funções do Lambda ao seu grupo do Greengrass.
- Crie assinaturas que permitam a comunicação MQTT entre as funções Lambda e. AWS IoT
- Configure o registro local para componentes e de sistema do Greengrass e funções do Lambda.
- Implante um grupo do Greengrass com suas funções do Lambda e assinaturas.
- Envie mensagens de funções locais do Lambda para o. AWS IoT
- Invoque funções locais do Lambda de. AWS IoT
- Teste funções sob demanda e de longa duração.

Módulos 4 e 5

O Módulo 4 mostra como os dispositivos cliente se conectam ao núcleo e se comunicam entre si.

O Módulo 5 mostra como os dispositivos cliente podem usar sombras para controlar o estado.

- Registre e provisione AWS IoT dispositivos (representados por terminais de linha de comando).
- Instale o AWS IoT Device SDK para Python. Ele é usado pelos dispositivos cliente para descobrir o núcleo Greengrass.
- Adicione os dispositivos cliente ao seu grupo do Greengrass.
- Crie assinaturas que permitam a comunicação MQTT.
- Implante um grupo do Greengrass com seus dispositivos cliente.

- Teste a comunicação dispositivo a dispositivo.
- Teste as atualizações de estado de sombra.

Módulo 6

O Módulo 6 mostra como as funções do Lambda podem acessar a Nuvem AWS.

- Crie uma função do grupo do Greengrass que permita o acesso aos recursos do Amazon DynamoDB.
- Adicione uma função do Lambda ao seu grupo do Greengrass. Essa função usa o AWS SDK para Python para interagir com o DynamoDB.
- Crie assinaturas que permitam a comunicação MQTT.
- Teste a interação com o DynamoDB.

Módulo 7

O Módulo 7 ensina a configurar a simulação de um módulo de segurança de hardware (HSM) para uso com um núcleo do Greengrass.

> **Important**
>
> Este módulo avançado é fornecido apenas para experimentação e testes iniciais. Ele não é direcionado para o uso em produção de qualquer tipo.

- Instale e configure um HSM baseado em software e uma chave privada.
- Configure o núcleo do Greengrass para usar a segurança do hardware.
- Teste a configuração de segurança do hardware.

Requisitos

Para concluir este tutorial, você precisará do seguinte:

- Um Mac, PC com Windows ou UNIX-like sistema.
- Um Conta da AWS. Se você não tiver uma, consulte the section called “Crie um Conta da AWS”.
- O uso de uma AWS região que oferece suporte AWS IoT Greengrass. Para ver a lista de regiões com suporte para AWS IoT Greengrass, consulte AWS endpoints e cotas no. Referência geral da AWS

> **Note**
>
> Anote sua Região da AWS e certifique-se de que ela seja usada de forma consistente ao longo deste tutorial. Se você trocar o seu Região da AWS durante o tutorial, poderá ter problemas para concluir as etapas.

- Um Raspberry Pi 4 Model B ou Raspberry Pi 3 Model B/B +, com um cartão microSD de 8 GB ou uma instância do Amazon EC2. Como o ideal seria que o AWS IoT Greengrass fosse usado com hardware físico, recomendamos que você use um Raspberry Pi.

  > **Note**
  >
  > Execute o comando a seguir para obter o modelo do Raspberry Pi:
  >
  > ```
  > cat /proc/cpuinfo
  > ```
  >
  > Na parte inferior da listagem, anote o valor do atributo `Revision` e, em seguida, consulte a tabela Qual Pi eu tenho?. Por exemplo, se o valor de `Revision` for `a02082`, a tabela mostrará que o Pi é do 3º modelo na versão B.
  > Execute o comando a seguir para determinar a arquitetura do Raspberry Pi:
  >
  > ```
  > uname -m
  > ```
  >
  > Para este tutorial, o resultado deve ser maior ou igual a `armv71`.

- Familiaridade básica com o Python.

Embora este tutorial seja destinado a ser executado AWS IoT Greengrass em um Raspberry Pi, AWS IoT Greengrass também oferece suporte a outras plataformas. Para obter mais informações, consulte the section called “Plataformas compatíveis e requisitos”.

Inscreva-se para um Conta da AWS

Para começar AWS, você precisa de um Conta da AWS. Há duas maneiras de se inscrever em AWS. Se você quiser ter controle total sobre a configuração da conta e onde seu conteúdo é armazenado, ou se tiver cargas de trabalho regulamentadas, como HIPAA ou FedRAMP, use

Inscrever-se em (avançado). AWS Se você for um novo usuário e quiser usar um login que já possui para entrar AWS e criar um AWS ambiente com padrões pré-configurados em select Regiões da AWS, use Inscrever-se em. AWS

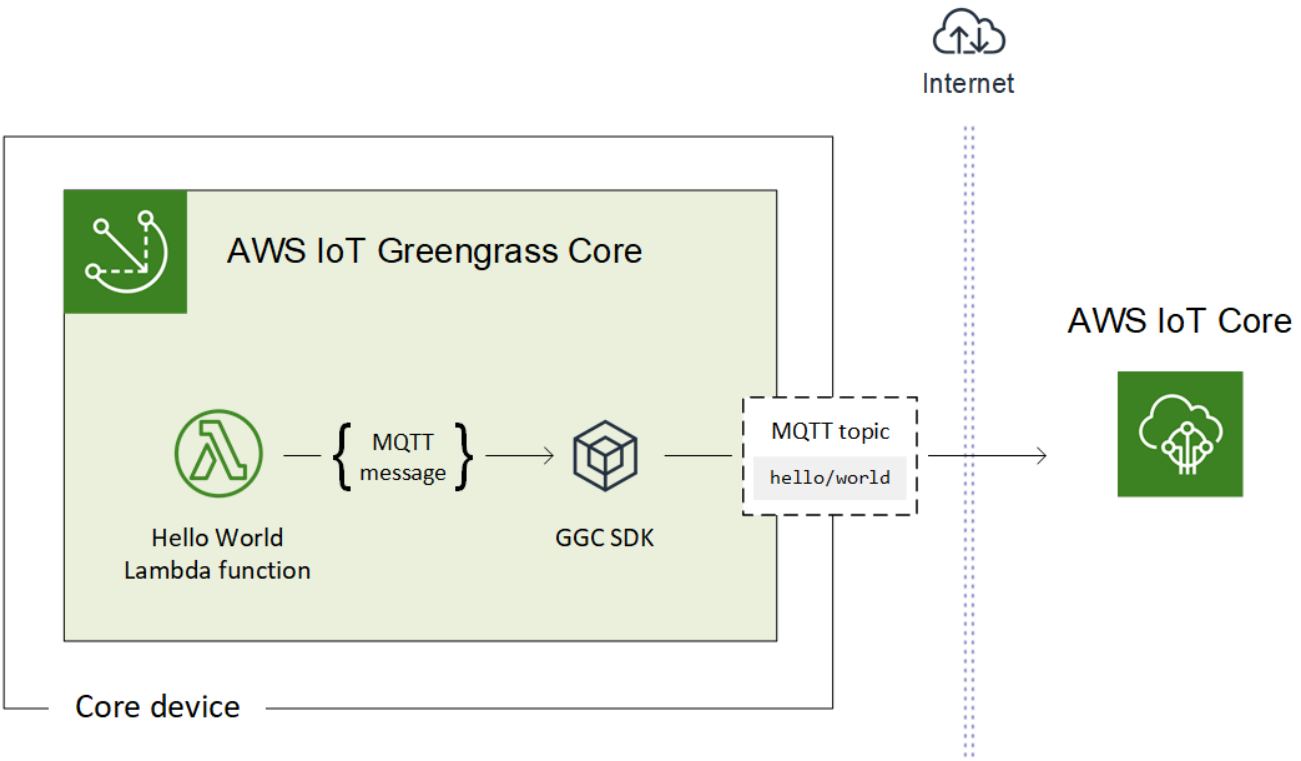
Crie um Conta da AWS

> **Important**
>
> Para este tutorial, consideramos que sua conta de usuário do IAM tem as permissões de acesso de administrador.

Início rápido: Configuração do dispositivo do Greengrass

A configuração do dispositivo do Greengrass é um script que configura seu dispositivo principal em minutos, para que você possa começar a usar AWS IoT Greengrass. Use esse script para:

1. Configure seu dispositivo e instale o software AWS IoT Greengrass Core.
2. Configure os recursos baseados na nuvem.
3. Opcionalmente, implante um grupo do Greengrass com uma função Hello World Lambda que envia mensagens MQTT para o AWS IoT núcleo. AWS IoT Greengrass Isso configura o ambiente do Greengrass conforme o diagrama a seguir.



Requisitos

A configuração do dispositivo do Greengrass tem os seguintes requisitos:

- O dispositivo de núcleo deve usar uma plataforma compatível. O dispositivo deve ter um gerenciador de pacotes apropriado instalado: `apt`, `yum` ou `opkg`.

- O usuário do Linux que executa o script deve ter permissões para executar como `sudo`.

- Você deve fornecer suas Conta da AWS credenciais. Para obter mais informações, consulte the section called “Forneça Conta da AWS credenciais”.

> **Note**
>
> A configuração do dispositivo Greengrass instala a versão mais recente do software AWS IoT Greengrass Core no dispositivo. Ao instalar o software AWS IoT Greengrass Core, você concorda com o Contrato de Licença do Software Greengrass Core.

Executar a configuração do dispositivo do Greengrass

A configuração do dispositivo do Greengrass é feita em poucos passos. Depois de fornecer suas Conta da AWS credenciais, o script provisiona seu dispositivo principal do Greengrass e implanta um grupo do Greengrass em minutos. Execute os seguintes comandos em uma janela de terminal no dispositivo de destino.

> **Note**
>
> Estas etapas mostram como executar o script no modo interativo, que solicita que você insira ou aceite cada valor de entrada. Para obter informações sobre como executar o script silenciosamente, consulte the section called “Execute a configuração do dispositivo do Greengrass no modo silencioso”.

1. Forneça suas credenciais. Neste procedimento, consideramos que você fornecerá as credenciais de segurança temporárias como variáveis do ambiente.

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

   > **Note**
   >
   > Se você estiver executando a configuração do dispositivo Greengrass em um Raspbian ou OpenWrt plataforma, faça uma cópia desses comandos. Você deve fornecê-los novamente depois de reiniciar o dispositivo.

2. Faça download e inicie o script. Você pode usar `wget` ou `curl` para fazer download do script.

   `wget`:

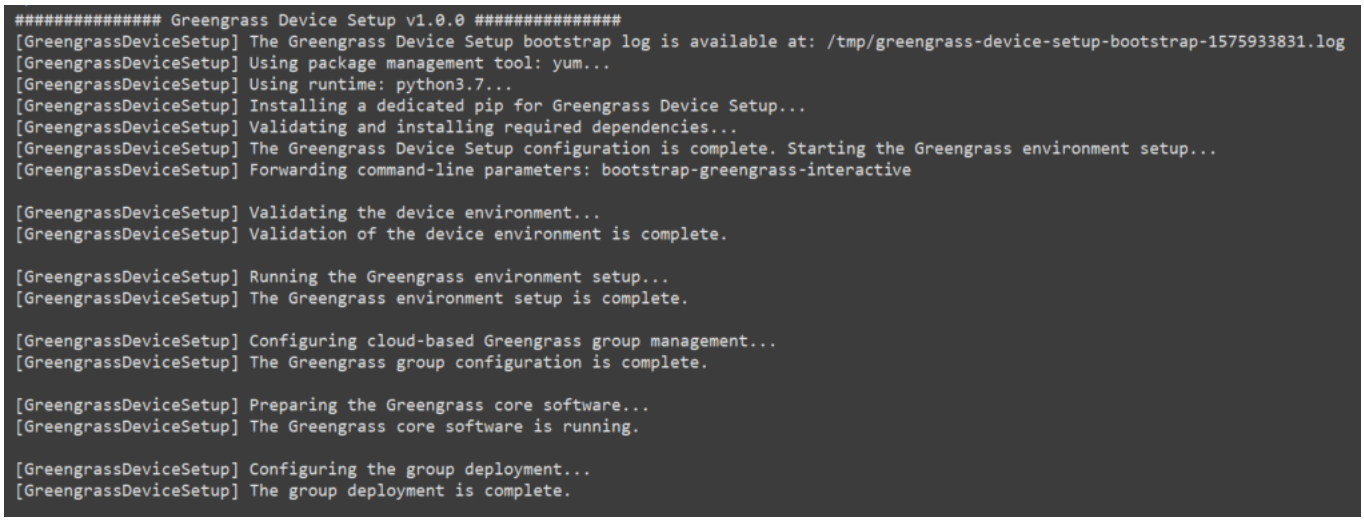
   ```
   wget -q -O ./gg-device-setup-latest.sh https://d1onfpft10uf5o.cloudfront.net/
   greengrass-device-setup/downloads/gg-device-setup-latest.sh && chmod +x ./
   gg-device-setup-latest.sh && sudo -E ./gg-device-setup-latest.sh bootstrap-
   greengrass-interactive
   ```

`curl`:

```
curl https://d1onfpft10uf5o.cloudfront.net/greengrass-device-setup/downloads/
gg-device-setup-latest.sh > gg-device-setup-latest.sh && chmod +x ./gg-device-
setup-latest.sh && sudo -E ./gg-device-setup-latest.sh bootstrap-greengrass-
interactive
```

3. Prossiga pelos prompts de comando e insira os valores. Você pode pressionar a tecla Enter para utilizar o valor padrão ou digitar um valor personalizado e pressionar Enter.

   O script grava mensagens de status no terminal semelhantes às seguintes.



4. Se o seu dispositivo principal estiver executando o Raspbian ou OpenWrt, reinicie o dispositivo quando solicitado, forneça suas credenciais e reinicie o script.

   a. Quando solicitado a reiniciar o dispositivo, execute um dos seguintes comandos.

      Em plataformas Raspbian:

      ```
      sudo reboot
      ```

      Para OpenWrt plataformas:

      ```
      reboot
      ```

b. Depois que o dispositivo for reinicializado, abra o terminal e forneça suas credenciais como variáveis de ambiente.

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

c. Reinicie o script.

```
sudo -E ./gg-device-setup-latest.sh bootstrap-greengrass-interactive
```

d. Quando solicitado se deseja usar os valores de entrada da sessão anterior ou iniciar uma nova instalação, insira `yes` para reutilizar os valores de entrada.

> **Note**
>
> Em plataformas que exigem uma reinicialização, seus valores de entrada da sessão anterior, exceto as credenciais, são temporariamente armazenados no arquivo `GreengrassDeviceSetup.config.info`.

Quando a instalação estiver concluída, o terminal exibirá uma mensagem de status de êxito semelhante à seguinte.

```
================================================================================

Your device is running the Greengrass core software.
Your Greengrass group and Hello World Lambda function were deployed to the core device.


Setup information:

Device info: Linux-4.14.152-127.182.amzn2.x86_64-x86_64-with-glibc2.2.5
Greengrass core software location: /
Installed Greengrass core software version: 1.10.0
Greengrass core: arn:aws:iot:us-west-2:012345678910:thing/GreengrassDeviceSetup_Core_d46a0ea4-18ae-4376-8f44-4a504cdea608
Greengrass core IoT certificate: arn:aws:iot:us-west-2:012345678910:cert/23fbf0f4b6a5ea369f2b97f1a1b558180a240faa8e059ce19dc58f4a4c0d3b77
Greengrass core IoT certificate location: /greengrass/certs/23fbf0f4b6.cert.pem
Greengrass core IoT key location: /greengrass/certs/23fbf0f4b6.private.key
Deployed Greengrass group name: GreengrassDeviceSetup_Group_ee70f777-9af0-43b6-8612-a18b418e8b4a
Deployed Greengrass group ID: 6f5c8410-f3a6-43a2-acf3-33158e10fb8e
Deployed Greengrass group version: arn:aws:greengrass:us-west-2:012345678910:/greengrass/groups/6f5c8410-f3a6-43a2-acf3-33158e10fb8e/versi
Greengrass service role: arn:aws:iam::012345678910:role/GreengrassServiceRole_mui1v
GreengrassDeviceSetup log location: GreengrassDeviceSetup-20191209-232356.log
Deployed hello-world Lambda function: arn:aws:lambda:us-west-2:012345678910:function:Greengrass_HelloWorld_uNTf2:1
Hello-world subscriber topic: hello/world

You can now use the AWS IoT Console to subscribe
to the 'hello/world' topic to receive messages published from your
Greengrass core.

================================================================================
```

5. Analise o novo grupo do Greengrass que o script configura usando os valores de entrada fornecidos por você.

a. Faça login Console de gerenciamento da AWS no seu computador e abra o AWS IoT console.

> **Note**
>
> Certifique-se de que o Região da AWS selecionado no console seja o mesmo que você usou para configurar seu ambiente Greengrass. Por padrão, a região é Oeste dos EUA (Oregon).

b. No painel de navegação, expanda os dispositivos Greengrass e selecione Grupos (V1) para localizar o grupo recém-criado.

6. Se você incluiu a função do Lambda Hello World, a configuração do dispositivo do Greengrass implantará o grupo do Greengrass no seu dispositivo de núcleo. Para testar a função do Lambda ou para obter informações sobre como remover a função do Lambda do grupo, prossiga para the section called “Verificar se a função do Lambda está em execução no dispositivo de núcleo” no Módulo 3-1 do tutorial Conceitos básicos.

> **Note**
>
> Certifique-se de que o Região da AWS selecionado no console seja o mesmo que você usou para configurar seu ambiente Greengrass. Por padrão, a região é Oeste dos EUA (Oregon).

Se você não incluiu a função do Lambda Hello World, você pode criar sua própria função do Lambda ou testar outros atributos do Greengrass. Por exemplo, você pode adicionar o conector de implantação do aplicativo Docker ao seu grupo e usá-lo para implantar contêineres do Docker no seu dispositivo de núcleo.

Solução de problemas

Você pode usar as informações a seguir para solucionar problemas com a configuração do AWS IoT Greengrass dispositivo.

Erro: Python (python3.7) não encontrado. Tentando instalá-lo…

Solução: esse erro pode ser exibido ao se trabalhar com uma instância do Amazon EC2. Esse erro ocorre quando o Python não está instalado na pasta `/usr/bin/python3.7`. Para resolver esse erro, mova o Python para o diretório correto depois de instalá-lo:

```
sudo ln -s /usr/local/bin/python3.7 /usr/bin/python3.7
```

Soluções para outros problemas

Para solucionar problemas adicionais com a configuração do AWS IoT Greengrass dispositivo, você pode procurar informações de depuração nos arquivos de log:

- No caso de problemas com a configuração do dispositivo do Greengrass, verifique o arquivo `/tmp/greengrass-device-setup-bootstrap-`*`epoch-timestamp`*`.log`.
- Para problemas com o grupo do Greengrass ou com a configuração do ambiente do núcleo, verifique o arquivo `GreengrassDeviceSetup-`*`date`*`-`*`time`*`.log` no mesmo diretório que o `gg-device-setup-latest.sh`, ou no local que você especificou.

Para obter mais ajuda na solução de problemas, consulte Solução de problemas ou verifique a AWS IoT Greengrass tag em AWS re:POST.

Opções de configuração do dispositivo do Greengrass

Você configura a configuração do dispositivo Greengrass para acessar seus AWS recursos e configurar seu ambiente Greengrass.

Forneça Conta da AWS credenciais

A configuração do dispositivo Greengrass usa suas Conta da AWS credenciais para acessar seus AWS recursos. Você pode usar credenciais de longo prazo para um usuário do IAM ou de segurança temporárias de um perfil do IAM.

Primeiro, obtenha as credenciais.

- Para usar credenciais de longo prazo, forneça o ID da chave de acesso e a chave de acesso secreta do usuário do IAM. Para obter informações sobre como criar chaves de acesso para credenciais de longo prazo, consulte Gerenciando chaves de acesso para usuários do IAM no Guia do usuário do IAM.

- Para usar credenciais de segurança temporárias (recomendado), forneça o ID da chave de acesso, a chave de acesso secreta e o token de sessão de um perfil do IAM assumido. Para obter informações sobre como extrair credenciais de segurança temporárias do AWS STS `assume-role` comando, consulte Como usar credenciais de segurança temporárias com o AWS CLI no Guia do usuário do IAM.

> **Note**
>
> Neste tutorial, consideramos que o usuário do IAM ou perfil do IAM tem permissões de acesso de administrador.

Selecione uma das maneiras a seguir para fornecer suas credenciais para a configuração do dispositivo do Greengrass:

- Como variáveis de ambiente. Defina as variáveis de ambiente `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY` e `AWS_SESSION_TOKEN` (se necessário) antes de iniciar o script, conforme mostrado no passo 1 do the section called “Executar a configuração do dispositivo do Greengrass”.

- Como valores de entrada. Insira os valores de ID de chave de acesso, a chave de acesso secreta e o token de sessão (se necessário) diretamente no terminal depois de iniciar o script.

A configuração do dispositivo do Greengrass não salva e nem armazena suas credenciais.

Fornecer valores de entrada

No modo interativo, a configuração do dispositivo do Greengrass solicita valores de entrada. Você pode pressionar a tecla Enter para utilizar o valor padrão ou digitar um valor personalizado e pressionar Enter. No modo silencioso, forneça os valores de entrada depois de iniciar o script.

Valores de entrada

AWS ID de chave de acesso

O ID da chave de acesso das credenciais de segurança temporárias ou de longo prazo. Especifique esta opção como valor de entrada somente se você não fornecer suas credenciais como variáveis de ambiente. Para obter mais informações, consulte the section called “Forneça Conta da AWS credenciais”.

Nome da opção para o modo silencioso: `--aws-access-key-id`

AWS chave de acesso secreta = ""

A chave de acesso secreta das credenciais de segurança temporárias ou de longo prazo. Especifique esta opção como valor de entrada somente se você não fornecer suas credenciais como variáveis de ambiente. Para obter mais informações, consulte the section called “Forneça Conta da AWS credenciais”.

Nome da opção para o modo silencioso: `--aws-secret-access-key`

AWS token de sessão

O token de sessão das credenciais de segurança temporárias. Especifique esta opção como valor de entrada somente se você não fornecer suas credenciais como variáveis de ambiente. Para obter mais informações, consulte the section called “Forneça Conta da AWS credenciais”.

Nome da opção para o modo silencioso: `--aws-session-token`

Região da AWS

Região da AWS Onde você deseja criar o grupo Greengrass. Para obter a lista de Região da AWS s suportados, consulte AWS IoT Greengrass no Referência geral da Amazon Web Services.

Valor padrão: `us-west-2`

Nome da opção para o modo silencioso: `--region`

Group name

O nome do grupo do Greengrass.

Valor padrão: `GreengrassDeviceSetup_Group_`*`guid`*

Nome da opção para o modo silencioso: `--group-name`

Nome do núcleo

O nome do núcleo do Greengrass. O núcleo é um AWS IoT dispositivo (coisa) que executa o software AWS IoT Greengrass Core. O núcleo é adicionado ao AWS IoT registro e ao grupo Greengrass. Se você fornecer um nome, ele deverá ser exclusivo no Conta da AWS Região da AWS e.

Valor padrão: `GreengrassDeviceSetup_Core_`*`guid`*

Nome da opção para o modo silencioso: `--core-name`

AWS IoT Greengrass Caminho de instalação do software principal

O local no sistema de arquivos do dispositivo em que você deseja instalar o software AWS IoT Greengrass Core.

Valor padrão: `/`

Nome da opção para o modo silencioso: `--ggc-root-path`

Função Hello World do Lambda

Indica se uma função Hello World do Lambda deve ser incluída no grupo do Greengrass. A função publica uma mensagem MQTT para o tópico `hello/world` a cada cinco segundos.

O script cria e publica essa função do Lambda definida pelo usuário AWS Lambda e a adiciona ao seu grupo do Greengrass. O script também cria uma assinatura no grupo que permite que a função envie mensagens MQTT para a AWS IoT.

> **Note**
>
> Ela é uma função Python 3.7 do Lambda. Se o Python 3.7 não estiver instalado no dispositivo e o script não conseguir instalá-lo, o script imprimirá uma mensagem de erro no terminal. Para incluir a função do Lambda no grupo, você deve instalar o Python 3.7 manualmente e reiniciar o script. Para criar o grupo do Greengrass sem a função do Lambda, reinicie o script e digite no quando solicitado a incluir a função.

Valor padrão: no

Nome da opção para o modo silencioso: `--hello-world-lambda` - Esta opção não assume um valor. Inclua-o no comando se quiser criar a função.

Tempo limite de implantação

O número de segundos antes da configuração do dispositivo do Greengrass parar de verificar o status da implantação do grupo do Greengrass. Ele é usado somente quando o grupo inclui a função Hello World do Lambda. Caso contrário, o grupo não será implantado.

O tempo de implantação depende da velocidade da rede. Para velocidades lentas de rede, você pode aumentar o valor.

Valor padrão: `180`

Nome da opção para o modo silencioso: `--deployment-timeout`

Caminho do log.

O local do arquivo de registros que contém as informações sobre as operações de configuração do grupo do Greengrass e do núcleo. Use esse log para solucionar problemas de implantação e outros problemas com o grupo do Greengrass e com a configuração principal.

Valor padrão: `./`

Nome da opção para o modo silencioso: `--log-path`

Verbosidade

Indica se as informações detalhadas de log devem ser impressas no terminal enquanto o script é executado. Você pode usar essas informações para solucionar problemas de configuração do dispositivo.

Valor padrão: no

Nome da opção para o modo silencioso: `--verbose` - Esta opção não assume um valor. Inclua-o no comando se quiser imprimir informações detalhadas de log.

Execute a configuração do dispositivo do Greengrass no modo silencioso

Você pode executar a configuração do dispositivo do Greengrass no modo silencioso para que o script não solicite nenhum valor. Para execução no modo silencioso, especifique o modo `bootstrap-greengrass` e os valores de entrada depois de iniciar o script. Você pode omitir valores de entrada se quiser usar seus valores padrão.

O procedimento depende de você fornecer suas Conta da AWS credenciais como variáveis de ambiente antes de iniciar o script ou como valores de entrada após iniciar o script.

Fornecer credenciais como variáveis de ambiente

1. Fornecer credenciais como variáveis de ambiente. O exemplo a seguir exporta credenciais temporárias, que incluem o token de sessão.

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

> **Note**
>
> Se você estiver executando a configuração do dispositivo Greengrass em um Raspbian ou OpenWrt plataforma, faça uma cópia desses comandos. Você deve fornecê-los novamente depois de reiniciar o dispositivo.

2. Faça download e inicie o script. Forneça valores de entrada conforme necessário. Por exemplo:

   - Como usar todos os valores padrão:

```
wget -q -O ./gg-device-setup-latest.sh https://d1onfpft10uf5o.cloudfront.net/
greengrass-device-setup/downloads/gg-device-setup-latest.sh && chmod +x ./
gg-device-setup-latest.sh && sudo -E ./gg-device-setup-latest.sh bootstrap-
greengrass
```

   - Como especificar valores personalizados:

```
wget -q -O ./gg-device-setup-latest.sh https://d1onfpft10uf5o.cloudfront.net/
greengrass-device-setup/downloads/gg-device-setup-latest.sh && chmod +x ./
gg-device-setup-latest.sh && sudo -E ./gg-device-setup-latest.sh bootstrap-
greengrass
--region us-east-1
--group-name Custom_Group_Name
--core-name Custom_Core_Name
--ggc-root-path /custom/ggc/root/path
--deployment-timeout 300
--log-path /customized/log/path
--hello-world-lambda
--verbose
```

> **Note**
>
> Para usar `curl` para fazer download do script, substitua `wget -q -O` por `curl` no comando.

3. Se o seu dispositivo principal estiver executando o Raspbian ou OpenWrt, reinicie o dispositivo quando solicitado, forneça suas credenciais e reinicie o script.

   a. Quando solicitado a reiniciar o dispositivo, execute um dos seguintes comandos.

      Em plataformas Raspbian:

      ```
      sudo reboot
      ```

      Para OpenWrt plataformas:

      ```
      reboot
      ```

   b. Depois que o dispositivo for reinicializado, abra o terminal e forneça suas credenciais como variáveis de ambiente.

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

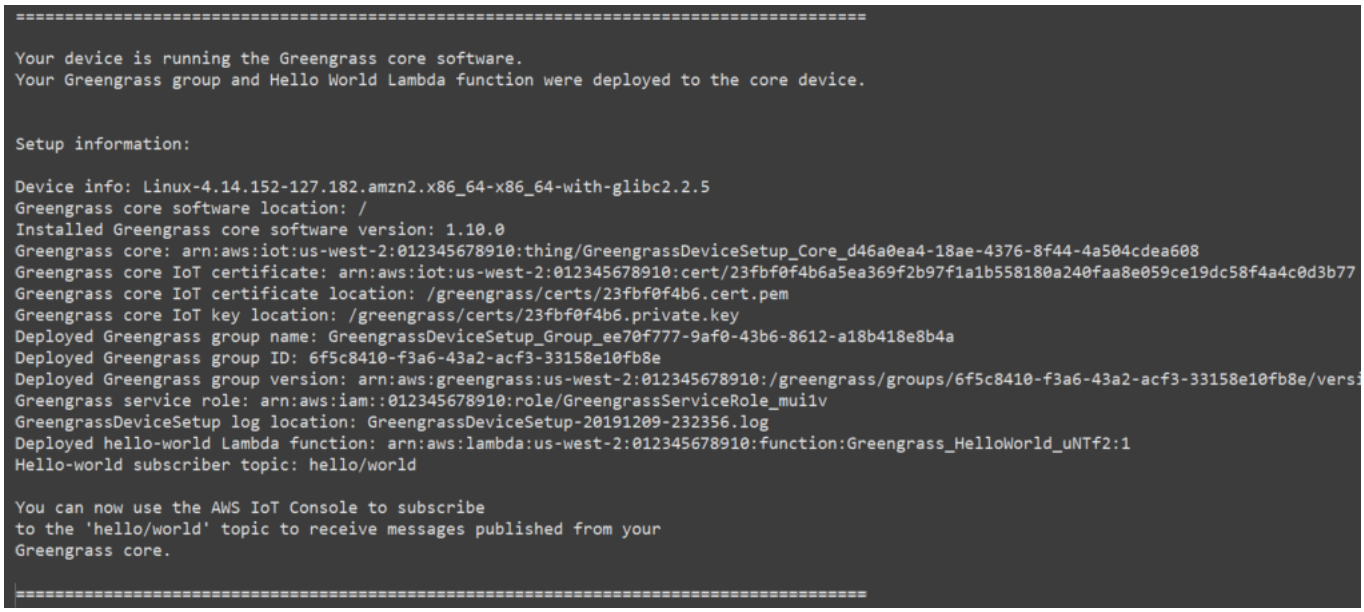
   c. Reinicie o script.

      ```
      sudo -E ./gg-device-setup-latest.sh bootstrap-greengrass
      ```

   d. Quando solicitado se deseja usar os valores de entrada da sessão anterior ou iniciar uma nova instalação, insira `yes` para reutilizar os valores de entrada.

> **Note**
>
> Em plataformas que exigem uma reinicialização, seus valores de entrada da sessão anterior, exceto as credenciais, são temporariamente armazenados no arquivo `GreengrassDeviceSetup.config.info`.

Quando a instalação estiver concluída, o terminal exibirá uma mensagem de status de êxito semelhante à seguinte.



4. Se você incluiu a função do Lambda Hello World, a configuração do dispositivo do Greengrass implantará o grupo do Greengrass no seu dispositivo de núcleo. Para testar a função do Lambda ou para obter informações sobre como remover a função do Lambda do grupo, prossiga para the section called “Verificar se a função do Lambda está em execução no dispositivo de núcleo” no Módulo 3-1 do tutorial Conceitos básicos.

> **Note**
>
> Certifique-se de que o Região da AWS selecionado no console seja o mesmo que você usou para configurar seu ambiente Greengrass. Por padrão, a região é Oeste dos EUA (Oregon).

Se você não incluiu a função do Lambda Hello World, você pode criar sua própria função do Lambda ou testar outros atributos do Greengrass. Por exemplo, você pode adicionar o conector de implantação do aplicativo Docker ao seu grupo e usá-lo para implantar contêineres do Docker no seu dispositivo de núcleo.

Fornecer credenciais como valores de entrada

1. Faça download e inicie o script. Fornecer credenciais e quaisquer outros valores de entrada que você deseje especificar. Os exemplos a seguir mostram como fornecer credenciais temporárias, que incluem o token de sessão.

   - Como usar todos os valores padrão:

   ```
   wget -q -O ./gg-device-setup-latest.sh https://d1onfpft10uf5o.cloudfront.net/
   greengrass-device-setup/downloads/gg-device-setup-latest.sh && chmod +x ./
   gg-device-setup-latest.sh && sudo -E ./gg-device-setup-latest.sh bootstrap-
   greengrass
   --aws-access-key-id AKIAIOSFODNN7EXAMPLE
   --aws-secret-access-key wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
   --aws-session-token AQoDYXdzEJr1K...o5OytwEXAMPLE=
   ```

   - Como especificar valores personalizados:

   ```
   wget -q -O ./gg-device-setup-latest.sh https://d1onfpft10uf5o.cloudfront.net/
   greengrass-device-setup/downloads/gg-device-setup-latest.sh && chmod +x ./
   gg-device-setup-latest.sh && sudo -E ./gg-device-setup-latest.sh bootstrap-
   greengrass
   --aws-access-key-id AKIAIOSFODNN7EXAMPLE
   --aws-secret-access-key wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
   --aws-session-token AQoDYXdzEJr1K...o5OytwEXAMPLE=
   --region us-east-1
   --group-name Custom_Group_Name
   --core-name Custom_Core_Name
   --ggc-root-path /custom/ggc/root/path
   --deployment-timeout 300
   --log-path /customized/log/path
   --hello-world-lambda
   ```

```
--verbose
```

> **Note**
>
> Se você estiver executando a configuração do dispositivo Greengrass em um Raspbian ou OpenWrt plataforma, faça uma cópia de suas credenciais. Você deve fornecê-los novamente depois de reiniciar o dispositivo.
> Para usar `curl` para fazer download do script, substitua `wget -q -O` por `curl` no comando.

2. Se o seu dispositivo principal estiver executando o Raspbian ou OpenWrt, reinicie o dispositivo quando solicitado, forneça suas credenciais e reinicie o script.

   a. Quando solicitado a reiniciar o dispositivo, execute um dos seguintes comandos.

      Em plataformas Raspbian:

      ```
      sudo reboot
      ```

      Para OpenWrt plataformas:

      ```
      reboot
      ```

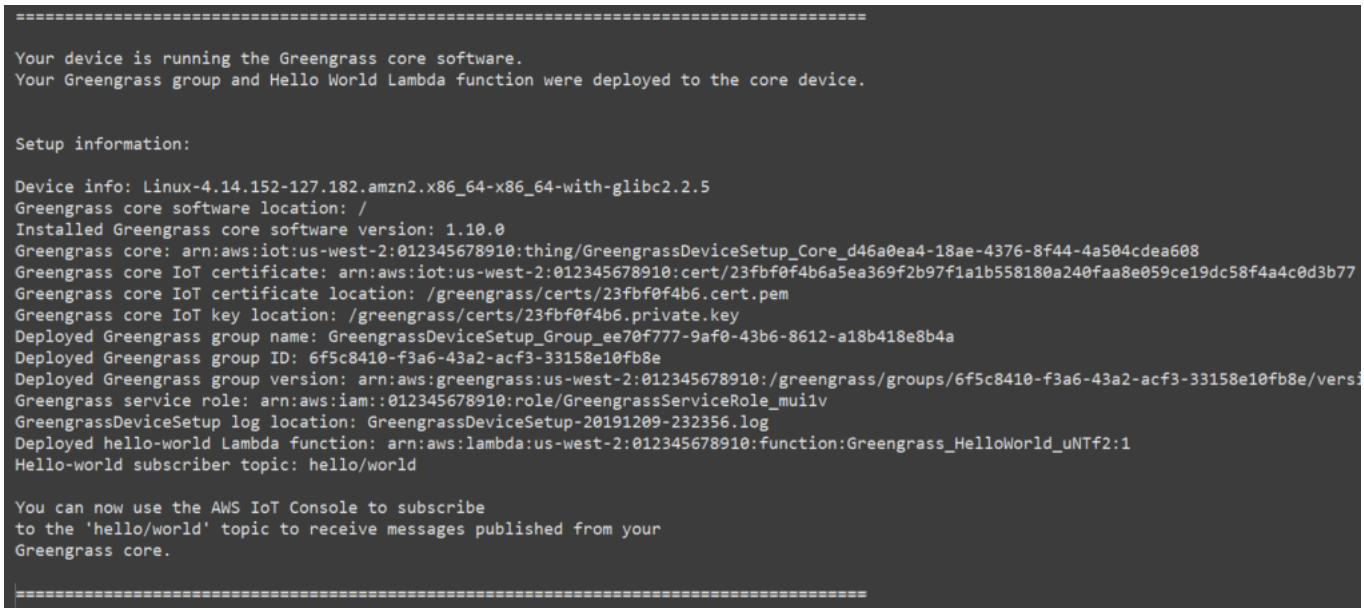
   b. Reinicie o script. Você deve incluir as credenciais no comando, mas não os outros valores de entrada. Por exemplo:

      ```
      sudo -E ./gg-device-setup-latest.sh bootstrap-greengrass
      --aws-access-key-id AKIAIOSFODNN7EXAMPLE
      --aws-secret-access-key wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
      --aws-session-token AQoDYXdzEJr1K...o5OytwEXAMPLE=
      ```

   c. Quando solicitado se deseja usar os valores de entrada da sessão anterior ou iniciar uma nova instalação, insira `yes` para reutilizar os valores de entrada.

> **Note**
>
> Em plataformas que exigem uma reinicialização, seus valores de entrada da sessão anterior, exceto as credenciais, são temporariamente armazenados no arquivo `GreengrassDeviceSetup.config.info`.

Quando a instalação estiver concluída, o terminal exibirá uma mensagem de status de êxito semelhante à seguinte.



3. Se você incluiu a função do Lambda Hello World, a configuração do dispositivo do Greengrass implantará o grupo do Greengrass no seu dispositivo de núcleo. Para testar a função do Lambda ou para obter informações sobre como remover a função do Lambda do grupo, prossiga para the section called “Verificar se a função do Lambda está em execução no dispositivo de núcleo” no Módulo 3-1 do tutorial Conceitos básicos.

> **Note**
>
> Certifique-se de que o Região da AWS selecionado no console seja o mesmo que você usou para configurar seu ambiente Greengrass. Por padrão, a região é Oeste dos EUA (Oregon).

Se você não incluiu a função do Lambda Hello World, você pode criar sua própria função do Lambda ou testar outros atributos do Greengrass. Por exemplo, você pode adicionar o conector de implantação do aplicativo Docker ao seu grupo e usá-lo para implantar contêineres do Docker no seu dispositivo de núcleo.

Módulo 1: Configuração do ambiente para o Greengrass

Este módulo mostra como preparar um Raspberry Pi, uma instância do Amazon EC2 ou outro dispositivo pronto para ser usado como seu dispositivo principal. AWS IoT Greengrass AWS IoT Greengrass

> **Tip**
>
> Ou, para usar um script que configure o dispositivo de núcleo para você, consulte the section called “Início rápido: Configuração do dispositivo do Greengrass”.

Este módulo deve demorar menos de 30 minutos para ser concluído.

Antes de começar, leia os requisitos para este tutorial. Depois, siga as instruções de configuração em um dos tópicos a seguir. Selecione apenas aquele que se aplica ao seu tipo de dispositivo de núcleo.

Tópicos

- Configurar um Raspberry Pi
- Configurando uma instância do Amazon EC2
- Configurar outros dispositivos

> **Note**
>
> Para saber como usar a AWS IoT Greengrass execução em um contêiner Docker pré-construído, consulte. the section called “Executar AWS IoT Greengrass em um contêiner Docker”

Configurar um Raspberry Pi

Siga as etapas deste tópico para configurar um Raspberry Pi para usar como AWS IoT Greengrass núcleo.

> **Tip**
>
> AWS IoT Greengrass também oferece outras opções para instalar o software AWS IoT Greengrass Core. Por exemplo, você pode usar a configuração do dispositivo Greengrass para configurar seu ambiente e instalar a versão mais recente do software AWS IoT Greengrass Core. Ou, em plataformas Debian suportadas, você pode usar o gerenciador de pacotes APT para instalar ou atualizar o software AWS IoT Greengrass Core. Para obter mais informações, consulte the section called “Instale o AWS IoT Greengrass Software principal”.

Se você estiver configurando um Raspberry Pi pela primeira vez, siga todas estas etapas. Caso contrário, pule para o passo 9. No entanto, recomendamos que você crie uma nova imagem do seu Raspberry Pi com o sistema operacional, conforme recomendado na Etapa 2.

1. Faça download e instale um formatador de cartão SD, como o Formatador de cartão de memória SD. Insira o cartão SD no seu computador. Inicie o programa e selecione a unidade em que você inseriu seu cartão SD. Você pode executar uma formatação rápida do cartão SD.
2. Faça download do sistema operacional Raspbian Buster como um arquivo `zip`.
3. Usando uma ferramenta de gravação de cartão SD (como Etcher), siga as instruções da ferramenta para instalar a imagem do arquivo `zip` obtido por download no cartão SD. Como a imagem do sistema operacional é grande, essa etapa deve levar algum tempo. Ejete seu cartão SD do computador e insira o cartão microSD no seu Raspberry Pi.
4. Durante a primeira inicialização, recomendamos que você conecte o Raspberry Pi a um monitor (por meio de HDMI), teclado e mouse. Em seguida, conecte seu Pi a uma fonte de energia micro USB e o sistema operacional Raspbian deverá ser iniciado.
5. Você pode configurar o layout de teclado do Pi antes de continuar. Para fazer isso, clique no ícone Raspberry no canto superior direito, selecione Preferences (Preferências) e, em seguida, Mouse and Keyboard Settings (Configurações de mouse e teclado). Depois, clique na guia

Keyboard (Teclado), Keyboard Layout (Layout de teclado) e, em seguida, selecione uma variante apropriada do teclado.

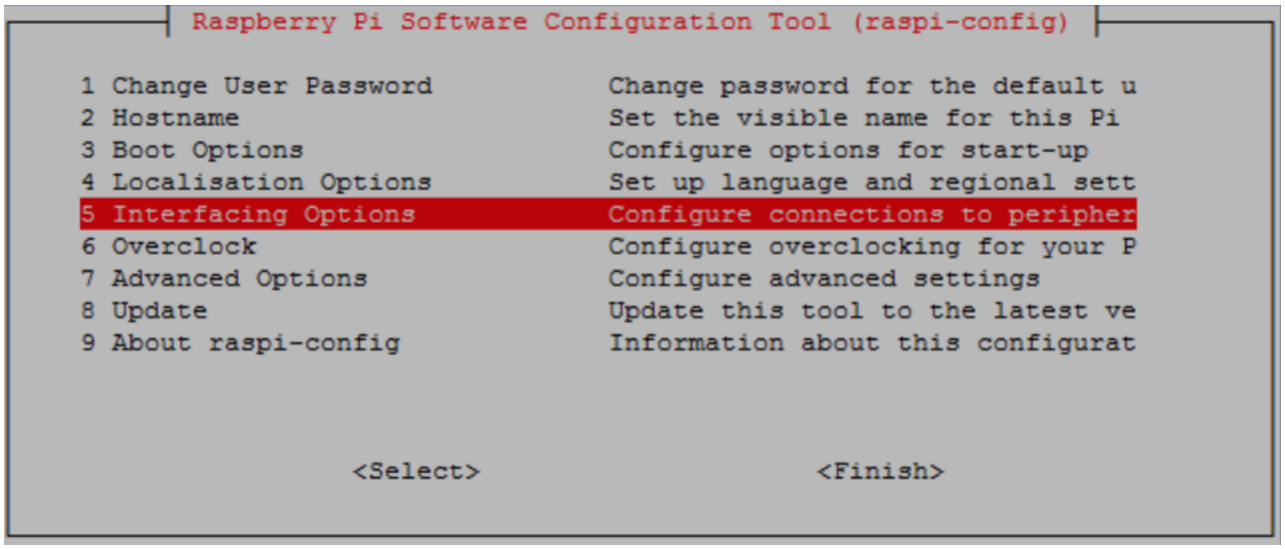
6. Em seguida, conecte seu Raspberry Pi à Internet por meio de uma Wi-Fi rede ou de um cabo Ethernet.

> **Note**
>
> Conecte seu Raspberry Pi à mesma rede que o seu computador está conectado, e certifique-se de que o computador e o Raspberry Pi tenham acesso à Internet antes de continuar. Se você estiver em um ambiente de trabalho ou atrás de um firewall, talvez seja necessário conectar o Pi e o computador à rede de convidado para colocar ambos os dispositivos na mesma rede. No entanto, essa abordagem pode desconectar o computador dos recursos da rede local, como sua intranet. Uma solução é conectar o Pi à Wi-Fi rede de convidados e conectar seu computador à rede de convidados e à sua Wi-Fi rede local por meio de um cabo Ethernet. Nessa configuração, seu computador deve ser capaz de se conectar ao Raspberry Pi por meio da Wi-Fi rede de convidados e aos recursos da rede local por meio do cabo Ethernet.

7. Você deve configurar o SSH no seu Pi para se conectar remotamente a ele. No Raspberry Pi, abra uma janela de terminal e execute o seguinte comando:

```
sudo raspi-config
```

Você deve ver o seguinte:



Role para baixo e selecione Interfacing Options e, em seguida, selecione P2 SSH. Quando solicitado, selecione Sim. (Use a tecla Tab seguida por Enter). Agora, o SSH deve estar habilitado. Selecione OK. Use a tecla Tab para escolher Finish (Concluir) e, em seguida, pressione Enter. Se o Raspberry Pi não reinicializar automaticamente, execute o seguinte comando:

```
sudo reboot
```

8. No Raspberry Pi, execute o seguinte comando no terminal:

```
hostname -I
```

Isso retorna o endereço IP do seu Raspberry Pi.

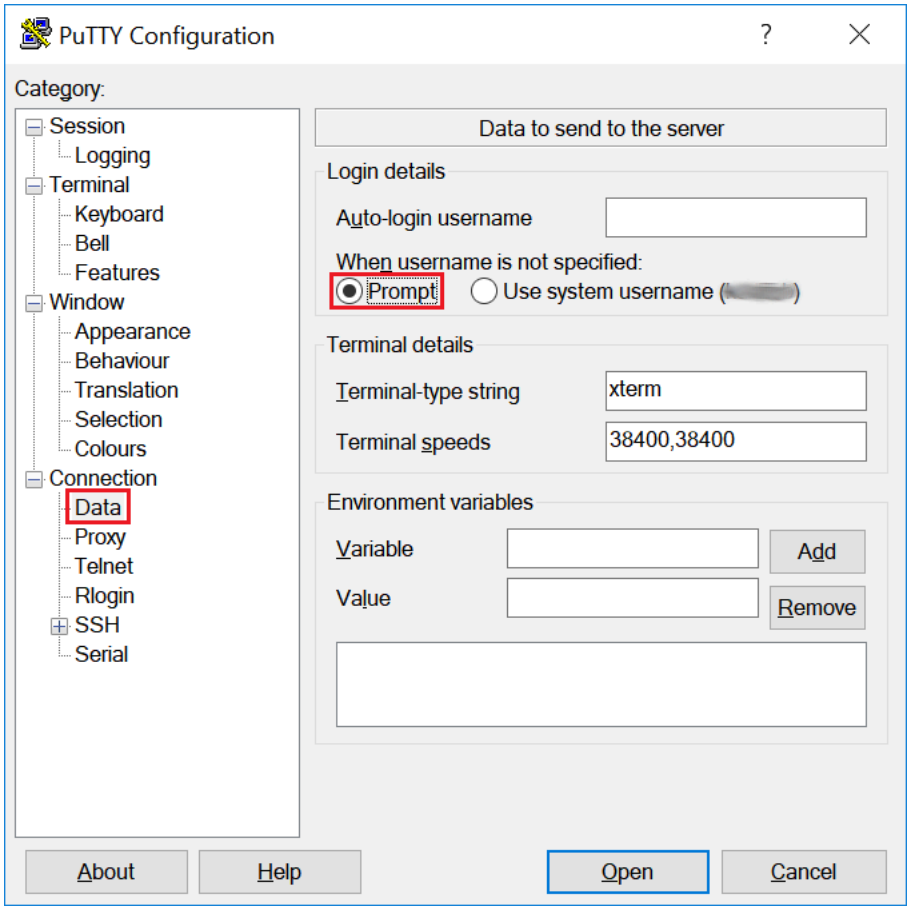
> **Note**
>
> Para o seguinte, se você receber uma mensagem relacionada à impressão digital da chave ECDSA (`Are you sure you want to continue connecting (yes/no)?`), insira `yes`. A senha padrão do Raspberry Pi é **`raspberry`**.

Se você estiver usando macOS, abra uma janela de terminal e insira o seguinte:

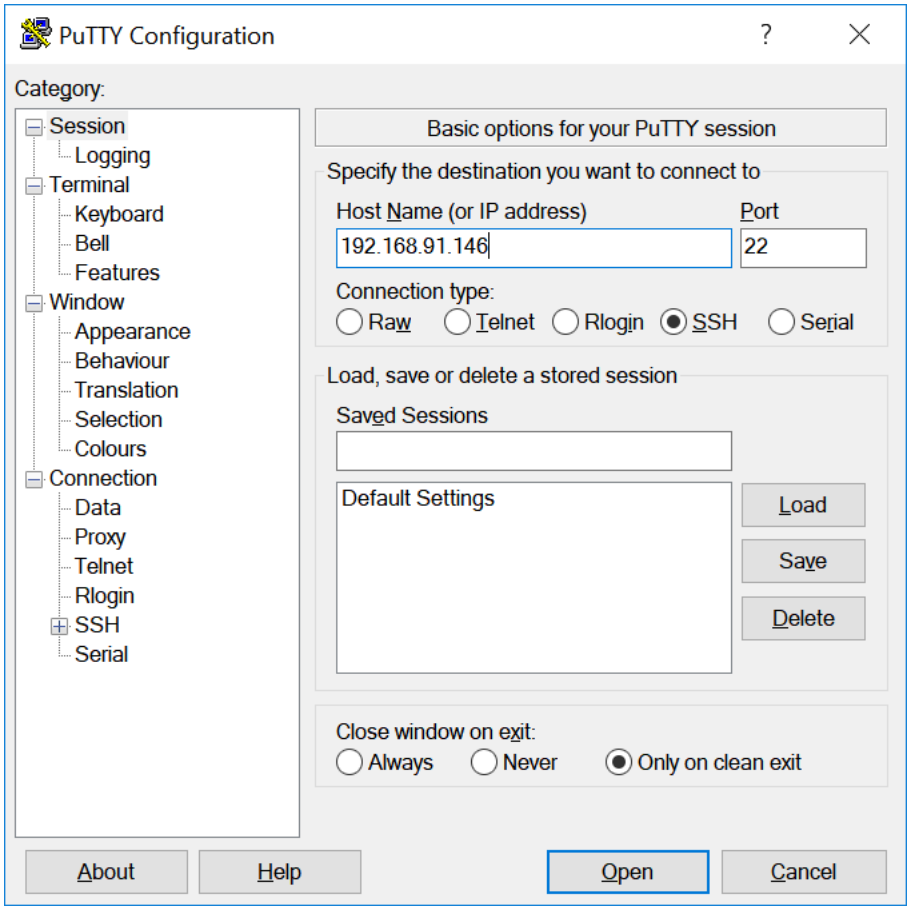
```
ssh pi@IP-address
```

*`IP-address`*é o endereço IP do seu Raspberry Pi que você obteve usando o `hostname -I` comando.

Se você estiver usando uma máquina com Windows, precisará instalar e configurar o PuTTY. Expanda Connection (Conexão), selecione Data (Dados) e certifique-se de que Prompt (Solicitar) esteja selecionado:

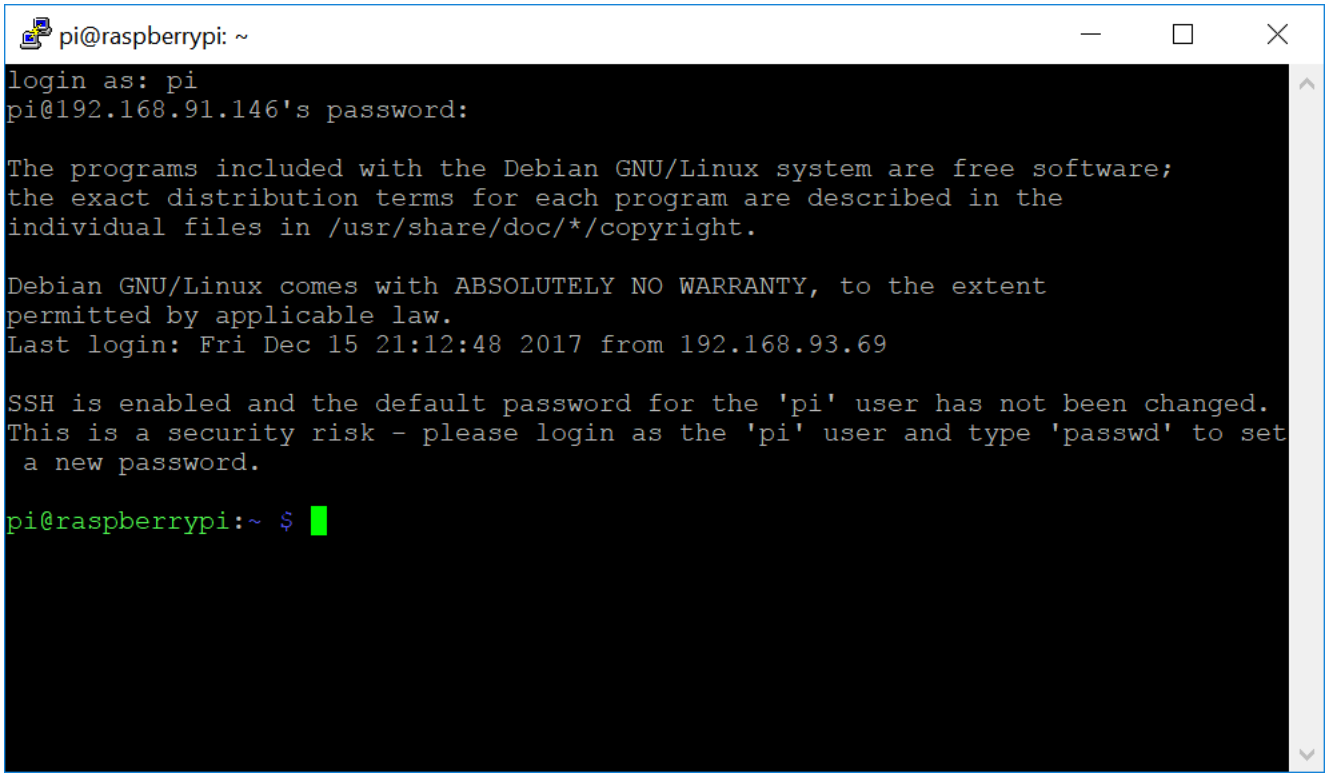


Em seguida, selecione Session (Sessão), insira o endereço IP do Raspberry Pi e selecione Open (Abrir) usando as configurações padrão.



Se um alerta de segurança do PuTTY for exibido, selecione Yes (Sim).

O login e senha padrão do Raspberry Pi são **`pi`** e **`raspberry`**, respectivamente.



> **Note**
>
> Se o seu computador estiver conectado a uma rede remota usando VPN , isso poderá dificultar a conexão do computador com o Raspberry Pi usando SSH.

9. Agora você está pronto para configurar o Raspberry Pi para AWS IoT Greengrass. Primeiro, execute os seguintes comandos a partir de uma janela de terminal local do Raspberry Pi ou uma janela de terminal SSH:

> **Tip**
>
> AWS IoT Greengrass também oferece outras opções para instalar o software AWS IoT Greengrass Core. Por exemplo, você pode usar a configuração do dispositivo Greengrass para configurar seu ambiente e instalar a versão mais recente do software AWS IoT Greengrass Core. Ou, em plataformas Debian suportadas, você pode usar o gerenciador de pacotes APT para instalar ou atualizar o software AWS IoT Greengrass Core. Para obter mais informações, consulte the section called “Instale o AWS IoT Greengrass Software principal”.

```
sudo adduser --system ggc_user
sudo addgroup --system ggc_group
```

10. Para aumentar a segurança no dispositivo Pi, habilite as proteções de hardlink e softlink (symlink) no sistema operacional durante a startup.

    a. Navegue até o arquivo `98-rpi.conf`.

    ```
    cd /etc/sysctl.d
    ls
    ```

    > **Note**
    >
    > Se você não encontrar o arquivo `98-rpi.conf`, siga as instruções no arquivo `README.sysctl`.

    b. Use um editor de texto (como Leafpad, GNU nano ou vi) para adicionar as duas linhas a seguir ao final do arquivo. Pode ser necessário usar o comando `sudo` para editar como raiz (por exemplo, `sudo nano 98-rpi.conf`).

    ```
    fs.protected_hardlinks = 1
    fs.protected_symlinks = 1
    ```

    c. Reinicialize o Pi.

    ```
    sudo reboot
    ```

    Depois de cerca de um minuto, conecte-se ao Pi usando SSH e então execute o comando a seguir para confirmar a alteração:

    ```
    sudo sysctl -a 2> /dev/null | grep fs.protected
    ```

    Você deve ver `fs.protected_hardlinks = 1` e `fs.protected_symlinks = 1`.

11. Edite o arquivos de inicialização da linha de comando para habilitar e montar cgroups de memória. Isso permite AWS IoT Greengrass definir o limite de memória para as funções do

Lambda. Também é necessário que os Cgroups sejam executados AWS IoT Greengrass no modo de conteinerização padrão.

a. Navegue até o diretório `boot`.

```
cd /boot/
```

b. Use um editor de texto para abrir `cmdline.txt`. Anexe o seguinte ao final da linha existente, e não como uma nova linha. Pode ser necessário usar o comando `sudo` para editar como raiz (por exemplo, `sudo nano cmdline.txt`).

```
cgroup_enable=memory cgroup_memory=1
```

c. Agora, reinicialize o Pi.

```
sudo reboot
```

Agora, seu Raspberry Pi deve estar pronto para o AWS IoT Greengrass.

12. Opcional. Instale o tempo de execução do Java 8, que é exigido pelo gerenciador de fluxo. Este tutorial não usa o gerenciador de fluxo, mas usa o fluxo de trabalho de Criação de grupo padrão que habilita o gerenciador de fluxo por padrão. Use os comandos a seguir para instalar o módulo de tempo de execução do Java 8 no dispositivo de núcleo ou desabilite o gerenciador de fluxo antes de implantar seu grupo. As instruções para desabilitar o gerenciador de fluxo são fornecidas no Módulo 3.

```
sudo apt install openjdk-8-jdk
```

13. Para garantir que você tenha todas as dependências necessárias, baixe e execute o verificador de dependências do Greengrass a partir do AWS IoT Greengrass repositório Samples em. GitHub Esses comandos descompactam e executam o script verificador de dependências no diretório `Downloads`.

> **Note**
>
> O verificador de dependências pode falhar se você estiver executando a versão 5.4.51 do kernel Raspbian. Essa versão não monta cgroups de memória corretamente. Isso pode fazer com que as funções do Lambda em execução no modo contêiner falhem.

Para obter mais informações sobre como atualizar seu kernel, consulte os Cgroups não carregados após a atualização do kernel nos fóruns do Raspberry Pi.

```
cd /home/pi/Downloads
mkdir greengrass-dependency-checker-GGCv1.11.x
cd greengrass-dependency-checker-GGCv1.11.x
wget https://github.com/aws-samples/aws-greengrass-samples/raw/master/greengrass-
dependency-checker-GGCv1.11.x.zip
unzip greengrass-dependency-checker-GGCv1.11.x.zip
cd greengrass-dependency-checker-GGCv1.11.x
sudo modprobe configs
sudo ./check_ggc_dependencies | more
```

Se `more` for exibido, pressione a tecla Spacebar para exibir outra tela de texto.

Important

Este tutorial requer o Python 3.7 Runtime para executar funções locais do Lambda. Quando o gerenciador de fluxo está habilitado, ele também requer o Java 8 Runtime. Se o script `check_ggc_dependencies` gerar avisos sobre esses pré-requisitos de tempo de execução ausentes, certifique-se de instalá-los antes de continuar. Você pode ignorar os avisos sobre outros pré-requisitos de tempo de execução opcionais ausentes.

Para obter informações sobre o comando modprobe, execute man modprobe no terminal.

A configuração do Raspberry Pi está concluída. Avance para the section called “Módulo 2: Instalando o AWS IoT Greengrass Software principal”.

Configurando uma instância do Amazon EC2

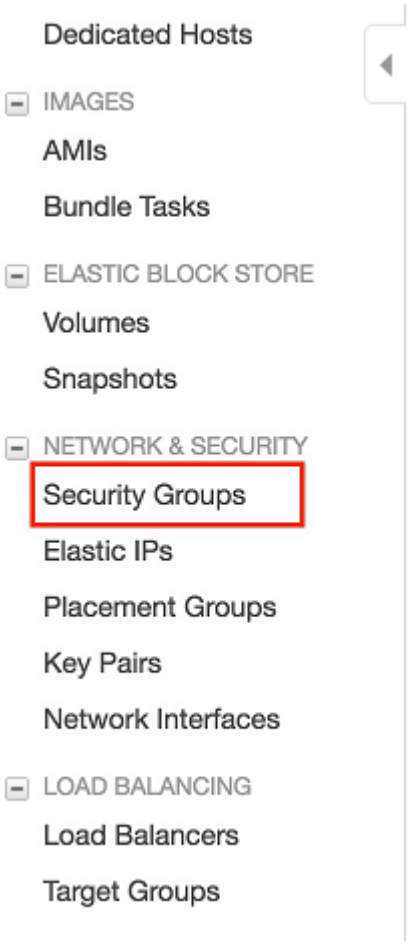
Siga as etapas neste tópico para configurar uma instância do Amazon EC2 a ser usada como núcleo do AWS IoT Greengrass .

> **Tip**
>
> Ou, para usar um script que configura seu ambiente e instala o software AWS IoT Greengrass Core para você, consultethe section called “Início rápido: Configuração do dispositivo do Greengrass”.

Embora você possa concluir este tutorial usando uma instância do Amazon EC2, AWS IoT Greengrass o ideal é que seja usado com hardware físico. Sempre que possível, recomendamos configurar um Raspberry Pi em vez de usar uma instância do Amazon EC2. Se estiver usando um Raspberry Pi, você não precisará seguir as etapas deste tópico.

1. Faça login no Console de gerenciamento da AWS e execute uma instância do Amazon EC2 usando uma AMI do Amazon Linux. Para obter informações sobre as instâncias do Amazon EC2, consulte o Guia de conceitos básicos do Amazon EC2.
2. Depois que sua instância do Amazon EC2 estiver em execução, habilite a porta 8883 para permitir a entrada de comunicações MQTT para que outros dispositivos possam se conectar ao núcleo. AWS IoT Greengrass

   a. No painel de navegação da Amazon EC2, selecione Grupos de segurança.



b. Selecione o grupo de segurança da instância que você acabou de executar e, em seguida, selecione a guia Regras de entrada.

c. Selecione Edit inbound rules (Editar regras de entrada).

   Para habilitar a porta 8883, você adiciona uma regra de TCP personalizada ao grupo de segurança. Para ter mais informações, consulte Adding rules to a security group no Manual do usuário do EC2.

d. Na página Edita regras de entrada, selecione Adicionar regra, insira as configurações a seguir e, depois, selecione Salvar.

   - Para Tipo, selecione Regra TCP personalizada.
   - Em Intervalo de portas, insira **`8883`**.
   - Para Source (Origem), selecione Anywhere (Qualquer lugar).
   - Em Descrição, insira **`MQTT Communications`**.

3. Conecte-se à sua instância Amazon EC2.

   a. No painel de navegação, selecione Instances (Instâncias), selecione sua instância e, em seguida, selecione Connect (Conectar).

   b. Siga as instruções na página Connect To Your Instance (Conectar à sua instância) para conectar-se à instância usando SSH e o arquivo da chave privada.

   Você pode usar PuTTY para Windows ou Terminal para macOS. Para ter mais informações, consulte Conectar-se à instância do Linux usando SSH no Manual do usuário do Amazon EC2.

   Agora você está pronto para configurar sua instância do Amazon EC2 para o AWS IoT Greengrass.

4. Depois de se conectar à sua instância do Amazon EC2, crie as contas `ggc_user` e `ggc_group`:

```
sudo adduser --system ggc_user
sudo groupadd --system ggc_group
```

   > **Note**
   >
   > Se o comando `adduser` não estiver disponível no sistema, use o comando a seguir.
   >
   > ```
   > sudo useradd --system ggc_user
   > ```

5. Para aumentar a segurança, certifique-se de que as proteções de hardlink e softlink (symlink) estão habilitadas no sistema operacional da startup da instância do Amazon EC2.

   > **Note**
   >
   > As etapas para habilitar as proteções de hardlink e softlink variam de acordo com o sistema operacional. Consulte a documentação para obter sua distribuição.

   a. Execute o seguinte comando para verificar se as proteções de hardlink e softlink estão habilitadas:

```
sudo sysctl -a | grep fs.protected
```

Se hardlinks e softlinks estão definidos como 1, suas proteções estão habilitadas corretamente. Prossiga para a etapa 6.

> **Note**
>
> Os softlinks são representados por `fs.protected_symlinks`.

b. Se hardlinks e softlinks não estão definidos como 1, habilite essas proteções. Navegue até o arquivo de configuração do sistema.

```
cd /etc/sysctl.d
ls
```

c. Usando o editor de texto de sua preferência (Leafpad, GNU nano ou vi), adicione as duas linhas a seguir ao final do arquivo de configuração do sistema. No Amazon Linux 1, esse é o arquivo `00-defaults.conf`. No Amazon Linux 2, esse é o arquivo `99-amazon.conf`. Talvez seja necessário alterar as permissões (usando o comando `chmod`) para gravar no arquivo ou usar o comando `sudo` para editar como raiz (por exemplo, `sudo nano 00-defaults.conf`).

```
fs.protected_hardlinks = 1
fs.protected_symlinks = 1
```

d. Reinicialize a instância do Amazon EC2.

```
sudo reboot
```

Depois de alguns minutos, conecte-se à sua instância usando SSH e então execute o comando a seguir para confirmar a alteração.

```
sudo sysctl -a | grep fs.protected
```

Os hardlinks e softlinks estão definidos como 1.

6. Extraia e execute o script a seguir para montar Grupos de controle do Linux (cgroups). Isso permite AWS IoT Greengrass definir o limite de memória para as funções do Lambda.

Também é necessário que os Cgroups sejam executados AWS IoT Greengrass no modo de conteinerização padrão.

```
curl https://raw.githubusercontent.com/tianon/cgroupfs-
mount/951c38ee8d802330454bdede20d85ec1c0f8d312/cgroupfs-mount > cgroupfs-mount.sh
chmod +x cgroupfs-mount.sh
sudo bash ./cgroupfs-mount.sh
```

Agora, sua instância do Amazon EC2 deve estar pronta para o AWS IoT Greengrass.

7. Opcional. Instale o tempo de execução do Java 8, que é exigido pelo gerenciador de fluxo. Este tutorial não usa o gerenciador de fluxo, mas usa o fluxo de trabalho de Criação de grupo padrão que habilita o gerenciador de fluxo por padrão. Use os comandos a seguir para instalar o módulo de tempo de execução do Java 8 no dispositivo de núcleo ou desabilite o gerenciador de fluxo antes de implantar seu grupo. As instruções para desabilitar o gerenciador de fluxo são fornecidas no Módulo 3.

   - Para Debian-based distribuições:

     ```
     sudo apt install openjdk-8-jdk
     ```

   - Para Hat-based distribuições vermelhas:

     ```
     sudo yum install java-1.8.0-openjdk
     ```

8. Para garantir que você tenha todas as dependências necessárias, baixe e execute o verificador de dependências do Greengrass a partir do AWS IoT Greengrass repositório Samples. GitHub Esses comandos fazem download, descompactam e executam o script verificador de dependências na sua instância do Amazon EC2.

   ```
   mkdir greengrass-dependency-checker-GGCv1.11.x
   cd greengrass-dependency-checker-GGCv1.11.x
   wget https://github.com/aws-samples/aws-greengrass-samples/raw/master/greengrass-
   dependency-checker-GGCv1.11.x.zip
   unzip greengrass-dependency-checker-GGCv1.11.x.zip
   cd greengrass-dependency-checker-GGCv1.11.x
   sudo ./check_ggc_dependencies | more
   ```

> **Important**
>
> Este tutorial requer o Python 3.7 Runtime para executar funções locais do Lambda. Quando o gerenciador de fluxo está habilitado, ele também requer o Java 8 Runtime. Se o script `check_ggc_dependencies` gerar avisos sobre esses pré-requisitos de tempo de execução ausentes, certifique-se de instalá-los antes de continuar. Você pode ignorar os avisos sobre outros pré-requisitos de tempo de execução opcionais ausentes.

A configuração da instância doAmazon EC2 está completa. Avance para the section called “Módulo 2: Instalando o AWS IoT Greengrass Software principal”.

Configurar outros dispositivos

Siga as etapas deste tópico para configurar um dispositivo (que não seja um Raspberry Pi) para usar como seu AWS IoT Greengrass núcleo.

> **Tip**
>
> Ou, para usar um script que configura seu ambiente e instala o software AWS IoT Greengrass principal para você, consultethe section called “Início rápido: Configuração do dispositivo do Greengrass”.

Se você é novato AWS IoT Greengrass, recomendamos que você use uma instância do Raspberry Pi ou do Amazon EC2 como seu dispositivo principal e siga as etapas de configuração apropriadas para o seu dispositivo.

Se você planeja criar um Linux-based sistema personalizado usando o Projeto Yocto, você pode usar a Receita AWS IoT Greengrass Bitbake do projeto. `meta-aws` Essa receita também ajuda você a desenvolver uma plataforma de software que ofereça suporte a software de AWS ponta para aplicativos incorporados. A compilação do Bitbake instala, configura e executa automaticamente o software AWS IoT Greengrass Core em seu dispositivo.

Projeto Yocto

Um projeto de colaboração de código aberto que ajuda você a criar Linux-based sistemas personalizados para aplicativos incorporados, independentemente da arquitetura de hardware. Para obter mais informações, consulte o Projeto Yocto.

`meta-aws`

Um projeto AWS gerenciado que fornece receitas de Yocto. Você pode usar as receitas para desenvolver software de AWS ponta em Linux-based sistemas construídos com OpenEmbeddedo Projeto Yocto. Para obter mais informações sobre esse recurso suportado pela comunidade, consulte o `meta-aws`projeto em GitHub.

`meta-aws-demos`

Um projeto AWS gerenciado que contém demonstrações do `meta-aws` projeto. Para obter mais exemplos sobre o processo de integração, consulte o `meta-aws-demos`projeto em GitHub.

Para usar outro dispositivo ou plataforma compatível, siga as etapas deste tópico.

1. Se o seu dispositivo principal for um dispositivo NVIDIA Jetson, você deve primeiro atualizar o firmware com o instalador JetPack 4.3. Se estiver configurando um dispositivo diferente, vá para a etapa 2.

   > **Note**
   >
   > A versão JetPack do instalador que você usa é baseada na versão de destino do CUDA Toolkit. As instruções a seguir usam JetPack 4.3 e o CUDA Toolkit 10.0. Para obter informações sobre como usar as versões apropriadas para o seu dispositivo, consulte How to Install Jetpack na documentação do NVIDIA.

   a. Em um desktop físico que esteja executando o Ubuntu 16.04 ou posterior, atualize o firmware com o instalador JetPack 4.3, conforme descrito em Download e instalação JetPack (4.3) na documentação da NVIDIA.

      Siga as instruções no instalador para instalar todos os pacotes e dependências na placa Jetson, que devem ser conectados ao desktop com um Micro-B cabo.

   b. Reinicialize sua placa no modo normal e conecte um monitor à placa.

> **Note**
>
> Quando você usar o SSH para se conectar à placa Jetson, use o nome de usuário padrão (**`nvidia`**) e a senha padrão (**`nvidia`**).

2. Execute um dos comandos a seguir para criar um usuário `ggc_user` e grupo `ggc_group`. Os comandos que você executa são diferentes, dependendo da distribuição instalada no dispositivo de núcleo.

   - Se o seu dispositivo principal estiver em execução OpenWrt, execute os seguintes comandos:

   ```
   opkg install shadow-useradd
   opkg install shadow-groupadd
   useradd --system ggc_user
   groupadd --system ggc_group
   ```

   - Caso contrário, execute os seguintes comandos:

   ```
   sudo adduser --system ggc_user
   sudo addgroup --system ggc_group
   ```

   > **Note**
   >
   > Se o comando `addgroup` não estiver disponível no sistema, use o comando a seguir.
   >
   > ```
   > sudo groupadd --system ggc_group
   > ```

3. Opcional. Instale o tempo de execução do Java 8, que é exigido pelo gerenciador de fluxo. Este tutorial não usa o gerenciador de fluxo, mas usa o fluxo de trabalho de Criação de grupo padrão que habilita o gerenciador de fluxo por padrão. Use os comandos a seguir para instalar o módulo de tempo de execução do Java 8 no dispositivo de núcleo ou desabilite o gerenciador de fluxo antes de implantar seu grupo. As instruções para desabilitar o gerenciador de fluxo são fornecidas no Módulo 3.

   - Para Debian-based nossas Ubuntu-based distribuições:

   ```
   sudo apt install openjdk-8-jdk
   ```

- Para Hat-based distribuições vermelhas:

```
sudo yum install java-1.8.0-openjdk
```

4. Para garantir que você tenha todas as dependências necessárias, baixe e execute o verificador de dependências do Greengrass a partir do AWS IoT Greengrass repositório Samples em. GitHub Esses comandos descompactam e executam o script verificador de dependências.

```
mkdir greengrass-dependency-checker-GGCv1.11.x
cd greengrass-dependency-checker-GGCv1.11.x
wget https://github.com/aws-samples/aws-greengrass-samples/raw/master/greengrass-
dependency-checker-GGCv1.11.x.zip
unzip greengrass-dependency-checker-GGCv1.11.x.zip
cd greengrass-dependency-checker-GGCv1.11.x
sudo ./check_ggc_dependencies | more
```

> **Note**
>
> O `check_ggc_dependencies` script é executado em plataformas AWS IoT Greengrass suportadas e requer comandos específicos do sistema Linux. Para obter mais informações, consulte o Readme do verificador de dependências.

5. Instale todas as dependências necessárias no seu dispositivo, conforme indicado pelo resultado do verificador de dependências. Para as dependências ausentes no nível do kernel, pode ser necessário recompilar o kernel. Para montar grupos de controle do Linux (`cgroups`), você pode executar o script cgroupfs-mount. Isso permite AWS IoT Greengrass definir o limite de memória para funções Lambda. Também é necessário que os Cgroups sejam executados AWS IoT Greengrass no modo de contêinerização padrão.

   Se nenhum erro aparecer na saída, AWS IoT Greengrass deve ser capaz de ser executado com sucesso em seu dispositivo.

> **Important**
>
> Este tutorial requer o Python 3.7 Runtime para executar funções locais do Lambda. Quando o gerenciador de fluxo está habilitado, ele também requer o Java 8 Runtime. Se o script `check_ggc_dependencies` gerar avisos sobre esses pré-requisitos de tempo

de execução ausentes, certifique-se de instalá-los antes de continuar. Você pode ignorar os avisos sobre outros pré-requisitos de tempo de execução opcionais ausentes.

Para obter a lista de AWS IoT Greengrass requisitos e dependências, consultethe section called "Plataformas compatíveis e requisitos".

Módulo 2: Instalando o AWS IoT Greengrass Software principal

Este módulo mostra como instalar o software AWS IoT Greengrass Core no dispositivo escolhido. Neste módulo, você primeiramente cria um grupo e um núcleo do Greengrass. Depois, você faz download, configura e inicia o software no dispositivo de núcleo. Para obter mais informações sobre a funcionalidade do software AWS IoT Greengrass Core, consulte the section called "Configurar o AWS IoT Greengrass núcleo".

Antes de começar, certifique-se de ter concluído as etapas de configuração no Módulo 1 para o dispositivo escolhido.

> **Tip**
>
> AWS IoT Greengrass também fornece outras opções para instalar o software AWS IoT Greengrass Core. Por exemplo, você pode usar a configuração do dispositivo Greengrass para configurar seu ambiente e instalar a versão mais recente do software AWS IoT Greengrass Core. Ou, em plataformas Debian suportadas, você pode usar o gerenciador de pacotes APT para instalar ou atualizar o software AWS IoT Greengrass Core. Para obter mais informações, consulte the section called "Instale o AWS IoT Greengrass Software principal".

Este módulo deve demorar menos de 30 minutos para ser concluído.

Tópicos

- Provisão e AWS IoT coisa para usar como núcleo do Greengrass
- Crie um AWS IoT Greengrass grupo para o núcleo
- Instale e execute AWS IoT Greengrass no dispositivo principal

Provisão e AWS IoT coisa para usar como núcleo do Greengrass

Os núcleos do Greengrass são dispositivos que executam o software AWS IoT Greengrass Core para gerenciar processos locais de IoT. Para configurar um núcleo do Greengrass, você cria uma AWS IoT coisa que representa um dispositivo ou entidade lógica que se conecta a AWS IoT. Quando você registra um dispositivo como uma AWS IoT coisa, esse dispositivo pode usar um certificado digital e chaves que permitem seu acesso AWS IoT. Você usa uma AWS IoT política para permitir que o dispositivo se comunique com os AWS IoT Greengrass serviços AWS IoT e.

Nesta seção, você registra seu dispositivo como AWS IoT algo para usá-lo como núcleo do Greengrass.

Para criar um AWS IoT objeto

1. Navegue até o console do AWS IoT.
2. Em Gerenciar, expanda Todos os dispositivos e, em seguida selecione Coisas.
3. Na página Coisas, selecione Criar coisas.
4. Na página Criar coisas, selecione Criar uma única coisa, em seguida, selecione Avançar.
5. Na página Especificar propriedades da coisa, faça o seguinte:

   a. Em Nome da coisa, insira um nome que represente seu dispositivo, como **`MyGreengrassV1Core`**.

   b. Selecione Next (Próximo).
6. Na página Configurar certificado do dispositivo, selecione Avançar.
7. Na página Anexar políticas ao certificado, execute uma das seguintes ações:
   - Selecione uma política existente que conceda as permissões exigidas pelos núcleos e, em seguida selecione Criar coisa.

     Um modal é aberto, onde você pode baixar os certificados e as chaves que o dispositivo usa para se conectar ao Nuvem AWS.
   - Crie e anexe uma nova política que conceda permissões ao dispositivo de núcleo. Faça o seguinte:

     a. Selecione Create policy (Criar política).

        A página Create policy (Criar política) é aberta em uma nova guia.

     b. Na página Create policy (Criar política) faça o seguinte:

i. Em Nome da política, insira um nome que descreva a política, como **`GreengrassV1CorePolicy`**.

ii. Na guia Instruções da política, em Documento da política, selecione JSON.

iii. Insira o seguinte documento de política. Essa política permite que o núcleo se comunique com o AWS IoT Core serviço, interaja com as sombras do dispositivo e se comunique com o AWS IoT Greengrass serviço. Para obter informações sobre como restringir o acesso a essa política com base no seu caso de uso, consulte Mínimo AWS IoT política para o AWS IoT Greengrass dispositivo principal.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish",
        "iot:Subscribe",
        "iot:Connect",
        "iot:Receive"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "iot:GetThingShadow",
        "iot:UpdateThingShadow",
        "iot:DeleteThingShadow"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
```

```
        "greengrass:*"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

iv. Selecione Create (Criar) para criar a política.

c. Volte para a guia do navegador com a página Anexar políticas ao certificado aberta. Faça o seguinte:

i. Na lista Políticas, selecione a política que você criou, como GreengrassV1CorePolicy.

Se a política não for exibida, selecione o botão de atualização.

ii. Selecione Criar coisa.

Um modal é aberto, onde você pode baixar os certificados e as chaves que o núcleo usa para se conectar ao AWS IoT.

8. Volte para a guia do navegador com a página Anexar políticas ao certificado aberta. Faça o seguinte:

a. Na lista Políticas, selecione a política que você criou, como GreengrassV1CorePolicy.

Se a política não for exibida, selecione o botão de atualização.

b. Selecione Criar coisa.

Um modal é aberto, onde você pode baixar os certificados e as chaves que o núcleo usa para se conectar ao AWS IoT.

9. No modal Baixar certificados e chaves, baixe os certificados do dispositivo.

> **Important**
>
> Antes de selecionar Concluído, faça download dos recursos de segurança.

Faça o seguinte:

a. Em Certificado do dispositivo, selecione Download para baixar o certificado do dispositivo.

b. Em Arquivo de chave pública, selecione Download para baixar a chave pública do certificado.

c. Em Arquivo de chave privada, selecione Download para baixar o arquivo de chave privada para o certificado.

d. Revise Autenticação do servidor no Guia do desenvolvedor da AWS IoT e, em seguida selecione o certificado de CA raiz apropriado. Recomendamos que você use endpoints do Amazon Trust Services (ATS) e certificados raiz da CA do ATS. Em Certificados CA raiz, selecione Download para obter um certificado de CA raiz.

e. Selecione Done (Concluído).

Anote a ID do certificado, que é comum nos nomes dos arquivos do certificado e das chaves do dispositivo. Você precisará disso mais tarde.

Crie um AWS IoT Greengrass grupo para o núcleo

AWS IoT Greengrass os grupos contêm configurações e outras informações sobre seus componentes, como dispositivos cliente, funções Lambda e conectores. Um grupo define a configuração de um núcleo, incluindo como seus componentes podem interagir uns com os outros.

Nesta seção, você cria um grupo para o seu núcleo.

> **Tip**
>
> Para ver um exemplo que usa a AWS IoT Greengrass API para criar e implantar um grupo, consulte o repositório gg_group_setup em. GitHub

Para criar um grupo para o núcleo

1. Navegue até o console do AWS IoT.
2. Em Gerenciar, expanda Dispositivos Greengrass e selecione Grupos (V1).

> **Note**
>
> Se você não vê o menu de dispositivos do Greengrass, mude para um Região da AWS que suporte. AWS IoT Greengrass V1 Para obter a lista de regiões compatíveis, consulte Endpoints e cotas do AWS IoT Greengrass V1 no Referência geral da AWS. Você deve criar AWS IoT algo para seu núcleo em uma região onde AWS IoT Greengrass V1 esteja disponível.

3. Na página Grupos do Greengrass, selecione Criar grupo.
4. Na página Criar grupo do Greengrass faça o seguinte:

   a. Em Nome do grupo do Greengrass, insira um nome que descreva o grupo, como **`MyGreengrassGroup`**.

   b. Para o Greengrass Core, escolha o AWS IoT que você criou anteriormente, como. MyGreengrassV1Core

      O console seleciona automaticamente o certificado de dispositivo da coisa para você.

   c. Selecione Criar grupo.

Instale e execute AWS IoT Greengrass no dispositivo principal

> **Note**
>
> Este tutorial fornece instruções para você executar o software AWS IoT Greengrass Core em um Raspberry Pi, mas você pode usar qualquer dispositivo compatível.

Nesta seção, você configura, instala e executa o software AWS IoT Greengrass Core em seu dispositivo principal.

Para instalar e executar AWS IoT Greengrass

1. Na seção Software AWS IoT Greengrass principal deste guia, baixe o pacote de instalação do software AWS IoT Greengrass Core. Selecione o pacote que melhor se adapta à arquitetura da CPU, à distribuição e ao sistema operacional do dispositivo de núcleo.

   - Para o Raspberry Pi, baixe o pacote para a arquitetura ARMv7l e o sistema operacional Linux.

- Para uma instância do Amazon EC2, faça download do pacote para a arquitetura x86_64 e o sistema operacional Linux.
- Para NVIDIA Jetson TX2, faça download do pacote para a arquitetura Armv8 (AArch64) e o sistema operacional Linux.
- Para o Intel Atom, faça download do pacote para a arquitetura x86_64 e o sistema operacional Linux.

2. Em etapas anteriores, você obteve cinco arquivos por download em seu computador:

   - `greengrass-`*`OS-architecture`*`-1.11.6.tar.gz`— Esse arquivo compactado contém o software AWS IoT Greengrass principal que é executado no dispositivo principal.
   - *`certificateId`*`-certificate.pem.crt` – o arquivo de certificado do dispositivo.
   - *`certificateId`*`-public.pem.key` – o arquivo de chave pública do certificado do dispositivo.
   - *`certificateId`*`-private.pem.key` – o arquivo de chave privada do certificado do dispositivo.
   - `AmazonRootCA1.pem` – o arquivo da autoridade de certificação (CA) raiz da Amazon.

   Nesta etapa, você transfere esses arquivos do computador para o dispositivo principal. Faça o seguinte:

   a. Se você não sabe qual é o endereço IP do dispositivo de núcleo do Greengrass, abra um terminal no dispositivo de núcleo e execute o seguinte comando.

      > **Note**
      >
      > Pode ser que este comando não retorne o endereço IP correto para alguns dispositivos. Consulte a documentação do seu dispositivo para recuperar o endereço IP dele.

      ```
      hostname -I
      ```

   b. Transfira esses arquivos do computador para o dispositivo de núcleo. As etapas de transferência de arquivos variam em função do sistema operacional do computador. Selecione o seu sistema operacional para as etapas que mostram como transferir arquivos para o dispositivo Raspberry Pi.

> **Note**
>
> Para um Raspberry Pi, o nome de usuário padrão é **`pi`** e a senha padrão é **`raspberry`**.
> Para um NVIDIA Jetson TX2, o nome de usuário padrão é **`nvidia`** e a senha padrão é **`nvidia`**.

Windows

Para transferir os arquivos compactados do computador para um dispositivo de núcleo do Raspberry Pi, use uma ferramenta como WinSCP ou o comando PuTTY pscp. Para usar o comando pscp, abra uma janela de prompt de comando no computador e execute o seguinte:

```
cd path-to-downloaded-files
pscp -pw Pi-password greengrass-OS-architecture-1.11.6.tar.gz pi@IP-
address:/home/pi
pscp -pw Pi-password certificateId-certificate.pem.crt pi@IP-address:/home/
pi
pscp -pw Pi-password certificateId-public.pem.key pi@IP-address:/home/pi
pscp -pw Pi-password certificateId-private.pem.key pi@IP-address:/home/pi
pscp -pw Pi-password AmazonRootCA1.pem pi@IP-address:/home/pi
```

> **Note**
>
> O número da versão nesse comando deve corresponder à versão do seu pacote de software AWS IoT Greengrass Core.

macOS

Para transferir os arquivos compactados do Mac para um dispositivo de núcleo do Raspberry Pi, abra uma janela do terminal no computador e execute os comandos a seguir. O *path-to-downloaded-files* é tipicamente`~/Downloads`.

> **Note**
>
> Podem ser solicitadas duas senhas. Nesse caso, a primeira senha será para o comando `sudo` do Mac e a segunda será a senha do Raspberry Pi.

```
cd path-to-downloaded-files
scp greengrass-OS-architecture-1.11.6.tar.gz pi@IP-address:/home/pi
scp certificateId-certificate.pem.crt pi@IP-address:/home/pi
scp certificateId-public.pem.key pi@IP-address:/home/pi
scp certificateId-private.pem.key pi@IP-address:/home/pi
scp AmazonRootCA1.pem pi@IP-address:/home/pi
```

> **Note**
>
> O número da versão nesse comando deve corresponder à versão do seu pacote de software AWS IoT Greengrass Core.

UNIX-like system

Para transferir os arquivos compactados do computador para um dispositivo de núcleo do Raspberry Pi, abra uma janela do terminal no computador e execute os seguintes comandos:

```
cd path-to-downloaded-files
scp greengrass-OS-architecture-1.11.6.tar.gz pi@IP-address:/home/pi
scp certificateId-certificate.pem.crt pi@IP-address:/home/pi
scp certificateId-public.pem.key pi@IP-address:/home/pi
scp certificateId-private.pem.key pi@IP-address:/home/pi
scp AmazonRootCA1.pem pi@IP-address:/home/pi
```

> **Note**
>
> O número da versão nesse comando deve corresponder à versão do seu pacote de software AWS IoT Greengrass Core.

Raspberry Pi web browser

Se você tiver usado o navegador da web do Raspberry Pi para fazer download dos arquivos compactados, os arquivos deverão estar na pasta `~/Downloads` do Pi, como a `/home/pi/Downloads`. Caso contrário, os arquivos compactados deverão estar na pasta ~ do Pi, como a `/home/pi`.

3. No dispositivo principal do Greengrass, abra um terminal e navegue até a pasta que contém o software AWS IoT Greengrass principal e os certificados. *path-to-transferred-files*Substitua pelo caminho para o qual você transferiu os arquivos no dispositivo principal. Por exemplo, em um Raspberry Pi, execute `cd /home/pi`.

```
cd path-to-transferred-files
```

4. Descompacte o software AWS IoT Greengrass Core no dispositivo principal. Execute o comando a seguir para descompactar o software de arquivo que você transferiu para o dispositivo de núcleo. Esse comando usa o argumento `-C /` para criar a pasta `/greengrass` na pasta raiz do dispositivo de núcleo.

```
sudo tar -xzvf greengrass-OS-architecture-1.11.6.tar.gz -C /
```

> **Note**
>
> O número da versão nesse comando deve corresponder à versão do seu pacote de software AWS IoT Greengrass Core.

5. Mova os certificados e as chaves para a pasta do software AWS IoT Greengrass principal. Execute os comandos a seguir para criar uma pasta para certificados e transferir os certificados e as chaves para ela. *path-to-transferred-files*Substitua pelo caminho para o qual você transferiu os arquivos no dispositivo principal e *certificateId* substitua pela ID do certificado nos nomes dos arquivos. Por exemplo, em um Raspberry Pi, substitua por *path-to-transferred-files* **/home/pi**

```
sudo mv path-to-transferred-files/certificateId-certificate.pem.crt /greengrass/
certs
sudo mv path-to-transferred-files/certificateId-public.pem.key /greengrass/certs
sudo mv path-to-transferred-files/certificateId-private.pem.key /greengrass/certs
```

```
sudo mv path-to-transferred-files/AmazonRootCA1.pem /greengrass/certs
```

6. O software AWS IoT Greengrass Core usa um arquivo de configuração que especifica parâmetros para o software. Esse arquivo de configuração especifica os caminhos dos arquivos de certificado e os Nuvem AWS endpoints a serem usados. Nesta etapa, você cria o arquivo de configuração do software AWS IoT Greengrass Core para seu núcleo. Faça o seguinte:

   a. Obtenha o Amazon Resource Name (ARN) para sua essência. AWS IoT Faça o seguinte:

      i. No AWS IoT console, em Gerenciar, em Dispositivos Greengrass, selecione Grupos (V1).

      ii. Na página de Grupos do Greengrass, selecione o grupo criado por você anteriormente.

      iii. Em Visão geral, selecione Núcleo do Greengrass.

      iv. Na página de detalhes do núcleo, copie o ARN do AWS IoT item e salve-o para usar no arquivo de configuração AWS IoT Greengrass principal.

   b. Obtenha o endpoint de dados do AWS IoT dispositivo para você Conta da AWS na região atual. Os dispositivos usam esse endpoint para se conectar AWS às AWS IoT coisas. Faça o seguinte:

      i. Em AWS IoT console, selecione Configurações.

      ii. Em Endpoint de dados do dispositivo, copie o Endpoint e salve-o para usar no arquivo de configuração do AWS IoT Greengrass Core.

   c. Crie o arquivo de configuração do software AWS IoT Greengrass principal. Por exemplo, você pode executar o seguinte comando para usar o GNU nano para criar o arquivo.

      ```
      sudo nano /greengrass/config/config.json
      ```

      Substitua o conteúdo do arquivo com o seguinte documento JSON.

      ```
      {
        "coreThing" : {
          "caPath": "AmazonRootCA1.pem",
          "certPath": "certificateId-certificate.pem.crt",
          "keyPath": "certificateId-private.pem.key",
          "thingArn": "arn:aws:iot:region:account-id:thing/MyGreengrassV1Core",
          "iotHost": "device-data-prefix-ats.iot.region.amazonaws.com",
          "ggHost": "greengrass-ats.iot.region.amazonaws.com",
          "keepAlive": 600
      ```

```
  },
  "runtime": {
    "cgroup": {
      "useSystemd": "yes"
    }
  },
  "managedRespawn": false,
  "crypto": {
    "caPath": "file:///greengrass/certs/AmazonRootCA1.pem",
    "principals": {
      "SecretsManager": {
        "privateKeyPath": "file:///greengrass/certs/certificateId-
private.pem.key"
      },
      "IoTCertificate": {
        "privateKeyPath": "file:///greengrass/certs/certificateId-
private.pem.key",
        "certificatePath": "file:///greengrass/certs/certificateId-
certificate.pem.crt"
      }
    }
  }
}
```

Faça o seguinte:

- Se você baixou um certificado de CA raiz da Amazon diferente do Amazon Root CA 1, substitua cada instância de *AmazonRootCA1.pem* pelo nome do arquivo da CA raiz da Amazon.
- Substitua cada instância do *certificateId* pelo ID do certificado no nome do certificado e dos arquivos-chave.
- *arn:aws:iot:region:account-id:thing/MyGreengrassV1Core*Substitua pelo ARN do item do seu núcleo que você salvou anteriormente.
- *MyGreengrassV1core*Substitua pelo nome da coisa mais importante do seu núcleo.
- *device-data-prefix-ats.iot.region.amazonaws.com*Substitua pelo endpoint de dados do AWS IoT dispositivo que você salvou anteriormente.
- *region*Substitua pelo seu Região da AWS.

Para obter mais informações sobre as opções de configuração que você pode especificar nessa configuração, consulte AWS IoT Greengrass arquivo de configuração principal.

7. Verifique se o dispositivo de núcleo está conectado à Internet. Depois, inicie o AWS IoT Greengrass no dispositivo de núcleo.

```
cd /greengrass/ggc/core/
sudo ./greengrassd start
```

Você deverá uma mensagem `Greengrass successfully started`. Anote o PID da função.

> **Note**
>
> Para configurar seu dispositivo principal para iniciar AWS IoT Greengrass na inicialização do sistema, consultethe section called “Iniciar o Greengrass na inicialização do sistema”.

Você pode executar o comando a seguir para confirmar se o software AWS IoT Greengrass Core (daemon Greengrass) está funcionando. *`PID-number`*Substitua pelo seu PID:

```
ps aux | grep PID-number
```

Você deve ver uma entrada do PID com um caminho para o daemon do Greengrass que está em execução (por exemplo, `/greengrass/ggc/packages/1.11.6/bin/daemon`). Se você tiver problemas ao iniciar AWS IoT Greengrass, consulteSolução de problemas.

Módulo 3 (parte 1): funções Lambda em AWS IoT Greengrass

Este módulo mostra como criar e implantar uma função Lambda que envia mensagens MQTT do seu AWS IoT Greengrass dispositivo principal. O módulo descreve configurações de funções do Lambda, assinaturas usadas para permitir mensagens MQTT e implantações em um dispositivo de núcleo.

O módulo 3 (Parte 2) aborda as diferenças entre funções Lambda sob demanda e de longa duração executadas no núcleo. AWS IoT Greengrass

Antes de começar, verifique se você concluiu o Módulo 1 e o Módulo 2 e tem um dispositivo AWS IoT Greengrass principal em execução.

> **Tip**
>
> Ou, para usar um script que configure o dispositivo de núcleo para você, consulte the section called “Início rápido: Configuração do dispositivo do Greengrass”. O script também pode criar e implantar a função do Lambda usada neste módulo.

Este modulo levará aproximadamente 30 minutos para ser concluído.

Tópicos

- Crie e empacote uma função do Lambda
- Configure a função Lambda para AWS IoT Greengrass
- Implantar configurações de nuvem em um dispositivo de núcleo do Greengrass
- Verificar se a função do Lambda está em execução no dispositivo de núcleo

Crie e empacote uma função do Lambda

O exemplo de função do Lambda em Python deste módulo usa o SDK do AWS IoT Greengrass Core para que o Python publique mensagens MQTT.

Nesta etapa:

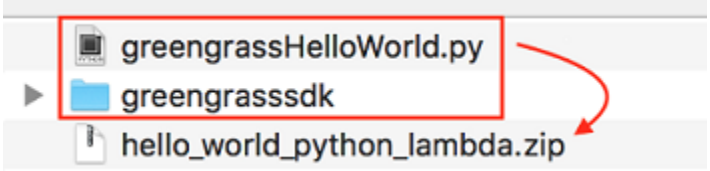
- Faça o download do AWS IoT Greengrass Core SDK para Python em seu computador (não no dispositivo AWS IoT Greengrass principal).
- Crie um pacote de implantação de funções do Lambda que contém o código da função e as dependências.
- Use o console do Lambda para criar uma função do Lambda e fazer upload do pacote de implantação.
- Publique uma versão da função do Lambda e crie um alias que aponte para a versão.

Para concluir este módulo, o Python 3.7 deve ser instalado no dispositivo principal.

1. Na página de downloads do AWS IoT Greengrass Core SDK, baixe o AWS IoT Greengrass Core SDK para Python em seu computador.
2. Descompacte o pacote baixado para obter o código da função do Lambda e o SDK.

   A função do Lambda neste módulo usa:

   - O arquivo `greengrassHelloWorld.py` em `examples\HelloWorld`. Este é o código da função do Lambda. A cada cinco segundos, a função publica uma das duas possíveis mensagens no tópico `hello/world`.
   - A pasta `greengrasssdk`. Esse é o SDK.
3. Copie a pasta `greengrasssdk` para a pasta `HelloWorld` que contém `greengrassHelloWorld.py`.
4. Para criar o pacote de implantação de funções do Lambda, salve `greengrassHelloWorld.py` o `greengrasssdk` e a pasta `zip` em um arquivo compactado denominado `hello_world_python_lambda.zip`. O arquivo py e a pasta `greengrasssdk` devem estar na raiz do diretório.

   Em UNIX-like sistemas (incluindo o terminal Mac), você pode usar o seguinte comando para empacotar o arquivo e a pasta:

   ```
   zip -r hello_world_python_lambda.zip greengrasssdk greengrassHelloWorld.py
   ```

   > **Note**
   >
   > Dependendo da sua distribuição, pode ser necessário instalar `zip` primeiro (por exemplo, ao executar `sudo apt-get install zip`). O comando de instalação pode ser diferente para sua distribuição.

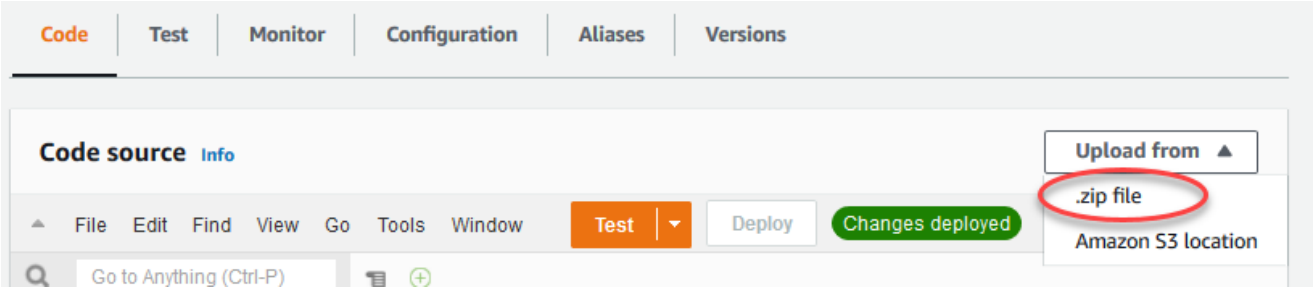
   Agora, você está pronto para criar sua função do Lambda e fazer upload do pacote de implantação.
5. Abra o console do Lambda e selecione Criar função.

6. Selecione Author from scratch.
7. Dê à função o nome **`Greengrass_HelloWorld`** e defina os campos restantes da seguinte forma:

   - Em Runtime (Tempo de execução), selecione Python 3.7.
   - Em Permissões, mantenha a configuração padrão. Isso cria uma função de execução que concede permissões básicas do Lambda. Essa função não é usada por AWS IoT Greengrass.

   Escolha a opção Criar função.
8. Faça upload do pacote de implantação da função Lambda:

   a. Na guia Código, em Fonte do código, selecione Fazer upload a partir de. No menu suspenso, selecione o arquivo .zip.

   b. Selecione Upload e, em seguida, selecione seu pacote de implantação `hello_world_python_lambda.zip`. Selecione Salvar.

   c. Na guia Código da função, em Configurações de runtime, selecione Editar e insira os valores a seguir.

      - Em Runtime (Tempo de execução), selecione Python 3.7.
      - Em Handler (Manipulador), insira **`greengrassHelloWorld.function_handler`**.

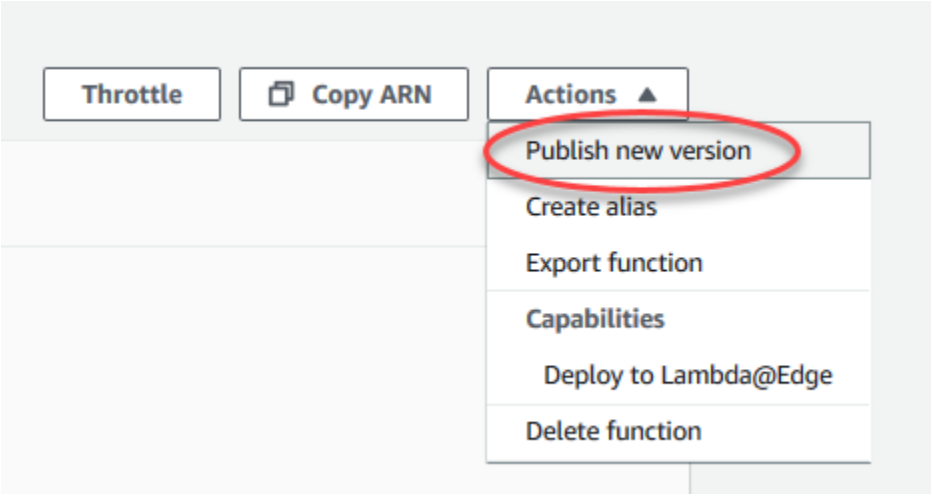


d. Escolha Salvar.

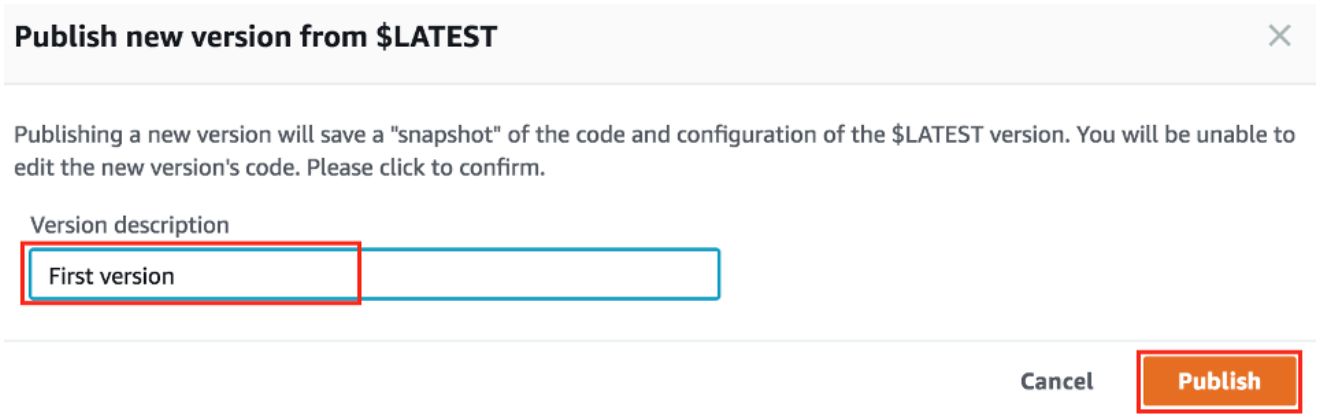
> **Note**
>
> O botão Testar no AWS Lambda console não funciona com essa função. O SDK AWS IoT Greengrass Core não contém módulos necessários para executar suas funções do Greengrass Lambda de forma independente no console. AWS Lambda Esses módulos (por exemplo, `greengrass_common`) são fornecidos às funções depois de serem implantados no núcleo do Greengrass.

9. Publique a função do Lambda:

a. No menu Ações na parte superior da página, selecione Publicar versão nova.



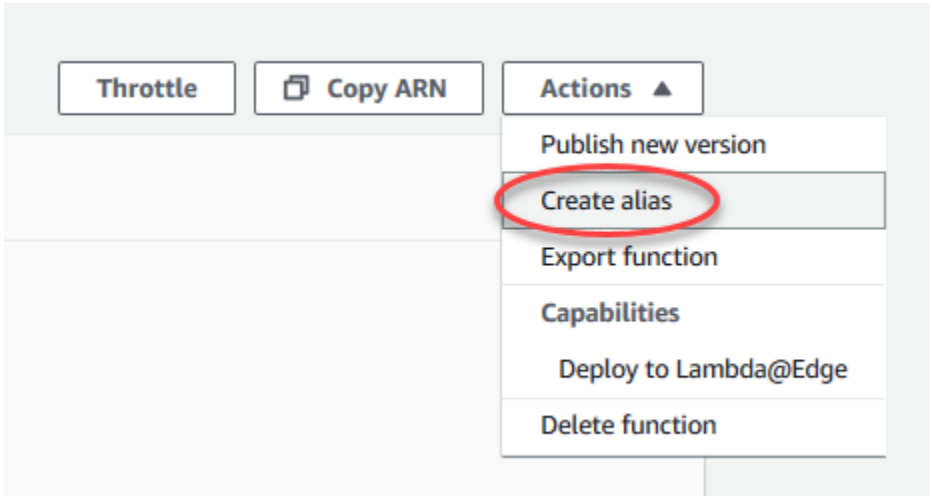
b. Em Version description (Descrição da versão), insira **`First version`** e, em seguida, selecione Publish (Publicar).



10. Crie um alias para a versão da função do Lambda:

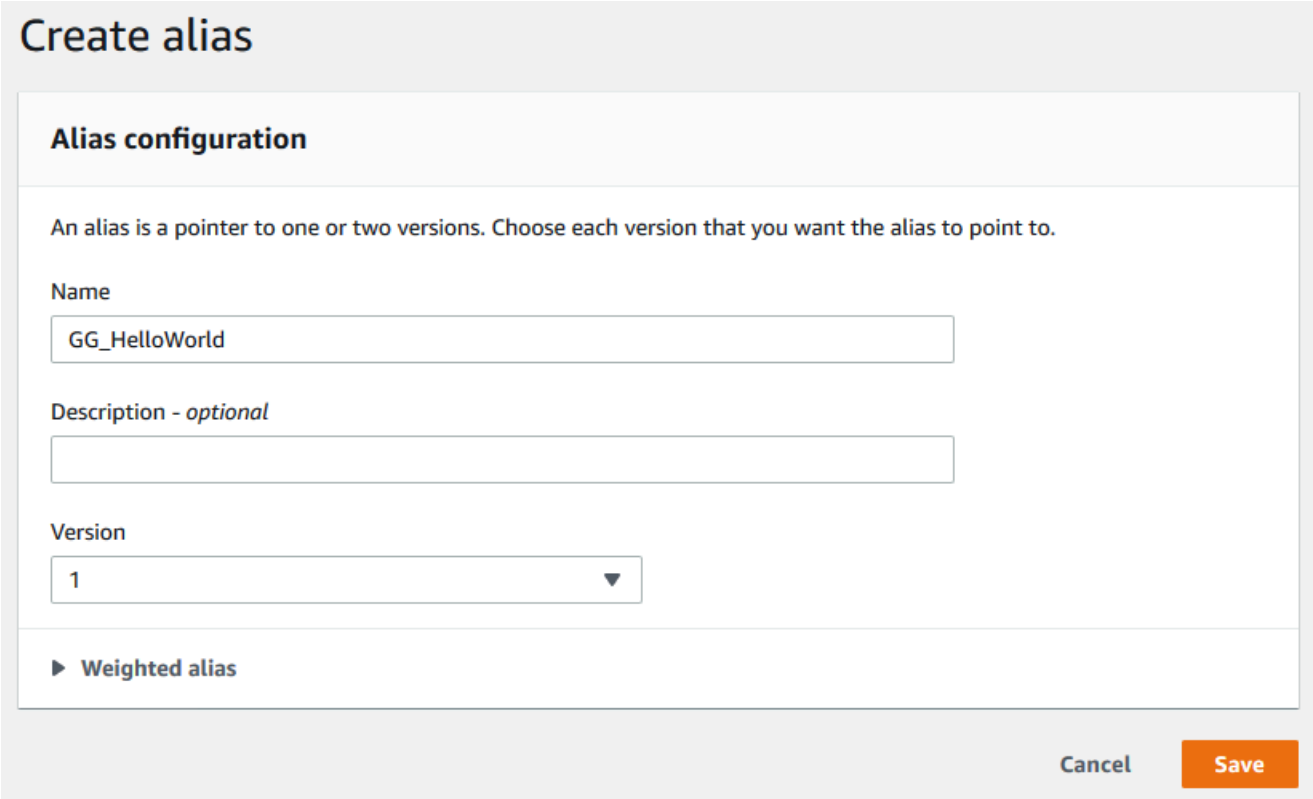
> **Note**
>
> Os grupos do Greengrass podem fazer referência a uma função do Lambda por alias (recomendado) ou por versão. Usar um alias facilita o gerenciamento de atualizações de código porque você não precisa alterar a tabela de assinaturas nem a definição do grupo ao atualizar a função do código. Em vez disso, você pode simplesmente apontar o alias para a nova versão da função.

a. No menu Ações na parte superior da página, selecione Criar alias.



b. Nomeie o alias como **`GG_HelloWorld`**, defina a versão como **`1`** (que corresponde à versão que você acabou de publicar) e selecione Salvar.

> **Note**
>
> AWS IoT Greengrass não oferece suporte a aliases do Lambda para as versões $LATEST.



Configure a função Lambda para AWS IoT Greengrass

Agora você está pronto para configurar sua função do Lambda para o AWS IoT Greengrass.

Nesta etapa:

- Use o AWS IoT console para adicionar a função Lambda ao seu grupo do Greengrass.
- Defina configurações específicas do grupo para a função do Lambda.
- Adicione uma assinatura ao grupo que permita que a função do Lambda publique mensagens MQTT no AWS IoT.
- Defina as configurações de log local para o grupo.

1. No painel de navegação do AWS IoT console, em Gerenciar, expanda dispositivos Greengrass e escolha Grupos (V1).
2. Em Grupos do Greengrass, selecione o grupo criado por você no Módulo 2.
3. Na página de configuração do grupo, selecione a guia Funções do Lambda e, em seguida, role para baixo até a seção Minhas funções do Lambda e selecione Adicionar função do Lambda.
4. Selecione o nome da função Lambda que você criou na etapa anterior (Greengrass_HelloWorlde não o nome do alias).
5. Para a versão, escolha Alias: HelloWorld GG_.
6. Na seção Configuração da função do Lambda, faça as seguintes alterações:

   - Defina o Usuário e grupo do sistema como Usar padrão do grupo.
   - Defina Conteinerização da função do Lambda para Usar padrão do grupo.
   - Defina Timeout (Tempo limite) como 25 segundos. Essa função do Lambda permanece em espera por 5 segundos antes de cada invocação.
   - Para Fixado, selecione Verdadeiro.

   > **Note**
   >
   > Uma função Lambda de longa duração (ou fixada) é iniciada automaticamente após o AWS IoT Greengrass início e continua sendo executada em seu próprio contêiner. Isso contrasta com uma função do Lambda sob demanda, que é iniciada quando invocada e interrompida quando não há tarefas a serem executadas. Para obter mais informações, consulte the section called “Configuração do ciclo de vida”.

7. Selecione Adicionar função do Lambda para salvar as alterações. Para obter informações sobre as propriedades de funções do Lambda, consulte the section called “Controlando a execução da função do Lambda do Greengrass”.

   Em seguida, crie uma assinatura que permita que a função do Lambda envie mensagens MQTT para o AWS IoT Core.

   A função do Lambda do Greengrass pode trocar mensagens MQTT com:

- Dispositivos no grupo do Greengrass.
- Conectores no grupo.
- Outras funções do Lambda no grupo.
- AWS IoT Core.
- O serviço de sombra local. Para obter mais informações, consulte the section called “Módulo 5: Interagir com sombras de dispositivos”.

O grupo usa assinaturas para controlar a maneira como essas entidades podem se comunicar umas com as outras. As assinaturas fornecem interações previsíveis e uma layer de segurança.

Uma assinatura consiste em uma origem, um destino e um tópico. A origem é o originador da mensagem. O destino é o destino da mensagem. O tópico permite que você filtre os dados enviados da origem para o destino. A origem ou o destino podem ser um dispositivo do Greengrass, uma função do Lambda, um conector, uma sombra do dispositivo ou o AWS IoT Core.

> **Note**
>
> Uma assinatura é direcionada de tal forma que as mensagens fluem em uma direção específica: da origem para o destino. Para permitir uma comunicação bidirecional, você deve configurar duas assinaturas.

> **Note**
>
> Atualmente, o filtro de tópicos da assinatura não permite mais do que um único caractere + em um tópico. O filtro de tópicos permite apenas um único caractere # no final de um tópico.

A função `Greengrass_HelloWorld` Lambda envia mensagens somente para o `hello/world` tópico em AWS IoT Core, então você só precisa criar uma assinatura da função Lambda para. AWS IoT Core Isso será feito na próxima etapa.

8. Na página de configuração do grupo, selecione a guia Assinaturas e, em seguida, Adicionar assinatura.

Para obter um exemplo que mostra como criar uma assinatura usando a AWS CLI, consulte create-subscription-definition na Referência de comandos da AWS CLI .

9. No Tipo de fonte, escolha a função Lambda e, para a Fonte, escolha. Greengrass_HelloWorld
10. Para Tipo de destino, selecione Serviço e, para Destino, selecione IoT Cloud.
11. Em Filtro de tópicos, insira **`hello/world`** e, em seguida selecione Criar assinatura.
12. Defina as configurações de registro do grupo. Neste tutorial, configure os componentes do sistema do AWS IoT Greengrass e funções do Lambda definidas pelo usuário para gravar logs no sistema de arquivos do dispositivo do núcleo.

    a. Na página de configuração do grupo, selecione a guia Logs.
    b. Na seção Configuração de logs local, selecione Editar.
    c. Na caixa de diálogo Editar configuração de logs locais, mantenha os valores padrão para níveis de log e tamanhos de armazenamento e, em seguida, selecione Salvar.

    Você pode usar logs para solucionar quaisquer problemas que encontrar ao executar este tutorial. Ao solucionar problemas, é possível alterar temporariamente o nível de log para Depurar. Para obter mais informações, consulte the section called “Acessar os logs do sistema de arquivos”.
13. Se o módulo de tempo de execução do Java 8 não estiver instalado no dispositivo de núcleo, você deverá instalá-lo ou desabilitar o gerenciador de fluxo.

    > **Note**
    >
    > Este tutorial não usa o gerenciador de fluxo, mas usa o fluxo de trabalho de Criação de grupo padrão que habilita o gerenciador de fluxo por padrão. Se o gerenciador de fluxo estiver habilitado, mas o Java 8 não estiver instalado, haverá falha na implantação do grupo. Para obter mais informações, consulte os requisitos do gerenciador de fluxo.

    Para desabilitar o gerenciador de fluxo:

    a. Na página de configurações do grupo, selecione a guia Funções do Lambda.
    b. Na seção Funções do Lambda do sistema, selecione Gerenciador de fluxo e selecione Editar.
    c. Selecione Desabilitar e Salvar.

Implantar configurações de nuvem em um dispositivo de núcleo do Greengrass

1. Verifique se o dispositivo de núcleo está conectado à Internet. Por exemplo, tente navegar até uma página da Web.
2. Verifique se o daemon do Greengrass está em execução no dispositivo de núcleo. No terminal do dispositivo de núcleo, execute os comandos a seguir para verificar se o daemon está em execução e o inicie, caso necessário.

   a. Para verificar se o daemon está em execução:

   ```
   ps aux | grep -E 'greengrass.*daemon'
   ```

   Se a saída contém uma entrada `root` para `/greengrass/ggc/packages/1.11.6/bin/daemon`, o daemon está em execução.

   b. Para iniciar o daemon:

   ```
   cd /greengrass/ggc/core/
   sudo ./greengrassd start
   ```

   Agora você está pronto para implantar a função do Lambda e as configurações de assinatura no dispositivo de núcleo do Greengrass.
3. No painel de navegação do AWS IoT console, em Gerenciar, expanda dispositivos Greengrass e escolha Grupos (V1).
4. Em Grupos do Greengrass, selecione o grupo criado por você no Módulo 2.
5. Na página de configuração do grupo, selecione Implantar.
6. Na guia Funções do Lambda, na seção Funções do Lambda do sistema, selecione Detector de IP.
7. Selecione Editar e, em seguida selecione Automatically detect and override MQTT broker endpoints (Detectar e substituir automaticamente os endpoints de atendente MQTT). Isso permite que os dispositivos adquiram automaticamente as informações de conectividade para o núcleo, como endereço IP, DNS e o número da porta. A detecção automática é recomendada, mas o AWS IoT Greengrass também oferece suporte a endpoints especificados manualmente. Você só é solicitado pelo método de descoberta na primeira vez em que o grupo for implantado.

A primeira implantação pode demorar alguns minutos. Quando a implantação estiver concluída, você deverá ver Successfully completed (Concluído com êxito) na coluna Status da página Deployments (Implantações):

> **Note**
>
> O status de implantação também é exibido abaixo do nome do grupo no cabeçalho da página.

Para obter ajuda sobre a solução de problemas, consulte Solução de problemas.

Verificar se a função do Lambda está em execução no dispositivo de núcleo

1. No painel de navegação do console do AWS IoT, em Teste, selecione Cliente de teste MQTT.
2. Selecione a guia Subscribe to topic (Inscrever-se no tópico).
3. Insira **`hello/world`** em Filtro de tópicos e expanda Configuração adicional.
4. Insira as informações listadas em cada um dos seguintes campos:
   - Para Quality of Service (Qualidade de Serviço), selecione 0.
   - Em MQTT payload display (Exibição de carga MQTT), selecione Display payloads as strings (Exibir cargas como strings).
5. Selecione Subscribe.

Supondo que a função do Lambda esteja sendo executada em seu dispositivo, ela publicará no tópico `hello/world` mensagens semelhantes à seguinte:



Embora a função Lambda continue enviando mensagens MQTT para o `hello/world` tópico, não interrompa o AWS IoT Greengrass daemon. Os módulos restantes são gravados com a suposição de que ele esteja em execução.

Você pode excluir a função e a assinatura do grupo:

- Na página de configuração de grupos, na guia Funções do Lambda, selecione a função do Lambda que você deseja remover e selecione Remover.
- Na página de configuração de grupos, na guia Assinaturas, selecione a assinatura e, em seguida, selecione Excluir.

A função e a assinatura serão removidas do núcleo durante a próxima implantação de grupo.

Módulo 3 (parte 2): funções Lambda em AWS IoT Greengrass

Este módulo explora as diferenças entre funções Lambda sob demanda e de longa duração executadas no núcleo. AWS IoT Greengrass

Antes de começar, execute o script de configuração do dispositivo do Greengrass ou verifique se você concluiu o módulo 1, o módulo 2 e o módulo 3 (parte 1).

Este modulo levará aproximadamente 30 minutos para ser concluído.

Tópicos

- Criar e empacotar a função do Lambda
- Configure funções Lambda de longa duração para AWS IoT Greengrass
- Testar funções de longa duração do Lambda

- Teste as funções do Lambda sob demanda

Criar e empacotar a função do Lambda

Nesta etapa:

- Crie um pacote de implantação de funções do Lambda que contém o código da função e as dependências.
- Use o console do Lambda para criar uma função do Lambda e fazer upload do pacote de implantação.
- Publique uma versão da função do Lambda e crie um alias que aponte para a versão.

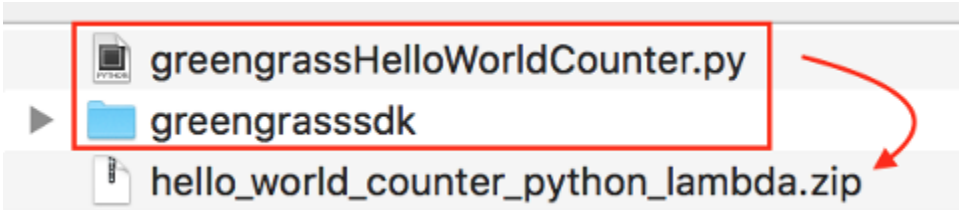
1. No seu computador, acesse o AWS IoT Greengrass Core SDK para Python que você baixou e extraiu the section called “Crie e empacote uma função do Lambda” no Módulo 3-1.

   A função do Lambda neste módulo usa:

   - O arquivo `greengrassHelloWorldCounter.py` em `examples\HelloWorldCounter`. Este é o código da função do Lambda.
   - A pasta `greengrasssdk`. Esse é o SDK.
2. Crie um pacote de implantação da função do Lambda;

   a. Copie a pasta `greengrasssdk` para a pasta `HelloWorldCounter` que contém `greengrassHelloWorldCounter.py`.

   b. Salve `greengrassHelloWorldCounter.py` e a pasta `greengrasssdk` em um arquivo `zip` chamado `hello_world_counter_python_lambda.zip`. O arquivo py e a pasta `greengrasssdk` devem estar na raiz do diretório.

   Em UNIX-like sistemas (incluindo o terminal Mac) que foram `zip` instalados, você pode usar o seguinte comando para empacotar o arquivo e a pasta:

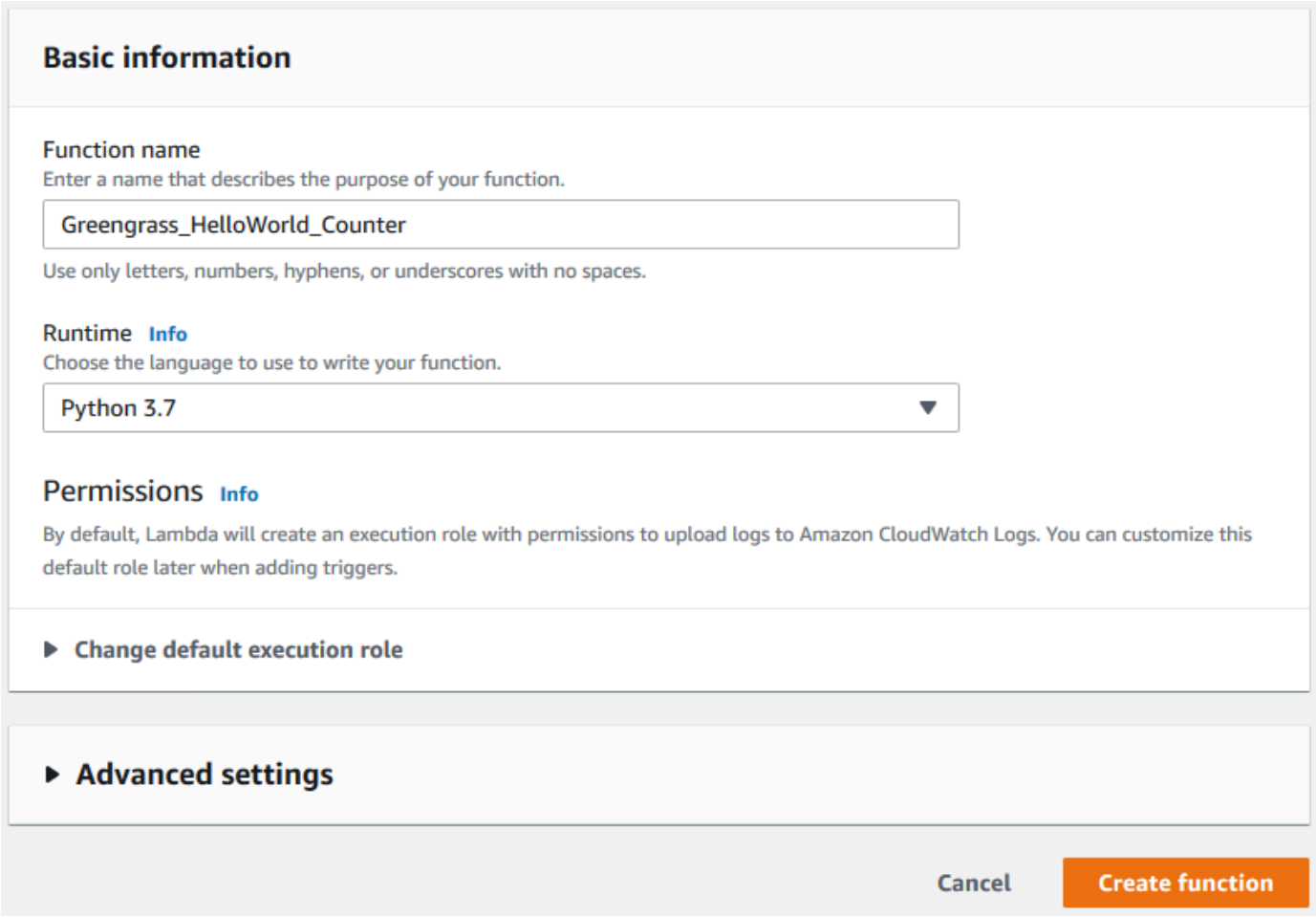
```
zip -r hello_world_counter_python_lambda.zip greengrasssdk
 greengrassHelloWorldCounter.py
```

Agora, você está pronto para criar sua função do Lambda e fazer upload do pacote de implantação.

3. Abra o console do Lambda e selecione Criar função.
4. Selecione Author from scratch.
5. Dê à função o nome **`Greengrass_HelloWorld_Counter`** e defina os campos restantes da seguinte forma:

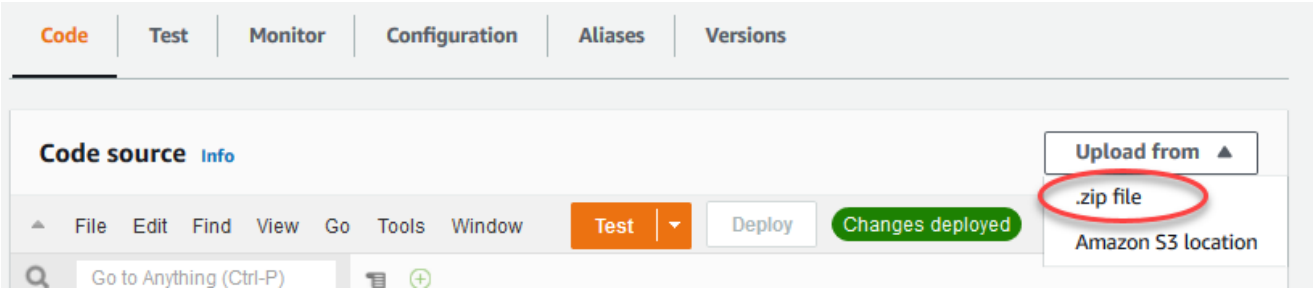
   - Em Runtime (Tempo de execução), selecione Python 3.7.
   - Em Permissões, mantenha a configuração padrão. Isso cria uma função de execução que concede permissões básicas do Lambda. Essa função não é usada por AWS IoT Greengrass. Também é possível reutilizar a função criada no módulo 3-1.

   Selecione Criar função.



6. Faça upload do pacote de implantação da função do Lambda.

   a. Na guia Código, em Fonte do código, selecione Fazer upload a partir de. No menu suspenso, selecione o arquivo .zip.



   b. Selecione Upload e, em seguida, selecione seu pacote de implantação `hello_world_counter_python_lambda.zip`. Selecione Salvar.

   c. Na guia Código da função, em Configurações de runtime, selecione Editar e insira os valores a seguir.

      - Em Runtime (Tempo de execução), selecione Python 3.7.

- Em Handler (Manipulador), insira **`greengrassHelloWorldCounter.function_handler`**.

d. Escolha Salvar.

> **Note**
>
> O botão Testar no AWS Lambda console não funciona com essa função. O SDK AWS IoT Greengrass Core não contém módulos necessários para executar suas funções do Greengrass Lambda de forma independente no console. AWS Lambda Esses módulos (por exemplo, `greengrass_common`) são fornecidos às funções depois de serem implantados no núcleo do Greengrass.

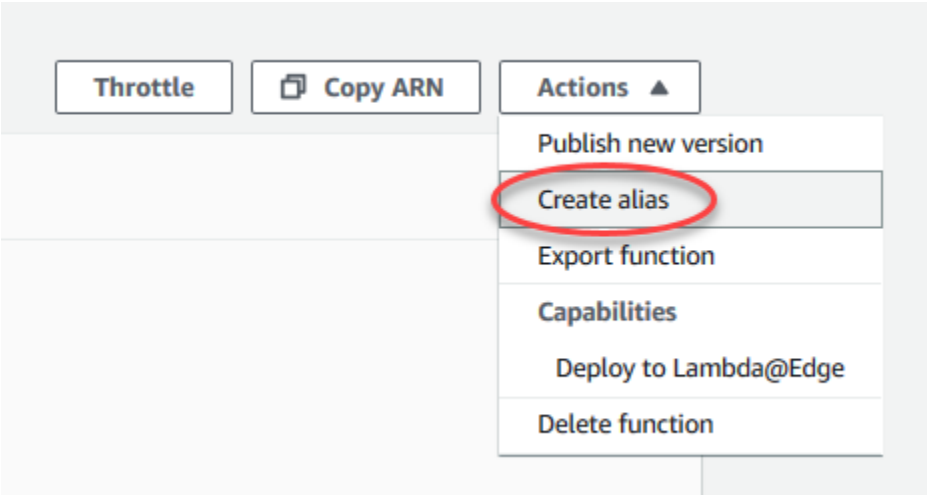
7. Publique a primeira versão da função.

   a. No menu Ações na parte superior da página, selecione Publicar versão nova. Em Version description (Descrição da versão), insira **`First version`**.

   b. Selecione Publish.

8. Crie um alias para a versão da função.

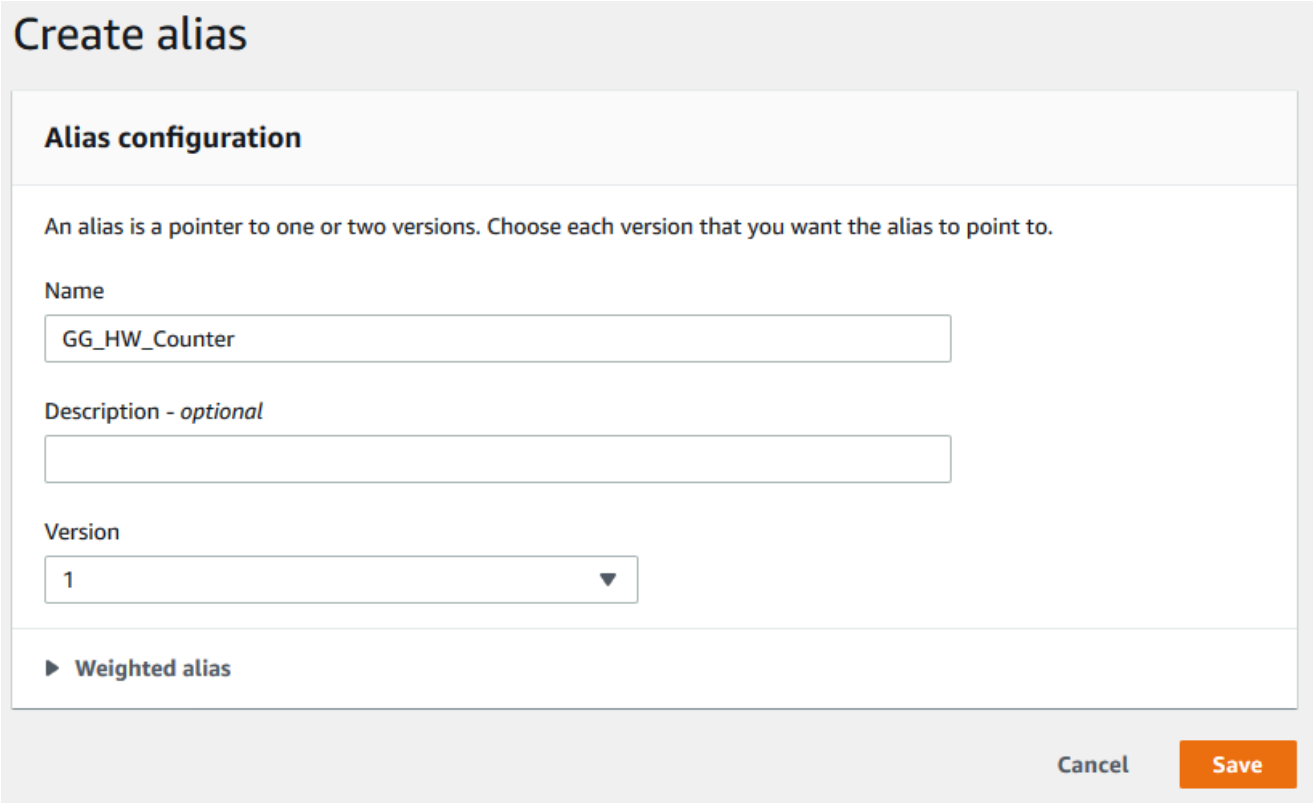
   a. No menu Ações na parte superior da página, selecione Criar alias.



   b. Em Nome, digite **`GG_HW_Counter`**.

   c. Em Version, selecione 1.

   d. Selecione Salvar.



Os aliases criam uma única entidade para a sua função do Lambda que os Dispositivos Greengrass podem assinar. Dessa forma, não é necessário atualizar assinaturas com novos números de versão da função do Lambda sempre que a função é modificada.

Configure funções Lambda de longa duração para AWS IoT Greengrass

Agora você está pronto para configurar sua função do Lambda para o AWS IoT Greengrass.

1. No painel de navegação do AWS IoT console, em Gerenciar, expanda dispositivos Greengrass e escolha Grupos (V1).
2. Em Grupos do Greengrass, selecione o grupo criado por você no Módulo 2.
3. Na página de configuração do grupo, selecione a guia Funções do Lambda e, em seguida, em Minhas funções do Lambda, selecione Adicionar.
4. Para a função Lambda, escolha. Greengrass_HelloWorld_Counter
5. Em Versão da função do Lambda, selecione o alias da versão que você publicou.
6. Em Tempo limite (segundos), insira **25**. Essa função do Lambda permanece em espera por 20 segundos antes de cada invocação.

7. Para Fixado, selecione Verdadeiro.
8. Deixe todos os outros campos nos valores padrão e selecione Adicionar função do Lambda.

Testar funções de longa duração do Lambda

Uma função Lambda de longa duração é iniciada automaticamente quando o AWS IoT Greengrass núcleo é iniciado e executado em um único contêiner (ou sandbox). Todas as variáveis e lógica de pré-processamento definidas fora do manipulador de funções são mantidas para cada invocação do manipulador de funções. As várias invocações do manipulador de funções são colocadas em fila até que a execução das invocações anteriores tenha sido executada.

O código `greengrassHelloWorldCounter.py` usado neste módulo define uma variável `my_counter` fora do manipulador de funções.

> **Note**
>
> Você pode visualizar o código no AWS Lambda console ou no AWS IoT Greengrass Core SDK para Python em. GitHub

Nesta etapa, você cria assinaturas que permitem a função Lambda e a troca de mensagens AWS IoT MQTT. Em seguida, você implanta o grupo e testa a função.

1. Na página de configuração do grupo, selecione Inscrições e, então, selecione Adicionar.
2. Em Tipo de origem, escolha Função Lambda e, em seguida, escolha Greengrass_HelloWorld_Counter.
3. Em Tipo de destino, selecione Serviço, selecione IoT Cloud.
4. Para Topic filter, insira **`hello/world/counter`**.
5. Selecione Criar assinatura.

   Essa assinatura única segue apenas uma direção: da função `Greengrass_HelloWorld_Counter` Lambda para AWS IoT. Para invocar (ou trigger) essa função do Lambda na nuvem, é necessário criar uma assinatura na direção oposta.
6. Siga as etapas de 1 a 5 para adicionar outra assinatura que use o valores a seguir. Essa assinatura permite que a função do Lambda receba mensagens da AWS IoT. Você usa essa assinatura ao enviar uma mensagem do AWS IoT console que invoca a função.

- Para a origem, selecione Serviço e, em seguida, selecione IoT Cloud.
- Para o destino, escolha a função Lambda e, em seguida, escolha Greengrass_HelloWorld_Counter.
- Para o filtro de tópico, insira **`hello/world/counter/trigger`**.

A extensão `/trigger` é usada neste filtro de tópicos porque você criou duas assinaturas e não deseja que uma interfira na outra.

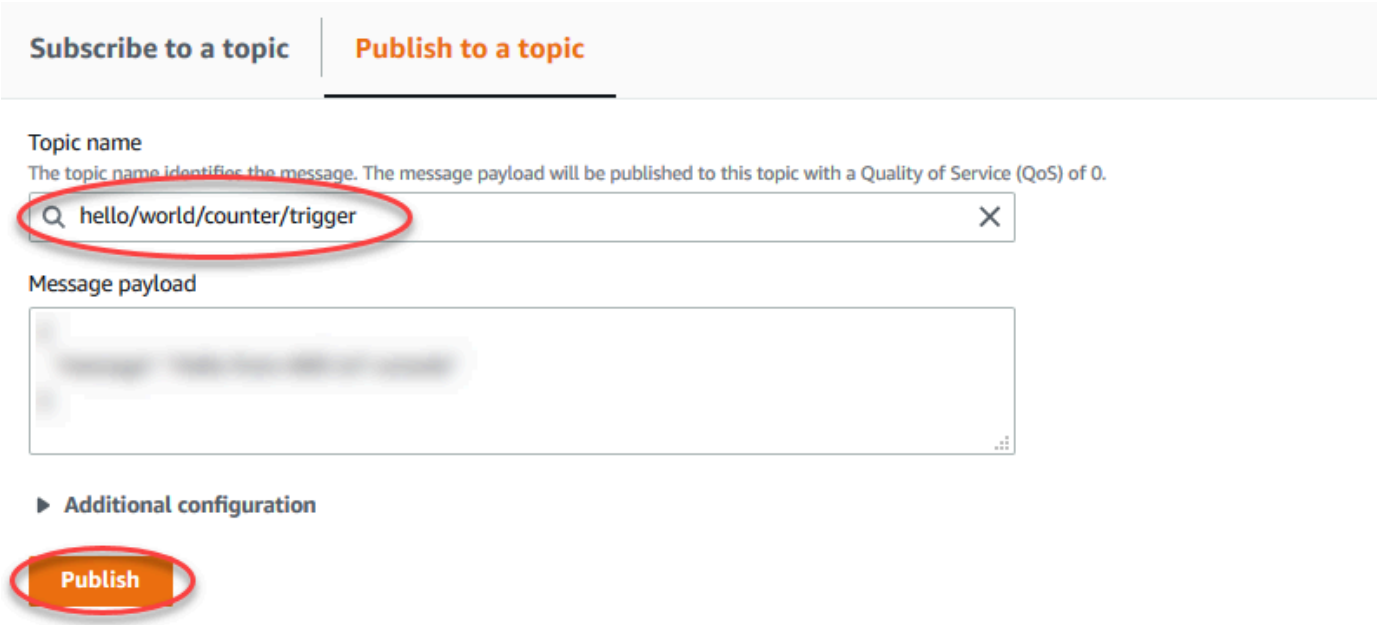
7. Verifique se o daemon do Greengrass está em execução, como descrito em Implantar configurações de nuvem em um dispositivo de núcleo.
8. Na página de configuração do grupo, selecione Implantar.
9. Após a conclusão da implantação, retorne à página inicial do AWS IoT console e escolha Testar.
10. Configure os campos a seguir.
    - Em Subscription topic (Tópico de assinatura), insira **`hello/world/counter`**.
    - Para Quality of Service (Qualidade de Serviço), selecione 0.
    - Em MQTT payload display (Exibição de carga MQTT), selecione Display payloads as strings (Exibir cargas como strings).
11. Selecione Subscribe.

    Ao contrário da Parte 1 deste módulo, você deverá ver nenhuma mensagem depois de assinar `hello/world/counter`. Isso ocorre porque o código `greengrassHelloWorldCounter.py` que publica para o tópico `hello/world/counter` está dentro do manipulador de funções, que é executado somente quando a função é invocada.

    Neste módulo, você configurou a função `Greengrass_HelloWorld_Counter` do Lambda para ser invocada quando ela recebe uma mensagem MQTT no tópico `hello/world/counter/trigger`.

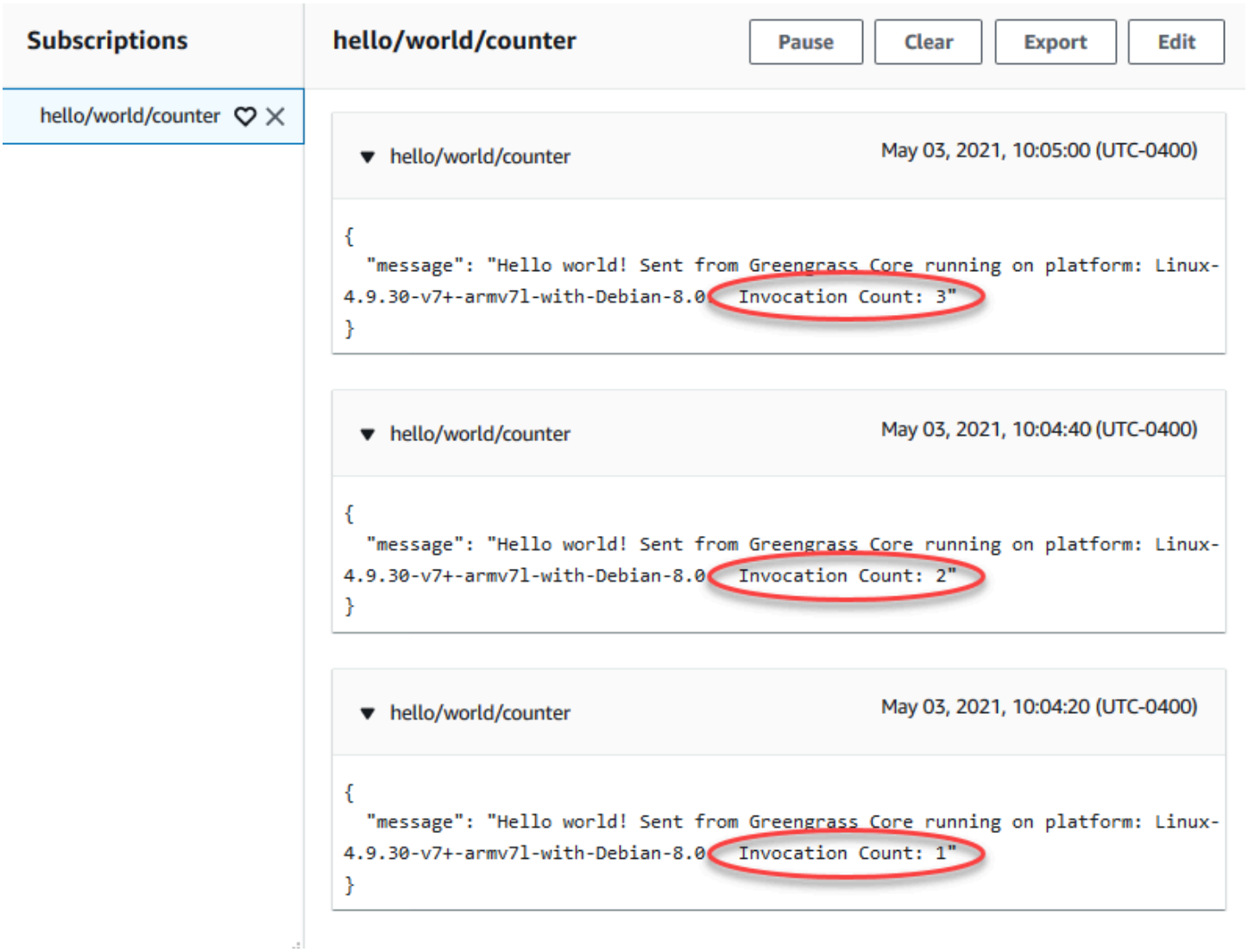
    Greengrass_HelloWorld_CounterA assinatura do IoT Cloud permite que a função envie mensagens AWS IoT sobre o `hello/world/counter` tópico. A Greengrass_HelloWorld_Counter assinatura do IoT Cloud permite AWS IoT enviar mensagens para a função sobre o `hello/world/counter/trigger` tópico.

12. Para testar o ciclo de vida de longa duração, invoque a função do Lambda publicando uma mensagem no tópico `hello/world/counter/trigger`. Você pode usar a mensagem padrão.



> **Note**
>
> A função `Greengrass_HelloWorld_Counter` ignora o conteúdo das mensagens recebidas. Ela só executa o código no `function_handler`, que envia uma mensagem ao tópico `hello/world/counter`. Você pode revisar esse código no AWS IoT Greengrass Core SDK para Python em. GitHub

Sempre que uma mensagem é publicada no tópico `hello/world/counter/trigger`, a variável `my_counter` é incrementada. Essa contagem de invocação é mostrada nas mensagens enviadas da função do Lambda. Como o manipulador de funções inclui um ciclo de hibernação de 20 segundos (`time.sleep(20)`), acionar repetidamente o manipulador enfileira as respostas do núcleo. AWS IoT Greengrass



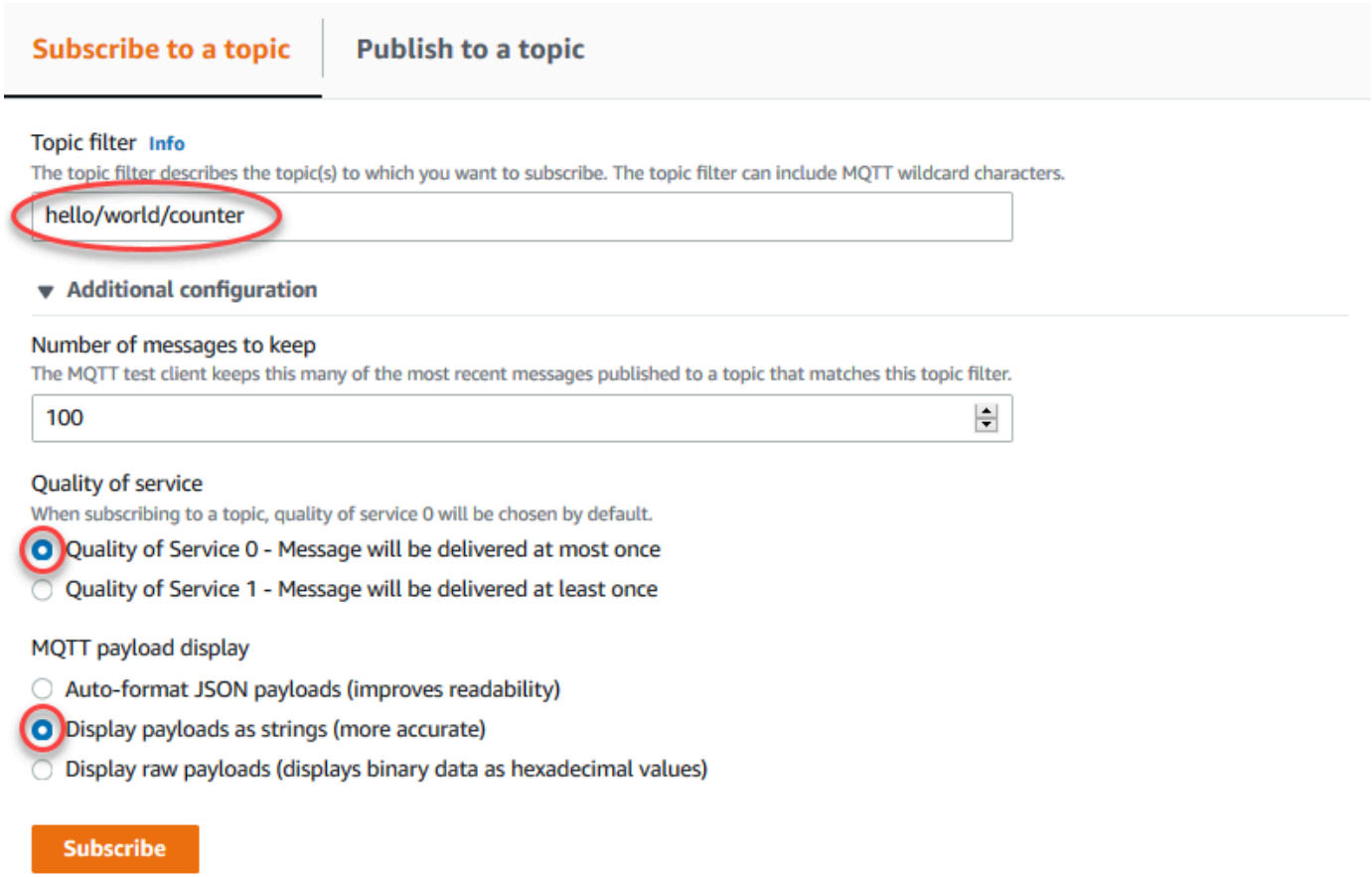
Teste as funções do Lambda sob demanda

Uma função Lambda sob demanda tem funcionalidade semelhante a uma função baseada em nuvem AWS Lambda . Várias invocações de uma função do Lambda sob demanda podem ser executadas em paralelo. Uma invocação da função do Lambda cria um contêiner separado para processar invocações ou reutiliza um contêiner existente caso os recursos permitam. Quaisquer variáveis ou pré-processamento definidos fora do manipulador de funções não serão mantidos quando os novos contêineres forem criados.

1. Na página de configuração do grupo, selecione a guia Funções do Lambda.
2. Em Minhas funções do Lambda, selecione a função Greengrass_HelloWorld_Counter do Lambda.
3. Na página de detalhes Greengrass_HelloWorld_Counter, escolha Editar.

4. Em Fixado, selecione Falso e, em seguida, selecione Salvar.
5. Na página de configuração do grupo, selecione Implantar.
6. Após a conclusão da implantação, retorne à página inicial do AWS IoT console e escolha Testar.
7. Configure os campos a seguir.

   - Em Subscription topic (Tópico de assinatura), insira **`hello/world/counter`**.
   - Para Quality of Service (Qualidade de Serviço), selecione 0.
   - Em MQTT payload display (Exibição de carga MQTT), selecione Display payloads as strings (Exibir cargas como strings).

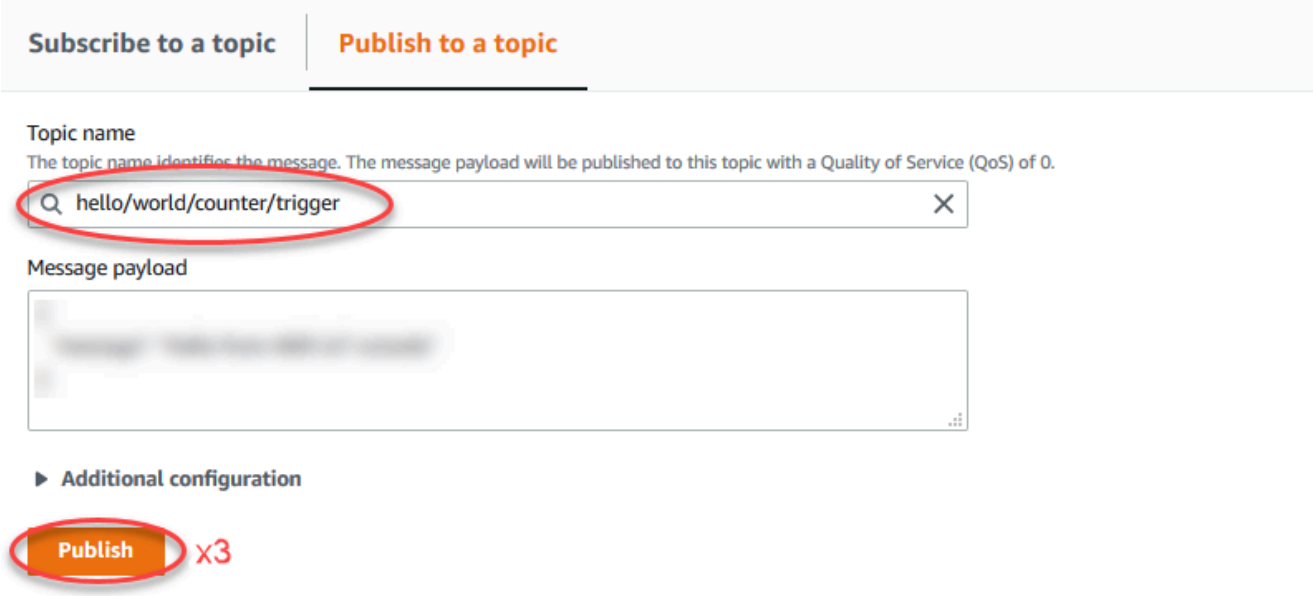


8. Selecione Subscribe.

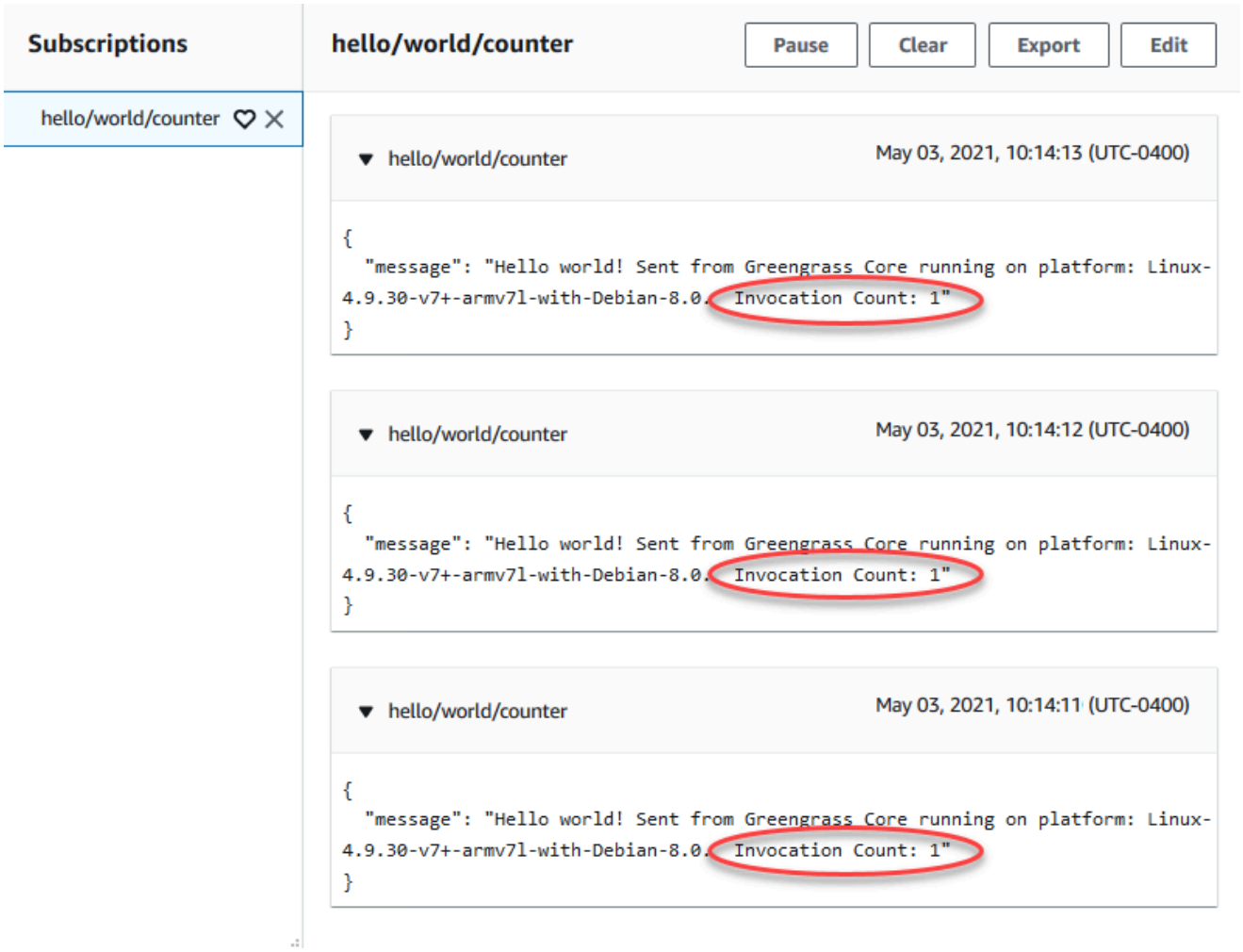
> **Note**
>
> Você não deverá ver nenhuma mensagem depois de fazer a assinatura.

9. Para testar o ciclo de vida sob demanda, invoque a função publicando uma mensagem no tópico `hello/world/counter/trigger`. Você pode usar a mensagem padrão.

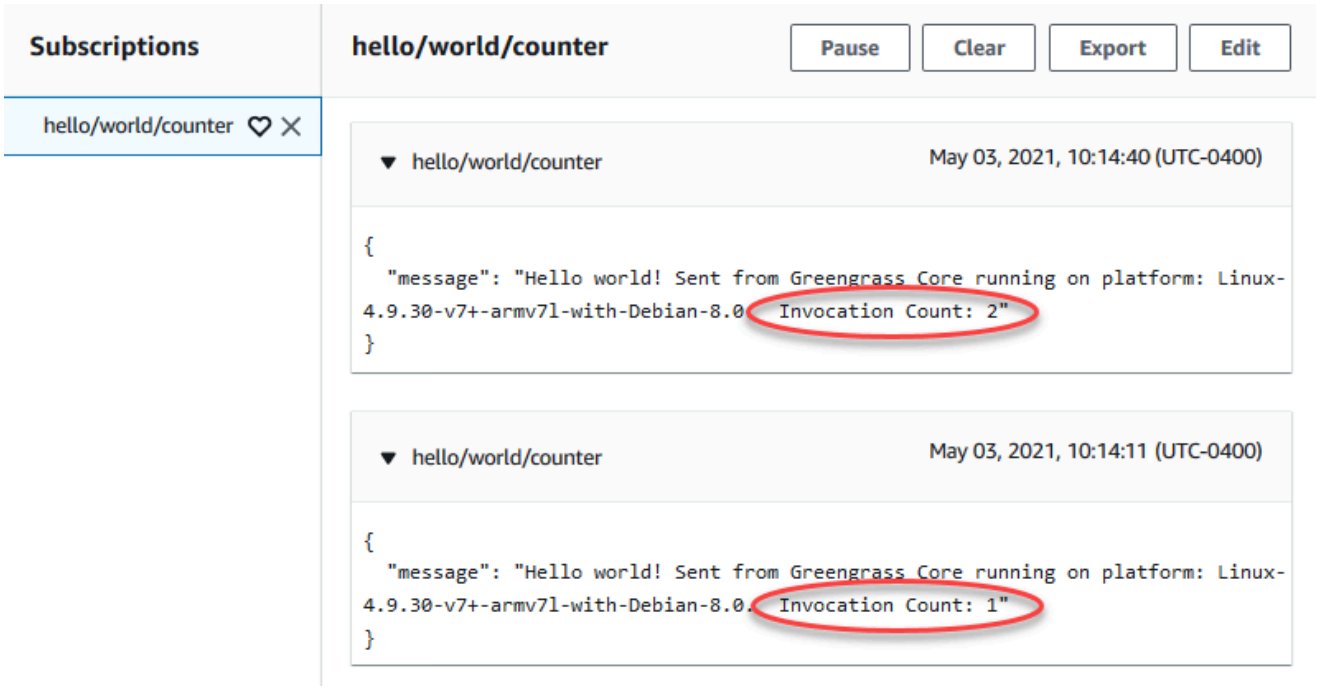
   a. Selecione Publicar três vezes rapidamente, dentro de cinco segundos entre cada vez que pressionar o botão.



   Cada publicação invoca o manipulador de funções e cria um contêiner para cada invocação. A contagem de invocações não é incrementada nas três vezes em que você acionou a função, pois cada função Lambda sob demanda tem a sua própria. container/sandbox



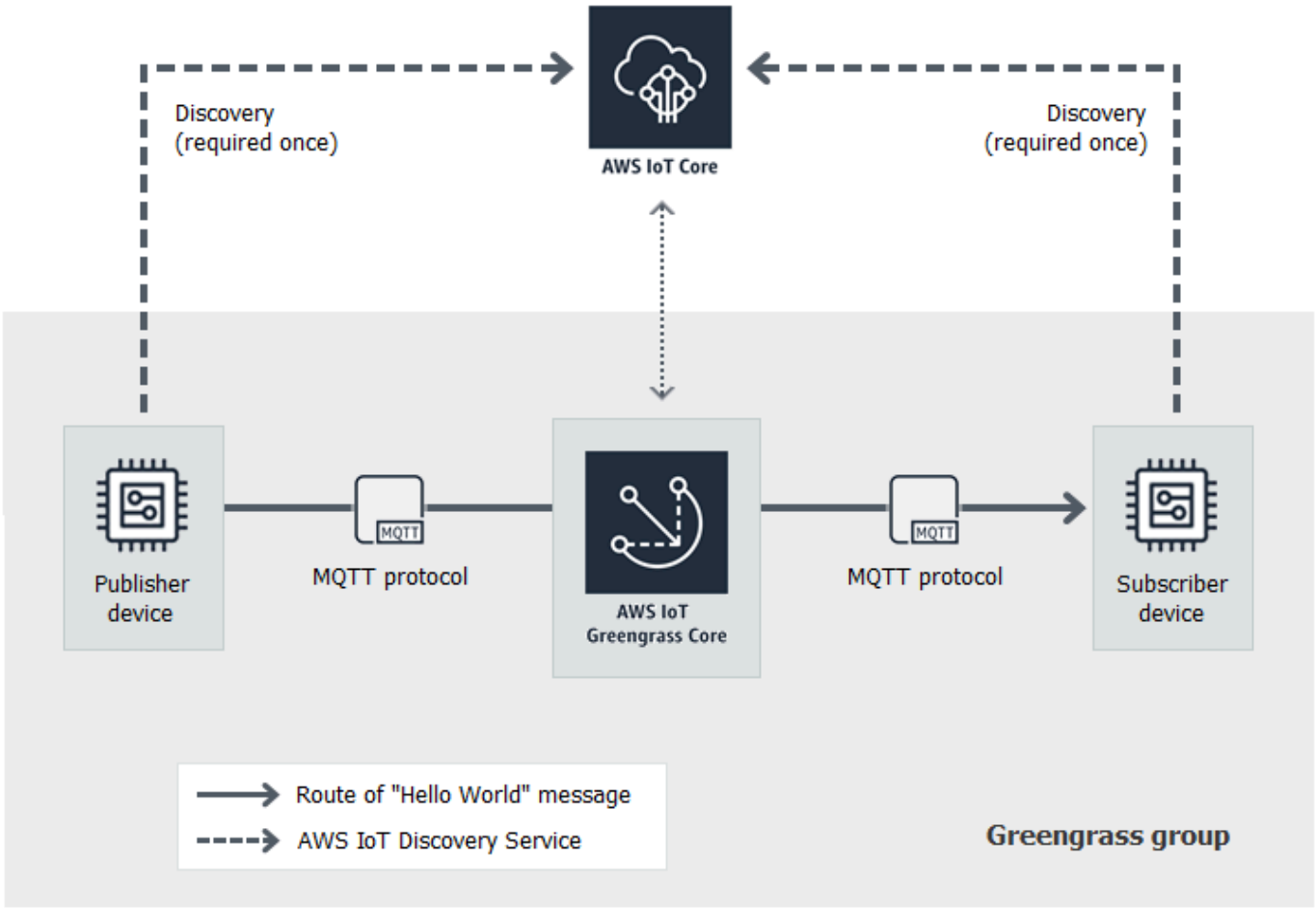
b. Depois de aproximadamente 30 segundos, selecione Publish to topic (Publicar em um tópico). A contagem de invocações deve ser incrementada para 2. Isso mostra que um contêiner, criado primeiro de uma invocação anterior está sendo reutilizado e que as variáveis de pré-processamento fora do manipulador de funções foram armazenadas.



Agora, você deve conhecer os dois tipos de funções do Lambda que podem ser executadas no núcleo AWS IoT Greengrass . O próximo módulo, Módulo 4, mostra como os dispositivos locais de IoT podem interagir em um AWS IoT Greengrass grupo.

Módulo 4: Interagindo com dispositivos clientes em um AWS IoT Greengrass grupo

Este módulo mostra como os dispositivos locais de IoT, chamados dispositivos ou dispositivos cliente, podem se conectar e se comunicar com um dispositivo AWS IoT Greengrass principal. Os dispositivos cliente que se conectam a um AWS IoT Greengrass núcleo fazem parte de um AWS IoT Greengrass grupo e podem participar do paradigma de AWS IoT Greengrass programação. Nesse módulo, um dispositivo cliente envia uma mensagem Hello World para outro dispositivo cliente no grupo do Greengrass:



Antes de começar, execute o script de configuração do dispositivo do Greengrass ou conclua o Módulo 1 e o Módulo 2. Este módulo cria dois dispositivos cliente simulados. Você não precisa de outros componentes ou dispositivos.

Este módulo deve demorar menos de 30 minutos para ser concluído.

Tópicos

- Crie dispositivos cliente em um AWS IoT Greengrass grupo
- Configurar assinaturas
- Instale o AWS IoT Device SDK para Python
- Testar comunicações

Crie dispositivos cliente em um AWS IoT Greengrass grupo

Nesta etapa, adicione dois dispositivos cliente ao grupo do Greengrass. Esse processo inclui registrar os dispositivos como AWS IoT itens e configurar certificados e chaves para permitir que eles se conectem. AWS IoT Greengrass

1. No painel de navegação do AWS IoT console, em Gerenciar, expanda dispositivos Greengrass e escolha Grupos (V1).
2. Selecione o grupo de destino.
3. Na página de configuração do grupo, selecione Dispositivos cliente e, em seguida, selecione Associar.
4. No modal Associar um dispositivo cliente a este grupo, escolha Criar AWS IoT coisa nova.

   A página Criar coisas é aberta em uma nova guia.
5. Na página Criar coisas, selecione Criar uma única coisa, em seguida, selecione Avançar.
6. Na página Especificar propriedades da coisa, registre esse dispositivo cliente como **`HelloWorld_Publisher`** e, em seguida selecione Avançar.
7. Na página Configurar certificado do dispositivo, selecione Avançar.
8. Na página Anexar políticas ao certificado, execute uma das seguintes ações:
   - Selecione uma política existente que conceda as permissões exigidas pelos dispositivos cliente e, em seguida, selecione Criar coisa.

     Um modal é aberto, onde você pode baixar os certificados e as chaves que o dispositivo usa para se conectar ao Nuvem AWS núcleo.
   - Crie e anexe uma nova política que conceda permissões ao dispositivo cliente. Faça o seguinte:

     a. Selecione Create policy (Criar política).

        A página Create policy (Criar política) é aberta em uma nova guia.

     b. Na página Create policy (Criar política) faça o seguinte:

        i. Em Nome da política, insira um nome que descreva a política, como **`GreengrassV1ClientDevicePolicy`**.

        ii. Na guia Instruções da política, em Documento da política, selecione JSON.

iii. Insira o seguinte documento de política. Essa política permite que o dispositivo cliente descubra os núcleos do Greengrass e se comunique sobre todos os tópicos do MQTT. Para obter informações sobre como restringir o acesso a essa política, consulte Autenticação e autorização de dispositivos para AWS IoT Greengrass.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish",
        "iot:Subscribe",
        "iot:Connect",
        "iot:Receive"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "greengrass:*"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

iv. Selecione Create (Criar) para criar a política.

c. Volte para a guia do navegador com a página Anexar políticas ao certificado aberta. Faça o seguinte:

i. Na lista Políticas, selecione a política que você criou, como GreengrassV1ClientDevicePolicy.

Se a política não for exibida, selecione o botão de atualização.

ii. Selecione Criar coisa.

Um modal é aberto, onde você pode baixar os certificados e as chaves que o dispositivo usa para se conectar ao Nuvem AWS núcleo.

9. No modal Baixar certificados e chaves, baixe os certificados do dispositivo.

> **Important**
>
> Antes de selecionar Concluído, faça download dos recursos de segurança.

Faça o seguinte:

a. Em Certificado do dispositivo, selecione Download para baixar o certificado do dispositivo.

b. Em Arquivo de chave pública, selecione Download para baixar a chave pública do certificado.

c. Em Arquivo de chave privada, selecione Download para baixar o arquivo de chave privada para o certificado.

d. Revise Autenticação do servidor no Guia do desenvolvedor da AWS IoT e, em seguida selecione o certificado de CA raiz apropriado. Recomendamos que você use endpoints do Amazon Trust Services (ATS) e certificados raiz da CA do ATS. Em Certificados CA raiz, selecione Download para obter um certificado de CA raiz.

e. Selecione Done (Concluído).

Anote a ID do certificado, que é comum nos nomes dos arquivos do certificado e das chaves do dispositivo. Você precisará disso mais tarde.

10. Volte para a guia do navegador com o modal Associar um dispositivo cliente a este grupo aberto. Faça o seguinte:

a. Para o nome da coisa AWS IoT , selecione a coisa HelloWorld_Publisher que você criou.

Se a coisa não for exibida, selecione o botão Atualizar.

b. Selecione Associar.

11. Repita as etapas 3 a 10 para adicionar um segundo dispositivo cliente ao grupo.

Dê um nome a esse dispositivo cliente **`HelloWorld_Subscriber`**. Faça download dos certificados e das chaves para esse dispositivo cliente em seu computador. Novamente, anote a ID do certificado que é comum nos nomes dos arquivos do dispositivo HelloWorld _Subscriber.

Agora você deve ter dois dispositivos cliente em seu grupo do Greengrass:

- HelloWorld_Editora
- HelloWorld_Assinante

12. Crie uma pasta no seu computador para as credenciais de segurança desses dispositivos cliente. Copie os certificados e as chaves para essa pasta.

Configurar assinaturas

Nesta etapa, você habilita o dispositivo cliente HelloWorld _Publisher a enviar mensagens MQTT para o dispositivo cliente HelloWorld _Subscriber.

1. Na página de configuração do grupo, selecione a guia Inscrições e, em seguida, selecione Adicionar.
2. Na página Criar uma assinatura, faça o seguinte para configurar a assinatura:

   a. Em Tipo de origem, escolha Dispositivo cliente e, em seguida, escolha HelloWorld_Publisher.

   b. Em Tipo de destino, escolha Dispositivo cliente e, em seguida, escolha HelloWorld_Assinante.

   c. Para Topic filter, insira **`hello/world/pubsub`**.

   > **Note**
   >
   > Você pode excluir assinaturas dos módulos anteriores. Na página Assinaturas do grupo, selecione as assinaturas a serem excluídas e, em seguida, selecione Excluir.

   d. Selecione Criar assinatura.
3. Certifique-se de que a detecção automática esteja habilitada para que o núcleo do Greengrass possa publicar uma lista de seus endereços IP. Os dispositivos cliente usam essa informação para descobrir o núcleo. Faça o seguinte:

   a. Na página de configuração do grupo, selecione a guia Funções do Lambda.

b. Em Funções do Lambda do sistema, selecione Detector de IP e, em seguida, selecione Editar.

c. Em Editar configurações do detector de IP, selecione Detectar e substituir automaticamente os endpoints do broker MQTT e, em seguida, selecione Salvar.

4. Verifique se o daemon do Greengrass está em execução, como descrito em Implantar configurações de nuvem em um dispositivo de núcleo.

5. Na página de configuração do grupo, selecione Implantar.

O status da implantação é exibido abaixo do nome do grupo no cabeçalho da página. Para ver os detalhes da implantação, selecione a guia Implantações.

Instale o AWS IoT Device SDK para Python

Os dispositivos cliente podem usar o AWS IoT Device SDK for Python para se comunicar com AWS IoT dispositivos AWS IoT Greengrass principais (usando a linguagem de programação Python). Para obter mais informações, incluindo requisitos, consulte o AWS IoT Device SDK Readme para Python em. GitHub

Nesta etapa, instale o SDK e obtenha a função de exemplo `basicDiscovery.py` usada pelos dispositivos cliente simulados em seu computador.

1. Para instalar o SDK no computador, com todos os componentes necessários, selecione o sistema operacional:

   Windows

   1. Abra um prompt de comando com privilégios elevados e execute o comando a seguir:

      ```
      python --version
      ```

      Se nenhuma informação sobre versão for retornada ou se o número da versão for menor do que 2.7 para o Python 2 ou menor do que 3.3 para o Python 3, siga as instruções em Fazer download do Python para instalar o Python 2.7 ou superior ou o Python 3.3 ou superior. Para obter mais informações, consulte Como usar o Python no Windows.

   2. Faça download do AWS IoT Device SDK para Python como arquivo `zip` e extraia-o para um local apropriado em seu computador.

Anote o caminho do arquivo para a pasta extraída `aws-iot-device-sdk-python-master` que contém o arquivo `setup.py`. Na próxima etapa, esse caminho de arquivo é indicado por*`path-to-SDK-folder`*.

3. No prompt de comandos com privilégios elevados, execute o seguinte:

```
cd path-to-SDK-folder
python setup.py install
```

macOS

1. Abra uma janela do terminal e execute o seguinte comando:

```
python --version
```

Se nenhuma informação sobre versão for retornada ou se o número da versão for menor do que 2.7 para o Python 2 ou menor do que 3.3 para o Python 3, siga as instruções em Fazer download do Python para instalar o Python 2.7 ou superior ou o Python 3.3 ou superior. Para obter mais informações, consulte Como usar o Python em um Macintosh.

2. Na janela do terminal, execute os seguintes comandos para determinar a versão do OpenSSL:

```
python
>>>import ssl
>>>print ssl.OPENSSL_VERSION
```

Anote o valor da versão do OpenSSL.

> **Note**
>
> Se você estiver executando o Python 3, use print(ssl.OPENSSL_VERSION).

Para fechar o shell do Python, execute o seguinte comando:

```
>>>exit()
```

Se a versão do OpenSSL for 1.0.1 ou posterior, vá para a etapa c. Do contrário, siga estas etapas:

- Na janela do terminal, execute o seguinte comando para determinar se o computador está usando o Simple Python Version Management:

```
which pyenv
```

Se um caminho de arquivo for retornado, selecione a guia Using **pyenv** (Usando). Se nada for retornado, selecione a guia Not using **pyenv** (Não está usando).

Using pyenv

1. Consulte Python Releases for macOS X (ou semelhante) para determinar a versão estável mais recente do Python. No exemplo a seguir, esse valor é indicado por*latest-Python-version*.
2. Na janela do terminal, execute os seguintes comandos:

```
pyenv install latest-Python-version
pyenv global latest-Python-version
```

   Por exemplo, se a versão mais recente do Python 2 for a 2.7.14, esses comandos serão:

```
pyenv install 2.7.14
pyenv global 2.7.14
```

3. Feche e então reabra a janela de terminal e execute os seguintes comandos:

```
python
>>>import ssl
>>>print ssl.OPENSSL_VERSION
```

   A versão do OpenSSL deve ser no mínimo 1.0.1. Se a versão for inferior a 1.0.1, então a atualização falhou. Verifique o valor da versão do Python usada nos comandos pyenv install e pyenv global e tente novamente.
4. Execute o seguinte comando para sair do shell do Python:

```
exit()
```

Not using pyenv

1. Em uma janela do terminal, execute o seguinte comando para determinar se o brew está instalado:

   ```
   which brew
   ```

   Se o caminho de um arquivo não for retornado, instale o `brew` da seguinte forma:

   ```
   /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/
   Homebrew/install/master/install)"
   ```

   > **Note**
   >
   > Siga as instruções de instalação. O download para as ferramentas de linha de comando do Xcode podem demorar algum tempo.

2. Execute os seguintes comandos :

   ```
   brew update
   brew install openssl
   brew install python@2
   ```

   O AWS IoT Device SDK for Python requer a versão 1.0.1 (ou posterior) do OpenSSL compilada com o executável Python. O comando brew install python instala um executável do `python2` que atende a esse requisito. O executável `python2` é instalado no diretório `/usr/local/bin`, que deve fazer parte da variável de ambiente `PATH`. Para confirmar, execute o seguinte comando:

   ```
   python2 --version
   ```

   Se as informações de versão do `python2` forem fornecidas, vá para a próxima etapa. Caso contrário, adicione permanentemente o caminho `/usr/local/bin` à sua variável de ambiente `PATH` anexando a seguinte linha em seu perfil de shell:

```
export PATH="/usr/local/bin:$PATH"
```

Por exemplo, se estiver usando `.bash_profile` ou ainda não tiver um perfil de shell, execute o seguinte comando em uma janela do terminal:

```
echo 'export PATH="/usr/local/bin:$PATH"' >> ~/.bash_profile
```

Em seguida, execute source em seu perfil e confirme se o `python2 --version` fornece informações sobre versão. Por exemplo, se estiver usando `.bash_profile`, execute os seguintes comandos:

```
source ~/.bash_profile
python2 --version
```

As informações da versão do `python2` devem ser retornadas.

3. Acrescente a seguinte linha a seu perfil de shell:

```
alias python="python2"
```

Por exemplo, se estiver usando `.bash_profile` ou ainda não tiver um perfil de shell, execute o seguinte comando:

```
echo 'alias python="python2"' >> ~/.bash_profile
```

4. Em seguida, execute source em seu perfil de shell. Por exemplo, se estiver usando `.bash_profile`, execute o seguinte comando:

```
source ~/.bash_profile
```

A invocação do comando python executa o executável do Python que contém a versão do OpenSSL (`python2`).

5. Execute os seguintes comandos :

```
python
 import ssl
 print ssl.OPENSSL_VERSION
```

A versão do OpenSSL deve ser a 1.0.1 ou posterior.

6. Para sair do shell do Python, execute o seguinte comando:

```
exit()
```

3. Execute os seguintes comandos para instalar o AWS IoT Device SDK para Python:

```
cd ~
git clone https://github.com/aws/aws-iot-device-sdk-python.git
cd aws-iot-device-sdk-python
sudo python setup.py install
```

UNIX-like system

1. Na janela de terminal do , execute o seguinte comando:

```
python --version
```

Se nenhuma informação sobre versão for retornada ou se o número da versão for menor do que 2.7 para o Python 2 ou menor do que 3.3 para o Python 3, siga as instruções em Fazer download do Python para instalar o Python 2.7 ou superior ou o Python 3.3 ou superior. Para obter mais informações, consulte Como usar o Python em plataformas Unix.

2. No terminal, execute os seguintes comandos para determinar a versão do OpenSSL:

```
python
>>>import ssl
>>>print ssl.OPENSSL_VERSION
```

Anote o valor da versão do OpenSSL.

> **Note**
>
> Se você estiver executando o Python 3, use print(ssl.OPENSSL_VERSION).

Para fechar o shell do Python, execute o seguinte comando:

```
exit()
```

Se a versão do OpenSSL for 1.0.1 ou posterior, vá para a próxima etapa. Caso contrário, execute o(s) comando(s) para atualizar o OpenSSL para sua distribuição (por exemplo, `sudo yum update openssl`, `sudo apt-get update` e assim por diante).

Confirme se a versão do OpenSSL é a 1.0.1 ou posterior executando os seguintes comandos:

```
python
>>>import ssl
>>>print ssl.OPENSSL_VERSION
>>>exit()
```

3. Execute os seguintes comandos para instalar o AWS IoT Device SDK para Python:

```
cd ~
git clone https://github.com/aws/aws-iot-device-sdk-python.git
cd aws-iot-device-sdk-python
sudo python setup.py install
```

2. Depois que o AWS IoT Device SDK for Python for instalado, navegue até a `samples` pasta e abra a `greengrass` pasta.

   Para este tutorial, você copiará a função de exemplo `basicDiscovery.py`, que usa os certificados e as chaves baixadas em the section called “Crie dispositivos cliente em um AWS IoT Greengrass grupo”.

3. Copie `basicDiscovery.py` para a pasta que contém os certificados e chaves dos dispositivos HelloWorld _Publisher e HelloWorld _Subscriber.

Testar comunicações

1. Certifique-se de que seu computador e o dispositivo AWS IoT Greengrass principal estejam conectados à Internet usando a mesma rede.

   a. No dispositivo AWS IoT Greengrass principal, execute o comando a seguir para encontrar seu endereço IP.

```
hostname -I
```

b. Em seu computador, execute o comando a seguir usando o endereço IP do núcleo. Você pode usar Ctrl + C para interromper o comando ping.

```
ping IP-address
```

Uma saída semelhante à seguinte indica uma comunicação bem-sucedida entre o computador e o dispositivo AWS IoT Greengrass principal (0% de perda de pacotes):

```
$ping 176.32.103.205
PING 176.32.103.205 (176.32.103.205) 56(84) bytes of data.
64 bytes from 176.32.103.205: icmp_seq=1 ttl=230 time=77.2 ms
64 bytes from 176.32.103.205: icmp_seq=2 ttl=230 time=77.1 ms
64 bytes from 176.32.103.205: icmp_seq=3 ttl=230 time=77.1 ms
64 bytes from 176.32.103.205: icmp_seq=4 ttl=230 time=77.1 ms
64 bytes from 176.32.103.205: icmp_seq=5 ttl=230 time=77.1 ms
64 bytes from 176.32.103.205: icmp_seq=6 ttl=230 time=77.1 ms
^C
--- 176.32.103.205 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5549ms
rtt min/avg/max/mdev = 77.107/77.172/77.256/0.361 ms
```

> **Note**
>
> Se você não conseguir executar ping em uma instância do EC2 em execução AWS IoT Greengrass, certifique-se de que as regras do grupo de segurança de entrada da instância permitam tráfego ICMP para mensagens de solicitação do Echo. Para ter mais informações, consulte Adding rules to a security group no Manual do usuário do EC2.
>
> Em computadores host do Windows, no aplicativo Firewall do Windows com Segurança Avançada, talvez você também precise habilitar uma regra de entrada que permita solicitações de eco de entrada (por exemplo, Compartilhamento de arquivos e impressoras (Solicitação de eco - ICMPv4-In)) ou criar uma.

2. Obtenha seu AWS IoT endpoint.

a. No painel de navegação do console do AWS IoT, selecione Configurações.

b. Em Endpoint de dados do dispositivo, anote o valor do Endpoint. Você usa esse valor para substituir o *AWS_IOT_ENDPOINT* espaço reservado nos comandos nas etapas a seguir.

> **Note**
>
> Certifique-se de que os endpoints correspondem ao seu tipo de certificado.

3. No seu computador (não no dispositivo AWS IoT Greengrass principal), abra duas janelas de linha de comando (terminal ou prompt de comando). Uma janela representa o dispositivo cliente HelloWorld _Publisher e a outra representa o dispositivo cliente HelloWorld _Subscriber.

   Após a execução, `basicDiscovery.py` tenta coletar informações sobre a localização do AWS IoT Greengrass núcleo em seus terminais. Essas informações são armazenadas depois que o dispositivo cliente descobre e se conecta ao núcleo. Isso permite que mensagens e operações futuras sejam executadas localmente (sem a necessidade de uma conexão com a internet).

   > **Note**
   >
   > Os IDs de cliente usados para conexões MQTT devem corresponder ao nome do dispositivo cliente. O script `basicDiscovery.py` define o ID do cliente para conexões MQTT com o nome do item que você especifica ao executar o script.
   > Execute o seguinte comando da pasta que contém o arquivo `basicDiscovery.py` para obter informações detalhadas sobre o uso de scripts:
   >
   > ```
   > python basicDiscovery.py --help
   > ```

4. Na janela do dispositivo cliente HelloWorld _Publisher, execute os seguintes comandos.

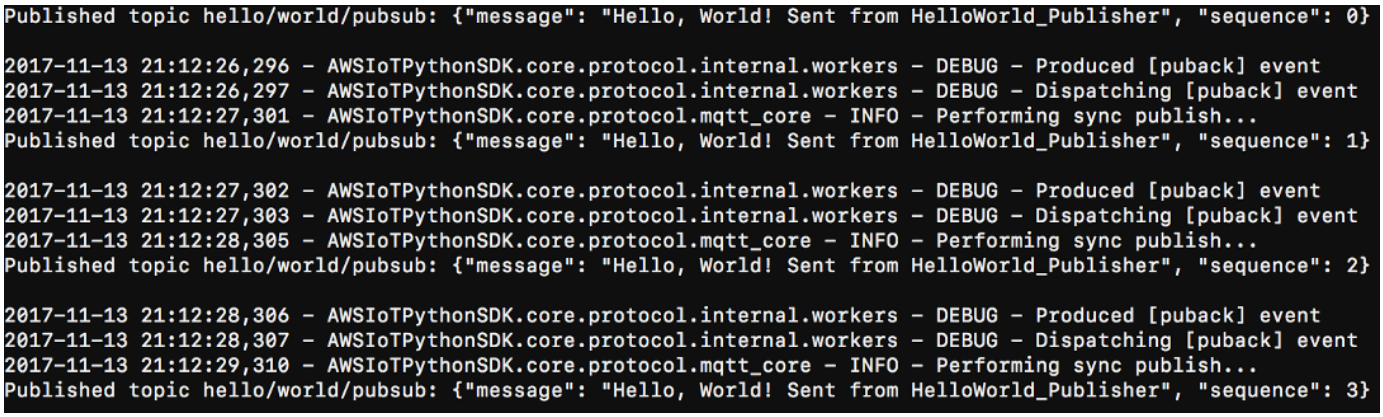
   - *path-to-certs-folder*Substitua pelo caminho para a pasta que contém os certificados, as chaves `basicDiscovery.py` e.
   - *AWS_IOT_ENDPOINT*Substitua pelo seu endpoint.
   - Substitua as duas *publisherCertId* instâncias pelo ID do certificado no nome do arquivo do seu dispositivo cliente HelloWorld _Publisher.

```
cd path-to-certs-folder
python basicDiscovery.py --endpoint AWS_IOT_ENDPOINT --rootCA AmazonRootCA1.pem
 --cert publisherCertId-certificate.pem.crt --key publisherCertId-private.pem.key
```

```
 --thingName HelloWorld_Publisher --topic 'hello/world/pubsub' --mode publish --
message 'Hello, World! Sent from HelloWorld_Publisher'
```

Você deverá ver uma saída semelhante à seguinte, que inclui entradas como `Published topic 'hello/world/pubsub': {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 1}`.

> **Note**
>
> Se o script retornar uma mensagem `error: unrecognized arguments`, altere as aspas simples para aspas duplas para os parâmetros `--topic` e `--message` e execute o comando novamente.
> Para solucionar problemas de conexão, você pode tentar usar a detecção manual de IP.



5. Na janela do dispositivo cliente HelloWorld _Subscriber, execute os seguintes comandos.

   - *`path-to-certs-folder`*Substitua pelo caminho para a pasta que contém os certificados, as chaves `basicDiscovery.py` e.
   - *`AWS_IOT_ENDPOINT`*Substitua pelo seu endpoint.
   - Substitua as duas *`subscriberCertId`* instâncias pelo ID do certificado no nome do arquivo do seu dispositivo cliente HelloWorld _Subscriber.

```
cd path-to-certs-folder
python basicDiscovery.py --endpoint AWS_IOT_ENDPOINT --rootCA AmazonRootCA1.pem --
cert subscriberCertId-certificate.pem.crt --key subscriberCertId-private.pem.key --
thingName HelloWorld_Subscriber --topic 'hello/world/pubsub' --mode subscribe
```

Você deverá ver a seguinte saída, que inclui entradas como `Received message on topic hello/world/pubsub: {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 1}`.

```
Received message on topic hello/world/pubsub: {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 0}

2017-11-13 21:12:27,435 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Produced [message] event
2017-11-13 21:12:27,435 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Dispatching [message] event
2017-11-13 21:12:27,436 - AWSIoTPythonSDK.core.protocol.internal.clients - DEBUG - Invoking custom event callback...
Received message on topic hello/world/pubsub: {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 1}

2017-11-13 21:12:28,320 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Produced [message] event
2017-11-13 21:12:28,324 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Dispatching [message] event
2017-11-13 21:12:28,324 - AWSIoTPythonSDK.core.protocol.internal.clients - DEBUG - Invoking custom event callback...
Received message on topic hello/world/pubsub: {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 2}

2017-11-13 21:12:29,547 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Produced [message] event
2017-11-13 21:12:29,552 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Dispatching [message] event
2017-11-13 21:12:29,552 - AWSIoTPythonSDK.core.protocol.internal.clients - DEBUG - Invoking custom event callback...
```

Feche a janela HelloWorld_Publisher para impedir que as mensagens sejam acumuladas na janela HelloWorld_Subscriber.

Testar em uma rede corporativa pode interferir com a conexão ao núcleo. Como alternativa, você pode inserir manualmente o endpoint. Isso garante que o `basicDiscovery.py` script se conecte ao endereço IP correto do dispositivo AWS IoT Greengrass principal.

Para inserir o endpoint manualmente

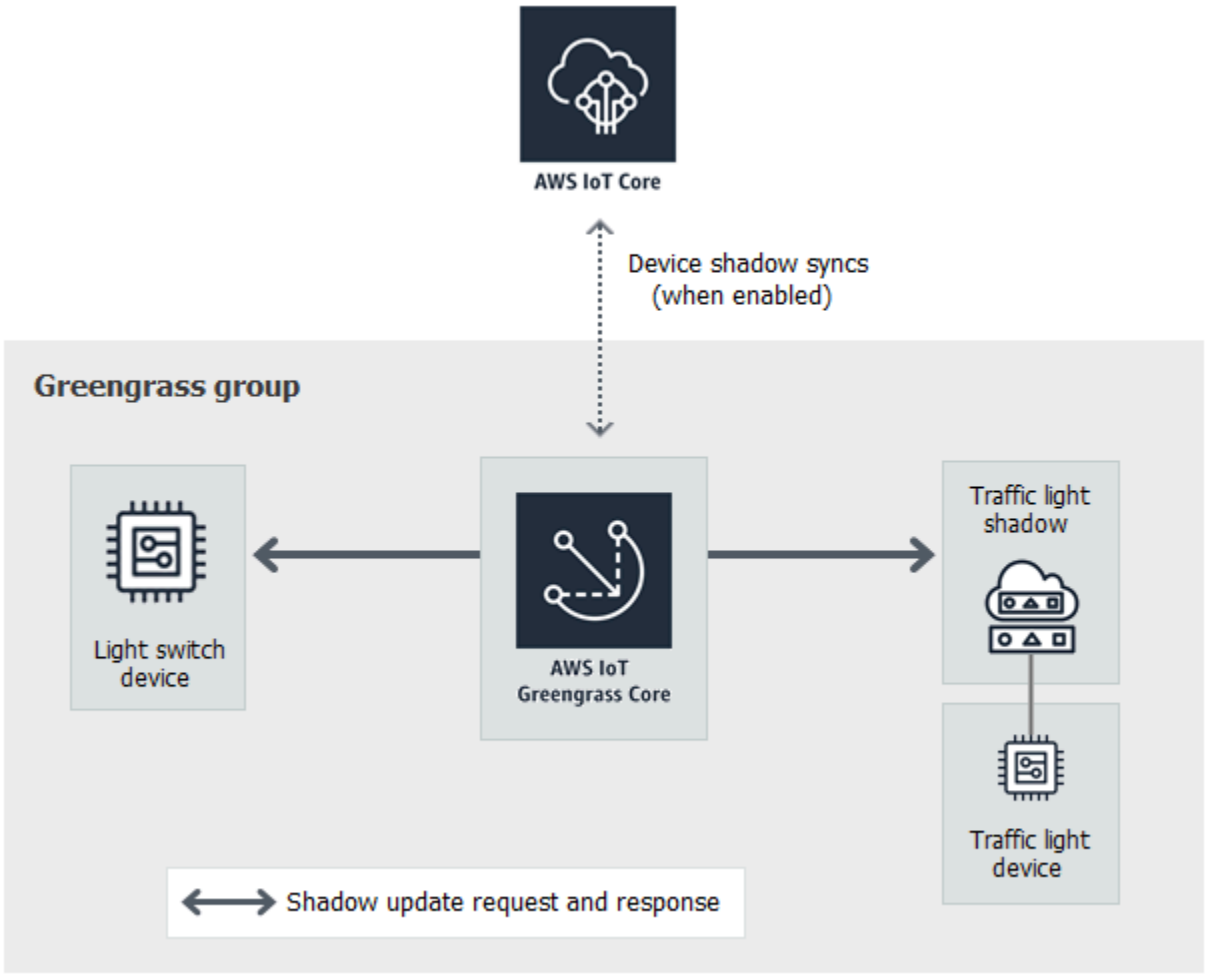
1. No painel de navegação do AWS IoT console, em Gerenciar, expanda dispositivos Greengrass e escolha Grupos (V1).
2. Em Grupos do Greengrass, selecione seu grupo.
3. Configure o núcleo para gerenciar manualmente os endpoints do corretor MQTT. Faça o seguinte:
   a. Na página de configuração do grupo, selecione a guia Funções do Lambda.
   b. Em Funções do Lambda do sistema, selecione Detector de IP e, em seguida, selecione Editar.
   c. Em Editar configurações do detector de IP, selecione Gerenciar manualmente os endpoints do corretor MQTT e, em seguida, selecione Salvar.
4. Insira o endpoint do corretor MQTT para o núcleo. Faça o seguinte:
   a. Em Visão geral, selecione Núcleo do Greengrass.
   b. Em Endpoints do corretor MQTT, selecione Gerenciar endpoints.

c. Selecione Adicionar endpoint e verifique se você tem apenas um valor de endpoint. Esse valor deve ser o endpoint do endereço IP da porta 8883 do seu dispositivo AWS IoT Greengrass principal (por exemplo,`192.168.1.4`).

d. Selecione Atualizar.

Módulo 5: Interagir com sombras de dispositivos

Este módulo avançado mostra como os dispositivos cliente podem interagir com as sombras AWS IoT do dispositivo em um AWS IoT Greengrass grupo. Sombra é um documento JSON usado para armazenar informações sobre o estado atual ou o desejado de uma coisa. Neste módulo, você aprenderá como um dispositivo cliente (`GG_Switch`) pode modificar o estado de outro dispositivo cliente (`GG_TrafficLight`), e como esses estados podem ser sincronizados com a nuvem do AWS IoT Greengrass :



Antes de começar, execute o script de configuração do dispositivo do Greengrass ou conclua o Módulo 1 e o Módulo 2. Você também deve entender como conectar dispositivos cliente a um AWS IoT Greengrass núcleo (Módulo 4). Você não precisa de outros componentes ou dispositivos.

Este modulo levará aproximadamente 30 minutos para ser concluído.

Tópicos

- Configurar dispositivos e assinaturas
- Fazer download de arquivos necessários
- Testar comunicações (sincronizações de dispositivos desativadas)
- Testar comunicações (sincronizações de dispositivos ativadas)

Configurar dispositivos e assinaturas

As sombras podem ser sincronizadas com AWS IoT quando o AWS IoT Greengrass núcleo está conectado à Internet. Neste módulo, você primeiro usará sombras locais sem sincronizar com a nuvem. Em seguida, você habilita a sincronização de nuvem.

Cada dispositivo cliente tem sua própria sombra. Para obter mais informações, consulte Serviço de sombra do dispositivo para AWS IoT no Guia do desenvolvedor do AWS IoT .

1. Na página de configuração do grupo, selecione a guia Dispositivos cliente.
2. Na guia Dispositivos clientes, adicione dois novos dispositivos clientes ao seu AWS IoT Greengrass grupo. Para ver as etapas detalhadas desse processo, consulte the section called “Crie dispositivos cliente em um AWS IoT Greengrass grupo”.
   - Nomeie os dispositivos cliente **`GG_Switch`** e **`GG_TrafficLight`**:
   - Gere e faça download dos recursos de segurança para ambos os dispositivos cliente.
   - Anote o ID do certificado nos nomes de arquivo dos recursos de segurança para os dispositivos cliente. Você usará esses valores depois.
3. Crie uma pasta no seu computador para as credenciais de segurança desses dispositivos cliente. Copie os certificados e as chaves para essa pasta.
4. Certifique-se de que os dispositivos cliente estejam configurados para usar sombras locais e de que eles não se sincronizem com o Nuvem AWS. Caso contrário, selecione o dispositivo cliente, selecione Sincronizar sombra e, em seguida, selecione Desativar sincronização da sombra com a nuvem.

5. Adicione as assinaturas na tabela a seguir ao seu grupo. Por exemplo, para criar a primeira assinatura:

   a. Na página de configuração do grupo, selecione a guia Inscrições e, em seguida, selecione Adicionar.

   b. Em Tipo de origem, selecione Dispositivo cliente e, em seguida, selecione GG_switch.

   c. Para Tipo de destino, selecione Serviço e Serviço de sombra local.

   d. Em Topic filter (Filtro de tópicos), insira **`$aws/things/GG_TrafficLight/shadow/update`**.

   e. Selecione Criar assinatura.

   Os tópicos devem ser inseridos exatamente como mostrado na tabela. Embora seja possível usar curingas para consolidar algumas das assinaturas, não recomendamos essa prática. Para obter mais informações, consulte Tópicos da sombra MQTT no Guia do desenvolvedor do AWS IoT .

Fonte	Destino	Tópico	Observações
GG_Switch	Serviço de shadow local	$aws/things/GG_TrafficLight/shadow/atualização	O GG_Switch envia uma solicitação de atualização para atualizar tópico.
Serviço de shadow local	GG_Switch	$aws/things/GG_TrafficLight/shadow/update/accepted	O GG_Switch precisa saber se a solicitação de atualização foi aceita.
Serviço de shadow local	GG_Switch	$aws/things/GG_TrafficLight/shadow/update/rejected	O GG_Switch precisa saber se a solicitação de atualização foi rejeitada.

Fonte	Destino	Tópico	Observações
GG_ TrafficLight	Serviço de shadow local	$aws/things/GG_TrafficLight/shadow/atualização	O GG_ TrafficLight envia uma atualização de seu estado para o tópico de atualização.
Serviço de shadow local	GG_ TrafficLight	$aws/things/GG_TrafficLight/shadow/update/delta	O Local Shadow Service envia uma atualização recebida para GG_ TrafficLight por meio do tópico delta.
Serviço de shadow local	GG_ TrafficLight	$aws/things/GG_TrafficLight/shadow/update/accepted	O GG_ TrafficLight precisa saber se sua atualização de estado foi aceita.
Serviço de shadow local	GG_ TrafficLight	$aws/things/GG_TrafficLight/shadow/update/rejected	O GG_ TrafficLight precisa saber se sua atualização de estado foi rejeitada.

As nova assinaturas são exibidas na guia Assinaturas.

> **Note**
>
> Para obter informações sobre o caractere $, consulte Tópicos reservados.

6. Certifique-se de que a detecção automática esteja habilitada para que o núcleo do Greengrass possa publicar uma lista de seus endereços IP. Os dispositivos cliente usam essa informação para descobrir o núcleo. Faça o seguinte:

   a. Na página de configuração do grupo, selecione a guia Funções do Lambda.

b. Em Funções do Lambda do sistema, selecione Detector de IP e, em seguida, selecione Editar.

c. Em Editar configurações do detector de IP, selecione Detectar e substituir automaticamente os endpoints do broker MQTT e, em seguida, selecione Salvar.

7. Verifique se o daemon do Greengrass está em execução, como descrito em Implantar configurações de nuvem em um dispositivo de núcleo.

8. Na página de configuração do grupo, selecione Implantar.

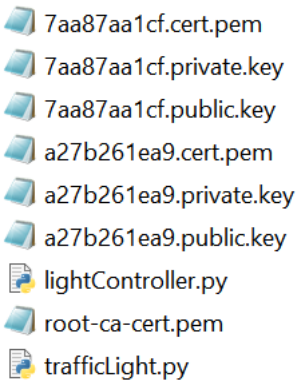
Fazer download de arquivos necessários

1. Se você ainda não fez isso, instale o AWS IoT Device SDK para Python. Para obter instruções, consulte a etapa 1 em the section called “Instale o AWS IoT Device SDK para Python”.

   Esse SDK é usado por dispositivos clientes para se comunicar com AWS IoT e com dispositivos AWS IoT Greengrass principais.

2. Da pasta de TrafficLight exemplos em diante GitHub, baixe os `trafficLight.py` arquivos `lightController.py` e para o seu computador. Salve-os na pasta que contém os certificados e chaves do dispositivo TrafficLight cliente GG_Switch e GG_.

   O `lightController.py` script corresponde ao dispositivo cliente GG_Switch e o `trafficLight.py` script corresponde ao dispositivo cliente GG_TrafficLight .



> **Note**
>
> Os arquivos Python de exemplo são armazenados no repositório AWS IoT Greengrass Core SDK para Python por conveniência, mas eles não usam o Core SDK. AWS IoT Greengrass

Testar comunicações (sincronizações de dispositivos desativadas)

1. Certifique-se de que seu computador e o dispositivo AWS IoT Greengrass principal estejam conectados à Internet usando a mesma rede.

   a. No dispositivo AWS IoT Greengrass principal, execute o comando a seguir para encontrar seu endereço IP.

   ```
   hostname -I
   ```

   b. Em seu computador, execute o comando a seguir usando o endereço IP do núcleo. Você pode usar Ctrl + C para interromper o comando ping.

   ```
   ping IP-address
   ```

   Uma saída semelhante à seguinte indica uma comunicação bem-sucedida entre o computador e o dispositivo AWS IoT Greengrass principal (0% de perda de pacotes):

   ```
   $ping 176.32.103.205
   PING 176.32.103.205 (176.32.103.205) 56(84) bytes of data.
   64 bytes from 176.32.103.205: icmp_seq=1 ttl=230 time=77.2 ms
   64 bytes from 176.32.103.205: icmp_seq=2 ttl=230 time=77.1 ms
   64 bytes from 176.32.103.205: icmp_seq=3 ttl=230 time=77.1 ms
   64 bytes from 176.32.103.205: icmp_seq=4 ttl=230 time=77.1 ms
   64 bytes from 176.32.103.205: icmp_seq=5 ttl=230 time=77.1 ms
   64 bytes from 176.32.103.205: icmp_seq=6 ttl=230 time=77.1 ms
   ^C
   --- 176.32.103.205 ping statistics ---
   6 packets transmitted, 6 received, 0% packet loss, time 5549ms
   rtt min/avg/max/mdev = 77.107/77.172/77.256/0.361 ms
   ```

   > **Note**
   >
   > Se você não conseguir executar ping em uma instância do EC2 em execução AWS IoT Greengrass, certifique-se de que as regras do grupo de segurança de entrada da instância permitam tráfego ICMP para mensagens de solicitação do Echo. Para ter mais informações, consulte Adding rules to a security group no Manual do usuário do EC2.
   >
   > Em computadores host do Windows, no aplicativo Firewall do Windows com Segurança Avançada, talvez você também precise habilitar uma regra de entrada

que permita solicitações de eco de entrada (por exemplo, Compartilhamento de arquivos e impressoras (Solicitação de eco - ICMPv4-In)) ou criar uma.

2. Obtenha seu AWS IoT endpoint.

   a. No painel de navegação do console do AWS IoT, selecione Configurações.

   b. Em Endpoint de dados do dispositivo, anote o valor do Endpoint. Você usa esse valor para substituir o *AWS_IOT_ENDPOINT* espaço reservado nos comandos nas etapas a seguir.

   > **Note**
   >
   > Certifique-se de que os endpoints correspondem ao seu tipo de certificado.

3. No seu computador (não no dispositivo AWS IoT Greengrass principal), abra duas janelas de linha de comando (terminal ou prompt de comando). Uma janela representa o dispositivo cliente GG_Switch e a outra representa o dispositivo cliente GG_TrafficLight .

   a. Na janela do dispositivo cliente GG_Switch, execute os comandos a seguir.

   - *path-to-certs-folder*Substitua pelo caminho para a pasta que contém os certificados, as chaves e os arquivos Python.
   - *AWS_IOT_ENDPOINT*Substitua pelo seu endpoint.
   - Substitua as duas *switchCertId* instâncias pelo ID do certificado no nome do arquivo do seu dispositivo cliente GG_Switch.

   ```
   cd path-to-certs-folder
   python lightController.py --endpoint AWS_IOT_ENDPOINT --rootCA
    AmazonRootCA1.pem --cert switchCertId-certificate.pem.crt --key switchCertId-
   private.pem.key --thingName GG_TrafficLight --clientId GG_Switch
   ```

   b. Na janela do dispositivo TrafficLight cliente GG_, execute os seguintes comandos.

   - *path-to-certs-folder*Substitua pelo caminho para a pasta que contém os certificados, as chaves e os arquivos Python.
   - *AWS_IOT_ENDPOINT*Substitua pelo seu endpoint.
   - Substitua as duas *lightCertId* instâncias pelo ID do certificado no nome do arquivo do seu dispositivo TrafficLight cliente GG_.

```
cd path-to-certs-folder
python trafficLight.py --endpoint AWS_IOT_ENDPOINT --rootCA AmazonRootCA1.pem
 --cert lightCertId-certificate.pem.crt --key lightCertId-private.pem.key --
thingName GG_TrafficLight --clientId GG_TrafficLight
```

A cada 20 segundos, o interruptor atualiza o estado de shadow para G, Y e R, e a luz exibe seu novo estado, conforme a seguir.

Resultado do GG_Switch:

```
{"state":{"desired":{"property":"R"}}}
2018-12-20 12:23:01,446 - AWSIoTPythonSDK.core.protocol.mqtt_core - INFO - Performing sync publish...
~~~~~~~~~~Shadow Update Accepted~~~~~~~~~~~~~~~
Update request with token: 3b22e27c-930d-4c6a-8562-9f86088249f4 accepted!
property: R
~~~~~~~~~~~~~~~~~~~~~~~~~
```

TrafficLight Saída GG_:

```
++++++++ Received Shadow Delta ++++++++++
{u'state': {u'property': u'R'}, u'metadata': {u'property': {u'timestamp': 1545337381}}, u'version': 33, u'clientToken':
u'3b22e27c-930d-4c6a-8562-9f86088249f4'}
property: R
version: 33
++++++++++++++++++++++


Light changed to: R
{"state":{"reported":{"property":"R"}}}
2018-12-20 12:23:01,539 - AWSIoTPythonSDK.core.protocol.mqtt_core - INFO - Performing sync publish...
~~~~~~~~~~ Shadow Update Accepted ~~~~~~~~~~~~~~~
Update request with token: f552109f-c1c2-4ae6-a841-8443506eefcb accepted!
property: R
~~~~~~~~~~~~~~~~~~~~~~~~~
```

Quando executado pela primeira vez, cada script de dispositivo cliente executa o serviço de AWS IoT Greengrass descoberta para se conectar ao AWS IoT Greengrass núcleo (por meio da Internet). Depois que um dispositivo cliente for descoberto e conectado com êxito ao AWS IoT Greengrass núcleo, as operações futuras poderão ser executadas localmente.

> **Note**
>
> Os scripts `lightController.py` e `trafficLight.py` armazenam informações de conexão na pasta `groupCA`, que é criada na mesma pasta que os scripts. Se você receber erros de conexão, certifique-se de que o endereço IP no arquivo `ggc-host` corresponde ao endpoint de endereço IP para seu núcleo.

4. No AWS IoT console, escolha seu AWS IoT Greengrass grupo, escolha a guia Dispositivos do cliente e escolha GG_ TrafficLight para abrir a página de detalhes do AWS IoT dispositivo cliente.
5. Selecione a guia Sombras do dispositivo. Depois que o GG_Switch mudar de estado, não deverá haver atualizações nesta sombra. Isso porque o GG_ TrafficLight está configurado para Desativar a sincronização de sombra com a nuvem.
6. Pressione Ctrl + C na janela do dispositivo cliente (`lightController.py`) do GG_Switch. Você deve ver que a janela GG_ TrafficLight (`trafficLight.py`) para de receber mensagens de mudança de estado.

   Mantenha essas janelas abertas para que você possa executar os comandos na próxima seção.

Testar comunicações (sincronizações de dispositivos ativadas)

Para esse teste, você configura a sombra do TrafficLight dispositivo GG_ para sincronizar AWS IoT. Você executa os mesmos comandos como no teste anterior, mas desta vez o estado da sombra na nuvem será atualizado quando GG_Switch enviar uma solicitação de atualização.

1. No AWS IoT console, escolha seu AWS IoT Greengrass grupo e, em seguida, escolha a guia Dispositivos clientes.
2. Selecione o TrafficLight dispositivo GG_, escolha Sincronizar sombra e, em seguida, escolha Ativar sincronização de sombra com a nuvem.

   Você deve receber uma notificação de que o status de sincronização da sombra do dispositivo foi atualizado.
3. Na página de configuração do grupo, selecione Implantar.
4. Em suas duas janelas de linha de comando, execute os comandos do teste anterior para os dispositivos cliente GG_Switch e GG_. TrafficLight
5. Agora, verifique o estado da sombra no AWS IoT console. Escolha seu AWS IoT Greengrass grupo, escolha a guia Dispositivos do cliente, escolha GG_ TrafficLight, escolha a guia Device Shadows e, em seguida, escolha Classic Shadow.

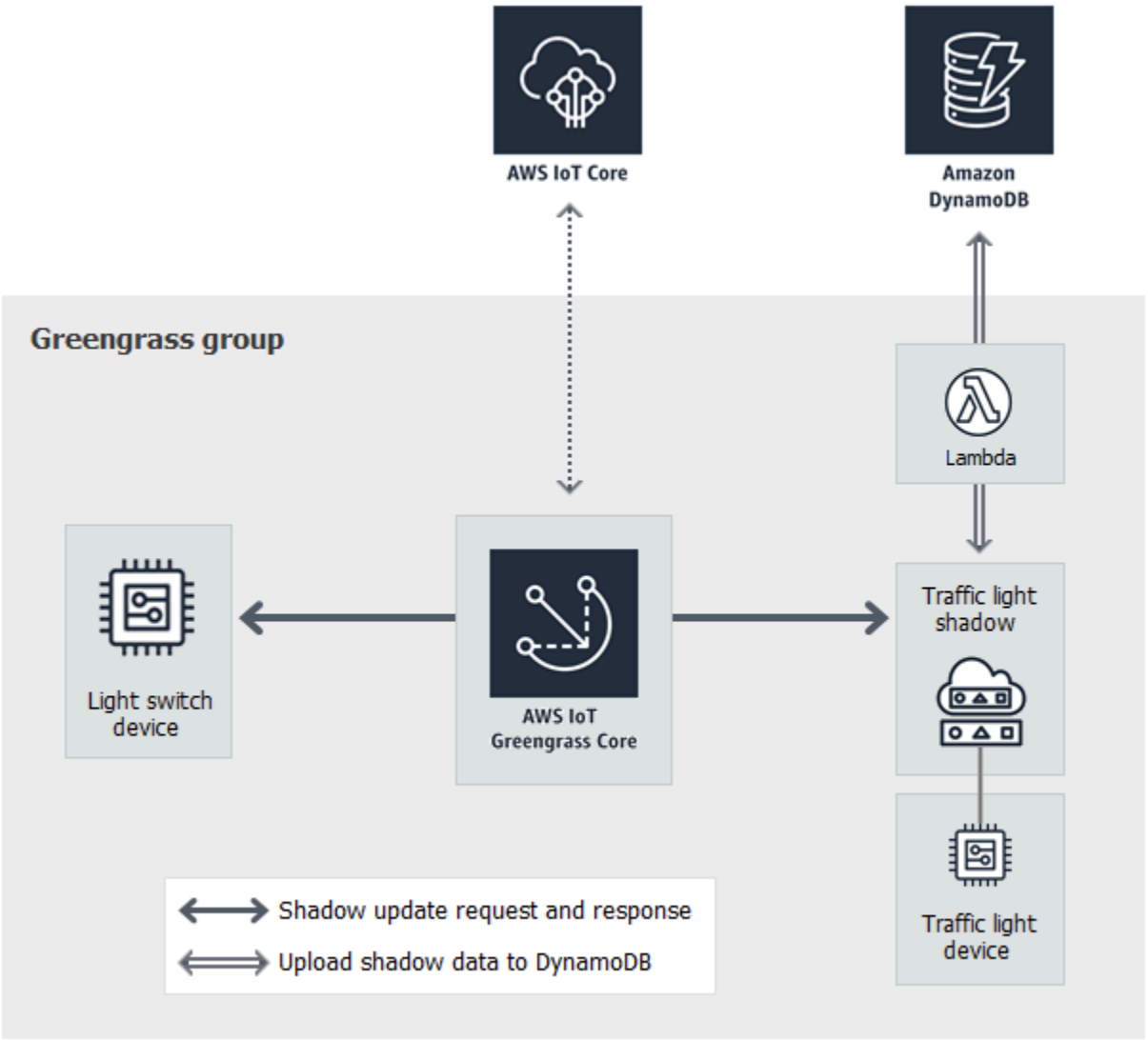
   Como você ativou a sincronização da TrafficLight sombra GG_ com AWS IoT, o estado da sombra na nuvem deve ser atualizado sempre que o GG_switch envia uma atualização. Essa funcionalidade pode ser usada para expor o estado de um dispositivo cliente com a AWS IoT.

> **Note**
>
> Se necessário, você pode solucionar problemas visualizando os registros AWS IoT Greengrass principais, especialmente`runtime.log`:
>
> ```
> cd /greengrass/ggc/var/log
> sudo cat system/runtime.log | more
> ```
>
> Você também pode visualizar `GGShadowSyncManager.log` e `GGShadowService.log`. Para obter mais informações, consulte Solução de problemas.

Mantenha os dispositivos cliente e as assinaturas configurados. Você poderá usá-los no próximo módulo. Você também pode executar os mesmos comandos.

Módulo 6: Acessando outros AWS serviços

Este módulo avançado mostra como AWS IoT Greengrass os núcleos podem interagir com outros AWS serviços na nuvem. Ele se baseia no exemplo de semáforo fornecido no Módulo 5 e adiciona uma função do Lambda que processa estados de sombra e faz upload de um resumo em uma tabela do Amazon DynamoDB.



Antes de começar, execute o script de configuração do dispositivo do Greengrass ou conclua o Módulo 1 e o Módulo 2. Também é necessário concluir o módulo 5. Você não precisa de outros componentes ou dispositivos.

Este modulo levará aproximadamente 30 minutos para ser concluído.

> **Note**
>
> Este módulo cria e atualiza uma tabela no DynamoDB. Embora a maioria das operações seja pequena e se enquadre no Amazon Web Services Free Tier, executar algumas das etapas

deste módulo pode resultar em cobranças na sua conta. Para obter informações sobre a definição de preços, consulte a Documentação de definição de preços do DynamoDB.

Tópicos

- Configurar a função do grupo
- Crie e configure a função do Lambda
- Configurar assinaturas
- Testar comunicações

Configurar a função do grupo

A função do grupo é um perfil do IAM que você cria e associa ao grupo do Greengrass. Essa função contém as permissões que as funções do Lambda implantadas (e AWS IoT Greengrass outros recursos) usam para AWS acessar os serviços. Para obter mais informações, consulte the section called “Função do grupo do Greengrass.”.

Use as seguintes etapas de alto nível para criar uma função de grupo no console do IAM.

1. Crie uma política que permita ou negue ações em um ou mais recursos.
2. Crie uma função que use o serviço do Greengrass como entidade confiável.
3. Anexe sua política à função.

Em seguida, no AWS IoT console, você adiciona a função ao grupo Greengrass.

> **Note**
>
> Um grupo do Greengrass tem uma função de grupo única. Se quiser adicionar permissões, você pode editar políticas anexadas ou anexar mais políticas.

Neste tutorial, você criará uma política de permissões que permita descrever, criar e atualizar ações em uma tabela do Amazon DynamoDB. Em seguida, você anexará a política a uma nova função e associará a função ao seu grupo do Greengrass.

Primeiramente, crie uma política gerenciada pelo cliente que conceda as permissões necessárias pela função do Lambda neste módulo.

1. No console do IAM no painel de navegação, selecione Policies e Create policy.
2. Na guia JSON, substitua o conteúdo do espaço reservado pela política a seguir. A função do Lambda neste módulo usa essas permissões para criar e atualizar uma tabela do DynamoDB chamada `CarStats`.

   JSON

   ```
   {
       "Version":"2012-10-17",
       "Statement": [
           {
               "Sid": "PermissionsForModule6",
               "Effect": "Allow",
               "Action": [
                   "dynamodb:DescribeTable",
                   "dynamodb:CreateTable",
                   "dynamodb:PutItem"
               ],
               "Resource": "arn:aws:dynamodb:*:*:table/CarStats"
           }
       ]
   }
   ```

3. Selecione Próximo: Tags e Próximo: Revisar. As tags não são usadas neste tutorial.
4. Em Nome, insira e **`greengrass_CarStats_Table`** e selecione Create policy (Criar política).

   Em seguida, crie uma função que use a nova política.
5. No painel de navegação, selecione Funções e Create role (Criar função).
6. Em Tipo de entidade confiável, selecione Serviço da AWS .
7. Em Caso de uso, Casos de uso para outros AWS serviços, selecione Greengrass, selecione Greengrass e, em seguida, selecione Avançar.
8. Em Políticas de permissões, selecione a nova política **`greengrass_CarStats_Table`** e selecione Avançar.
9. Em Nome do perfil, insira **`Greengrass_Group_Role`**.

10. Em Descrição, insira **`Greengrass group role for connectors and user-defined Lambda functions`**.
11. Selecione Criar perfil.

    Agora, adicione a função ao seu grupo do Greengrass.

12. No painel de navegação do AWS IoT console, em Gerenciar, expanda dispositivos Greengrass e escolha Grupos (V1).
13. Em Grupos do Greengrass, selecione seu grupo.
14. Selecione Configurações e, em seguida, Associar função.
15. Selecione Greengrass_Group_Role na sua lista de funções e, em seguida, selecione Associar função.

Crie e configure a função do Lambda

Nesta etapa, você cria uma função do Lambda que monitora o número de carros que avançam o semáforo. Sempre que o estado da sombra `GG_TrafficLight` mudar para `G`, a função do Lambda simulará a passagem de um número aleatório de carros (de 1 a 20). Sempre que a terceira luz `G` mudar, a função do Lambda enviará estatísticas básicas, como mínimo e máximo, para uma tabela do DynamoDB.

1. No computador, crie uma pasta chamada `car_aggregator`.
2. Da pasta de TrafficLight exemplos em diante GitHub, baixe o `carAggregator.py` arquivo para a `car_aggregator` pasta. Este é o código da função do Lambda.

   > **Note**
   >
   > Esse exemplo de arquivo Python é armazenado no repositório AWS IoT Greengrass Core SDK por conveniência, mas não usa o AWS IoT Greengrass Core SDK.

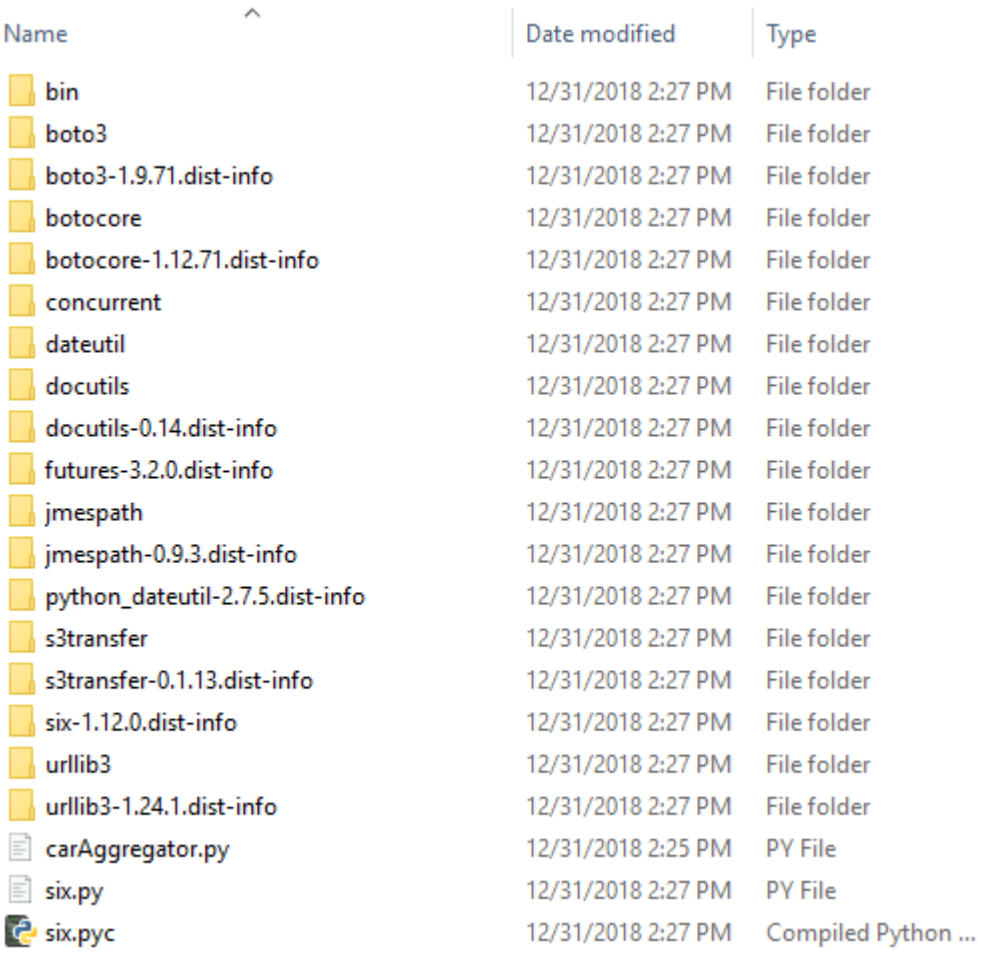
3. Se você não estiver trabalhando na região Leste dos EUA (Norte da Virgínia), abra `carAggregator.py` e `region_name` altere a linha a seguir para a Região da AWS que está atualmente selecionada no AWS IoT console. Para obter a lista de Região da AWS s suportados, consulte AWS IoT Greengrass no Referência geral da Amazon Web Services.

   ```
   dynamodb = boto3.resource('dynamodb', region_name='us-east-1')
   ```

4. Execute o seguinte comando em uma janela da linha de comando para instalar o pacote AWS SDK para Python (Boto3) e suas dependências na pasta `car_aggregator`. As funções do Greengrass Lambda usam o AWS SDK para acessar outros serviços. AWS (Para o Windows, use um prompt de comando elevado.)

   ```
   pip install boto3 -t path-to-car_aggregator-folder
   ```

   Isso resultará em uma listagem de diretórios semelhante à seguinte:

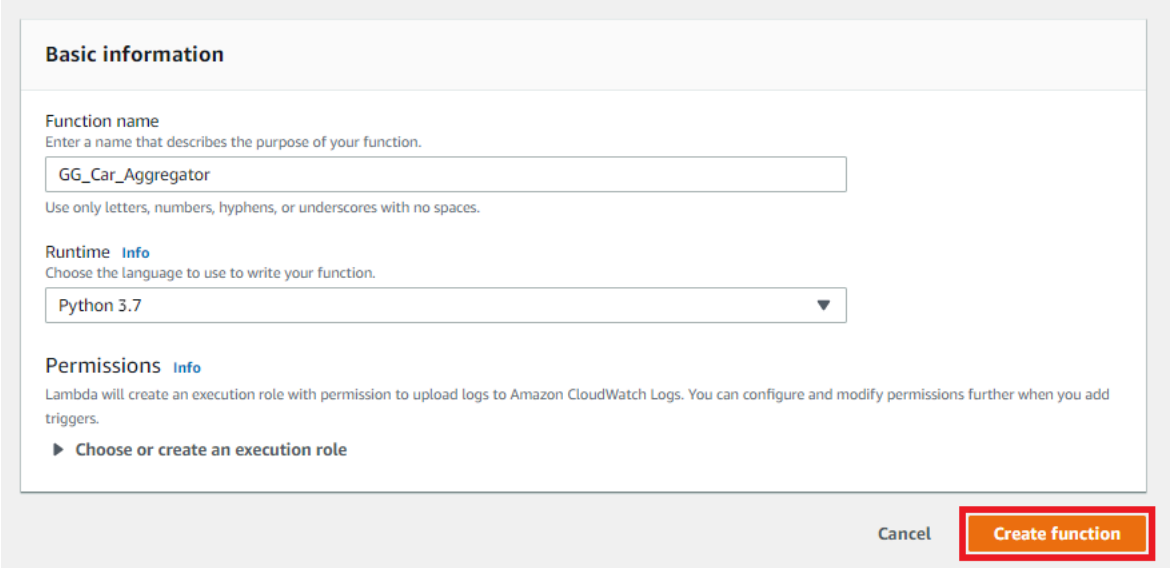
5. Compacte o conteúdo da pasta `car_aggregator` em um arquivo `.zip` chamado `car_aggregator.zip`. (Compacte o conteúdo da pasta, e não a pasta.) Esse é o pacote de implantação de sua função do Lambda.

6. No console do Lambda, crie uma função chamada **`GG_Car_Aggregator`** e defina os campos restantes da seguinte maneira:

   - Em Runtime (Tempo de execução), selecione Python 3.7.

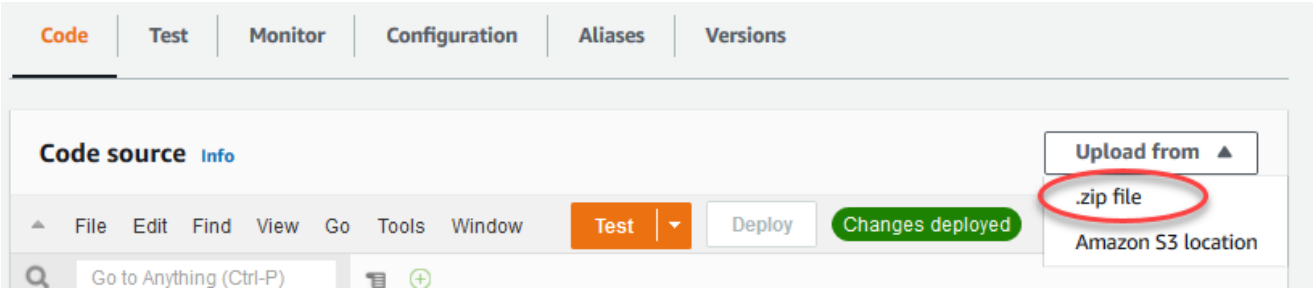
- Em Permissões, mantenha a configuração padrão. Isso cria uma função de execução que concede permissões básicas do Lambda. Essa função não é usada por AWS IoT Greengrass.

Escolha a opção Criar função.



7. Faça upload do pacote de implantação da função Lambda:

   a. Na guia Código, em Fonte do código, selecione Fazer upload a partir de. No menu suspenso, selecione o arquivo .zip.



   b. Selecione Upload e, em seguida, seu pacote de implantação `car_aggregator.zip`. Selecione Salvar.

   c. Na guia Código da função, em Configurações de runtime, selecione Editar e insira os valores a seguir.

      - Em Runtime (Tempo de execução), selecione Python 3.7.
      - Em Handler (Manipulador), insira **`carAggregator.function_handler`**.

   d. Selecione Salvar.

8. Publique a função do Lambda e crie um alias para chamado **`GG_CarAggregator`**. Para obter instruções passo a passo, consulte as etapas publicar a função do Lambda e criar um alias no Módulo 3 (Parte 1).
9. No AWS IoT console, adicione a função Lambda que você acabou de criar ao seu AWS IoT Greengrass grupo:

   a. Na página de configuração do grupo, selecione Função do Lambda e, em seguida, em Minhas funções do Lambda, selecione Adicionar.

   b. Para a função Lambda, escolha Car_Aggregator GG_.

   c. Em Versão da função do Lambda, selecione o alias da versão que você publicou.

   d. Em Memory limit (Limite de memória), insira **`64 MB`**.

   e. Para Fixado, selecione Verdadeiro.

   f. Selecione Adicionar função do Lambda.

> **Note**
>
> Você pode remover outras funções do Lambda de módulos anteriores.

Configurar assinaturas

Nesta etapa, você cria uma assinatura que permite que a TrafficLight sombra GG_ envie informações de estado atualizadas para a função GG_ LambdaCar_Aggregator . Essa assinatura é adicionada às assinaturas que você criou no Módulo 5, que são todas exigidas por esse módulo.

1. Na página de configuração do grupo, selecione a guia Inscrições e, em seguida, selecione Adicionar.
2. Na página Criar assinatura do evento, faça o seguinte:

   a. Em Tipo de origem, selecione Serviço e Serviço de sombra local.

   b. Em Tipo de destino, selecione Função do Lambda e, em seguida, selecione GG_Car_Aggregator.

   c. Em Topic filter (Filtro de tópicos), insira **`$aws/things/GG_TrafficLight/shadow/update/documents`**.

   d. Selecione Criar assinatura.

Este módulo requer as novas assinaturas e as assinaturas que você criou no Módulo 5.

3. Verifique se o daemon do Greengrass está em execução, como descrito em Implantar configurações de nuvem em um dispositivo de núcleo.
4. Na página de configuração do grupo, selecione Implantar.

Testar comunicações

1. No computador, abra duas janelas command-line. Assim como no Módulo 5, uma janela é para o dispositivo cliente GG_Switch e a outra é para o dispositivo cliente GG_TrafficLight . Você as usará para executar os mesmos comandos executados no Módulo 5.

   Execute os seguintes comandos para o dispositivo cliente GG_Switch:

   ```
   cd path-to-certs-folder
   python lightController.py --endpoint AWS_IOT_ENDPOINT --rootCA AmazonRootCA1.pem
    --cert switchCertId-certificate.pem.crt --key switchCertId-private.pem.key --
   thingName GG_TrafficLight --clientId GG_Switch
   ```

   Execute os seguintes comandos para o dispositivo TrafficLight cliente GG_:

   ```
   cd path-to-certs-folder
   python trafficLight.py --endpoint AWS_IOT_ENDPOINT --rootCA AmazonRootCA1.pem --
   cert lightCertId-certificate.pem.crt --key lightCertId-private.pem.key --thingName
    GG_TrafficLight --clientId GG_TrafficLight
   ```

   A cada 20 segundos, o interruptor atualiza o estado da sombra para G, Y e R, e a luz exibe o novo estado.
2. O manipulador da função do Lambda é acionado a cada terceira luz verde (a cada três minutos), e um novo registro do DynamoDB é criado. `trafficLight.py`Depois `lightController.py` de executar por três minutos, acesse e abra o Console de gerenciamento da AWS console do DynamoDB.
3. Escolha Leste dos EUA (Norte da Virgínia) no Região da AWS menu. Esta é a região onde a função `GG_Car_Aggregator` cria a tabela.
4. No painel de navegação, escolha Tabelas e, em seguida, escolha a CarStatstabela.
5. Selecione Exibir itens para ver as entradas na tabela.

As entradas devem ser exibidas com estatísticas básicas sobre carros que passaram (uma entrada a cada três minutos). Talvez você precise escolher o botão de atualização para ver as atualizações feitas na tabela.

6. Se o teste não for bem-sucedido, você pode procurar informações sobre solução de problemas nos logs do Greengrass.

   a. Alterne para o usuário raiz e navegue até o diretório `log`. O acesso aos AWS IoT Greengrass registros requer permissões de root.

   ```
   sudo su
   cd /greengrass/ggc/var/log
   ```

   b. Verifique se há erros em `runtime.log`.

   ```
   cat system/runtime.log | grep 'ERROR'
   ```

   c. Verifique o log gerado pela função do Lambda.

   ```
   cat user/region/account-id/GG_Car_Aggregator.log
   ```

   Os scripts `lightController.py` e `trafficLight.py` armazenam informações de conexão na pasta `groupCA`, que é criada na mesma pasta que os scripts. Se você receber erros de conexão, certifique-se de que o endereço IP no arquivo `ggc-host` corresponde ao endpoint de endereço IP para seu núcleo.

   Para obter mais informações, consulte Solução de problemas.

Este é o final do tutorial básico. Agora você deve entender o modelo de AWS IoT Greengrass programação e seus conceitos fundamentais, incluindo AWS IoT Greengrass núcleos, grupos, assinaturas, dispositivos cliente e o processo de implantação de funções Lambda executadas na borda.

Você pode excluir a tabela do DynamoDB e as funções e assinaturas do Lambda do Greengrass. Para interromper a comunicação entre o dispositivo AWS IoT Greengrass principal e a AWS IoT nuvem, abra um terminal no dispositivo principal e execute um dos seguintes comandos:

- Para desligar o dispositivo AWS IoT Greengrass principal:

```
sudo halt
```

- Para parar o AWS IoT Greengrass daemon:

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

Módulo 7: Simular a integração de segurança de hardware

Esse recurso está disponível para AWS IoT Greengrass Core v1.7 e versões posteriores.

Esse módulo avançado mostra como configurar a simulação de um módulo de segurança de hardware (HSM) para uso com um núcleo do Greengrass. A configuração usa SoftHSM puro, que é uma implementação de software de PKCS#11 que usa a interface de programação de aplicativo (API). O objetivo deste módulo é permitir que você configure um ambiente no qual você pode aprender e fazer testes iniciais em uma implementação apenas para software da API do PKCS#11. Ele é fornecido apenas para fins de aprendizagem e testes iniciais, não é destinado a nenhum uso para produção.

Você pode usar esta configuração para teste usando um serviço compatível com PKCS#11 a fim de armazenar suas chaves privadas. Para obter mais informações sobre a implementação somente para software, consulte SoftHSM. Para obter mais informações sobre a integração da segurança de hardware em um AWS IoT Greengrass núcleo, incluindo requisitos gerais, consultethe section called “Integração de segurança de hardware”.

> **Important**
>
> Este módulo é destinado somente a fins de experimentos. Não recomendamos em hipótese alguma o uso de SoftHSM em um ambiente de produção, pois ele pode fornecer uma falsa sensação de segurança adicional. A configuração resultante não fornece todos os benefícios de segurança reais. As chaves armazenadas em SoftHSM não são armazenadas com mais segurança do que qualquer outro meio de armazenamento de segredos no ambiente do Greengrass.
>
> O objetivo deste módulo é permitir que você saiba mais sobre a especificação PKCS#11 e que faça testes iniciais do seu software se, futuramente, você planejar usar um HSM com base em hardware real

Você deve testar sua implementação futura de hardware de forma separada e completa antes de qualquer uso de produção, pois pode haver diferenças entre a implementação PKCS#11 fornecida no SoftHSM e uma implementação baseada em hardware.

Se precisar de ajuda com a integração de um módulo de segurança de hardware compatível, entre em contato com seu representante do AWS Enterprise Support.

Antes de começar, execute o script de configuração do dispositivo do Greengrass ou verifique se você concluiu o módulo 1 e o módulo 2 do tutorial Conceitos básicos. Neste módulo, presumimos que seu núcleo já esteja provisionado e se comunicando com. AWS Este modulo levará aproximadamente 30 minutos para ser concluído.

Instalar o software SoftHSM

Nesta etapa, você instala o SoftHSM e as ferramentas pkcs11, que são usados para gerenciar sua instância SoftHSM.

- Em um terminal no seu dispositivo AWS IoT Greengrass principal, execute o seguinte comando:

  ```
  sudo apt-get install softhsm2 libsofthsm2-dev pkcs11-dump
  ```

  Para obter mais informações sobre esses pacotes, consulte Instalar softhsm2, Instalar libsofthsm2-dev, e Instalar pkcs11-despejo.

  > **Note**
  >
  > Se você encontrar problemas ao usar esse comando em seu sistema, consulte SoftHSM versão 2 em. GitHub Esse site fornece mais informações de instalação, incluindo a criação a partir do código-fonte.

Configure o SoftHSM

Nesta etapa, você configura o SoftHSM.

1. Mude para o usuário raiz.

```
sudo su
```

2. Use a página do manual para encontrar o `softhsm2.conf` no âmbito do sistema local. Um local comum é `/etc/softhsm/softhsm2.conf`, mas a localização pode ser diferente em alguns sistemas.

```
man softhsm2.conf
```

3. Crie o diretório para o arquivo de configuração softhsm2 no âmbito do sistema local. Neste exemplo, supomos que o local é `/etc/softhsm/softhsm2.conf`.

```
mkdir -p /etc/softhsm
```

4. Crie o diretório do token no diretório `/greengrass`.

> **Note**
>
> Se esta etapa for ignorada, o softhsm2-util relatará `ERROR: Could not initialize the library`.

```
mkdir -p /greengrass/softhsm2/tokens
```

5. Configure o diretório do token.

```
echo "directories.tokendir = /greengrass/softhsm2/tokens" > /etc/softhsm/
softhsm2.conf
```

6. Configure um back-end baseado em arquivo.

```
echo "objectstore.backend = file" >> /etc/softhsm/softhsm2.conf
```

> **Note**
>
> Essas definições de configuração são projetadas para fins de experimentação apenas. Para consultar todas as opções de configuração, leia a página do manual para o arquivo de configuração.

```
man softhsm2.conf
```

Importar a chave privada para o SoftHSM

Nesta etapa, você inicialize o token do SoftHSM, converte a chave privada e, em seguida, importa a chave privada.

1. Inicialize o token do SoftHSM.

   ```
   softhsm2-util --init-token --slot 0 --label greengrass --so-pin 12345 --pin 1234
   ```

   > **Note**
   >
   > Se solicitado, insira um pin de SO `12345` e um pin de usuário de `1234`. O AWS IoT Greengrass não usa o pin de SO (supervisor), portanto, você pode utilizar qualquer valor.
   > Se você receber o erro `CKR_SLOT_ID_INVALID: Slot 0 does not exist`, tente o seguinte comando em vez disso:
   >
   > ```
   > softhsm2-util --init-token --free --label greengrass --so-pin 12345 --pin
   >  1234
   > ```

2. Converta a chave privada em um formato que possa ser usado pela ferramenta de importação do SoftHSM. Para este tutorial, converta a chave privada obtida na opção Default Group creation (Criação de grupo padrão) no Módulo 2 do tutorial de Conceitos básicos.

   ```
   openssl pkcs8 -topk8 -inform PEM -outform PEM -nocrypt -in hash.private.key -
   out hash.private.pem
   ```

3. Importar a chave privada para o SoftHSM. Execute apenas um dos seguintes comandos, dependendo da sua versão do softhsm2-util.

   Raspbian softhsm2-util sintaxe v2.2.0

   ```
   softhsm2-util --import hash.private.pem --token greengrass --label iotkey --id
    0000 --pin 12340
   ```

Ubuntu softhsm2-util sintaxe v2.0.0

```
softhsm2-util --import hash.private.pem --slot 0 --label iotkey --id 0000 --pin
 1234
```

Esse comando identifica o slot como `0` e define o rótulo de chave como `iotkey`. Você usa esses valores na próxima seção.

Após a chave privada ser importada, você tem a opção de removê-la do diretório `/greengrass/certs`. Certifique-se de manter a CA raiz e os certificados do dispositivo no diretório.

Configurar o núcleo do Greengrass para usar o SoftHSM

Nesta etapa, você modifica o arquivo de configuração do núcleo do Greengrass para usar o SoftHSM.

1. Encontre o caminho para a biblioteca do provedor SoftHSM (`libsofthsm2.so`) no seu sistema:

   a. Obtenha a lista de pacotes instalados para a biblioteca.

   ```
   sudo dpkg -L libsofthsm2
   ```

   O arquivo `libsofthsm2.so` está localizado no diretório `softhsm`.

   b. Copiar o caminho completo para o arquivo (por exemplo, `/usr/lib/x86_64-linux-gnu/softhsm/libsofthsm2.so`). Você usará esse valor posteriormente.

2. Pare o daemon do Greengrass.

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

3. Abra o arquivo de configuração do Greengrass. Este é o arquivo `config.json` no diretório `/greengrass/config`.

> **Note**
>
> Os exemplos neste procedimento são gravados com a suposição de que o arquivo `config.json` usa o formato gerado a com a opção Default Group creation (Criação de grupo padrão) no Módulo 2 do tutorial de Conceitos básicos.

4. No objeto `crypto.principals`, insira o seguinte objeto de certificado do servidor MQTT. Adicione uma vírgula onde for preciso criar um arquivo JSON válido.

```
"MQTTServerCertificate": {
  "privateKeyPath": "path-to-private-key"
}
```

5. No objeto `crypto`, insira o seguinte objeto de PKCS11. Adicione uma vírgula onde for preciso criar um arquivo JSON válido.

```
"PKCS11": {
  "P11Provider": "/path-to-pkcs11-provider-so",
  "slotLabel": "crypto-token-name",
  "slotUserPin": "crypto-token-user-pin"
}
```

O arquivo deverá ser semelhante ao seguinte:

```
{
  "coreThing" : {
    "caPath" : "root.ca.pem",
    "certPath" : "hash.cert.pem",
    "keyPath" : "hash.private.key",
    "thingArn" : "arn:partition:iot:region:account-id:thing/core-thing-name",
    "iotHost" : "host-prefix.iot.region.amazonaws.com",
    "ggHost" : "greengrass.iot.region.amazonaws.com",
    "keepAlive" : 600
  },
  "runtime" : {
    "cgroup" : {
      "useSystemd" : "yes"
    }
  },
  "managedRespawn" : false,
```

```
  "crypto": {
    "PKCS11": {
      "P11Provider": "/path-to-pkcs11-provider-so",
      "slotLabel": "crypto-token-name",
      "slotUserPin": "crypto-token-user-pin"
    },
    "principals" : {
      "MQTTServerCertificate": {
        "privateKeyPath": "path-to-private-key"
      },
      "IoTCertificate" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key",
        "certificatePath" : "file:///greengrass/certs/hash.cert.pem"
      },
      "SecretsManager" : {
        "privateKeyPath" : "file:///greengrass/certs/hash.private.key"
      }
    },
    "caPath" : "file:///greengrass/certs/root.ca.pem"
  }
}
```

> **Note**
>
> Para usar atualizações OTA (over-the-air) com a segurança de hardware, o objeto `PKCS11` também deve conter a propriedade `OpenSSLEngine`. Para obter mais informações, consulte the section called “Configurar as atualizações OTA”.

6. Edite o objeto `crypto`:

   a. Configure o objeto `PKCS11`.

      - Para `P11Provider`, insira o caminho completo para `libsofthsm2.so`.
      - Em `slotLabel`, digite `greengrass`.
      - Em `slotUserPin`, digite `1234`.

   b. Configure os caminhos da chave privada no objeto `principals`. Não edite a propriedade `certificatePath`.

- Para as propriedades `privateKeyPath`, insira o seguinte caminho RFC 7512 PKCS#11 (que especifica o rótulo da chave). Faça isso para as entidades principais `IoTCertificate`, `SecretsManager`, e `MQTTServerCertificate`.

```
pkcs11:object=iotkey;type=private
```

c. Verifique o objeto `crypto`. A aparência deve ser semelhante à seguinte:

```
"crypto": {
  "PKCS11": {
    "P11Provider": "/usr/lib/x86_64-linux-gnu/softhsm/libsofthsm2.so",
    "slotLabel": "greengrass",
    "slotUserPin": "1234"
  },
  "principals": {
    "MQTTServerCertificate": {
      "privateKeyPath": "pkcs11:object=iotkey;type=private"
    },
    "SecretsManager": {
      "privateKeyPath": "pkcs11:object=iotkey;type=private"
    },
    "IoTCertificate": {
      "certificatePath": "file://certs/core.crt",
      "privateKeyPath": "pkcs11:object=iotkey;type=private"
    }
  },
  "caPath": "file://certs/root.ca.pem"
}
```

7. Remova os valores `caPath`, `certPath` e `keyPath` do objeto `coreThing`. A aparência deve ser semelhante à seguinte:

```
"coreThing" : {
  "thingArn" : "arn:partition:iot:region:account-id:thing/core-thing-name",
  "iotHost" : "host-prefix-ats.iot.region.amazonaws.com",
  "ggHost" : "greengrass-ats.iot.region.amazonaws.com",
  "keepAlive" : 600
}
```

> **Note**
>
> Para este tutorial, você especifica a mesma chave privada para todas as entidades principais. Para obter mais informações sobre como escolher a chave privada para o servidor MQTT local, consulte Desempenho. Para obter mais informações sobre o secrets manager local, consulte Implantar segredos no núcleo.

Testar a configuração

- Inicie o daemon do Greengrass.

  ```
  cd /greengrass/ggc/core/
  sudo ./greengrassd start
  ```

  Se o daemon for iniciado com êxito, seu núcleo será configurado corretamente.

  Agora você está pronto para saber mais sobre a especificação PKCS#11 e fazer testes iniciais com a API do PKCS#11 fornecida pela implementação do SoftHSM.

  > **Important**
  >
  > Mais uma vez, é extremamente importante saber que este módulo é destinado somente a aprendizagem e testes. Na verdade, ele não fortalece a postura de segurança do seu ambiente do Greengrass.
  > Em vez disso, a finalidade do módulo é permitir que você inicie a aprendizagem e os testes como preparação para usar um verdadeiro HSM baseado em hardware no futuro. Quando isso ocorrer, você deverá testar seu software em relação ao HSM baseado em hardware antes de qualquer uso de produção, pois pode haver diferenças entre a implementação PKCS#11 fornecida no SoftHSM e uma implementação baseada em hardware.

Consulte também

- Guia de uso da interface de token criptográfico PKCS #11 Versão 2.40. Editada por John Leiseboer e Robert Griffin. 16 de novembro de 2014. Nota 02 do Comitê OASIS. http://docs.oasis-

open.org/pkcs11/pkcs11-ug/v2.40/cn02/pkcs11-ug-v2.40-cn02.html. Versão mais recente: http://docs.oasis-open.org/pkcs11/pkcs11-ug/v2.40/pkcs11-ug-v2.40.html.

- RFC 7512

Atualizações OTA de AWS IoT Greengrass Software principal

O pacote de software AWS IoT Greengrass Core inclui um agente de atualização que pode realizar atualizações de software sem fio (OTA). AWS IoT Greengrass Você pode usar atualizações OTA para instalar a versão mais recente do software de núcleo do AWS IoT Greengrass ou do software do atendente de atualizações OTA em um ou mais núcleos. Com as atualizações OTA, seus dispositivos principais não precisam estar fisicamente presentes.

É recomendável usar atualizações OTA quando possível. Elas fornecem um mecanismo que você pode usar para rastrear o status da atualização e o histórico de atualizações. Se ocorrer uma atualização com falha, o atendente de atualizações OTA reverterá para a versão anterior do software.

> **Note**
>
> As atualizações OTA não são suportadas quando você usa `apt` para instalar o software AWS IoT Greengrass Core. Para essas instalações, é recomendável usar `apt` para atualizar o software. Para obter mais informações, consulte the section called “Instalar de um repositório do APT”.

As atualizações OTA são mais eficientes para:

- Corrigir vulnerabilidades de segurança.
- Resolver problemas de estabilidade do software.
- Implantar atributos novos ou melhorados.

Esse atributo se integra aos Trabalhos do AWS IoT.

Requisitos

Os requisitos a seguir se aplicam às atualizações de AWS IoT Greengrass software OTA.

- O núcleo do Greengrass deve ter pelo menos 400 MB de espaço em disco disponível no armazenamento local. O agente de atualização OTA exige cerca de três vezes o requisito de uso

em tempo de execução do software AWS IoT Greengrass Core. Para obter mais informações, consulte Cotas de serviço para o núcleo do Greengrass no Referência geral da Amazon Web Services.

- O núcleo do Greengrass deve ter uma conexão com a Nuvem AWS.
- O núcleo do Greengrass deve ser configurado corretamente e provisionado com certificados e chaves para autenticação com e. AWS IoT Core AWS IoT Greengrass Para obter mais informações, consulte the section called “X.509 certificados”.
- O núcleo do Greengrass não pode ser configurado para usar um proxy de rede.

> **Note**
>
> A partir da AWS IoT Greengrass v1.9.3, as atualizações OTA são suportadas em núcleos que configuram o tráfego MQTT para usar a porta 443 em vez da porta padrão 8883. No entanto, o atendente de atualizações OTA não oferece suporte a atualizações por meio de um proxy de rede. Para obter mais informações, consulte the section called “Conectar-se à porta 443 ou por meio de um proxy de rede”.

- A inicialização confiável não pode ser habilitada na partição que contém o software AWS IoT Greengrass Core.

> **Note**
>
> Você pode instalar e executar o software AWS IoT Greengrass Core em uma partição com inicialização confiável habilitada, mas as atualizações OTA não são suportadas.

- AWS IoT Greengrass deve ter read/write permissões na partição que contém o software AWS IoT Greengrass Core.
- Ao usar um sistema init para gerenciar o núcleo do Greengrass, você deve configurar as atualizações OTA para se integrarem ao sistema init. Para obter mais informações, consulte the section called “Integração com sistemas Init”.
- Você deve criar uma função que seja usada para pré-atribuir os URLs do Amazon S3 AWS IoT Greengrass aos artefatos de atualização de software. Essa função de signatário permite AWS IoT Core acessar artefatos de atualização de software armazenados no Amazon S3 em seu nome. Para obter mais informações, consulte the section called “Permissões do IAM para atualizações OTA”.

Permissões do IAM para atualizações OTA

Ao AWS IoT Greengrass lançar uma nova versão do software AWS IoT Greengrass Core, AWS IoT Greengrass atualiza os artefatos de software armazenados no Amazon S3 que são usados para a atualização do OTA.

Você Conta da AWS deve incluir uma função de signatário de URL do Amazon S3 que possa ser usada para acessar esses artefatos. A função deve ter uma política de permissões que permita a `s3:GetObject` ação nos buckets no destino Região da AWS s. A função também deve ter uma política de confiança que permita que o `iot.amazonaws.com` assuma a função como uma entidade confiável.

Política de permissões

Para permissões de função, você pode usar a política AWS gerenciada ou criar uma política personalizada.

- Use a política AWS gerenciada

  A política GreengrassOTAUpdateArtifactAccessgerenciada é fornecida por AWS IoT Greengrass. Use essa política se quiser permitir o acesso em todas as regiões da Amazon Web Services suportadas pela AWS IoT Greengrass, tanto atuais quanto futuras.

- Crie uma política personalizada

  Você deve criar uma política personalizada se quiser especificar explicitamente as regiões da Amazon Web Services em que seus núcleos estão implantados. O exemplo de política a seguir permite acesso a atualizações de AWS IoT Greengrass software em seis regiões.

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAccessToGreengrassOTAUpdateArtifacts",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::us-east-1-greengrass-updates/*",
```

```
                "arn:aws:s3:::us-west-2-greengrass-updates/*",
                "arn:aws:s3:::ap-northeast-1-greengrass-updates/*",
                "arn:aws:s3:::ap-southeast-2-greengrass-updates/*",
                "arn:aws:s3:::eu-central-1-greengrass-updates/*",
                "arn:aws:s3:::eu-west-1-greengrass-updates/*"
            ]
        }
    ]
}
```

Política de confiança

A política de confiança anexada à função deve permitir a ação `sts:AssumeRole` e definir `iot.amazonaws.com` como um principal. Isso permite que o AWS IoT Core assuma a função como uma entidade confiável. Veja a seguir um exemplo de documento de política:

JSON

```
{
     "Version":"2012-10-17",
     "Statement": [
         {
             "Sid": "AllowIotToAssumeRole",
             "Action": "sts:AssumeRole",
             "Principal": {
                  "Service": "iot.amazonaws.com"
             },
             "Effect": "Allow"
        }
    ]
}
```

Além disso, o usuário que inicia uma atualização OTA deve ter permissões para usar `greengrass:CreateSoftwareUpdateJob` e `iot:CreateJob` e para usar `iam:PassRole` para passar as permissões da função de assinante. Veja a seguir um exemplo de política do IAM:

JSON

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

```
        {
            "Action": [
                "greengrass:CreateSoftwareUpdateJob"
            ],
            "Effect": "Allow",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:CreateJob"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": "arn:aws:iam::123456789012:role/name-of-s3-url-signer-
role"
        }
    ]
}
```

Considerações

Antes de iniciar uma atualização OTA do software de núcleo do Greengrass, esteja ciente do impacto sobre os dispositivos no seu grupo do Greengrass, tanto no dispositivo de núcleo quanto nos dispositivos cliente conectados localmente a esse núcleo:

- O núcleo é desligado durante a atualização.
- Todas as funções Lambda em execução no núcleo serão desativadas. Se essas funções gravarem em recursos locais, elas poderão deixá-los em um estado incorreto, a menos que seja encerrado corretamente.
- Durante o tempo de inatividade do núcleo, todas as conexões com o Nuvem AWS são perdidas. As mensagens roteadas pelo núcleo por dispositivos cliente são perdidas.
- Os caches da credencial são perdidos.
- As filas que retêm trabalhos pendentes para funções Lambda são perdidas.

- Long-lived As funções Lambda perdem suas informações de estado dinâmico e todo o trabalho pendente é descartado.

As seguintes informações de estado são preservadas durante a atualização OTA:

- Arquivo de núcleo
- Configuração de grupo do Greengrass
- Sombras locais
- Logs do Greengrass
- Registros do atendente de atualizações OTA

Agente de atualizações OTA do Greengrass

O atendente de atualizações OTA do Greengrass é o componente de software do dispositivo que processa as tarefas de atualização criadas e implantadas na nuvem. O agente de atualização OTA é distribuído no mesmo pacote de software do software AWS IoT Greengrass Core. O atendente está localizado em */greengrass-root*`/ota/ota_agent/ggc-ota`. Ele grava logs em `/var/log/greengrass/ota/ggc_ota.txt`.

> **Note**
>
> *greengrass-root*representa o caminho em que o software AWS IoT Greengrass Core está instalado em seu dispositivo. Normalmente, esse é o diretório `/greengrass`.

É possível iniciar o atendente de atualizações OTA executando o binário manualmente ou integrando-o como parte de um script init, como um arquivo de serviço systemd. Se você executar o binário manualmente, ele deverá ser executado como root. Quando é iniciado, o agente de atualização OTA escuta os trabalhos de atualização de AWS IoT Greengrass software AWS IoT Core e os executa sequencialmente. O agente de atualização OTA ignora todos os outros tipos de AWS IoT trabalho.

O trecho a seguir mostra um exemplo de um arquivo de serviço systemd para iniciar, parar e reiniciar o atendente de atualizações OTA:

```
[Unit]
```

```
Description=Greengrass OTA Daemon

[Service]
Type=forking
Restart=on-failure
ExecStart=/greengrass/ota/ota_agent/ggc-ota

[Install]
WantedBy=multi-user.target
```

Um núcleo que é o destino de uma atualização não deve executar duas instâncias do atendente de atualizações OTA. Isso faz com que os dois atendente s processem as mesmas tarefas, o que cria conflitos.

Integração com sistemas Init

Durante uma atualização OTA, o atendente de atualizações OTA reinicia os binários no dispositivo de núcleo. Se os binários estiverem em execução, isso poderá causar conflitos quando um sistema de inicialização estiver monitorando o estado do software AWS IoT Greengrass principal ou do agente durante a atualização. Para ajudar a integrar o mecanismo de atualização OTA às suas estratégias de monitoramento, grave scripts de shell que são executados antes e depois de uma atualização. Por exemplo, você pode usar o script `ggc_pre_update.sh` para fazer backup de dados ou interromper processos antes que o dispositivo seja desligado.

Para instruir o atendente de atualizações OTA a executar esses scripts, é necessário incluir o sinalizador `"managedRespawn" : true` no arquivo config.json. Essa configuração é mostrada no seguinte trecho:

```
{
   "coreThing": {
       …
   },
   "runtime": {
       …
   },
   "managedRespawn": true
       …
}
```

Respawn gerenciado com atualizações OTA

Os requisitos a seguir se aplicam às atualizações do OTA com o `managedRespawn` definido como `true`:

- Os seguintes scripts de shell devem estar presentes no diretório */greengrass-root*`/usr/scripts`:
  - `ggc_pre_update.sh`
  - `ggc_post_update.sh`
  - `ota_pre_update.sh`
  - `ota_post_update.sh`
- Os scripts devem retornar um código de retorno bem-sucedido.
- Os scripts devem ser de propriedade de root e executáveis somente pelo root.
- O script `ggc_pre_update.sh` deve interromper o daemon do Greengrass.
- O script `ggc_post_update.sh` deve iniciar o daemon do Greengrass.

> **Note**
>
> Como o atendente de atualizações OTA gerencia seu próprio processo, os `ota_pre_update.sh` e scripts `ota_post_update.sh` não precisam parar ou iniciar o serviço OTA.

O atendente de atualizações OTA executa os scripts a partir do */greengrass-root*`/usr/scripts`. A árvore de diretórios deve ser semelhante a:

```
<greengrass_root>
|-- certs
|-- config
|   |-- config.json
|-- ggc
|-- usr/scripts
|   |-- ggc_pre_update.sh
|   |-- ggc_post_update.sh
|   |-- ota_pre_update.sh
|   |-- ota_post_update.sh
|-- ota
```

Se `managedRespawn` estiver definido como `true`, o atendente de atualizações OTA verificará o diretório */greengrass-root*`/usr/scripts` para localizar esses scripts antes e depois da atualização de software. Se os scripts não existirem, a atualização falhará. AWS IoT Greengrass não validará o conteúdo desses scripts. Como prática recomendada, verifique se seus scripts funcionam corretamente e emita códigos de saída apropriados para os erros.

Para atualizações OTA do AWS IoT Greengrass Software principal:

- Antes de iniciar a atualização, o atendente executa o script `ggc_pre_update.sh`. Use esse script para comandos que precisam ser executados antes que o agente de atualização OTA inicie a atualização do software AWS IoT Greengrass Core, como fazer backup de dados ou interromper qualquer processo em execução. O exemplo a seguir mostra um script simples para interromper o daemon do Greengrass.

  ```
  #!/bin/bash
  set -euo pipefail
  systemctl stop greengrass
  ```

- Depois de concluir a atualização, o atendente executa o script `ggc_post_update.sh`. Use esse script para comandos que precisam ser executados após o agente de atualização OTA iniciar a atualização do software AWS IoT Greengrass Core, como para reiniciar processos. O exemplo a seguir mostra um script simples para iniciar o daemon do Greengrass.

  ```
  #!/bin/bash
  set -euo pipefail
  systemctl start greengrass
  ```

Para atualizações OTA do atendente de atualizações OTA:

- Antes de iniciar a atualização, o atendente executa o script `ota_pre_update.sh`. Use esse script para comandos que precisam ser executados antes que o atendente de atualizações OTA se atualize, como fazer backup de dados ou interromper qualquer processo em execução.
- Depois de concluir a atualização, o atendente executa o script `ota_post_update.sh`. Use esse script para comandos que precisam ser executados após a atualização do atendente de atualizações OTA, como para reiniciar processos.

> **Note**
>
> Se `managedRespawn` estiver definido como `false`, o atendente de atualizações OTA não executará os scripts.

Criar uma atualização OTA

Siga estas etapas para realizar uma atualização OTA do AWS IoT Greengrass software em um ou mais núcleos:

1. Verifique se seus núcleos atendem aos requisitos para atualizações OTA.

   > **Note**
   >
   > Se você configurou um sistema de inicialização para gerenciar o software AWS IoT Greengrass Core ou o agente de atualização OTA, verifique o seguinte em seus núcleos:
   >
   > - O arquivo config.json especifica `"managedRespawn" : true`.
   > - O usr/scripts diretório*greengrass-root*//contém os seguintes scripts:
   >   - ggc_pre_update.sh
   >   - ggc_post_update.sh
   >   - ota_pre_update.sh
   >   - ota_post_update.sh
   >
   > Para obter mais informações, consulte the section called “Integração com sistemas Init”.

2. Em um terminal de dispositivo núcleo, inicie o atendente de atualizações OTA.

   ```
   cd /greengrass-root/ota/ota_agent
   sudo ./ggc-ota
   ```

> **Note**
>
> *greengrass-root* representa o caminho em que o software AWS IoT Greengrass Core está instalado em seu dispositivo. Normalmente, esse é o diretório `/greengrass`.

Não inicie várias instâncias do atendente de atualizações OTA em um núcleo porque isso pode causar conflitos.

3. Use a AWS IoT Greengrass API para criar um trabalho de atualização de software.

   a. Chame a API CreateSoftwareUpdateJob. Neste exemplo de procedimento, usamos AWS CLI comandos.

      O comando a seguir cria um trabalho que atualiza o software AWS IoT Greengrass Core em um núcleo. Substitua os valores de exemplo e execute o comando.

      Linux or macOS terminal

      ```
      aws greengrass create-software-update-job \
      --update-targets-architecture x86_64 \
      --update-targets [\"arn:aws:iot:region:123456789012:thing/myCoreDevice\"] \
      --update-targets-operating-system ubuntu \
      --software-to-update core \
      --s3-url-signer-role arn:aws:iam::123456789012:role/myS3UrlSignerRole \
      --update-agent-log-level WARN \
      --amzn-client-token myClientToken1
      ```

      Windows command prompt

      ```
      aws greengrass create-software-update-job ^
      --update-targets-architecture x86_64 ^
      --update-targets [\"arn:aws:iot:region:123456789012:thing/myCoreDevice\"] ^
      --update-targets-operating-system ubuntu ^
      --software-to-update core ^
      --s3-url-signer-role arn:aws:iam::123456789012:role/myS3UrlSignerRole ^
      --update-agent-log-level WARN ^
      --amzn-client-token myClientToken1
      ```

O comando retorna a seguinte resposta.

```
{
    "IotJobId": "GreengrassUpdateJob_c3bd7f36-ee80-4d42-8321-a1da0EXAMPLE",
    "IotJobArn": "arn:aws:iot:region:123456789012:job/
GreengrassUpdateJob_c3bd7f36-ee80-4d42-8321-a1da0EXAMPLE",
    "PlatformSoftwareVersion": "1.10.1"
}
```

b. Copie o `IoTJobId` da resposta.

c. Ligue DescribeJobpara a AWS IoT Core API para ver o status do trabalho. Substitua o valor de exemplo pelo ID do trabalho e execute o comando.

```
aws iot describe-job --job-id GreengrassUpdateJob_c3bd7f36-ee80-4d42-8321-
a1da0EXAMPLE
```

O comando retorna um objeto de resposta que contém informações sobre o trabalho, incluindo `status` e `jobProcessDetails`.

```
{
    "job": {
        "jobArn": "arn:aws:iot:region:123456789012:job/
GreengrassUpdateJob_c3bd7f36-ee80-4d42-8321-a1da0EXAMPLE",
        "jobId": "GreengrassUpdateJob_c3bd7f36-ee80-4d42-8321-a1da0EXAMPLE",
        "targetSelection": "SNAPSHOT",
        "status": "IN_PROGRESS",
        "targets": [
            "arn:aws:iot:region:123456789012:thing/myCoreDevice"
        ],
        "description": "This job was created by Greengrass to update the
 Greengrass Cores in the targets with version 1.10.1 of the core software
 running on x86_64 architecture.",
        "presignedUrlConfig": {
            "roleArn": "arn:aws::iam::123456789012:role/myS3UrlSignerRole",
            "expiresInSec": 3600
        },
        "jobExecutionsRolloutConfig": {},
        "createdAt": 1588718249.079,
        "lastUpdatedAt": 1588718253.419,
        "jobProcessDetails": {
```

```
            "numberOfCanceledThings": 0,
            "numberOfSucceededThings": 0,
            "numberOfFailedThings": 0,
            "numberOfRejectedThings": 0,
            "numberOfQueuedThings": 1,
            "numberOfInProgressThings": 0,
            "numberOfRemovedThings": 0,
            "numberOfTimedOutThings": 0
        },
        "timeoutConfig": {}
    }
}
```

Para obter ajuda sobre a solução de problemas, consulte Solução de problemas.

CreateSoftwareUpdateJob API

Você pode usar a `CreateSoftwareUpdateJob` API para atualizar o software AWS IoT Greengrass Core ou o software do agente de atualização OTA em seus dispositivos principais. Essa API cria um trabalho de snapshot do AWS IoT que notifica os dispositivos quando uma atualização está disponível. Depois de ligar`CreateSoftwareUpdateJob`, você pode usar outros comandos de AWS IoT trabalho para rastrear a atualização do software. Para obter mais informações, consulte Trabalhos no Guia do desenvolvedor do AWS IoT .

O exemplo a seguir mostra como usar o AWS CLI para criar um trabalho que atualiza o software AWS IoT Greengrass Core em um dispositivo principal:

```
aws greengrass create-software-update-job \
--update-targets-architecture x86_64 \
--update-targets [\"arn:aws:iot:region:123456789012:thing/myCoreDevice\"] \
--update-targets-operating-system ubuntu \
--software-to-update core \
--s3-url-signer-role arn:aws:iam::123456789012:role/myS3UrlSignerRole \
--update-agent-log-level WARN \
--amzn-client-token myClientToken1
```

O comando `create-software-update-job` retorna uma resposta JSON que contém o ID do trabalho, o ARN do trabalho e a versão do software que foi instalada pela atualização:

```
{
    "IotJobId": "GreengrassUpdateJob_c3bd7f36-ee80-4d42-8321-a1da0EXAMPLE",
    "IotJobArn": "arn:aws:iot:region:123456789012:job/GreengrassUpdateJob_c3bd7f36-
ee80-4d42-8321-a1da0EXAMPLE",
    "PlatformSoftwareVersion": "1.9.2"
}
```

Para obter as etapas que mostram como usar `create-software-update-job` para atualizar um dispositivo de núcleo, consulte the section called “Criar uma atualização OTA”.

O comando `create-software-update-job` tem os seguintes parâmetros:

`--update-targets-architecture`

A arquitetura do dispositivo de núcleo.

Valores válidos: `armv7l`, `armv6l`, `x86_64` ou `aarch64`

`--update-targets`

Os núcleos a serem atualizados. A lista pode conter ARNs de núcleos individuais e ARNs de grupos de coisas cujos membros são núcleos. Para obter mais informações sobre grupos de coisas, consulte Grupos de coisas estáticas no Guia do usuário do AWS IoT .

`--update-targets-operating-system`

O sistema operacional do dispositivo de núcleo.

Valores válidos: `ubuntu`, `amazon_linux`, `raspbian` ou `openwrt`

`--software-to-update`

Especifica se o software do núcleo ou o software do atendente de atualizações OTA devem ser atualizados.

Valores válidos: `core` ou `ota_agent`

`--s3-url-signer-role`

O ARN do perfil do IAM usado para pré-assinar o URL do Amazon S3 que vincula aos artefatos de atualização de software do AWS IoT Greengrass . A política de permissões anexadas da função deve permitir a ação `s3:GetObject` nos buckets nas regiões de destino das Região da AWS s. A função também deve permitir que `iot.amazonaws.com` assuma a função como uma

entidade confiável. Para obter mais informações, consulte the section called “Permissões do IAM para atualizações OTA”.

`--amzn-client-token`

(Opcional) Um token do cliente usado para fazer solicitações idempotentes. Forneça um token exclusivo para impedir que atualizações duplicadas sejam criadas por causa de novas tentativas internas.

`--update-agent-log-level`

(Opcional) O nível de registro em log para instruções de log geradas pelo atendente de atualizações OTA. O padrão é `ERROR`.

Valores válidos: `NONE`, `TRACE`, `DEBUG`, `VERBOSE`, `INFO`, `WARN`, `ERROR` ou `FATAL`

> **Note**
>
> `CreateSoftwareUpdateJob` aceita solicitações somente para as seguintes combinações de arquitetura e sistema operacional compatíveis:
>
> - ubuntu/x86_64
> - ubuntu/aarch64
> - amazon_linux/x86_64
> - raspbian/armv7l
> - raspbian/armv6l
> - openwrt/aarch64
> - openwrt/armv7l

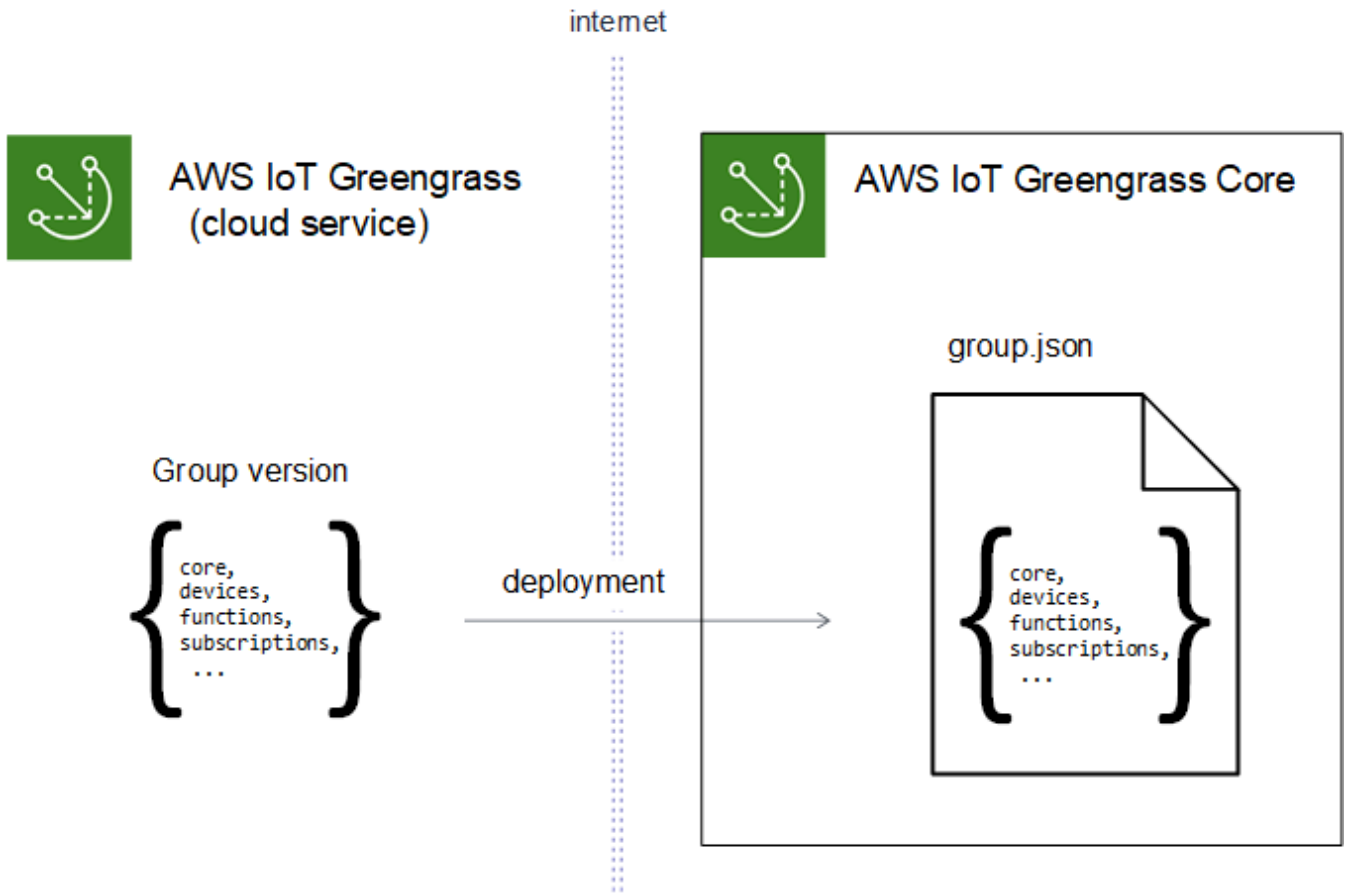
Implantar AWS IoT Greengrass grupos para um AWS IoT Greengrass core

Use AWS IoT Greengrass grupos para organizar entidades em seu ambiente periférico. Você também pode usar os grupos para controlar como as entidades no grupo interagem entre si e com a Nuvem AWS. Por exemplo, somente as funções do Lambda no grupo são implantadas para execução local e somente os dispositivos no grupo podem se comunicar usando o servidor MQTT local.

Um grupo deve incluir um núcleo, que é um AWS IoT dispositivo que executa o software AWS IoT Greengrass Core. O núcleo atua como um gateway de ponta e fornece AWS IoT Core recursos no ambiente periférico. Dependendo da sua necessidade de negócios, também é possível adicionar as seguintes entidades a um grupo:

- Dispositivos cliente. Representados como coisas no registro do AWS IoT . Esses dispositivos devem executar o FreeRTOS ou usar o SDK AWS IoT de dispositivo ou a API de descoberta do AWS IoT Greengrass para obter informações de conexão para o núcleo. Somente dispositivos cliente membros do grupo podem se conectar ao núcleo.
- Funções lambda. User-defined aplicativos sem servidor que executam código no núcleo. As funções do Lambda são criadas AWS Lambda e referenciadas a partir de um grupo do Greengrass. Para obter mais informações, consulte Execute funções locais do Lambda.
- Conectores. Aplicativos com tecnologia sem servidor predefinidos que executam código no núcleo. Os conectores podem fornecer integração integrada com infraestrutura local AWS, protocolos de dispositivos e outros serviços em nuvem. Para obter mais informações, consulte Integrar a serviços e protocolos usando conectores.
- Assinaturas. Define os editores, os assinantes e os tópicos MQTT (ou assuntos) autorizados para comunicação MQTT.
- Recursos. Referências a dispositivos e volumes locais, modelos de machine learning e segredos, usados para controle de acesso por conectores e funções do Lambda do Greengrass.
- Logs. Configurações de registro para componentes AWS IoT Greengrass do sistema e funções do Lambda. Para obter mais informações, consulte the section called “Monitoramento com logs do AWS IoT Greengrass ”.

Você gerencia seu grupo do Greengrass no Nuvem AWS e depois o implanta em um núcleo. A implantação copia a configuração do grupo para o arquivo `group.json` no dispositivo de núcleo. Esse arquivo está localizado em *`greengrass-root`*`/ggc/deployments/group`.



> **Note**
>
> Durante uma implantação, o processo de daemon do Greengrass no dispositivo de núcleo é interrompido e, depois, reiniciado.

Implantação de grupos a partir do AWS IoT console

Você pode implantar um grupo e gerenciar suas implantações na página de configuração do grupo no AWS IoT console.

> **Note**
>
> Para abrir essa página no console, selecione Dispositivos Greengrass, depois, Grupos (V1) e, em Grupos do Greengrass, selecione seu grupo.

Para implantar a versão atual do grupo

- Na página de configuração do grupo, selecione Implantar.

Para visualizar o histórico de implantações do grupo

O histórico de implantações de um grupo inclui a data e a hora, a versão do grupo e o status de cada tentativa de implantação.

1. Na página de configuração do grupo, selecione a guia Implantações.
2. Para ver mais informações sobre uma implantação, incluindo mensagens de erro, escolha Implantações no AWS IoT console, em Dispositivos do Greengrass.

Como reimplantar uma implantação de grupo

Você poderá querer reimplantar uma implantação se a implantação atual falhar ou reverter para uma versão de grupo diferente.

1. No AWS IoT console, escolha dispositivos Greengrass e, em seguida, escolha Grupos (V1).
2. Selecione a guia Implantações.
3. Selecione a implantação que você deseja reimplantar e, em seguida selecione Reimplantar.

Para redefinir implantações do grupo

Você pode querer redefinir implantações do grupo para mover ou excluir um grupo ou para remover as informações de implantação. Para obter mais informações, consulte the section called “Redefinir implantações”.

1. No AWS IoT console, escolha dispositivos Greengrass e, em seguida, escolha Grupos (V1).
2. Selecione a guia Implantações.

3. Selecione a implantação que você deseja redefinir e, em seguida selecione Redefinir implantações.

Implantando grupos com o AWS IoT Greengrass solicitações de

A AWS IoT Greengrass API fornece as seguintes ações para implantar AWS IoT Greengrass grupos e gerenciar implantações em grupo. Você pode chamar essas ações da AWS CLI AWS IoT Greengrass API ou do AWS SDK.

Ação	Description
CreateDeployment	Cria uma implantação `NewDeployment` ou `Redeployment` . Você poderá querer reimplantar uma implantação se a implantação atual falhar. Ou você pode querer reimplantar para reverter para uma versão de grupo diferente.
GetDeploymentStatus	Retorna o status de uma implantação: `Building`, `InProgress` , `Success` ou `Failure`. Você pode configurar EventBridge eventos da Amazon para receber notificações de implantação. Para obter mais informações, consulte the section called “Obter notificações de implantação”.
ListDeployments	Retorna o histórico de implantações do grupo.
ResetDeployments	Redefine as implantações do grupo. Você pode querer redefinir implantações do grupo para mover ou excluir um grupo ou para remover as informações de implantação. Para obter mais informações, consulte the section called “Redefinir implantações”.

> **Note**
>
> Para obter informações sobre operações de implantação em massa, consulte the section called “Criar implantações em massa”.

Obter o ID do grupo

O ID do grupo é normalmente usado em ações da API. Você pode usar a ListGroups ação para encontrar a ID do grupo-alvo na sua lista de grupos. Por exemplo, no AWS CLI, use o `list-groups` comando.

```
aws greengrass list-groups
```

Você também pode incluir a opção `query` para filtrar resultados. Por exemplo:

- Para obter o último grupo criado:

  ```
  aws greengrass list-groups --query "reverse(sort_by(Groups, &CreationTimestamp))[0]"
  ```

- Para obter um grupo pelo nome:

  ```
  aws greengrass list-groups --query "Groups[?Name=='MyGroup']"
  ```

  Os nomes de grupo não precisam ser exclusivos, portanto, vários grupos podem ser retornados.

Esta é uma resposta de exemplo do `list-groups`. As informações de cada grupo incluem o ID do grupo (na propriedade do `Id`) e o ID da versão mais recente do grupo (na propriedade da `LatestVersion`). Para obter outras IDs de versão para um grupo, use a ID do grupo com ListGroupVersions.

> **Note**
>
> Você também pode encontrar esses valores no AWS IoT console. O ID do grupo é exibido na página Settings (Configurações) do grupo. Os IDs de versão do grupo são exibidos na guia Implantações do grupo.

```
{
    "Groups": [
        {
            "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/00dedaaa-ac16-484d-ad77-c3eedEXAMPLE/versions/4cbc3f07-fc5e-48c4-
a50e-7d356EXAMPLE",
            "Name": "MyFirstGroup",
            "LastUpdatedTimestamp": "2019-11-11T05:47:31.435Z",
            "LatestVersion": "4cbc3f07-fc5e-48c4-a50e-7d356EXAMPLE",
            "CreationTimestamp": "2019-11-11T05:47:31.435Z",
            "Id": "00dedaaa-ac16-484d-ad77-c3eedEXAMPLE",
            "Arn": "arn:aws:us-west-2:123456789012:/greengrass/groups/00dedaaa-
ac16-484d-ad77-c3eedEXAMPLE"
        },
        {
            "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE/versions/8fe9e8ec-64d1-4647-
b0b0-01dc8EXAMPLE",
            "Name": "GreenhouseSensors",
            "LastUpdatedTimestamp": "2020-01-07T19:58:36.774Z",
            "LatestVersion": "8fe9e8ec-64d1-4647-b0b0-01dc8EXAMPLE",
            "CreationTimestamp": "2020-01-07T19:58:36.774Z",
            "Id": "036ceaf9-9319-4716-ba2a-237f9EXAMPLE",
            "Arn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE"
        },
        ...
    ]
}
```

Se você não especificar um Região da AWS, AWS CLI os comandos usarão a Região padrão do seu perfil. Para devolver grupos em uma região diferente, inclua a *`region`* opção. Por exemplo:

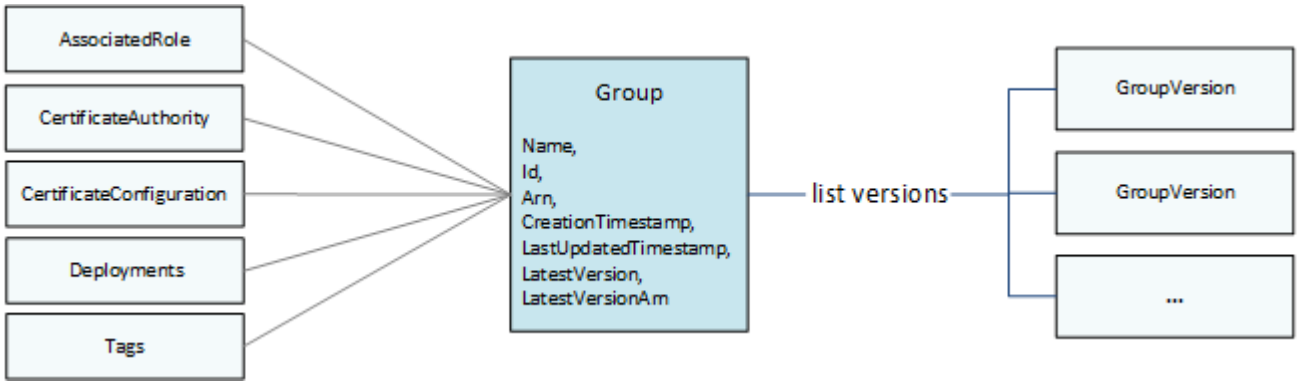
```
aws greengrass list-groups --region us-east-1
```

Visão geral do AWS IoT Greengrass modelo de objeto de grupo

Ao programar com a AWS IoT Greengrass API, é útil entender o modelo de objeto de grupo do Greengrass.

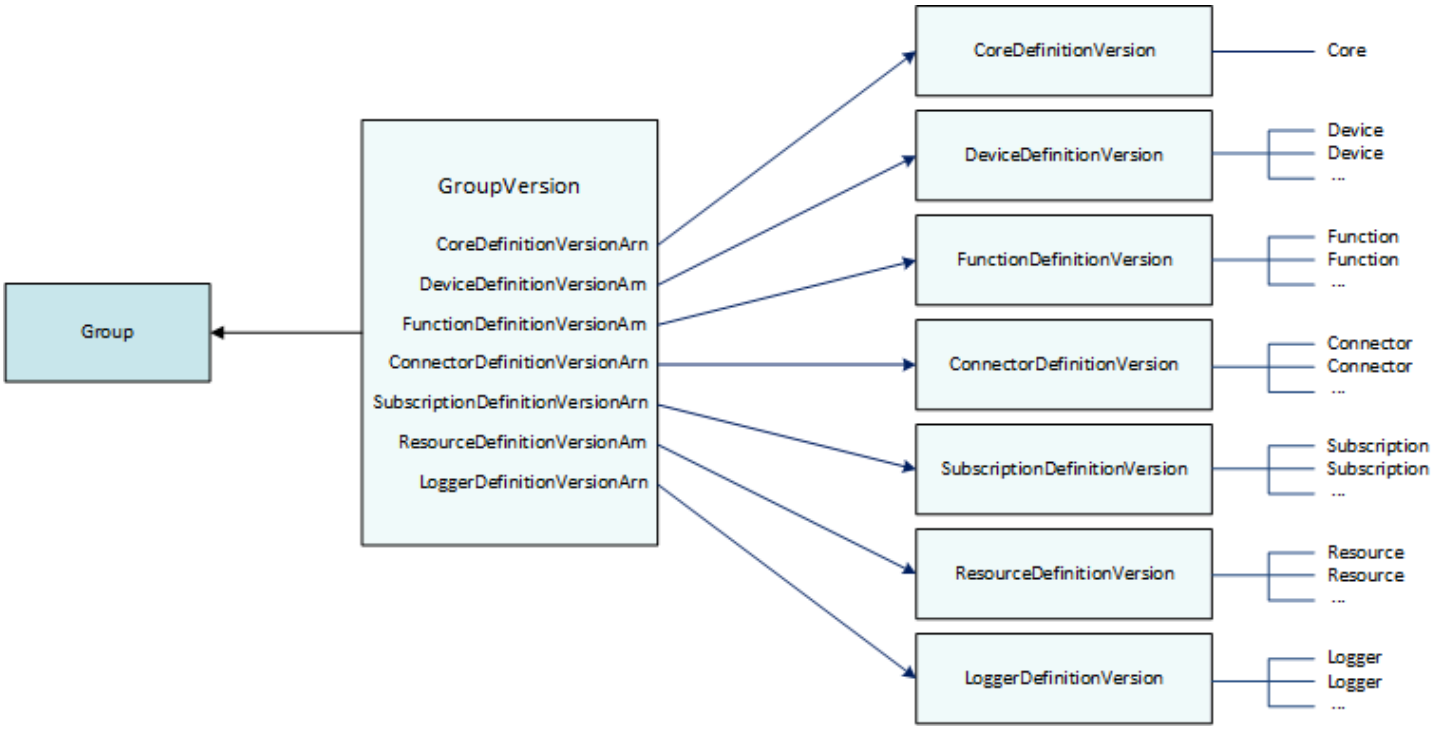
Groups (Grupos)

Na AWS IoT Greengrass API, o `Group` objeto de nível superior consiste em metadados e uma lista de `GroupVersion` objetos. `GroupVersion`os objetos são associados a um `Group` por ID.



Versões do grupo

Os objetos `GroupVersion` definem a associação ao grupo. Cada `GroupVersion` faz referência a um `CoreDefinitionVersion` e a outras versões de componente por ARN. Essas referências determinam quais entidades devem ser incluídas no grupo.



Por exemplo, para incluir três funções do Lambda, um dispositivo e duas assinaturas no grupo, o `GroupVersion` faz referência:

- Ao `CoreDefinitionVersion` que contém o núcleo necessário.
- Ao `FunctionDefinitionVersion` que contém as três funções.
- Ao `DeviceDefinitionVersion` que contém o dispositivo cliente.
- Ao `SubscriptionDefinitionVersion` que contém as duas assinaturas.

O `GroupVersion` implantado em um dispositivo de núcleo determina as entidades que estão disponíveis no ambiente local e como elas podem interagir.

Componentes do grupo

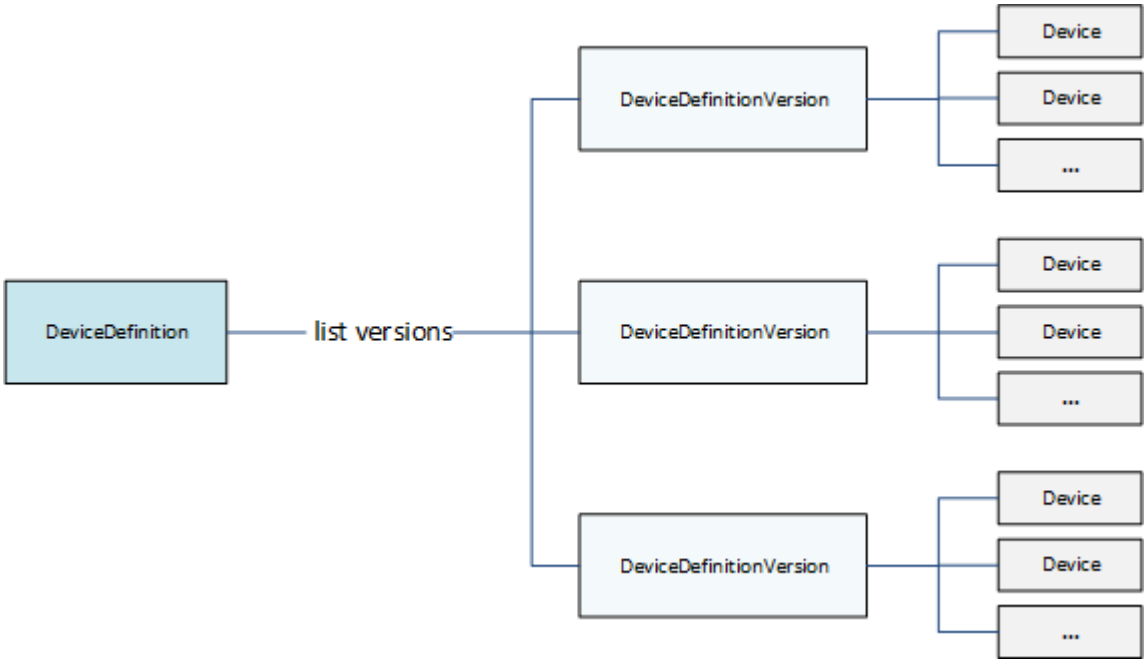
Os componentes que você adiciona a grupos têm uma hierarquia de três níveis:

- Uma definição que faz referência a uma lista de DefinitionVersion objetos de um determinado tipo. Por exemplo, um `DeviceDefinition` faz referência a uma lista de objetos `DeviceDefinitionVersion`.
- Um DefinitionVersion que contém um conjunto de entidades de um determinado tipo. Por exemplo, um `DeviceDefinitionVersion` contém uma lista de objetos `Device`.
- Entidades individuais que definem suas propriedades e seu comportamento. Por exemplo, a `Device` define o ARN do dispositivo cliente correspondente no AWS IoT registro, o ARN do certificado do dispositivo e se a sombra local é sincronizada automaticamente com a nuvem.

  É possível adicionar os seguintes tipos de entidades a um grupo:

  - Conector
  - Serviços
  - Dispositivo
  - Função
  - Logger
  - Recurso
  - Assinatura

O exemplo `DeviceDefinition` a seguir faz referência a três objetos `DeviceDefinitionVersion`, cada um contendo vários objetos `Device`. Somente um `DeviceDefinitionVersion` de cada vez é usado em um grupo.



Atualizar grupos

Na AWS IoT Greengrass API, você usa versões para atualizar a configuração de um grupo. As versões são imutáveis, portanto, para adicionar, remover ou alterar componentes do grupo, você deve criar DefinitionVersion objetos que contenham entidades novas ou atualizadas.

Você pode associar novos DefinitionVersions objetos a objetos de Definição novos ou existentes. Por exemplo, é possível usar a ação `CreateFunctionDefinition` para criar um `FunctionDefinition` que inclui o `FunctionDefinitionVersion` como uma versão inicial ou usar a ação `CreateFunctionDefinitionVersion` e fazer referência a um `FunctionDefinition` existente.

Depois de criar seus componentes de grupo, você cria um `GroupVersion` que contém todos os DefinitionVersion objetos que você deseja incluir no grupo. Depois, implante o `GroupVersion`.

Para implantar um `GroupVersion`, ele deve fazer referência a um `CoreDefinitionVersion` que contenha exatamente um `Core`. Todas as entidades referenciadas devem ser membros do grupo. Além disso, uma função de serviço do Greengrass deve ser associada à sua Conta da AWS no Região da AWS local em que você está implantando o. `GroupVersion`

> **Note**
>
> As ações `Update` na API são usadas para alterar o nome de um `Group` ou de um objeto Definition do componente.

Atualizando entidades que fazem referência a AWS recursos

As funções do Greengrass Lambda e os recursos secretos definem Greengrass-specific propriedades e também fazem referência aos recursos correspondentes AWS . Para atualizar essas entidades, você pode fazer alterações no AWS recurso correspondente em vez dos objetos do Greengrass. Por exemplo, as funções do Lambda fazem referência a uma função AWS Lambda e também definem o ciclo de vida e outras propriedades específicas do grupo Greengrass.

- Para atualizar o código da função do Lambda ou as dependências empacotadas, faça as alterações no AWS Lambda. Durante a próxima implantação em grupo, essas alterações serão recuperadas AWS Lambda e copiadas em seu ambiente local.
- Para atualizar Greengrass-specific as propriedades, você cria uma `FunctionDefinitionVersion` que contém as `Function` propriedades atualizadas.

> **Note**
>
> As funções do Lambda do Greengrass podem fazer referência a uma função do Lambda pelo ARN do alias ou da versão. Se você fizer referência ao ARN do alias (recomendado), não será necessário atualizar o `FunctionDefinitionVersion` (ou o `SubscriptionDefinitionVersion`) ao publicar uma nova versão da função no AWS Lambda. Para obter mais informações, consulte the section called “Fazer referência a funções por alias ou por versão”.

Consulte também

- the section called “Obter notificações de implantação”
- the section called “Redefinir implantações”
- the section called “Criar implantações em massa”
- Solução de problemas de implantação

- AWS IoT Greengrass Version 1 API Reference
- AWS IoT Greengrass comandos na Referência de AWS CLI Comandos

Obter notificações de implantação

As regras de EventBridge eventos da Amazon fornecem notificações sobre mudanças de estado nas implantações do seu grupo Greengrass. 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 para EventBridge pelo menos uma vez. Isso significa que AWS IoT Greengrass pode enviar várias cópias de um determinado evento para garantir a entrega. Além disso, os listeners do evento poderão não receber os eventos na ordem em que estes 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 dispositivos principais do Greengrass e notificações de implantação. Para obter mais informações, consulte O que é a Amazon EventBridge? no Guia do EventBridge usuário da Amazon.

AWS IoT Greengrass emite um evento quando as implantações em grupo mudam de estado. 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.

AWS IoT Greengrass emite um evento quando uma implantação entra nos seguintes estados: `BuildingInProgress`,`Success`, e. `Failure`

> **Note**
>
> Atualmente, não há suporte para o monitoramento do status de uma operação de implantação em massa . No entanto, AWS IoT Greengrass emite eventos de mudança de estado para implantações de grupos individuais que fazem parte de uma implantação em massa.

Evento de alteração de status da implantação do grupo

O evento para uma alteração no estado da implantação usa o seguinte formato:

```
{
    "version":"0",
    "id":" cd4d811e-ab12-322b-8255-EXAMPLEb1bc8",
    "detail-type":"Greengrass Deployment Status Change",
    "source":"aws.greengrass",
    "account":"123456789012",
    "time":"2018-03-22T00:38:11Z",
    "region":"us-west-2",
    "resources":[],
    "detail":{
        "group-id": "284dcd4e-24bc-4c8c-a770-EXAMPLEf03b8",
        "deployment-id": "4f38f1a7-3dd0-42a1-af48-EXAMPLE09681",
        "deployment-type": "NewDeployment|Redeployment|ResetDeployment|
ForceResetDeployment",
        "status": "Building|InProgress|Success|Failure"
    }
}
```

Você pode criar regras que se aplicam a um ou mais grupos. Você pode filtrar regras por um ou mais dos seguintes tipos e estados de implantação:

Tipos de implantação

- `NewDeployment`. A primeira implantação de uma versão de grupo.
- `ReDeployment`. Uma reimplantação de uma versão de grupo.
- `ResetDeployment`. Exclui as informações de implantação armazenadas no Nuvem AWS e no AWS IoT Greengrass núcleo. Para obter mais informações, consulte the section called “Redefinir implantações”.

- `ForceResetDeployment`. Exclui informações de implantação armazenadas na Nuvem AWS e relata sucesso sem esperar que o núcleo responda. Também exclui informações de implantação armazenadas no núcleo se ele estiver conectado ou quando ele se conectar.

Estados de implantação

- `Building`. AWS IoT Greengrass está validando a configuração do grupo e criando artefatos de implantação.
- `InProgress`. A implantação está em andamento no AWS IoT Greengrass núcleo.
- `Success`. A implantação foi bem-sucedida.
- `Failure`. Houve falha na implantação.

É possível que os eventos estejam duplicados ou fora de ordem. Para determinar a ordem dos eventos, use a propriedade `time`.

> **Note**
>
> AWS IoT Greengrass não usa a `resources` propriedade, então ela está sempre vazia.

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 implantação (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 Rule type, escolha Rule with an event pattern.
7. Selecione Next (Próximo).
8. Em Event source (Origem do evento), selecione AWS services (Serviços da).
9. Em Padrão do evento, selecione Serviços da AWS .
10. Em Serviço da AWS , selecione Greengrass.
11. Em Tipo de evento, selecione Greengrass Deployment Status Change (Alteração de status de Implantação do Greengrass).

    > **Note**
    >
    > A chamada de AWS API via tipo de CloudTrail evento é baseada na AWS IoT Greengrass integração com AWS CloudTrail. Você pode usar essa opção para criar regras iniciadas por chamadas de leitura ou gravação para a AWS IoT Greengrass API. Para obter mais informações, consulte the section called “Registro em log AWS IoT Greengrass Chamadas de API com AWS CloudTrail”.
12. Selecione os estados de implantação que iniciam uma notificação.

- Para receber notificações de todos os eventos de alteração de estado, selecione Any state (Qualquer estado).
- Para receber notificações apenas para alguns eventos de alteração de estado, selecione Specific state(s) (Estados específicos) e, depois, selecione os estados de destino.

13. Selecione os tipos de implantação que iniciam uma notificação.

    - Para receber notificações sobre todos os tipos de implantação, selecione Any state (Qualquer estado).
    - Para receber notificações somente para alguns tipos de implantação, selecione Specific state(s) (Estados específicos) e, depois, selecione os tipos de implantação de destino.

14. Selecione Next (Próximo).
15. Em Target types (Tipos de destinos), selecione AWS service (Serviço da).
16. 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. Selecione Next (Próximo).

17. Em Tags, defina tags para a regra ou deixe os campos em branco.
18. Selecione Next (Próximo).
19. Analise os detalhes da regra e selecione Criar regra.

Configurar notificações de implantação (CLI)

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.

1. Crie a regra.

    - *group-id*Substitua pelo ID do seu AWS IoT Greengrass grupo.

```
aws events put-rule \
  --name TestRule \
```

```
  --event-pattern "{\"source\": [\"aws.greengrass\"], \"detail\": {\"group-id\":
[\"group-id\"]}}"
```

As propriedades que são omitidas do padrão são ignoradas.

2. Adicione o tópico como um destino de regra.

   - *topic-arn*Substitua pelo ARN do seu tópico do Amazon SNS.

```
aws events put-targets \
  --rule TestRule \
  --targets "Id"="1","Arn"="topic-arn"
```

> **Note**
>
> Para permitir que EventBridge a Amazon chame 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 implantação (CloudFormation)

Use 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

- Implemente AWS IoT Greengrass grupos
- O que é a Amazon EventBridge? no Guia do EventBridge usuário da Amazon

Redefinir implantações

Esse recurso está disponível para o AWS IoT Greengrass Core v1.1 e versões posteriores.

Você pode querer redefinir as implantações de um grupo para:

- Exclua o grupo, por exemplo, quando você quiser mover o núcleo do grupo para outro grupo ou se o núcleo do grupo tiver sido recriado. Antes de excluir um grupo, você deve redefinir as implantações do grupo para usar o núcleo com outro grupo do Greengrass.
- Mover o núcleo do grupo para um grupo diferente.
- Reverter o grupo para seu estado antes de qualquer implantação.
- Remover a configuração de implantação do dispositivo de núcleo.
- Excluir dados confidenciais do dispositivo de núcleo ou da nuvem.
- Implantar um novo grupo de configuração em um núcleo sem a necessidade de substituir o núcleo por outro no grupo atual.

> **Note**
>
> A funcionalidade de redefinição de implantações não está disponível no software de núcleo do AWS IoT Greengrass v1.0.0. Não é possível excluir um grupo que foi implantado usando a v1.0.0.

A operação de redefinição de implantações primeiro limpa todas as informações de implantação armazenadas na nuvem para determinado grupo. Depois, ele também instrui o dispositivo de núcleo do grupo a limpar todas as informações relacionadas à implantação (funções do função do Lambda, logs de usuário, banco de dados shadow e certificado do servidor, mas não o `config.json` definido pelo usuário nem os certificados de núcleo do Greengrass). Não é possível iniciar uma redefinição de implantações para um grupo se o grupo tiver, no momento, uma implantação com o status `In Progress` ou `Building`.

Redefina as implantações a partir do AWS IoT console

Você pode redefinir as implantações de grupo na página de configuração do grupo no AWS IoT console.

1. No painel de navegação do AWS IoT console, em Gerenciar, expanda dispositivos Greengrass e escolha Grupos (V1).
2. Selecione o grupo de destino.
3. Na guia Implantações, selecione Redefinir implantações.

4. Na caixa de diálogo Redefinir implantações para este grupo do Greengrass, digite **`confirm`** para concordar e selecione Redefinir implantação.

Redefina as implantações com o AWS IoT Greengrass solicitações de

Você pode usar a `ResetDeployments` ação na AWS IoT Greengrass API ou no AWS CLI AWS SDK para redefinir as implantações. Os exemplos deste tópico usam a CLI.

```
aws greengrass reset-deployments --group-id GroupId [--force]
```

Argumentos para o comando da **`CLI reset-deployments`**:

`--group-id`

O ID do grupo. Use o comando `list-groups` para obter esse valor.

`--force`

Opcional. Use esse parâmetro se o dispositivo de núcleo do grupo foi perdido, roubado ou destruído. Essa opção faz com que o processo de implantação de redefinição relate êxito depois que todas as informações de implantação na nuvem tiverem sido limpas, sem esperar que um dispositivo de núcleo responda. No entanto, se o dispositivo de núcleo estiver ou se tornar ativo, ele também executará operações de limpeza.

A saída do comando `reset-deployments` da CLI tem a seguinte aparência:

```
{
    "DeploymentId": "4db95ef8-9309-4774-95a4-eea580b6ceef",
    "DeploymentArn": "arn:aws:greengrass:us-west-2:106511594199:/greengrass/groups/
b744ed45-a7df-4227-860a-8d4492caa412/deployments/4db95ef8-9309-4774-95a4-eea580b6ceef"
}
```

Você pode verificar o status da implantação de redefinição com comando `get-deployment-status` da CLI:

```
aws greengrass get-deployment-status --deployment-id DeploymentId --group-id GroupId
```

Argumentos para o comando da **`CLI get-deployment-status:`**

`--deployment-id`

O ID da implantação.

`--group-id`

O ID do grupo.

A saída do comando `get-deployment-status` da CLI tem a seguinte aparência:

```
{
    "DeploymentStatus": "Success",
    "UpdatedAt": "2017-04-04T00:00:00.000Z"
}
```

O `DeploymentStatus` é definido como `Building` quando a implantação de redefinição está sendo preparada. Quando a implantação redefinida está pronta, mas o AWS IoT Greengrass núcleo não captou a implantação redefinida, `DeploymentStatus` está`InProgress`.

Se a operação de redefinição falhar, as informações de erro serão retornados na resposta.

Consulte também

- Implemente AWS IoT Greengrass grupos
- ResetDeployments na Referência da AWS IoT Greengrass Version 1 API
- GetDeploymentStatusna Referência da AWS IoT Greengrass Version 1 API

Criar implantações em massa para grupos

Você pode usar chamadas de API simples para implantar um grande número de grupos do Greengrass ao mesmo tempo. Essas implantações são acionadas com uma taxa adaptável que tem um limite superior fixo.

Este tutorial descreve como usar o AWS CLI para criar e monitorar uma implantação de grupo em massa no AWS IoT Greengrass. O exemplo de implantação em massa deste tutorial contém vários grupos. Você pode usar o exemplo na sua implementação para adicionar quantos grupos precisar.

O tutorial contém as seguintes etapas de nível elevado:

1. Criar e fazer upload do arquivo de entrada de implementação em massa
2. Criar e configurar uma função de execução do IAM para implantação em lote
3. Permitir o acesso da função de execução ao bucket do S3
4. Implantar os grupos
5. Teste a implantação

Pré-requisitos

Para concluir este tutorial, é necessário:

- Um ou mais grupos implantáveis do Greengrass. Para obter mais informações sobre a criação de grupos e cores do AWS IoT Greengrass , consulte Conceitos básicos de AWS IoT Greengrass.
- O AWS CLI instalado e configurado em sua máquina. Para obter mais informações, consulte o Guia do usuário do AWS CLI.
- Um bucket S3 criado da Região da AWS mesma AWS IoT Greengrass forma que. Para obter mais informações, consulte Criando e configurando um bucket do S3 no Guia do usuário do Amazon Simple Storage Service.

> **Note**
>
> Atualmente, buckets habilitados SSE KMS não são compatíveis.

Etapa 1: Criar e fazer upload do arquivo de entrada de implementação em massa

Nesta etapa, você cria um arquivo de entrada para implantação e faz upload dele para o seu bucket do Amazon S3. Esse arquivo é um arquivo JSON serializado e delimitado por linhas que contém informações sobre cada grupo em sua implantação em massa. AWS IoT Greengrass usa essas informações para implantar cada grupo em seu nome quando você inicializa a implantação de grupos em massa.

1. Execute o seguinte comando a fim de obter o `groupId` para cada grupo que você deseja implantar. Você insere o `groupId` no seu arquivo de entrada da implantação em massa para que o AWS IoT Greengrass possa identificar cada grupo a ser implantado.

> **Note**
>
> Você também pode encontrar esses valores no AWS IoT console. O ID do grupo é exibido na página Settings (Configurações) do grupo. Os IDs de versão do grupo são exibidos na guia Implantações do grupo.

```
aws greengrass list-groups
```

A resposta contém informações sobre cada grupo em sua AWS IoT Greengrass conta:

```
{
  "Groups": [
    {
      "Name": "string",
      "Id": "string",
      "Arn": "string",
      "LastUpdatedTimestamp": "string",
      "CreationTimestamp": "string",
      "LatestVersion": "string",
      "LatestVersionArn": "string"
    }
  ],
  "NextToken": "string"
}
```

Execute o seguinte comando a fim de obter o `groupVersionId` de cada grupo que você deseja implantar.

```
list-group-versions --group-id groupId
```

A resposta contém informações sobre todas as versões no grupo. Anote o ID da valor `Version` para a versão do grupo você deseja usar.

```
{
  "Versions": [
    {
      "Arn": "string",
      "Id": "string",
      "Version": "string",
      "CreationTimestamp": "string"
    }
  ],
  "NextToken": "string"
}
```

2. No terminal do computador ou no editor de sua escolha, crie um arquivo,*MyBulkDeploymentInputFile*, a partir do exemplo a seguir. Esse arquivo contém informações sobre cada AWS IoT Greengrass grupo a ser incluído em uma implantação em massa. Embora esse exemplo defina vários grupos para esse tutorial, seu arquivo pode conter apenas um.

> **Note**
>
> O tamanho do arquivo deve ser menor que 100 MB.

```
{"GroupId":"groupId1", "GroupVersionId":"groupVersionId1",
 "DeploymentType":"NewDeployment"}
{"GroupId":"groupId2", "GroupVersionId":"groupVersionId2",
 "DeploymentType":"NewDeployment"}
{"GroupId":"groupId3", "GroupVersionId":"groupVersionId3",
 "DeploymentType":"NewDeployment"}
...
```

Cada registro (ou linha) contém um objeto do grupo. Cada objeto do grupo contém seu `GroupId` e `GroupVersionId` correspondentes e um `DeploymentType`. Atualmente, AWS IoT Greengrass suporta somente tipos de implantação em `NewDeployment` massa.

Salve e feche seu arquivo. Anote a localização do arquivo.

3. Use o comando a seguir em seu terminal para fazer upload do arquivo de entrada no seu bucket do Amazon S3. Substitua o caminho do arquivo pelo local e nome do seu arquivo. Para obter informações, consulte Adicionar um objeto a um bucket.

```
aws s3 cp path/MyBulkDeploymentInputFile s3://amzn-s3-demo-bucket/
```

Etapa 2: Criar e configurar uma função de execução do IAM

Nesta etapa, você usa o console do IAM para criar uma função de execução autônoma. Em seguida, você estabelece uma relação de confiança entre a função AWS IoT Greengrass e garante que seu usuário do IAM tenha `PassRole` privilégios para sua função de execução. Isso permite AWS IoT Greengrass assumir sua função de execução e criar as implantações em seu nome.

1. Use a política a seguir para criar uma função de execução. Esse documento de política permite que o AWS IoT Greengrass acesse seu arquivo de entrada da implantação em massa ao criar cada implantação em seu nome.

   Para obter mais informações sobre como criar um perfil do IAM e delegar permissões, consulte Criar perfis do IAM.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "greengrass:CreateDeployment",
            "Resource": [
            "arn:aws:greengrass:us-east-1:123456789012:/greengrass/
groups/groupId1",
    "arn:aws:greengrass:us-east-1:123456789012:/greengrass/groups/groupId2",
    "arn:aws:greengrass:us-east-1:123456789012:/greengrass/groups/groupId3"
            ]
        }
    ]
}
```

> **Note**
>
> Essa política deve ter um recurso para cada grupo ou versão do grupo do seu arquivo de entrada da implantação em massa a ser implantado pelo AWS IoT Greengrass. Para permitir o acesso a todos os grupos, para `Resource`, especifique um asterisco:
>
> ```
> "Resource": ["*"]
> ```

2. Modifique a relação de confiança da sua função de execução a fim de incluir o AWS IoT Greengrass. Isso permite ao AWS IoT Greengrass usar sua função de execução e as permissões anexadas a ele. Para obter informações, consulte Editar a relação de confiança de uma função existente.

   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 Cross-service prevenção delegada confusa.

   JSON

   ```
   {
     "Version":"2012-10-17",
     "Statement": [
       {
         "Sid": "",
         "Effect": "Allow",
         "Principal": {
           "Service": "greengrass.amazonaws.com"
         },
         "Action": "sts:AssumeRole",
         "Condition": {
           "StringEquals": {
           "aws:SourceAccount": "123456789012"
           },
           "ArnLike": {
   ```

```
            "aws:SourceArn": "arn:aws:greengrass:us-east-1:123456789012:*"
          }
        }
      }
    ]
}
```

3. Forneça permissões `PassRole` do IAM para a sua função de execução ao usuário do IAM. Esse usuário do IAM é o usado para iniciar a implantação em massa. As permissões `PassRole` permitem que o usuário do IAM passe a sua função de execução ao AWS IoT Greengrass para uso. Para obter mais informações, consulte Conceder permissões a um usuário para passar uma função para um AWS serviço.

   Use o seguinte exemplo para atualizar a política do IAM anexada ao seu perfil de execução. Modifique este exemplo, conforme necessário.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1508193814000",
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::123456789012:user/executionRoleArn"
            ],
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "greengrass.amazonaws.com"
                }
            }
        }
    ]
}
```

Etapa 3: Permitir o acesso da função de execução ao bucket do S3

Para iniciar sua implantação em massa, sua função de execução deve ser capaz de ler o arquivo de entrada da implantação em massa do seu bucket do Amazon S3. Anexe a política de exemplo a seguir ao seu bucket do Amazon S3 para que suas permissões `GetObject` sejam acessíveis à sua função de execução.

Para obter mais informações, consulte Como eu faço para adicionar uma política de bucket do S3?

JSON

```
{
    "Version":"2012-10-17",
    "Id": "examplePolicy",
    "Statement": [
        {
            "Sid": "Stmt1535408982966",
            "Effect": "Allow",
            "Principal": {
                "AWS": [
                    "executionRoleArn"
                ]
            },
            "Action": "s3:GetObject",
            "Resource":
            "arn:aws:s3:::amzn-s3-demo-bucket/objectKey"
        }
    ]
}
```

Você pode usar o comando a seguir em seu terminal para verificar a política do seu bucket:

```
aws s3api get-bucket-policy --bucket amzn-s3-demo-bucket
```

> **Note**
>
> Você pode modificar diretamente sua função de execução a fim de conceder a ela permissões `GetObject` para o seu bucket do Amazon S3. Para isso, anexe a seguinte política de exemplo à sua função de execução.
>
> JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::amzn-s3-demo-bucket/objectKey"
        }
    ]
}
```

Etapa 4: Implantar os grupos

Nesta etapa, você inicia uma operação de implantação em massa para todas as versões do grupo configuradas no seu arquivo de entrada da implantação em massa. A ação de implantação para cada uma de suas versões de grupo é do tipo `NewDeploymentType`.

> **Note**
>
> Você não pode chamar StartBulkDeployment enquanto outra implantação em massa da mesma conta ainda estiver em execução. A solicitação foi rejeitada.

1. Use o comando a seguir para iniciar a implantação em massa.

   Recomendamos que você inclua um token `X-Amzn-Client-Token` em cada solicitação StartBulkDeployment. Essas solicitações são idempotentes com relação ao token e os

parâmetros da solicitação. Esse token pode ser qualquer string exclusiva e com diferenciação entre maiúsculas de minúsculas de até 64 caracteres ASCII.

```
aws greengrass start-bulk-deployment --cli-input-json "{
          "InputFileUri":"URI of file in S3 bucket",
          "ExecutionRoleArn":"ARN of execution role",
          "AmznClientToken":"your Amazon client token"
          }"
```

O comando deve resultar em um código de status bem-sucedido `200`, junto com a seguinte resposta:

```
{
  "bulkDeploymentId": UUID
}
```

Anote o ID da implantação em massa. Ele pode ser usado para verificar o status de sua implantação em massa.

> **Note**
>
> Embora as operações de implantação em massa não sejam suportadas atualmente, você pode criar regras de EventBridge eventos da Amazon para receber notificações sobre mudanças de status de implantação para grupos individuais. Para obter mais informações, consulte the section called “Obter notificações de implantação”.

2. Use o comando a seguir para verificar o status da sua implantação em massa.

```
aws greengrass get-bulk-deployment-status --bulk-deployment-id 1234567
```

O comando deve retornar um código de status bem-sucedido `200`, além de uma carga JSON de informações:

```
 {
  "BulkDeploymentStatus": Running,
```

```
  "Statistics": {
     "RecordsProcessed": integer,
     "InvalidInputRecords": integer,
     "RetryAttempts": integer
  },
  "CreatedAt": "string",
  "ErrorMessage": "string",
  "ErrorDetails": [
    {
      "DetailedErrorCode": "string",
      "DetailedErrorMessage": "string"
    }
  ]
}
```

`BulkDeploymentStatus` contém o status atual da execução em massa. A execução pode ter um dos seis status diferentes:

- `Initializing`. A solicitação de implantação em massa foi recebida, e a execução está prestes a ser iniciada.
- `Running`. A execução da implantação em massa foi iniciada.
- `Completed`. A execução da implantação em massa finalizou o processamento de todos os registros.
- `Stopping`. A execução da implantação em massa recebeu um comando de interrupção e será encerrada em breve. Não é possível iniciar uma nova implantação em massa enquanto uma implantação anterior está em estado `Stopping`.
- `Stopped`. A execução da implantação em massa foi interrompida manualmente.
- `Failed`. A execução da implantação em massa encontrou um erro e foi encerrada. É possível encontrar detalhes do erro no campo `ErrorDetails`.

A carga JSON também inclui informações estatísticas sobre o progresso da implantação em massa. Você pode usar essas informações para determinar a quantidade de grupos que foram processados e quantos deles falharam. As informações estatísticas incluem:

- `RecordsProcessed`: o número de registro de tentativas do grupo.
- `InvalidInputRecords`: o número total de registros que não apresentaram erro de nova tentativa. Por exemplo, isso pode ocorrer se o registro de um grupo do arquivo de entrada usa

um formato inválido ou especifica uma versão de grupo inexistente, ou se a execução não concede permissão para implantar um grupo ou versão do grupo.

- `RetryAttempts`: o número de tentativas de implantação que apresentaram erro de nova tentativa. Por exemplo, uma nova tentativa é acionada se a tentativa de implantar um grupo retorna um erro de limitação. A implantação do grupo pode ser repetida até cinco vezes.

No caso de uma falha de execução da implantação em massa, essa carga também inclui uma seção `ErrorDetails` que pode ser usada para solução de problemas. Ela contém informações sobre a causa da falha de execução.

Você pode verificar periodicamente o status da implantação em massa para confirmar se ela está progredindo conforme esperado. Depois de concluir a implantação, `RecordsProcessed` deve ser igual ao número de grupos de implantação em seu arquivo de entrada da implantação em massa. Isso indica que cada registro foi processado.

Etapa 5: Testar a implantação

Use o comando ListBulkDeployments para encontrar o ID da implantação em massa.

```
aws greengrass list-bulk-deployments
```

Esse comando retorna uma lista de todas as suas implantações em massa da mais recente para a menos recente, incluindo o `BulkDeploymentId`.

```
{
  "BulkDeployments": [
    {
      "BulkDeploymentId": 1234567,
      "BulkDeploymentArn": "string",
      "CreatedAt": "string"
    }
  ],
  "NextToken": "string"
}
```

Agora, chame o comando ListBulkDeploymentDetailedReports para reunir informações detalhadas sobre cada implantação.

```
aws greengrass list-bulk-deployment-detailed-reports --bulk-deployment-id 1234567
```

O comando deve retornar um código de status bem-sucedido `200`, junto com uma carga JSON de informações:

```
{
  "BulkDeploymentResults": [
    {
      "DeploymentId": "string",
      "GroupVersionedArn": "string",
      "CreatedAt": "string",
      "DeploymentStatus": "string",
      "ErrorMessage": "string",
      "ErrorDetails": [
        {
          "DetailedErrorCode": "string",
          "DetailedErrorMessage": "string"
        }
      ]
    }
  ],
  "NextToken": "string"
}
```

Essa carga geralmente contém uma lista paginada de cada implantação e seu status de implantação da mais recente para a menos recente. Ela também contém mais informações no caso de uma falha de execução da implantação em massa. Novamente, o número total de implantações listadas deve ser igual ao número de grupos que você identificou no seu arquivo de entrada da implantação em massa.

As informações retornadas podem ser alteradas até que as implantações estejam em estado terminal (êxito ou falha). Você pode chamar esse comando periodicamente até lá.

Solução de problemas de implantações em massa

Se a implantação em massa não for bem-sucedida, você poderá tentar as etapas de solução de problemas a seguir. Execute os comandos no seu terminal.

Solução de problemas de erros do arquivo de entrada

A implantação em massa pode falhar em caso de erros de sintaxe no arquivo de entrada da implantação em massa. Isso retorna um status `Failed` de implantação em massa com uma mensagem de erro indicando o número da linha do primeiro erro de validação. Existem quatro erros possíveis:

- ```
  InvalidInputFile: Missing GroupId at line number: line number
  ```

  Esse erro indica que a linha fornecida do arquivo de entrada não é capaz de registrar o parâmetro especificado. Os possíveis parâmetros ausentes são `GroupId` e `GroupVersionId`.

- ```
  InvalidInputFile: Invalid deployment type at line number : line number. Only valid
   type is 'NewDeployment'.
  ```

  Esse erro indica a linha fornecida do arquivo de entrada lista um tipo inválido de implantação. No momento, o único tipo de implantação compatível é `NewDeployment`.

- ```
  Line %s is too long in S3 File. Valid line is less than 256 chars.
  ```

  Esse erro indica que a linha fornecida do arquivo de entrada é longa demais e deve ser reduzida.

- ```
  Failed to parse input file at line number: line number
  ```

  Esse erro indica que a linha fornecida do arquivo de entrada não é considerada um json válido.

Verifique se há implantações simultâneas em massa

Não é possível iniciar uma nova implantação em massa enquanto outra ainda está em execução ou em estado não terminal. Isso pode resultar em um `Concurrent Deployment Error`. Você

pode usar o comando ListBulkDeployments para verificar se uma implantação em massa está sendo executada no momento. Esse comando lista as implantações em massa da mais recente para a menos recente.

```
{
  "BulkDeployments": [
    {
      "BulkDeploymentId": BulkDeploymentId,
      "BulkDeploymentArn": "string",
      "CreatedAt": "string"
    }
  ],
  "NextToken": "string"
}
```

Use o `BulkDeploymentId` da primeira implantação em massa listada para executar o comando GetBulkDeploymentStatus. Se a sua implantação em massa mais recente estiver em um estado de execução (`Initializing` ou `Running`), use o comando a seguir para interromper a implantação em massa.

```
aws greengrass stop-bulk-deployment --bulk-deployment-id BulkDeploymentId
```

Essa ação resulta em um status `Stopping` até que a implantação seja `Stopped`. Após a implantação atingir o status `Stopped`, você pode iniciar uma nova implantação em massa.

Verifique ErrorDetails

Execute o comando `GetBulkDeploymentStatus` para retornar uma carga JSON que contém informações sobre qualquer falha de execução da implantação em massa.

```
  "Message": "string",
  "ErrorDetails": [
    {
      "DetailedErrorCode": "string",
      "DetailedErrorMessage": "string"
    }
  ]
```

Ao obter um erro, a carga JSON `ErrorDetails` retornada por essa chamada contém mais informações sobre a falha de execução da implantação em massa. Um código de status de erro na série `400`, por exemplo, indica um erro de entrada, nos parâmetros de entrada ou nas dependências do chamador.

Verifique o AWS IoT Greengrass registro principal

Você pode solucionar problemas visualizando os registros AWS IoT Greengrass principais. Use os comandos a seguir para visualizar o `runtime.log`:

```
cd /greengrass/ggc/var/log
sudo cat system/runtime.log | more
```

Para obter mais informações sobre AWS IoT Greengrass registro em log, consulteMonitoramento com AWS IoT Greengrass logs.

Consulte também

Para saber mais, consulte os seguintes recursos:

- Implemente AWS IoT Greengrass grupos
- Comandos do API do Amazon S3 na Referência de comandos da AWS CLI .
- AWS IoT Greengrass comandos na Referência de AWS CLI Comandos

Execute funções Lambda no AWS IoT Greengrass core

AWS IoT Greengrass fornece um ambiente de execução Lambda em contêineres para o código definido pelo usuário no qual você cria. AWS Lambda Funções Lambda que são implantadas em um AWS IoT Greengrass núcleo executadas no tempo de execução Lambda local do núcleo. As funções do Lambda locais podem ser acionadas por eventos locais, mensagens da nuvem e outras fontes, o que oferece funcionalidade de computação local para dispositivos cliente. Por exemplo, você pode usar funções do Lambda do Greengrass para filtrar dados do dispositivo antes de transmitir os dados para a nuvem.

Para implantar uma função do Lambda para um núcleo, você adiciona a função a um grupo do Greengrass (referenciando a função do Lambda existente), define as configurações específicas de grupo da função e implanta o grupo. Se a função acessar AWS serviços, você também deverá adicionar todas as permissões necessárias à função de grupo do Greengrass.

Você pode configurar parâmetros que determinam como as funções do Lambda são executadas, incluindo permissões, isolamento, limites de memória e muito mais. Para obter mais informações, consulte the section called “Controlando a execução da função do Lambda do Greengrass”.

> **Note**
>
> Essas configurações também possibilitam a execução AWS IoT Greengrass em um contêiner Docker. Para obter mais informações, consulte the section called “Executar AWS IoT Greengrass em um contêiner Docker”.

A tabela a seguir lista os AWS Lambda tempos de execução suportados e as versões do software AWS IoT Greengrass Core nas quais eles podem ser executados.

Linguagem ou plataforma	Versão do GGC
Python 3.8	1.11
Python 3.7	1.9 ou posterior
Python 2.7 *	1.0 ou posterior
Java 8	1.1 ou posterior

Linguagem ou plataforma	Versão do GGC
Node.js 12x*	1.10 ou posterior
Node.js 8,10 *	1.9 ou posterior
Node.js 6,10 *	1.1 ou posterior
C, C++	1.6 ou posterior

* Você pode executar funções Lambda que usam esses tempos de execução em versões compatíveis do AWS IoT Greengrass, mas não pode criá-las no. AWS Lambda Se o runtime de seu dispositivo for diferente do runtime do Lambda AWS especificado para essa função, você poderá escolher seu próprio runtime usando `FunctionRuntimeOverride` no `FunctionDefintionVersion`. Para obter mais informações, consulte CreateFunctionDefinition. Para obter mais informações sobre os runtimes compatíveis, consulte a Política de suporte do runtime no Guia do desenvolvedor do AWS Lambda .

SDKs para funções do Lambda do Greengrass

AWS fornece três SDKs que podem ser usados pelas funções do Greengrass Lambda executadas em um núcleo. AWS IoT Greengrass Esses SDKs estão contidos em pacotes separados para que as funções possam usá-los simultaneamente. Para usar um SDK em uma função do Lambda do Greengrass, inclua-o no pacote de implantação da função do Lambda que você transfere por upload para o AWS Lambda.

AWS IoT Greengrass SDK principal

Habilita funções do Lambda locais para interagir com o núcleo para:

- Troque mensagens MQTT com AWS IoT Core.
- Troque mensagens MQTT com conectores, dispositivos cliente e outras funções do Lambda no grupo do Greengrass.
- Interaja com o serviço de shadow local.
- Invoque outras funções locais do Lambda.
- Acesse recursos secretos.
- Interaja com o gerenciador de fluxo.

AWS IoT Greengrass fornece o SDK AWS IoT Greengrass principal nas seguintes linguagens e plataformas em GitHub.

- AWS IoT Greengrass SDK principal para Java
- AWS IoT Greengrass SDK principal para Node.js
- AWS IoT Greengrass SDK principal para Python
- AWS IoT Greengrass SDK principal para C

Para incluir a dependência do AWS IoT Greengrass Core SDK no pacote de implantação da função Lambda:

1. Baixe a linguagem ou a plataforma do pacote AWS IoT Greengrass Core SDK que corresponda ao tempo de execução da sua função Lambda.
2. Descompacte o pacote obtido por download para obter o SDK. O SDK é a pasta do `greengrasssdk`.
3. Inclua `greengrasssdk` no pacote de implantação da função do Lambda que contém seu código de função. Esse é o pacote para o qual você faz o upload AWS Lambda ao criar a função Lambda.

StreamManagerClient

Somente os seguintes SDKs AWS IoT Greengrass principais podem ser usados para operações do gerenciador de streams:

- SDK do Java (v1.4.0 ou posterior)
- SDK do Python (v1.5.0 ou posterior)
- Node.js SDK (v1.6.0 ou posterior)

Para usar o SDK AWS IoT Greengrass principal para Python para interagir com o gerenciador de streams, você deve instalar o Python 3.7 ou posterior. Você também deve instalar dependências para incluir em seus pacotes de implantação da função do Lambda em Python:

1. Navegue até o diretório SDK que contém o arquivo `requirements.txt`. Esse arquivo lista as dependências.
2. Instale as dependências do SDK. Por exemplo, execute o seguinte comando `pip` para instalar as dependências no diretório atual:

```
pip install --target . -r requirements.txt
```

Instale o SDK AWS IoT Greengrass principal para Python no dispositivo principal

Se você estiver executando funções do Python Lambda, você também pode usar `pip`para instalar o AWS IoT Greengrass SDK principal para Python no dispositivo principal. Em seguida, você poderá implantar suas funções sem incluir o SDK no pacote de implantação da função do Lambda. Para obter mais informações, consulte greengrasssdk.

Esse suporte destina-se a núcleos com restrição de tamanho. Recomendamos que você inclua o SDK em seus pacotes de implantação de função do Lambda quando possível.

AWS IoT Greengrass SDK de Machine Learning

Permite que as funções locais do Lambda consumam modelos de machine learning (ML) implantados no núcleo do Greengrass como recursos de ML. As funções do Lambda podem usar o SDK para invocar e interagir com um serviço de inferência local implantado no núcleo como um conector. As funções do Lambda e os conectores de ML também podem usar o SDK para enviar dados ao conector de feedback de ML para fazer uploads e publicações. Para obter mais informações, incluindo exemplos de código que usam o SDK, consulte the section called “Classificação de imagens do ML”, the section called “Detecção de objetos do ML” e the section called “Feedback do ML”.

A tabela a seguir lista os idiomas ou plataformas compatíveis com as versões do SDK e as versões do software AWS IoT Greengrass principal nas quais elas podem ser executadas.

Versão do SDK	Linguagem ou plataforma	Versão necessária do GGC	Changelog
1.1.0	Python 3.7 ou 2.7	1.9.3 ou posterior	Adicionado o suporte ao Python 3.7 e novo

Versão do SDK	Linguagem ou plataforma	Versão necessária do GGC	Changelog
			cliente `feedback`.
1.0.0	Python 2.7	1.7 ou posterior	Versão inicial.

Para baixar as informações, consulte the section called “AWS IoT Greengrass Software ML SDK”.

AWS SDKs

Permite que as funções locais do Lambda façam chamadas diretas para AWS serviços, como Amazon S3, DynamoDB e. AWS IoT AWS IoT Greengrass Para usar um SDK do AWS em uma função do Lambda do Greengrass, você deverá incluí-lo no pacote de implantação. Ao usar o AWS SDK no mesmo pacote do SDK AWS IoT Greengrass principal, certifique-se de que suas funções do Lambda usem os namespaces corretos. As funções do Lambda do Greengrass não podem se comunicar com serviços em nuvem quando o núcleo está off-line.

Faça o download dos AWS SDKs no Centro de recursos de introdução.

Para obter mais informações sobre como criar um pacote de implantação, consulte the section called “Crie e empacote uma função do Lambda” no tutorial Conceitos básicos ou Criar um pacote de implantação no Guia do desenvolvedor do AWS Lambda .

Migrando funções do Lambda baseadas em nuvem

O SDK AWS IoT Greengrass principal segue o modelo de programação do AWS SDK, o que facilita a portabilidade de funções Lambda desenvolvidas para a nuvem para funções Lambda executadas em um núcleo. AWS IoT Greengrass

Por exemplo, a função Python Lambda a seguir usa o AWS SDK para Python (Boto3) para publicar uma mensagem no tópico `some/topic` na nuvem:

```
import boto3

iot_client = boto3.client("iot-data")
```

```
response = iot_client.publish(
    topic="some/topic", qos=0, payload="Some payload".encode()
)
```

Para portar a função para um AWS IoT Greengrass núcleo, na `import` instrução e na `client` inicialização, altere o nome do `boto3` módulo para`greengrasssdk`, conforme mostrado no exemplo a seguir:

```
import greengrasssdk

iot_client = greengrasssdk.client("iot-data")
iot_client.publish(topic="some/topic", qos=0, payload="Some payload".encode())
```

> **Note**
>
> O SDK AWS IoT Greengrass principal suporta o envio de mensagens MQTT somente com QoS = 0. Para obter mais informações, consulte the section called “Enviar mensagem sobre a qualidade de serviço”.

A semelhança entre os modelos de programação também possibilita que você desenvolva suas funções Lambda na nuvem e depois as AWS IoT Greengrass migre com o mínimo esforço. Os executáveis do Lambda não são executados na nuvem, então você não pode usar o AWS SDK para desenvolvê-los na nuvem antes da implantação.

Referência de funções do Lambda por alias ou versão

Os grupos do Greengrass podem fazer referência a uma função do Lambda por alias (recomendado) ou por versão. Usar um alias facilita o gerenciamento de atualizações de código porque você não precisa alterar a tabela de assinaturas nem a definição do grupo ao atualizar a função do código. Em vez disso, você pode simplesmente apontar o alias para a nova versão da função. Os aliases se tornam números de versão durante a implantação do grupo. Quando você usa aliases, a versão resolvida é atualizada para a versão para a qual o alias aponta no momento da implantação.

AWS IoT Greengrass não oferece suporte a aliases Lambda para as versões $LATEST. As versões $LATEST não estão vinculadas a versões de função imutável e publicada e podem ser alteradas a qualquer momento, que é a resposta ao princípio do AWS IoT Greengrass de imutabilidade da versão.

Uma prática comum para manter as funções do Lambda do Greengrass atualizadas com alterações feitas no código é usar um alias chamado **`PRODUCTION`** no grupo e nas assinaturas do Greengrass. Conforme você promove novas versões da função do Lambda para produção, aponte o alias para a versão estável mais recente e, em seguida, reimplante o grupo. Você também pode usar esse método a fim de reverter para uma versão anterior.

Controlar a execução de funções do Lambda do Greengrass usando a configuração específica do grupo

AWS IoT Greengrass fornece gerenciamento baseado em nuvem das funções do Greengrass Lambda. Embora o código e as dependências de uma função Lambda sejam gerenciados usando AWS Lambda, você pode configurar como a função Lambda se comporta quando é executada em um grupo do Greengrass.

Group-specific definições de configuração

AWS IoT Greengrass fornece as seguintes configurações específicas do grupo para as funções do Greengrass Lambda.

Usuário e grupo do sistema

A identidade de acesso usada para executar cada função do Lambda. Por padrão, as funções do Lambda são executadas como identidade de acesso padrão do grupo. Normalmente, essas são as contas do sistema do AWS IoT Greengrass sistema padrão (ggc_user ggc_group). Você pode alterar a configuração de permissões e escolher o ID de usuário e o ID de grupo que tem as permissões necessárias para executar a função do Lambda. Você pode substituir o UID e o GID ou apenas um, se deixar o outro campo em branco. Essa configuração oferece a você um controle mais detalhado sobre o acesso aos recursos do dispositivo. Recomendamos que você configure seu hardware do Greengrass com os limites de recursos apropriados, permissões de arquivo e cotas de disco para os usuários e grupos cujas permissões são usadas para executar funções do Lambda.

Esse recurso está disponível para AWS IoT Greengrass Core v1.7 e versões posteriores.

> **Important**
>
> Convém evitar a execução das funções do Lambda como raiz, a menos que isso seja absolutamente necessário. Executar como root aumenta os seguintes riscos:

- O risco de alterações não intencionais, como excluir acidentalmente um arquivo essencial.
- O risco para seus dados e dispositivos originário de indivíduos mal-intencionados.
- O risco de fuga do contêiner quando os contêineres do Docker são executados com `--net=host` e `UID=EUID=0`.

Se você precisar executar como root, deverá atualizar a AWS IoT Greengrass configuração para habilitá-la. Para obter mais informações, consulte the section called "Executar uma função do Lambda como raiz".

ID do usuário do sistema (número)

O ID do usuário que tem as permissões necessárias para executar a função do Lambda. Essa configuração só estará disponível se você optar por executar como Outra ID/group ID de usuário. Você pode usar o getent passwd comando em seu dispositivo AWS IoT Greengrass principal para pesquisar o ID de usuário que deseja usar para executar a função Lambda.

Se você usar o mesmo UID para executar processos e a função do Lambda em um dispositivo de núcleo do Greengrass, sua função de grupo do Greengrass poderá conceder credenciais temporárias aos processos. Os processos podem usar as credenciais temporárias nas implantações do Greengrass no núcleo.

ID do grupo do sistema (número)

O ID do grupo que tem as permissões necessárias para executar a função do Lambda. Essa configuração só estará disponível se você optar por executar como Outra ID/group ID de usuário. Você pode usar o getent group comando no seu dispositivo AWS IoT Greengrass principal para pesquisar o ID do grupo que deseja usar para executar a função Lambda.

Conteinerização da função do Lambda

Selecione se a função do Lambda será executada com o conteinerização padrão para o grupo ou especifique a conteinerização que sempre deverá ser usada para essa função do Lambda.

O modo de conteinerização de uma função do Lambda determina seu nível de isolamento.

- As funções do Lambda conteinerizadas são executadas no modo Contêiner do Greengrass. A função Lambda é executada em um ambiente de tempo de execução isolado (ou namespace) dentro do contêiner. AWS IoT Greengrass

- Non-containerized As funções Lambda são executadas no modo Sem contêiner. As funções do Lambda são executadas como processo normal do Linux sem nenhum isolamento.

Esse recurso está disponível para AWS IoT Greengrass Core v1.7 e versões posteriores.

Recomendamos executar funções do Lambda em um contêiner do Greengrass, a menos que o seu caso de uso exija que essas funções sejam executadas sem conteinerização do Greengrass. Quando as funções do Lambda são executadas em um contêiner do Greengrass, é possível anexar os recursos locais e de dispositivos, além de obter os benefícios de isolamento e maior segurança. Antes de alterar a conteinerização, consulte the section called “Considerações ao escolher a conteinerização de função do Lambda”.

> **Note**
>
> Para executar sem habilitar o namespace e o cgroup do kernel do seu dispositivo, todas as funções do Lambda deverão ser executadas sem conteinerização. Você pode fazer isso facilmente ao configurar a conteinerização padrão para o grupo. Para mais informações, consulte the section called “Definir a conteinerização padrão para funções do Lambda em um grupo”.

Limite de memória

A alocação da memória para a função. O padrão é 16 MB.

> **Note**
>
> A configuração de limite de memória se torna indisponível quando você altera a função do Lambda para a execução sem conteinerização. As funções do Lambda que são executadas sem conteinerização não têm limite de memória. A configuração de limite de memória é descartada quando você altera a função do Lambda ou a configuração de conteinerização padrão do grupo para ser executada sem conteinerização.

Timeout (Tempo limite)

O tempo até a função ou a solicitação ser encerrada. O padrão é 3 segundos.

Pinned

O ciclo de vida da função do Lambda pode ser sob demanda ou de longa duração. O padrão é sob demanda.

Uma função do Lambda sob demanda começa em um contêiner novo ou reutilizado quando invocada. As solicitações para a função podem ser processadas por qualquer contêiner disponível. Uma função Lambda de longa duração ou fixa é iniciada automaticamente após o AWS IoT Greengrass início e continua sendo executada em seu próprio contêiner (ou sandbox). Todas as solicitações para a função são processadas pelo mesmo contêiner. Para obter mais informações, consulte the section called “Configuração do ciclo de vida”.

Acesso de leitura ao diretório /sys

Se a função pode acessar a pasta /sys do host. Use-a quando a função tiver de ler informações do dispositivo de /sys. O padrão é falso.

> **Note**
>
> Essa configuração não estará disponível quando você executar uma função do Lambda sem conteinerização. O valor dessa configuração é descartado quando você altera a função do Lambda para a execução sem conteinerização.

Tipo de codificação

O tipo de codificação esperada da carga de entrada para a função, JSON ou binário. O padrão é JSON.

O suporte para o tipo de codificação binária está disponível a partir do AWS IoT Greengrass Core Software v1.5.0 e do AWS IoT Greengrass Core SDK v1.1.0. Aceitar dados de entrada binários pode ser útil para funções que interagem com dados do dispositivo, porque os recursos de hardware restritos de dispositivos normalmente dificultam ou impossibilitam a criação de um tipo de dados JSON.

> **Note**
>
> Os executáveis do Lambda só oferecem suporte ao tipo de codificação binária, não JSON.

Argumentos do processo

Os argumentos de linha de comando são passados para a função do Lambda quando ela é executada.

Variáveis de ambiente

Key-value pares que podem passar dinamicamente configurações para códigos de funções e bibliotecas. As variáveis de ambiente locais funcionam da mesma maneira que variáveis de ambiente da função AWS Lambda, mas estão disponíveis no ambiente do núcleo.

Resource access policies (Políticas de acesso ao recurso)

Uma lista de até 10 recursos locais, recursos de segredos e recursos de machine learning que a função do Lambda tem permissão para acessar, além da permissão `read-only` ou `read-write` correspondente. No console, esses recursos afiliados estão listados na página de configuração do grupo na guia Recursos.

O modo de conteinerização afeta o modo como as funções do Lambda podem acessar recursos locais de dispositivo e volume e recursos de machine learning.

- Non-containerized As funções Lambda devem acessar recursos locais do dispositivo e do volume diretamente por meio do sistema de arquivos no dispositivo principal.
- Para permitir que funções do Lambda não conteinerizadas acessem recursos de machine learning no grupo do Greengrass, você deve definir o proprietário do recurso e as propriedades de permissões de acesso no recurso de machine learning. Para obter mais informações, consulte the section called “Acesse os recursos de machine learning”.

Para obter informações sobre como usar a AWS IoT Greengrass API para definir configurações específicas do grupo para funções Lambda definidas pelo usuário, CreateFunctionDefinitionconsulte na Referência da API ou AWS IoT Greengrass Version 1 create-function-definition na Referência de Comandos.AWS CLI Para implantar funções do Lambda em um núcleo do Greengrass, crie uma versão de definição de função que contenha suas funções, crie uma versão de grupo que faça referência à versão de definição de função e outros componentes do grupo e implante o grupo.

Executar uma função do Lambda como raiz

Esse recurso está disponível para AWS IoT Greengrass Core v1.7 e versões posteriores.

Antes de executar uma ou mais funções do Lambda como root, você deve primeiro atualizar a AWS IoT Greengrass configuração para habilitar o suporte. O suporte para executar as funções

do Lambda como raiz está desativado por padrão. A implantação falhará se você tentar implantar uma função Lambda e executá-la como root (UID e GID de 0) e não tiver atualizado a configuração. AWS IoT Greengrass Um erro como o seguinte aparece no registro de tempo de execução (*greengrass_root*/ggc/var/log/system/runtime.log):

```
lambda(s)
[list of function arns] are configured to run as root while Greengrass is not
 configured to run lambdas with root permissions
```

> **Important**
>
> Convém evitar a execução das funções do Lambda como raiz, a menos que isso seja absolutamente necessário. Executar como root aumenta os seguintes riscos:
>
> - O risco de alterações não intencionais, como excluir acidentalmente um arquivo essencial.
> - O risco para seus dados e dispositivos originário de indivíduos mal-intencionados.
> - O risco de fuga do contêiner quando os contêineres do Docker são executados com `--net=host` e `UID=EUID=0`.

Para permitir que as funções do Lambda sejam executadas como raiz

1. No seu AWS IoT Greengrass dispositivo, navegue até a pasta *greengrass-root* /config.

   > **Note**
   >
   > Por padrão, *greengrass-root* é o diretório /greengrass.

2. Atualize o arquivo config.json para adicionar `"allowFunctionsToRunAsRoot" : "yes"` ao campo `runtime`. Por exemplo:

   ```
   {
     "coreThing" : {
       ...
     },
     "runtime" : {
       ...
       "allowFunctionsToRunAsRoot" : "yes"
     },
   ```

```
  ...
}
```

3. Use os seguintes comandos para reiniciar AWS IoT Greengrass:

```
cd /greengrass/ggc/core
sudo ./greengrassd restart
```

Agora você pode definir o ID do usuário e o ID do grupo (UID/GID) das funções do Lambda como 0 para executar essa função do Lambda como root.

Você pode alterar o valor de `"allowFunctionsToRunAsRoot"` to `"no"` e reiniciar AWS IoT Greengrass se quiser impedir que as funções Lambda sejam executadas como root.

Considerações ao escolher a conteinerização de função do Lambda

Esse recurso está disponível para AWS IoT Greengrass Core v1.7 e versões posteriores.

Por padrão, as funções Lambda são executadas dentro de um AWS IoT Greengrass contêiner. Esse contêiner fornece isolamento entre as funções e o host, fornecendo segurança adicional para o host e as funções no contêiner.

Recomendamos executar funções do Lambda em um contêiner do Greengrass, a menos que o seu caso de uso exija que essas funções sejam executadas sem conteinerização do Greengrass. Ao executar suas funções do Lambda em um contêiner do Greengrass, você terá mais controle sobre como restringir o acesso a recursos.

Aqui estão alguns exemplos de casos de uso para a execução sem conteinerização:

- Você deseja executar AWS IoT Greengrass em um dispositivo que não ofereça suporte ao modo contêiner (por exemplo, porque você está usando uma distribuição Linux especial ou tem uma versão do kernel muito antiga).
- 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 de acesso a recursos locais com caminhos que não podem ser determinados no momento da implantação ou cujos caminhos podem mudar após a implantação, como alguns dispositivos conectáveis.
- Você tem um aplicativo antigo que foi escrito como um processo e encontrou problemas ao executá-lo como uma função do Lambda em contêiner.

Diferenças de conteinerização

Conteinerização	Observações
Contêiner do Greengrass	• Todos os AWS IoT Greengrass recursos estão disponíveis quando você executa uma função 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 se elas forem executadas com o mesmo ID do grupo. Em outras palavras, as funções do Lambda são executadas com maior isolamento umas das outras. • Como as funções Lambda que são executadas em um AWS IoT Greengrass contêiner têm todos os processos secundári os executados no mesmo contêiner da função Lambda, os processos secundários são encerrados quando a função Lambda é encerrada.

Nenhum contêiner	• Os atributos a seguir não estão disponíve is para funções do Lambda não conteiner izadas:
	  • Limites de memória de funções do Lambda.
	  • Recursos de volume e dispositivo locais. É necessário acessar esses recursos diretamente no dispositivo de núcleo em vez de acessá-los como membros do grupo do Greengrass.
	• Se sua função do Lambda não conteiner izada acessar um recurso de machine

Conteinerização	Observações
	learning, você deverá identificar um proprietá rio do recurso e definir permissões de acesso no recurso, não na função do Lambda. Isso requer o software AWS IoT Greengrass Core v1.10 ou posterior. Para obter mais informações, consulte the section called "Acesse os recursos de machine learning". • A função do Lambda tem acesso somente leitura ao código implantado de outras funções do Lambda em execução com o mesmo ID do grupo. • As funções Lambda que geram processos secundários em uma sessão de processo diferente ou com um manipulador SIGHUP (signal hangup) substituído, como com o utilitário nohup, não são encerradas automaticamente quando a função principal do Lambda é encerrada. AWS IoT Greengras s

> **Note**
>
> A configuração padrão de conteinerização para o grupo do Greengrass não se aplica aos conectores.

A alteração da conteinerização para uma função do Lambda pode causar problemas quando você a implantar. Se você tiver atribuído recursos locais à sua função do Lambda que não esteja mais disponível com suas novas configurações de conteinerização, a implantação falhará.

- Quando você altera uma função do Lambda da execução em um contêiner do Greengrass para a execução sem conteinerização, os limites de memória para a função serão descartados. Você deve acessar o sistema de arquivos diretamente em vez de usar recursos locais anexados. Você deve remover todos os recursos anexados antes da implantação.

- 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 padrão de 16 MB. Você pode definir essas configurações para cada função do Lambda antes da implantação.

Para alterar as configurações de conteinerização para uma função do Lambda

1. No painel de navegação do AWS IoT console, em Gerenciar, expanda dispositivos Greengrass e escolha Grupos (V1).
2. Selecione o grupo que contém a função do Lambda, cujas configurações você deseja alterar.
3. Selecione a guia Funções do Lambda.
4. Na função do Lambda que você deseja alterar, selecione as reticências (…) e então selecione Editar configuração.
5. Altere as configurações de conteinerização. Se você configurar a função do Lambda para executar um contêiner do Greengrass, também será necessário definir o Limite de memória e o Acesso de leitura ao diretório /sys.
6. Selecione Salvar e depois Confirmar para salvar as alterações na sua função do Lambda.

As alterações entram em vigor quando o grupo é implantado.

Você também pode usar o CreateFunctionDefinitione CreateFunctionDefinitionVersionna Referência da AWS IoT Greengrass API. Se você estiver alterando a configuração de conteinerização, atualize os outros parâmetros também. Por exemplo, se você estiver executando a alteração de uma função do Lambda em um contêiner do Greengrass para executar sem conteinerização, limpe o parâmetro `MemorySize`.

Determine os modos de isolamento compatíveis com o dispositivo do Greengrass

Você pode usar o verificador de AWS IoT Greengrass dependências para determinar quais modos de isolamento (container/no contêiner Greengrass) são compatíveis com seu dispositivo Greengrass.

Para executar o AWS IoT Greengrass verificador de dependência

1. Baixe e execute o verificador AWS IoT Greengrass de dependências do GitHub repositório.

```
wget https://github.com/aws-samples/aws-greengrass-samples/raw/master/greengrass-
dependency-checker-GGCv1.11.x.zip
unzip greengrass-dependency-checker-GGCv1.11.x.zip
cd greengrass-dependency-checker-GGCv1.11.x
sudo modprobe configs
sudo ./check_ggc_dependencies | more
```

2. Se `more` for exibido, pressione a tecla Spacebar para exibir outra página de texto.

Para obter informações sobre o comando modprobe, execute man modprobe no terminal.

Definir a identidade de acesso padrão para as funções do Lambda em um grupo

Esse recurso está disponível para o AWS IoT Greengrass Core v1.8 e versões posteriores.

Para obter mais controle sobre o acesso a recursos de dispositivo, você pode configurar a identidade de acesso padrão usada para executar as funções do Lambda no grupo. Essa configuração determina as permissões padrão para suas funções do Lambda quando elas são executadas no dispositivo de núcleo. Para substituir a configuração para funções individuais no grupo, você pode usar a propriedade Run as (Executar como) da função. Para obter mais informações, consulte Executar como.

Essa configuração em nível de grupo também é usada para executar o software AWS IoT Greengrass Core subjacente. Ela consiste em funções do Lambda que gerenciam operações, como o roteamento de mensagens, a sincronização de sombra local e a detecção automática de endereço IP.

A identidade de acesso padrão pode ser configurada para ser executada como contas padrão do AWS IoT Greengrass sistema (ggc_user e ggc_group) ou usar as permissões de outro usuário ou grupo. Recomendamos que você configure seu hardware do Greengrass com os limites de recursos, permissões de arquivo e cotas de disco apropriados para qualquer usuário e grupo com permissões que são usadas para executar funções do Lambda definidas pelo usuário ou sistema.

Para modificar a identidade de acesso padrão para seu AWS IoT Greengrass grupo

1. No painel de navegação do AWS IoT console, em Gerenciar, expanda dispositivos Greengrass e escolha Grupos (V1).
2. Selecione o grupo cujas configurações você deseja alterar.

3. Selecione a guia Funções do Lambda e, na seção Ambiente de runtime da função do Lambda padrão, selecione Editar.
4. Na página Editar ambiente de execução da função Lambda padrão, em Usuário e grupo padrão do sistema, escolha Outra ID de usuário ID/group.

   Quando você escolhe essa opção, os campos ID do usuário do sistema (número) e ID do grupo do sistema (número) são exibidos.
5. Insira um ID de usuário, ID de grupo ou ambos. Se você deixar um campo em branco, a respectiva conta do sistema do Greengrass (ggc_user ou ggc_group) será usada.

   - Para ID do usuário do sistema (número), insira o ID do usuário que tem as permissões que você deseja usar por padrão para executar as funções do Lambda no grupo. Você pode usar o comando getent passwd em seu dispositivo AWS IoT Greengrass para pesquisar o ID do usuário.
   - Para ID do grupo do sistema (número), insira o ID do grupo que tem as permissões que você deseja usar por padrão para executar as funções do Lambda no grupo. Você pode usar o comando getent group em seu dispositivo AWS IoT Greengrass para pesquisar o ID do grupo.

   > **Important**
   >
   > Executar como usuário raiz aumenta os riscos para seus dados e dispositivo. Não execute como root (UID/GID=0), a menos que seu caso de negócios exija isso. Para obter mais informações, consulte the section called “Executar uma função do Lambda como raiz”.

As alterações entram em vigor quando o grupo é implantado.

Definir a conteinerização padrão para funções do Lambda em um grupo

Esse recurso está disponível para AWS IoT Greengrass Core v1.7 e versões posteriores.

A configuração de conteinerização para um grupo do Greengrass determina a conteinerização padrão para as funções do Lambda no grupo.

- No modo de contêiner Greengrass, as funções Lambda são executadas em um ambiente de execução isolado dentro do AWS IoT Greengrass contêiner por padrão.

- No modo Sem contêiner, as funções do Lambda são executadas como processos regulares do Linux por padrão.

É possível modificar as configurações do grupo para especificar a conteinerização padrão para funções do Lambda no grupo. Você poderá substituir essa configuração para uma ou mais funções do Lambda no grupo se quiser que as funções do Lambda sejam executadas com conteinerização diferente do padrão do grupo. Antes de alterar as configurações de conteinerização, consulte the section called “Considerações ao escolher a conteinerização de função do Lambda”.

> **Important**
>
> Se você deseja alterar a conteinerização padrão para o grupo, mas tem uma ou mais funções que usam uma conteinerização diferente, altere as configurações para as funções do Lambda antes de alterar a configuração do grupo. Se você alterar a configuração de conteinerização do grupo primeiro, os valores das configurações Memory limit (Limite de memória) e Read access to /sys directory (Acesso de leitura ao diretório /sys) serão descartadas.

Para modificar as configurações de conteinerização de seu AWS IoT Greengrass grupo

1. No painel de navegação do AWS IoT console, em Gerenciar, expanda dispositivos Greengrass e escolha Grupos (V1).
2. Selecione o grupo cujas configurações você deseja alterar.
3. Selecione a guia Funções do Lambda.
4. Em Ambiente de runtime da função do Lambda padrão, selecione Editar.
5. Na página Editar Ambiente de runtime da função do Lambda padrão, em Conteinerização da função do Lambda padrão, altere a configuração de conteinerização.
6. Selecione Salvar.

As alterações entram em vigor quando o grupo é implantado.

Fluxos de comunicação para funções do Lambda do Greengrass

As funções do Greengrass Lambda oferecem suporte a vários métodos de comunicação com outros membros do AWS IoT Greengrass grupo, serviços locais e serviços em nuvem (incluindo serviços). AWS

Comunicação usando mensagens MQTT

As funções do Lambda podem enviar e receber mensagens MQTT usando um padrão de publicação/assinatura que é controlado por assinaturas.

Esse fluxo de comunicação permite que as funções do Lambda troquem mensagens com as seguintes entidades:

- Dispositivos cliente no grupo.
- Conectores no grupo.
- Outras funções do Lambda no grupo.
- AWS IoT.
- Serviço Device Shadow local.

Uma assinatura define uma origem de mensagem, um destino de mensagem e um tópico (ou assunto) que é usado para rotear mensagens da origem para o destino. As mensagens publicadas em uma função do Lambda são passadas para o manipulador registrado da função. As assinaturas permitem mais segurança e oferecem interações previsíveis. Para obter mais informações, consulte the section called “Assinaturas gerenciadas no fluxo de trabalho de mensagens MQTT”.

> **Note**
>
> Quando o núcleo está offline, as funções do Lambda do Greengrass podem trocar mensagens com dispositivos cliente, conectores, outras funções e sombras locais, mas as mensagens para o AWS IoT serão colocadas em fila. Para obter mais informações, consulte the section called “Fila de mensagens MQTT”.

Outros fluxos de comunicação

- Para interagir com modelos de machine learning e recursos locais de dispositivo e volume em um dispositivo de núcleo, as funções do Lambda do Greengrass usam interfaces do sistema operacional específicas da plataforma. Por exemplo, você pode usar o método `open` no módulo os em funções do Python. Para permitir que uma função acesse um recurso, a função deve ser afiliada ao recurso e receber a permissão `read-only` ou `read-write`. Para obter mais informações, incluindo a disponibilidade da versão AWS IoT Greengrass principal, consulte Acesso aos recursos locais the section called “Acessando os recursos de machine learning do código de função do Lambda” e.

  > **Note**
  >
  > Se executar a função do Lambda sem conteinerização, você não poderá usar recursos locais de dispositivo e volume anexados e deverá acessar esses recursos diretamente.

- As funções do Lambda podem usar o `Lambda` cliente no SDK AWS IoT Greengrass principal para invocar outras funções do Lambda no grupo Greengrass.
- As funções Lambda podem usar o AWS SDK para se comunicar com os serviços. AWS Para obter mais informações, consulte SDKs da AWS.
- As funções do Lambda podem usar interfaces de terceiros para se comunicar com serviços em nuvem externos, semelhantes a funções do Lambda baseadas em nuvem.

> **Note**
>
> As funções do Greengrass Lambda não conseguem se comunicar com AWS nem com outros serviços de nuvem quando o núcleo está off-line.

Recuperar o tópico MQTT de entrada (ou assunto)

AWS IoT Greengrass usa assinaturas para controlar a troca de mensagens MQTT entre dispositivos clientes, funções Lambda e conectores em um grupo e com AWS IoT o serviço paralelo local. As assinaturas definem a origem de uma mensagem, o destino de uma mensagem e um tópico MQTT usado para rotear mensagens. Quando o destino é uma função do Lambda, o manipulador da função

é invocado quando a origem publica uma mensagem. Para obter mais informações, consulte the section called “Comunicação usando mensagens MQTT”.

O exemplo a seguir mostra como uma função do Lambda pode obter o tópico de entrada do `context` que é passado para o manipulador. Ele faz isso acessando a `subject` chave da hierarquia de contexto (`context.client_context.custom['subject']`). O exemplo também analisa a mensagem de entrada do JSON e, em seguida, publica o tópico e a mensagem analisados.

> **Note**
>
> Na AWS IoT Greengrass API, o tópico de uma assinatura é representado pela `subject` propriedade.

```
import greengrasssdk
import logging

client = greengrasssdk.client('iot-data')

OUTPUT_TOPIC = 'test/topic_results'

def get_input_topic(context):
    try:
        topic = context.client_context.custom['subject']
    except Exception as e:
        logging.error('Topic could not be parsed. ' + repr(e))
    return topic

def get_input_message(event):
    try:
        message = event['test-key']
    except Exception as e:
        logging.error('Message could not be parsed. ' + repr(e))
    return message

def function_handler(event, context):
    try:
        input_topic = get_input_topic(context)
        input_message = get_input_message(event)
        response = 'Invoked on topic "%s" with message "%s"' % (input_topic,
 input_message)
        logging.info(response)
```

```
    except Exception as e:
        logging.error(e)

    client.publish(topic=OUTPUT_TOPIC, payload=response)

    return
```

Para testar a função, adicione-a ao seu grupo usando as definições de configuração padrão. Em seguida, adicione as assinaturas a seguir e implante o grupo. Para instruções, consulte the section called “Módulo 3 (parte 1): funções Lambda em AWS IoT Greengrass”.

Destino de tópicos

test/input_message

Destino de tópicos

test/topic_results

Depois que a implantação for concluída, invoque a função.

1. No AWS IoT console, abra a página do cliente de teste do MQTT.
2. Inscreva-se no tópico `test/topic_results` selecionando a guia Inscrever-se em um tópico.
3. Publique uma mensagem no tópico `test/input_message` selecionando a guia Publicar em um tópico. Para este exemplo, você deve incluir a propriedade `test-key` na mensagem JSON.

```
{
  "test-key": "Some string value"
}
```

Se for bem-sucedido, a função publicará o tópico de entrada e a string de mensagem para o tópico `test/topic_results`.

Configuração do ciclo de vida das funções do Lambda do Greengrass

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 as variáveis e a lógica de pré-processamento que estão fora do manipulador da função são retidas.

AWS IoT Greengrass suporta ciclos de vida sob demanda (padrão) ou de longa duração:

- On-demandas funções começam quando são invocadas e param quando não há mais tarefas para serem executadas. Uma invocação da função cria um contêiner à parte (ou sandbox) para processar invocações, a menos que um contêiner existente esteja disponível para reutilização. Os dados enviados para a função podem ser obtidos por qualquer um dos contêineres.

  Várias invocações de uma função sob demanda podem ser executadas em paralelo.

  Variáveis e lógica de pré-processamento que estão definidas fora do manipulador de funções não são mantidas quando novos contêineres são criados.

- Long-livedas funções (ou fixadas) são iniciadas automaticamente quando o AWS IoT Greengrass núcleo é iniciado e executado em um único contêiner. Todos os dados enviados para a função são obtidos pelo mesmo contêiner.

  Várias invocações serão enfileiradas até as invocações anteriores terem sido executadas.

  Variáveis e lógica de pré-processamento que estão definidas fora do manipulador de funções são mantidas para cada invocação do manipulador.

  Long-lived As funções Lambda são úteis quando você precisa começar a trabalhar sem nenhuma entrada inicial. Por exemplo, uma função de longa duração pode carregar e iniciar o processamento de um modelo ML para estar pronto quando a função começa a receber dados do dispositivo.

> **Note**
>
> Lembre-se de que as funções de longa duração têm tempos limite associados a invocações do manipulador. Se quiser executar indefinidamente executando código, você deverá iniciá-lo fora do manipulador. Certifique-se de que não haja código de bloqueio fora do manipulador que possa evitar que a função conclua a inicialização.
> Essas funções serão executadas, a menos que o núcleo seja interrompido (por exemplo, durante a implantação de um grupo ou a reinicialização de um dispositivo) ou a função inserirá um estado de erro (como um tempo limite do manipulador, exceção não capturada ou limite excedido de memória).

Para obter mais informações sobre a reutilização de contêineres, consulte Entendendo a reutilização de contêineres AWS Lambda no blog de AWS computação.

Executáveis do Lambda

Esse recurso está disponível para AWS IoT Greengrass Core v1.6 e versões posteriores.

Um executável do Lambda é um tipo de função do Lambda do Greengrass que você pode usar para executar o código binário no ambiente básico. Ele permite executar a funcionalidade específica do dispositivo de maneira nativa e se beneficiar do espaço menor do código compilado. Os executáveis do Lambda podem ser invocados por eventos, invocar outras funções e acessar recursos locais.

Os executáveis do Lambda só oferecem suporte ao tipo de codificação binária (e não JSON), mas você pode gerenciá-los no grupo do Greengrass e implantá-los como outras funções do Lambda do Greengrass. No entanto, o processo de criação de executáveis do Lambda é diferente da criação de funções em Python, Java e Lambda: Node.js

- Você não pode usar o AWS Lambda console para criar (ou gerenciar) um executável Lambda. Você pode criar um executável Lambda somente usando a API. AWS Lambda
- Você carrega o código da função AWS Lambda como um executável compilado que inclui o SDK AWS IoT Greengrass principal para C.
- Você especifica o nome do executável como o manipulador de funções.

Os executáveis do Lambda devem implementar determinadas chamadas e padrões de programação no código de função. Por exemplo, o método `main` deve:

- Chame `gg_global_init` para inicializar variáveis globais internas do Greengrass. Essa função deve ser chamada antes de criar qualquer thread e antes de chamar qualquer outra função AWS IoT Greengrass do Core SDK.
- Chame `gg_runtime_start` para registrar o manipulador de funções com o runtime do Lambda do Greengrass. Essa função deve ser chamada durante a inicialização. Chamar essa função faz o thread atual ser usado pelo runtime. O parâmetro `GG_RT_OPT_ASYNC` opcional orienta essa função a não bloquear, mas, em vez disso, criar um novo thread para o runtime. Essa função usa um manipulador `SIGTERM`.

O trecho a seguir é o `main` método do exemplo de código simple_handler.c em. GitHub

```
int main() {
    gg_error err = GGE_SUCCESS;

    err = gg_global_init(0);
    if(err) {
        gg_log(GG_LOG_ERROR, "gg_global_init failed %d", err);
        goto cleanup;
    }

    gg_runtime_start(handler, 0);

cleanup:
    return -1;
}
```

Para obter mais informações sobre requisitos, restrições e outros detalhes de implementação, consulte SDK do AWS IoT Greengrass Core para C.

Crie um executável do Lambda

Depois de compilar seu código junto com o SDK, use a AWS Lambda API para criar uma função Lambda e fazer o upload do executável compilado.

> **Note**
>
> A função deve ser compilado com um compilador C89 compatível.

O exemplo a seguir usa o comando da CLI create-function para criar um executável do Lambda. O comando especifica:

- O nome do executável do manipulador. Ele deve ser o nome exato do executável compilado.
- O caminho do arquivo `.zip` que contém o executável compilado.
- `arn:aws:greengrass:::runtime/function/executable` do runtime. Este é o runtime de todos os executáveis do Lambda.

> **Note**
>
> Para `role`, você pode especificar o ARN de qualquer função do Lambda de execução. O AWS IoT Greengrass não usa essa função, mas o parâmetro é obrigatório para a criação da função. Para obter informações sobre funções do Lambda de execução, consulte Modelo de permissões do AWS Lambda no Guia do desenvolvedor do AWS Lambda .

```
aws lambda create-function \
--region aws-region \
--function-name function-name \
--handler executable-name \
--role role-arn \
--zip-file fileb://file-name.zip \
--runtime arn:aws:greengrass:::runtime/function/executable
```

Em seguida, use a AWS Lambda API para publicar uma versão e criar um alias.

- Use publish-version para publicar uma versão da função.

```
aws lambda publish-version \
--function-name function-name \
--region aws-region
```

- Use create-alias para criar um alias que aponte para a versão recém-publicada. Recomendamos fazer referência a funções do Lambda por alias quando você as adiciona a um grupo do Greengrass.

```
aws lambda create-alias \
--function-name function-name \
--name alias-name \
--function-version version-number \
--region aws-region
```

> **Note**
>
> O AWS Lambda console não exibe executáveis Lambda. Para atualizar o código da função, você deve usar a AWS Lambda API.

Em seguida, adicione o executável do Lambda a um grupo do Greengrass, configure-o para aceitar dados de entrada binários nas configurações específicas do grupo e implantar o grupo. Você pode fazer isso no AWS IoT Greengrass console ou usando a AWS IoT Greengrass API.

Em execução AWS IoT Greengrass em um contêiner Docker

AWS IoT Greengrass pode ser configurado para ser executado em um contêiner Docker.

Você pode baixar um Dockerfile por meio da Amazon CloudFront que tenha o software AWS IoT Greengrass principal e as dependências instalados. Para modificar a imagem do Docker para executar em diferentes arquiteturas de plataforma ou reduzir o tamanho da imagem do Docker, consulte o arquivo `README` no download do pacote do Docker.

Para ajudar você a começar a experimentar AWS IoT Greengrass, AWS também fornece imagens pré-criadas do Docker que têm o software AWS IoT Greengrass principal e as dependências instalados. Faça download de uma imagem do Docker Hub ou do Amazon Elastic Container Registry (Amazon ECR). Essas imagens pré-compiladas usam as imagens base do Amazon Linux 2 (x86_64) e Alpine Linux (x86_64, Armv7l ou AArch64).

> **Important**
>
> Em 30 de junho de 2022, AWS IoT Greengrass encerrou a manutenção das imagens Docker AWS IoT Greengrass do software Core v1.x que são publicadas no Amazon Elastic Container Registry (Amazon ECR) e no Docker Hub. Você pode continuar baixando essas imagens do Docker do Amazon ECR e do Docker Hub até 30 de junho de 2023, ou seja, um ano após o término da manutenção. No entanto, as imagens do Docker do software AWS IoT Greengrass Core v1.x não recebem mais patches de segurança ou correções de erros após o término da manutenção em 30 de junho de 2022. Se você executa uma carga de trabalho de produção que depende dessas imagens do Docker, recomendamos que você crie suas próprias imagens do Docker usando os Dockerfiles fornecidos. AWS IoT Greengrass Para obter mais informações, consulte AWS IoT Greengrass Software Docker.

Este tópico descreve como baixar a imagem do AWS IoT Greengrass Docker do Amazon ECR e executá-la em uma plataforma Windows, macOS ou Linux (x86_64). O tópico inclui as etapas a seguir:

1. Obtenha a imagem do AWS IoT Greengrass contêiner do Amazon ECR
2. Criar e configurar o núcleo e o grupo do Greengrass
3. Execute AWS IoT Greengrass localmente
4. Configurar a conteinerização "Sem contêiner" para o grupo
5. Implantar funções do Lambda para o contêiner do Docker
6. (Opcional) Implantar dispositivos cliente que interagem com o Greengrass no contêiner do Docker

Os seguintes recursos não são compatíveis quando você executa AWS IoT Greengrass em um contêiner Docker:

- Conectores executados no modo de contêiner do Greengrass. Para executar um conector em um contêiner do Docker, o conector deve ser executado no modo Sem contêiner. Para localizar conectores compatíveis com o modo Sem contêiner consulte the section called “Conectores do Greengrass fornecidos pela AWS”. Alguns desses conectores têm um parâmetro de modo de isolamento que você deve definir como Sem contêiner.
- Recursos de volume e dispositivo locais. Suas funções do Lambda definidas pelo usuário executadas no contêiner do Docker devem acessar dispositivos e volumes diretamente no núcleo.

Esses recursos não são suportados quando o ambiente de execução do Lambda para o grupo Greengrass está definido como Sem contêiner, o que é necessário para ser executado AWS IoT Greengrass em um contêiner Docker.

Pré-requisitos

Antes de começar este tutorial, você deve fazer o seguinte.

- Você deve instalar os seguintes softwares e versões em seu computador host com base na versão AWS Command Line Interface (AWS CLI) que você escolher.

  AWS CLI version 2

  - Docker, versão 18.09 ou posterior. Versões anteriores também podem funcionar, mas recomendamos a versão 18.09 ou posterior.
  - AWS CLI versão 2.0.0 ou posterior.
    - Para instalar a AWS CLI versão 2, consulte Instalando a AWS CLI versão 2.
    - Para configurar o AWS CLI, consulte Configurando o. AWS CLI

    > **Note**
    >
    > Para atualizar para uma AWS CLI versão posterior 2 em um computador Windows, você deve repetir o processo de instalação do MSI.

  AWS CLI version 1

  - Docker, versão 18.09 ou posterior. Versões anteriores também podem funcionar, mas recomendamos a versão 18.09 ou posterior.
  - Python, versão 3.6 ou posterior.
  - pip versão 18.1 ou posterior.
  - AWS CLI versão 1.17.10 ou posterior
    - Para instalar a AWS CLI versão 1, consulte Instalando a AWS CLI versão 1.
    - Para configurar o AWS CLI, consulte Configurando o. AWS CLI
    - Para atualizar para a versão mais recente da AWS CLI versão 1, execute o comando a seguir.

      ```
      pip install awscli --upgrade --user
      ```

> **Note**
>
> Se você usa a instalação MSI da AWS CLI versão 1 no Windows, esteja ciente do seguinte:
>
> - Se a instalação da AWS CLI versão 1 falhar ao instalar o botocore, tente usar a instalação do Python e do pip.
> - Para atualizar para uma AWS CLI versão posterior 1, você deve repetir o processo de instalação do MSI.

- Para acessar os recursos do Amazon Elastic Container Registry (Amazon ECR), você deve conceder a seguinte permissão.
  - O Amazon ECR exige que os usuários concedam a `ecr:GetAuthorizationToken` permissão por meio de uma política AWS Identity and Access Management (IAM) antes de poderem se autenticar em um registro e enviar ou extrair imagens de um repositório do Amazon ECR. Para obter mais informações, consulte Exemplos de políticas do repositório do Amazon ECR e Acessando um repositório do Amazon ECR no Guia do usuário do Amazon Elastic Container Registry.

Etapa 1: Obtenha o AWS IoT Greengrass imagem de contêiner do Amazon ECR

AWS fornece imagens do Docker que têm o software AWS IoT Greengrass Core instalado.

> **Warning**
>
> A partir da versão 1.11.6 do software AWS IoT Greengrass Core, as imagens do Greengrass Docker não incluem mais o Python 2.7, porque o Python 2.7 chegou ao fim da vida útil em 2020 e não recebe mais atualizações de segurança. Se você optar por atualizar essas imagens do Docker, recomendamos verificar se seus aplicativos funcionam com as novas imagens do Docker antes de implantar as atualizações nos dispositivos de produção.
> Se você precisar do Python 2.7 para seu aplicativo que usa uma imagem do Docker do Greengrass, poderá modificar o Dockerfile do Greengrass para incluir o Python 2.7 em seu aplicativo.

Para ver as etapas que mostram como obter a imagem `latest` do Amazon ECR, selecione o seu sistema operacional:

Extrair a imagem de contêiner (Linux)

Execute os seguintes comandos no terminal do computador.

1. Faça login no AWS IoT Greengrass registro no Amazon ECR.

   ```
   aws ecr get-login-password --region  us-west-2 | docker login --username AWS --
   password-stdin https://216483018798.dkr.ecr.us-west-2.amazonaws.com
   ```

   Se for bem-sucedido, a saída imprimirá `Login Succeeded`.

2. Recupere a imagem do AWS IoT Greengrass contêiner.

   ```
   docker pull 216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
   ```

   > **Note**
   >
   > A `latest` imagem contém a versão estável mais recente do software AWS IoT Greengrass Core instalada em uma imagem base do Amazon Linux 2. Você também pode extrair outras imagens do repositório. Para encontrar todas as imagens disponíveis, verifique a página Tags no Hub do Docker ou use o comando aws ecr list-images. Por exemplo:
   >
   > ```
   > aws ecr list-images --region us-west-2 --registry-id 216483018798 --
   > repository-name aws-iot-greengrass
   > ```

3. Habilite as proteções symlink e hardlink. Se você estiver experimentando a execução AWS IoT Greengrass em um contêiner, poderá habilitar as configurações somente para a inicialização atual.

   > **Note**
   >
   > Talvez você precise usar sudo para executar esses comandos.

   - Para habilitar as configurações só para a inicialização atual:

```
echo 1 > /proc/sys/fs/protected_hardlinks
echo 1 > /proc/sys/fs/protected_symlinks
```

- Para habilitar as configurações de modo que sejam mantidas nas reinicializações:

```
echo '# AWS IoT Greengrass' >> /etc/sysctl.conf
echo 'fs.protected_hardlinks = 1' >> /etc/sysctl.conf
echo 'fs.protected_symlinks = 1' >> /etc/sysctl.conf

sysctl -p
```

4. Ative o encaminhamento de rede IPv4, necessário para que a implantação AWS IoT Greengrass na nuvem e as comunicações MQTT funcionem no Linux. No arquivo `/etc/sysctl.conf`, defina `net.ipv4.ip_forward` como 1 e, em seguida, recarregue `sysctls`.

```
sudo nano /etc/sysctl.conf
# set this net.ipv4.ip_forward = 1
sudo sysctl -p
```

> **Note**
>
> Você pode usar o editor de sua selecione no lugar do nano.

Extrair a imagem de contêiner (macOS)

Execute os seguintes comandos no terminal do computador.

1. Faça login no AWS IoT Greengrass registro no Amazon ECR.

```
aws ecr get-login-password --region  us-west-2 | docker login --username AWS --
password-stdin https://216483018798.dkr.ecr.us-west-2.amazonaws.com
```

   Se for bem-sucedido, a saída imprimirá `Login Succeeded`.

2. Recupere a imagem do AWS IoT Greengrass contêiner.

```
docker pull 216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

> **Note**
>
> A `latest` imagem contém a versão estável mais recente do software AWS IoT Greengrass Core instalada em uma imagem base do Amazon Linux 2. Você também pode extrair outras imagens do repositório. Para encontrar todas as imagens disponíveis, verifique a página Tags no Hub do Docker ou use o comando aws ecr list-images. Por exemplo:
>
> ```
> aws ecr list-images --region us-west-2 --registry-id 216483018798 --
> repository-name aws-iot-greengrass
> ```

Extrair a imagem de contêiner (Windows)

Execute os seguintes comandos em um prompt de comando. Antes de você poder usar comandos do Docker no Windows, o Docker Desktop deve estar em execução.

1. Faça login no AWS IoT Greengrass registro no Amazon ECR.

   ```
   aws ecr get-login-password --region  us-west-2 | docker login --username AWS --
   password-stdin https://216483018798.dkr.ecr.us-west-2.amazonaws.com
   ```

   Se for bem-sucedido, a saída imprimirá `Login Succeeded`.

2. Recupere a imagem do AWS IoT Greengrass contêiner.

   ```
   docker pull 216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
   ```

   > **Note**
   >
   > A `latest` imagem contém a versão estável mais recente do software AWS IoT Greengrass Core instalada em uma imagem base do Amazon Linux 2. Você também pode extrair outras imagens do repositório. Para encontrar todas as imagens disponíveis, verifique a página Tags no Hub do Docker ou use o comando aws ecr list-images. Por exemplo:

```
aws ecr list-images --region us-west-2 --registry-id 216483018798 --
repository-name aws-iot-greengrass
```

Etapa 2: Criar e configurar o núcleo e o grupo do Greengrass

A imagem do Docker tem o software AWS IoT Greengrass Core instalado, mas você deve criar um grupo e um núcleo do Greengrass. Isso inclui o download de certificados e do arquivo de configuração do núcleo.

- Siga as etapas em the section called "Módulo 2: Instalando o AWS IoT Greengrass Software principal". Ignore as etapas em que você baixa e executa o software AWS IoT Greengrass Core. O software do núcleo e suas dependências de runtime já estão configurados na imagem do Docker.

Etapa 3: executar AWS IoT Greengrass Localmente

Depois que seu grupo estiver configurado, você estará pronto para configurar e iniciar o núcleo. Para ver as etapas que mostram como fazer isso, selecione o seu sistema operacional:

Executar o Greengrass localmente (Linux)

Execute os seguintes comandos no terminal do computador.

1. Crie uma pasta para os recursos de segurança do dispositivo e mova o certificado e as chaves para essa pasta. Execute os seguintes comandos. *path-to-security-files*Substitua pelo caminho para os recursos de segurança e *certificateId* substitua pela ID do certificado nos nomes dos arquivos.

   ```
   mkdir /tmp/certs
   mv path-to-security-files/certificateId-certificate.pem.crt /tmp/certs
   mv path-to-security-files/certificateId-public.pem.key /tmp/certs
   mv path-to-security-files/certificateId-private.pem.key /tmp/certs
   mv path-to-security-files/AmazonRootCA1.pem /tmp/certs
   ```

2. Crie uma pasta para a configuração do dispositivo e mova o arquivo de configuração AWS IoT Greengrass principal para essa pasta. Execute os seguintes comandos. *path-to-config-file*Substitua pelo caminho para o arquivo de configuração.

```
mkdir /tmp/config
mv path-to-config-file/config.json /tmp/config
```

3. AWS IoT Greengrass Inicie e monte os certificados e o arquivo de configuração no contêiner do Docker.

   Substitua `/tmp` pelo caminho onde você descompactou seus certificados e o arquivo de configuração.

   ```
   docker run --rm --init -it --name aws-iot-greengrass \
   --entrypoint /greengrass-entrypoint.sh \
   -v /tmp/certs:/greengrass/certs \
   -v /tmp/config:/greengrass/config \
   -p 8883:8883 \
   216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
   ```

   A saída deve ser semelhante a este exemplo:

   ```
   Setting up greengrass daemon
   Validating hardlink/softlink protection
   Waiting for up to 30s for Daemon to start

   Greengrass successfully started with PID: 10
   ```

Executar o Greengrass localmente (macOS)

Execute os seguintes comandos no terminal do computador.

1. Crie uma pasta para os recursos de segurança do dispositivo e mova o certificado e as chaves para essa pasta. Execute os seguintes comandos. *path-to-security-files*Substitua pelo caminho para os recursos de segurança e *certificateId* substitua pela ID do certificado nos nomes dos arquivos.

   ```
   mkdir /tmp/certs
   mv path-to-security-files/certificateId-certificate.pem.crt /tmp/certs
   mv path-to-security-files/certificateId-public.pem.key /tmp/certs
   mv path-to-security-files/certificateId-private.pem.key /tmp/certs
   mv path-to-security-files/AmazonRootCA1.pem /tmp/certs
   ```

2. Crie uma pasta para a configuração do dispositivo e mova o arquivo de configuração AWS IoT Greengrass principal para essa pasta. Execute os seguintes comandos. *path-to-config-file*Substitua pelo caminho para o arquivo de configuração.

```
mkdir /tmp/config
mv path-to-config-file/config.json /tmp/config
```

3. AWS IoT Greengrass Inicie e monte os certificados e o arquivo de configuração no contêiner do Docker.

   Substitua `/tmp` pelo caminho onde você descompactou seus certificados e o arquivo de configuração.

```
docker run --rm --init -it --name aws-iot-greengrass \
--entrypoint /greengrass-entrypoint.sh \
-v /tmp/certs:/greengrass/certs \
-v /tmp/config:/greengrass/config \
-p 8883:8883 \
216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

   A saída deve ser semelhante a este exemplo:

```
Setting up greengrass daemon
Validating hardlink/softlink protection
Waiting for up to 30s for Daemon to start

Greengrass successfully started with PID: 10
```

Executar o Greengrass localmente (Windows)

1. Crie uma pasta para os recursos de segurança do dispositivo e mova o certificado e as chaves para essa pasta. Execute os seguintes comandos em um prompt de comando. *path-to-security-files*Substitua pelo caminho para os recursos de segurança e *certificateId* substitua pela ID do certificado nos nomes dos arquivos.

```
mkdir C:\Users\%USERNAME%\Downloads\certs
move path-to-security-files\certificateId-certificate.pem.crt C:\Users\%USERNAME%
\Downloads\certs
```

```
move path-to-security-files\certificateId-public.pem.key C:\Users\%USERNAME%
\Downloads\certs
move path-to-security-files\certificateId-private.pem.key C:\Users\%USERNAME%
\Downloads\certs
move path-to-security-files\AmazonRootCA1.pem C:\Users\%USERNAME%\Downloads\certs
```

2. Crie uma pasta para a configuração do dispositivo e mova o arquivo de configuração AWS IoT Greengrass principal para essa pasta. Execute os seguintes comandos em um prompt de comando. *path-to-config-file*Substitua pelo caminho para o arquivo de configuração.

```
mkdir C:\Users\%USERNAME%\Downloads\config
move path-to-config-file\config.json C:\Users\%USERNAME%\Downloads\config
```

3. AWS IoT Greengrass Inicie e monte os certificados e o arquivo de configuração no contêiner do Docker. Execute os seguintes comandos em seu prompt de comando.

```
docker run --rm --init -it --name aws-iot-greengrass --entrypoint /greengrass-
entrypoint.sh -v c:/Users/%USERNAME%/Downloads/certs:/greengrass/certs
 -v c:/Users/%USERNAME%/Downloads/config:/greengrass/config -p 8883:8883
 216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

Quando o Docker solicitar que você compartilhe sua unidade `C:\` com o daemon do Docker, permita que ele salve o diretório `C:\` como bind mount dentro do contêiner do Docker. Para obter mais informações, consulte Unidades compartilhadas na documentação do Docker.

A saída deve ser semelhante a este exemplo:

```
Setting up greengrass daemon
Validating hardlink/softlink protection
Waiting for up to 30s for Daemon to start

Greengrass successfully started with PID: 10
```

> **Note**
>
> Se o contêiner não abrir o shell e sair imediatamente, você poderá depurar o problema fazendo uma montagem bind dos logs de runtime do ao iniciar a imagem. Para obter mais informações, consulte the section called “Como manter os logs de runtime do Greengrass fora do contêiner do Docker”.

Etapa 4: Configurar a conteinerização "Sem contêiner" para o grupo do Greengrass

Quando você executa AWS IoT Greengrass em um contêiner Docker, todas as funções do Lambda devem ser executadas sem conteinerização. Nesta etapa, você definirá a conteinerização padrão para o grupo como No container (Nenhum contêiner). Você deve fazer isso antes de implantar o grupo pela primeira vez.

1. No painel de navegação do AWS IoT console, em Gerenciar, expanda dispositivos Greengrass e escolha Grupos (V1).
2. Selecione o grupo cujas configurações você deseja alterar.
3. Selecione a guia Funções do Lambda.
4. Em Ambiente de runtime da função do Lambda padrão, selecione Editar.
5. Em Editar o ambiente padrão de runtime da função do Lambda, em Conteinerização padrão da função do Lambda, altere as configurações de conteinerização.
6. Selecione Salvar.

As alterações entram em vigor quando o grupo é implantado.

Para obter mais informações, consulte the section called “Definir a conteinerização padrão para funções do Lambda em um grupo”.

> **Note**
>
> Por padrão, as funções do Lambda usam a configuração de conteinerização do grupo. Se você substituir a configuração Nenhum contêiner de qualquer função do Lambda quando o AWS IoT Greengrass estiver em execução em um contêiner do Docker, a implantação falhará.

Etapa 5: Implantar funções Lambda no AWS IoT Greengrass Contêiner do docker

Você pode implantar funções do Lam de longa duração para o contêiner do Docker do Greengrass.

- Siga as etapas em the section called “Módulo 3 (parte 1): funções Lambda em AWS IoT Greengrass” para implantar uma função do Lambda de longa duração "Hello World" para o contêiner.

Etapa 6: (opcional) implantar dispositivos cliente que interagem com o Greengrass em execução no contêiner do Docker

Você também pode implantar dispositivos cliente que interagem AWS IoT Greengrass quando estão sendo executados em um contêiner Docker.

- Siga as etapas em the section called “Módulo 4: Interagindo com dispositivos clientes em um AWS IoT Greengrass grupo” para implantar dispositivos cliente que se conectam ao núcleo e enviam mensagens MQTT.

Parando o AWS IoT Greengrass Contêiner do docker

Para parar o contêiner do AWS IoT Greengrass Docker, pressione Ctrl+C no terminal ou no prompt de comando. Essa ação envia `SIGTERM` para o processo de daemon do Greengrass a fim de descartar esse processo e todos os processos do Lambda que foram iniciados pelo processo de daemon. O contêiner do Docker é inicializado com o processo `/dev/init` como PID 1, o que ajuda a eliminar quaisquer processos zumbis restantes. Para obter mais informações, consulte a Referência de execução do Docker.

Solução de problemas AWS IoT Greengrass em um contêiner Docker

Use as informações a seguir para ajudar a solucionar problemas com a execução AWS IoT Greengrass em um contêiner do Docker.

Erro: não é possível realizar um login interativo em um dispositivo não TTY.

Solução: este erro pode ocorrer ao executar o comando `aws ecr get-login-password`. Verifique se você instalou a versão 2 ou a versão 1 mais recente do AWS CLI . Recomendamos que você use a AWS CLI versão 2. Para obter mais informações, consulte Instalar a AWS CLI no Guia do usuário da AWS Command Line Interface .

Erro: Opções desconhecidas: -no-include-email

Solução: este erro pode ocorrer ao executar o comando `aws ecr get-login`. Verifique se você tem a AWS CLI versão mais recente instalada (por exemplo, execute:`pip install awscli --upgrade --user`). Se você estiver usando o Windows e tiver instalado a CLI usando o instalador MSI, repita o processo de instalação. Para obter informações, consulte Instalar a AWS Command Line Interface no Microsoft Windows no Guia do usuário do AWS Command Line Interface .

Aviso: IPv4 está desabilitado. As redes não funcionarão.

Solução: você pode receber esse aviso ou uma mensagem semelhante ao executar AWS IoT Greengrass em um computador Linux. Ative o encaminhamento de rede IPv4 conforme descrito nesta etapa. AWS IoT Greengrass a implantação na nuvem e as comunicações MQTT não funcionam quando o encaminhamento IPv4 não está habilitado. Para obter mais informações, consulte Configurar parâmetros de kernel com namespace (sysctls) em runtime na documentação do Docker.

Erro: Um firewall está bloqueando o compartilhamento de arquivos entre janelas e os contêineres.

Solução: esse erro ou uma mensagem `Firewall Detected` pode ser recebida 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: não arn:aws:iam::<account-id>:user/<user-name> está autorizado a realizar: ecr: GetAuthorizationToken on resource: *

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.

Para obter ajuda geral para AWS IoT Greengrass solução de problemas, consulteSolução de problemas.

Depuração AWS IoT Greengrass em um contêiner 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.

Como manter os logs de runtime do Greengrass fora do contêiner do Docker

Você pode executar o contêiner do AWS IoT Greengrass Docker após a montagem do diretório. `/greengrass/ggc/var/log` Os logs serão mantidos mesmo depois que o contêiner for encerrado ou removido.

No Linux ou macOS

Pare qualquer contêiner do Docker do Greengrass em execução no host e, em seguida, execute o comando a seguir em um terminal. Isso faz a montagem bind do diretório `log` do Greengrass e inicia a imagem do Docker.

Substitua `/tmp` pelo caminho onde você descompactou seus certificados e o arquivo de configuração.

```
docker run --rm --init -it --name aws-iot-greengrass \
      --entrypoint /greengrass-entrypoint.sh \
      -v /tmp/certs:/greengrass/certs \
      -v /tmp/config:/greengrass/config \
      -v /tmp/log:/greengrass/ggc/var/log \
      -p 8883:8883 \
      216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

Em seguida, você poderá verificar seus logs em `/tmp/log` no host para ver o que aconteceu enquanto o Greengrass estava em execução dentro do contêiner do Docker.

No Windows

Pare qualquer contêiner do Docker do Greengrass em execução no host e, em seguida, execute o comando a seguir em um prompt de comando. Isso faz a montagem bind do diretório `log` do Greengrass e inicia a imagem do Docker.

```
cd C:\Users\%USERNAME%\Downloads
mkdir log
docker run --rm --init -it --name aws-iot-greengrass --entrypoint /greengrass-
entrypoint.sh -v c:/Users/%USERNAME%/Downloads/certs:/greengrass/certs -v c:/
Users/%USERNAME%/Downloads/config:/greengrass/config -v c:/Users/%USERNAME%/
```

```
Downloads/log:/greengrass/ggc/var/log -p 8883:8883 216483018798.dkr.ecr.us-
west-2.amazonaws.com/aws-iot-greengrass:latest
```

Em seguida, você poderá verificar seus logs em `C:/Users/%USERNAME%/Downloads/log` no host para ver o que aconteceu enquanto o Greengrass estava em execução dentro do contêiner do Docker.

Como anexar um shell interativo ao contêiner do Docker

Você pode anexar um shell interativo a um contêiner do AWS IoT Greengrass Docker em execução. Isso pode ajudar a investigar o estado do contêiner do Docker do Greengrass.

No Linux ou macOS

Enquanto o contêiner do Docker do Greengrass estiver em execução, execute o comando a seguir em um terminal à parte.

```
docker exec -it $(docker ps -a -q -f "name=aws-iot-greengrass") /bin/bash
```

No Windows

Enquanto o contêiner do Docker do Greengrass estiver em execução, execute os comandos a seguir em um prompt de comando à parte.

```
docker ps -a -q -f "name=aws-iot-greengrass"
```

*gg-container-id*Substitua pelo `container_id` resultado do comando anterior.

```
docker exec -it gg-container-id /bin/bash
```

Acesso aos recursos locais com funções e conectores do Lambda

Esse recurso está disponível para AWS IoT Greengrass Core v1.3 e versões posteriores.

Com AWS IoT Greengrass, você pode criar AWS Lambda funções, configurar conectores na nuvem e implantá-los nos dispositivos principais para execução local. Nos núcleos do Greengrass que executam Linux, essas funções do Lambda localmente implantadas e conectores podem acessar recursos locais fisicamente presentes no dispositivo de núcleo do Greengrass. Por exemplo, para se comunicar com dispositivos que são conectados por meio do Modbus ou CANbus, você pode permitir que a função do Lambda acesse a porta serial no dispositivo de núcleo. Para configurar o acesso seguro a recursos locais, você deve garantir a segurança do seu hardware físico e do SO do seu dispositivo de núcleo do Greengrass.

Para começar a acessar recursos locais, consulte os seguintes tutoriais:

- Como configurar o acesso a recursos locais usando o AWS Interface da linha de comando
- Como configurar o acesso a recursos locais usando o Console de gerenciamento da AWS

Tipos de recursos compatíveis

Você pode acessar dois tipos de recursos locais: recursos de volume e recursos do dispositivo.

Recursos do volume

Arquivos ou diretórios no sistema de arquivos raiz (exceto em `/sys`, `/dev` ou `/var`). Isso inclui:

- Pastas ou arquivos usados para ler ou gravar informações em funções do Lambda do Greengrass (por exemplo, `/usr/lib/python2.x/site-packages/local`).
- As pastas ou os arquivos no sistema de arquivos /proc do host (por exemplo, `/proc/net` ou `/proc/stat`). Compatível com a versão 1.6 ou posterior. Para requisitos adicionais, consulte the section called “Recursos de volume no diretório /proc”.

> **Tip**
>
> Para configurar os diretórios `/var`, `/var/run` e `/var/lib` como recursos de volume, primeiro monte o diretório em uma pasta diferente e, em seguida, configure a pasta como um volume de recursos.

Ao configurar recursos de volume, você especifica um caminho de origem e um caminho de destino. O caminho de origem é o caminho absoluto do recurso no host. O caminho de destino é o caminho absoluto do recurso no ambiente do namespace do Lambda. Este é o contêiner que é executado em uma função ou conector do Lambda do Greengrass. Qualquer alteração no caminho de destino é refletida no caminho de origem do sistema de arquivos do host.

> **Note**
>
> Os arquivos no caminho de destino são visíveis somente no namespace do Lambda. Não é possível vê-los em um namespace Linux regular.

Recursos de dispositivo

Arquivos em `/dev`. Somente dispositivos de caracteres ou dispositivos de bloco em `/dev` são permitidos para recursos do dispositivo. Isso inclui:

- Portas seriais usadas para se comunicar com dispositivos conectados por meio de portas seriais (por exemplo, `/dev/ttyS0` e `/dev/ttyS1`).
- USB usado para conectar periféricos USB (por exemplo, `/dev/ttyUSB0` ou `/dev/bus/usb`).
- GPIOs usadas para sensores e acionadores por meio da GPIO (por exemplo, `/dev/gpiomem`).
- GPUs usadas para acelerar a machine learning usando GPUs integradas (por exemplo, `/dev/nvidia0`).
- Câmeras usadas para capturar imagens e vídeos (por exemplo, `/dev/video0`).

> **Note**
>
> `/dev/shm` é uma exceção. Ele pode ser configurado apenas como um recursos de volume. Devem ser concedidas permissões de `rw` aos recursos em `/dev/shm`.

AWS IoT Greengrass também oferece suporte a tipos de recursos usados para realizar inferências de aprendizado de máquina. Para obter mais informações, consulte Executar a inferência de machine learning.

Requisitos

Os requisitos a seguir se aplicam ao configurar o acesso seguro a recursos locais:

- Você deve estar usando o AWS IoT Greengrass Core Software v1.3 ou posterior. Para criar recursos para o diretório /proc do host, você deve estar usando a v1.6 ou posterior.
- O recurso local (inclusive todos os drivers e bibliotecas obrigatórios) deve estar instalado corretamente no dispositivo básico do Greengrass e disponível de maneira consistente durante o uso.
- A operação desejada do recurso, bem como o acesso ao recurso, não devem exigir privilégios de raiz.
- Estão disponíveis somente permissões de `read` ou `read and write`. As funções do Lambda não podem realizar operações privilegiadas nos recursos.
- Você precisa fornecer o caminho completo do recurso local no sistema operacional do dispositivo de núcleo do Greengrass.
- Um nome de recurso ou ID tem um comprimento máximo de 128 caracteres e deve usar o padrão `[a-zA-Z0-9:_-]+`.

Recursos de volume no diretório /proc

As considerações a seguir se aplicam a recursos de volume no diretório /proc do host.

- Você deve estar usando o AWS IoT Greengrass Core Software v1.6 ou posterior.
- Você pode permitir acesso somente leitura para funções do Lambda, mas não acesso de leitura/gravação. Esse nível de acesso é gerenciado por AWS IoT Greengrass.
- Também pode ser necessário conceder ao grupo do sistema operacional permissões para permitir acesso de leitura no sistema de arquivos. Por exemplo, suponhamos que o diretório de origem ou o arquivo tenha uma permissão de 660 arquivos, o que significa que somente o proprietário ou o usuário no grupo tem acesso de leitura (e gravação). Nesse caso, você deve adicionar as permissões do proprietário do grupo do sistema operacional ao recurso. Para obter mais informações, consulte the section called “Permissão de acesso a arquivo do proprietário do grupo”.

- Como o ambiente de host e o namespace do Lambda contêm um diretório /proc, certifique-se de evitar conflitos de nomenclatura ao especificar o caminho de destino. Por exemplo, caso /proc seja o caminho de origem, você pode especificar /host-proc como o caminho de destino (ou qualquer nome de caminho que não seja "/proc").

Permissão de acesso a arquivo do proprietário do grupo

Um processo de função AWS IoT Greengrass Lambda normalmente é executado como e. `ggc_user` `ggc_group` No entanto, você pode conceder permissões de acesso adicionais a arquivos ao processo da função do Lambda na definição de recurso local, da seguinte maneira:

- Para adicionar as permissões do grupo Linux que possui o recurso, use o parâmetro `GroupOwnerSetting#AutoAddGroupOwner` ou a opção do console Automatically add file system permissions of the system group that owns the resource (Adicionar automaticamente permissões de sistema de arquivos do grupo de sistemas que possui o recurso).
- Para adicionar as permissões de um grupo de Linux diferente, use o parâmetro `GroupOwnerSetting#GroupOwner` ou a opção de console Specify another system group to add file system permissions (Especificar outro grupo de sistemas ao qual adicionar permissões de sistema de arquivos). O valor `GroupOwner` será ignorado se `GroupOwnerSetting#AutoAddGroupOwner` for verdadeiro.

Um processo de função AWS IoT Greengrass Lambda herda todas as permissões do sistema de arquivos de `ggc_userggc_group`, e do grupo Linux (se adicionado). Para a função do Lambda acessar um recurso, o processo de função do Lambda deve ter as permissões obrigatórias para o recurso. Você pode usar o comando `chmod(1)` para alterar a permissão do recurso, se necessário.

Consulte também

- Cotas de serviço para recursos no Referência geral da Amazon Web Services

Como configurar o acesso a recursos locais usando o AWS Interface da linha de comando

Esse recurso está disponível para AWS IoT Greengrass Core v1.3 e versões posteriores.

Para usar um recurso local, você deve adicionar uma definição de recurso à definição do grupo implantada no dispositivo básico do Greengrass. A definição do grupo também deve conter uma função do Lambda na qual você concede permissões de acesso dos recursos locais às suas funções do Lambda. Para obter mais informações, inclusive sobre os requisitos e restrições, consulte Acesso aos recursos locais com funções e conectores do Lambda.

Este tutorial descreve o processo para criar um recurso local e configurar o acesso a ele usando a AWS Command Line Interface (CLI). Para seguir as etapas do tutorial, você já deve ter criado um grupo do Greengrass, conforme descrito em Conceitos básicos de AWS IoT Greengrass.

Para obter um tutorial que usa o Console de gerenciamento da AWS, consulteComo configurar o acesso a recursos locais usando o Console de gerenciamento da AWS.

Criar recursos locais

Primeiro, você usa o comando `CreateResourceDefinition` para criar uma definição do recurso que especifica os recursos a serem acessados. Neste exemplo, criamos dois recursos, `TestDirectory` e `TestCamera`:

```
aws greengrass create-resource-definition  --cli-input-json '{
    "Name": "MyLocalVolumeResource",
    "InitialVersion": {
        "Resources": [
            {
                "Id": "data-volume",
                "Name": "TestDirectory",
                "ResourceDataContainer": {
                    "LocalVolumeResourceData": {
                        "SourcePath": "/src/LRAtest",
                        "DestinationPath": "/dest/LRAtest",
                        "GroupOwnerSetting": {
                            "AutoAddGroupOwner": true,
                            "GroupOwner": ""
                        }
                    }
                }
            },
            {
                "Id": "data-device",
                "Name": "TestCamera",
                "ResourceDataContainer": {
```

```
                    "LocalDeviceResourceData": {
                        "SourcePath": "/dev/video0",
                        "GroupOwnerSetting": {
                            "AutoAddGroupOwner": true,
                            "GroupOwner": ""
                        }
                    }
                }
            }
        ]
    }
}'
```

Recursos: Uma lista de objetos `Resource` no grupo do Greengrass. Um grupo do Greengrass pode ter até 50 recursos.

Resource#Id: O identificador exclusivo do recurso. O ID é usado para consultar um recurso na configuração da função do Lambda. Comprimento máximo 128 caracteres. Padrão: [a-z:_ A-Z0-9 -] +.

Resource#Name: O nome do recurso. O nome do recurso é exibido no console do Greengrass. Comprimento máximo 128 caracteres. Padrão: [a-z:_ A-Z0-9 -] +.

LocalDeviceResourceData# SourcePath: O caminho absoluto local do recurso do dispositivo. O caminho de origem de um recurso do dispositivo só pode se referir a um dispositivo de caracteres ou dispositivo de blocos em `/dev`.

LocalVolumeResourceData# SourcePath: O caminho absoluto local do recurso de volume no dispositivo principal do Greengrass. Esse local está fora do contêiner em que a função é executada. O caminho de origem de um tipo de recurso de volume não pode começar com `/sys`.

LocalVolumeResourceData# DestinationPath: O caminho absoluto do recurso de volume dentro do ambiente Lambda. Esse local está dentro do contêiner em que a função é executada.

GroupOwnerSetting: permite que você configure privilégios de grupo adicionais para o processo Lambda. Esse campo é opcional. Para obter mais informações, consulte Permissão de acesso a arquivo do proprietário do grupo.

GroupOwnerSetting# AutoAddGroupOwner: Se verdadeiro, o Greengrass adiciona automaticamente o proprietário do grupo de sistema operacional Linux especificado do recurso aos privilégios do processo Lambda. Assim, o processo do Lambda tem as permissões de acesso do arquivo do grupo do Linux adicionado.

GroupOwnerSetting# GroupOwner: especifica o nome do grupo do sistema operacional Linux cujos privilégios são adicionados ao processo Lambda. Esse campo é opcional.

O ARN da versão de definição do recurso é retornado por `CreateResourceDefinition`. O ARN deve ser usado durante a atualização de uma definição de grupo.

```
{
    "LatestVersionArn": "arn:aws:greengrass:us-west-2:012345678901:/greengrass/
definition/resources/ab14d0b5-116e-4951-a322-9cde24a30373/versions/a4d9b882-
d025-4760-9cfe-9d4fada5390d",
    "Name": "MyLocalVolumeResource",
    "LastUpdatedTimestamp": "2017-11-15T01:18:42.153Z",
    "LatestVersion": "a4d9b882-d025-4760-9cfe-9d4fada5390d",
    "CreationTimestamp": "2017-11-15T01:18:42.153Z",
    "Id": "ab14d0b5-116e-4951-a322-9cde24a30373",
    "Arn": "arn:aws:greengrass:us-west-2:123456789012:/greengrass/definition/resources/
ab14d0b5-116e-4951-a322-9cde24a30373"
}
```

Criar a função do Greengrass

Depois que os recursos forem criados, use o comando `CreateFunctionDefinition` para criar a função do Greengrass e conceder o acesso de função ao recurso:

```
aws greengrass create-function-definition --cli-input-json '{
    "Name": "MyFunctionDefinition",
    "InitialVersion": {
        "Functions": [
            {
                "Id": "greengrassLraTest",
                "FunctionArn": "arn:aws:lambda:us-
west-2:012345678901:function:lraTest:1",
                "FunctionConfiguration": {
                    "Pinned": false,
                    "MemorySize": 16384,
                    "Timeout": 30,
                    "Environment": {
                        "ResourceAccessPolicies": [
                            {
                                "ResourceId": "data-volume",
                                "Permission": "rw"
                            },
```

```
                            {
                                "ResourceId": "data-device",
                                "Permission": "ro"
                            }
                        ],
                        "AccessSysfs": true
                    }
                }
            }
        ]
    }
}'
```

ResourceAccessPolicies: contém o `resourceId` e `permission` que concede à função Lambda acesso ao recurso. Uma função do Lambda pode acessar no máximo 20 recursos.

ResourceAccessPolicy#Permission: especifica quais permissões a função Lambda tem no recurso. As opções disponíveis são `rw` (read/write) ou `ro` (somente leitura).

AccessSysfs: Se verdadeiro, o processo Lambda pode ter acesso de leitura à `/sys` pasta no dispositivo principal do Greengrass. Isso é usado nos casos em que a função do Lambda do Greengrass precisa ler informações do dispositivo de `/sys`.

Novamente, `CreateFunctionDefinition` retorna um ARN da versão de definição da função. O ARN deve ser usado na versão de definição do grupo.

```
{
    "LatestVersionArn": "arn:aws:greengrass:us-west-2:012345678901:/greengrass/
definition/functions/3c9b1685-634f-4592-8dfd-7ae1183c28ad/versions/37f0d50e-ef50-4faf-
b125-ade8ed12336e",
    "Name": "MyFunctionDefinition",
    "LastUpdatedTimestamp": "2017-11-22T02:28:02.325Z",
    "LatestVersion": "37f0d50e-ef50-4faf-b125-ade8ed12336e",
    "CreationTimestamp": "2017-11-22T02:28:02.325Z",
    "Id": "3c9b1685-634f-4592-8dfd-7ae1183c28ad",
    "Arn": "arn:aws:greengrass:us-west-2:123456789012:/greengrass/definition/
functions/3c9b1685-634f-4592-8dfd-7ae1183c28ad"
}
```

Adicionar a função do Lambda ao grupo

Por fim, use `CreateGroupVersion` para adicionar a função ao grupo. Por exemplo:

```
aws greengrass create-group-version --group-id "b36a3aeb-3243-47ff-9fa4-7e8d98cd3cf5" \
--resource-definition-version-arn "arn:aws:greengrass:us-west-2:123456789012:/
greengrass/definition/resources/db6bf40b-29d3-4c4e-9574-21ab7d74316c/versions/31d0010f-
e19a-4c4c-8098-68b79906fb87" \
--core-definition-version-arn "arn:aws:greengrass:us-west-2:123456789012:/
greengrass/definition/cores/adbf3475-f6f3-48e1-84d6-502f02729067/
versions/297c419a-9deb-46dd-8ccc-341fc670138b" \
--function-definition-version-arn "arn:aws:greengrass:us-west-2:123456789012:/
greengrass/definition/functions/d1123830-da38-4c4c-a4b7-e92eec7b6d3e/versions/a2e90400-
caae-4ffd-b23a-db1892a33c78" \
--subscription-definition-version-arn "arn:aws:greengrass:us-west-2:123456789012:/
greengrass/definition/subscriptions/7a8ef3d8-1de3-426c-9554-5b55a32fbcb6/
versions/470c858c-7eb3-4abd-9d48-230236bfbf6a"
```

> **Note**
>
> Para saber como obter o ID do grupo a ser usado com esses comandos, consulte the section called “Obter o ID do grupo”.

É apresentada uma nova versão do grupo:

```
{
    "Arn": "arn:aws:greengrass:us-west-2:012345678901:/greengrass/groups/
b36a3aeb-3243-47ff-9fa4-7e8d98cd3cf5/versions/291917fb-ec54-4895-823e-27b52da25481",
    "Version": "291917fb-ec54-4895-823e-27b52da25481",
    "CreationTimestamp": "2017-11-22T01:47:22.487Z",
    "Id": "b36a3aeb-3243-47ff-9fa4-7e8d98cd3cf5"
}
```

Seu grupo do Greengrass agora contém a função LRatest Lambda que tem acesso a dois recursos: e. TestDirectory TestCamera

Esta função do Lambda de exemplo, `lraTest.py`, escrito em Python, grava no recurso do volume local:

```
# Demonstrates a simple use case of local resource access.
# This Lambda function writes a file test to a volume mounted inside
# the Lambda environment under destLRAtest. Then it reads the file and
# publishes the content to the AWS IoT LRAtest topic.
```

```
import sys
import greengrasssdk
import platform
import os
import logging

# Setup logging to stdout
logger = logging.getLogger(__name__)
logging.basicConfig(stream=sys.stdout, level=logging.DEBUG)

# Create a Greengrass Core SDK client.
client = greengrasssdk.client('iot-data')
volumePath = '/dest/LRAtest'

def function_handler(event, context):
    try:
        client.publish(topic='LRA/test', payload='Sent from AWS IoT Greengrass Core.')
        volumeInfo = os.stat(volumePath)
        client.publish(topic='LRA/test', payload=str(volumeInfo))
        with open(volumePath + '/test', 'a') as output:
            output.write('Successfully write to a file.')
        with open(volumePath + '/test', 'r') as myfile:
            data = myfile.read()
        client.publish(topic='LRA/test', payload=data)
    except Exception as e:
        logger.error('Failed to publish message: ' + repr(e))
    return
```

Esses comandos são fornecidos pela API do Greengrass para criar e gerenciar definições de recursos e versões de definição de recursos:

- CreateResourceDefinition
- CreateResourceDefinitionVersion
- DeleteResourceDefinition
- GetResourceDefinition
- GetResourceDefinitionVersion
- ListResourceDefinitions
- ListResourceDefinitionVersions
- UpdateResourceDefinition

Solução de problemas

- P: Por que minha implantação do grupo do Greengrass falha com um erro semelhante a:

  ```
  group config is invalid:
      ggc_user or [ggc_group root tty] don't have ro permission on the file: /dev/tty0
  ```

  R: Esse erro indica que o processo do Lambda não tem permissão para acessar o recurso especificado. A solução é alterar a permissão do arquivo do recurso, de maneira que o Lambda possa acessá-lo. (Consulte Permissão de acesso a arquivo do proprietário do grupo para obter detalhes).

- P: quando eu configuro `/var/run` como um recurso do volume, por que a função do Lambda não consegue inicializar, gerando uma mensagem de erro no runtime.log?

  ```
  [ERROR]-container_process.go:39,Runtime execution error: unable to start lambda
   container.
  container_linux.go:259: starting container process caused "process_linux.go:345:
  container init caused \"rootfs_linux.go:62: mounting \\\"/var/run\\\" to rootfs \\\"/
  greengrass/ggc/packages/1.3.0/rootfs_sys\\\" at \\\"/greengrass/ggc/packages/1.3.0/
  rootfs_sys/run\\\"
  caused \\\"invalid argument\\\"\""
  ```

  R: o AWS IoT Greengrass núcleo atualmente não suporta a configuração de `/var/var/run`, e `/var/lib` como recursos de volume. Uma solução é primeiro montar `/var`, `/var/run` ou `/var/lib` em uma pasta diferente e, em seguida, configurar a pasta como recurso de volume.

- P: quando eu configuro `/dev/shm` como um recurso do volume com permissão somente leitura, por que a função do Lambda não consegue iniciar, gerando um erro no runtime.log?

  ```
  [ERROR]-container_process.go:39,Runtime execution error: unable to start lambda
   container.
  container_linux.go:259: starting container process caused "process_linux.go:345:
  container init caused \"rootfs_linux.go:62: mounting \\\"/dev/shm\\\" to rootfs \\\"/
  greengrass/ggc/packages/1.3.0/rootfs_sys\\\" at \\\"/greengrass/ggc/packages/1.3.0/
  rootfs_sys/dev/shm\\\"
  caused \\\"operation not permitted\\\"\""”
  ```

  R: só `/dev/shm` pode ser configurado como read/write. Altere a permissão do recurso para `rw` a fim de resolver o problema.

Como configurar o acesso a recursos locais usando o Console de gerenciamento da AWS

Esse recurso está disponível para AWS IoT Greengrass Core v1.3 e versões posteriores.

Você pode configurar funções do Lambda para acessar com segurança recursos locais no dispositivo de núcleo do Greengrass do host. Os recursos locais referem-se a barramentos e periféricos que estão fisicamente presentes no host, ou a volumes do sistema de arquivos no SO do host. Para obter mais informações, inclusive sobre os requisitos e restrições, consulte Acesso aos recursos locais com funções e conectores do Lambda.

Este tutorial descreve como usar o Console de gerenciamento da AWS para configurar o acesso aos recursos locais que estão presentes em um dispositivo AWS IoT Greengrass principal. Ele contém as seguintes etapas de nível elevado:

1. Crie um pacote de implantação para a função do Lambda
2. Crie e publique uma função do Lambda
3. Adicionar a função do Lambda ao grupo
4. Adicionar um recurso local ao grupo
5. Adicionar assinaturas ao grupo
6. Implantar o grupo

Para ver um tutorial que usa o AWS Command Line Interface, consulteComo configurar o acesso a recursos locais usando o AWS Interface da linha de comando.

Pré-requisitos

Para concluir este tutorial, é necessário:

- Um grupo do Greengrass e um núcleo do Greengrass (versão 1.3 ou posterior). Para criar um grupo ou núcleo do Greengrass, consulte Conceitos básicos de AWS IoT Greengrass.
- Os seguintes diretórios no dispositivo de núcleo do Greengrass:
  - /src/LRAtest
  - /dest/LRAtest

  O grupo de proprietários desses diretórios deve ter acesso de leitura e gravação aos diretórios. Você pode usar o seguinte comando para conceder acesso:

```
sudo chmod 0775 /src/LRAtest
```

Etapa 1: crie um pacote de implantação para a função do Lambda

Nesta etapa, você cria um pacote de implantação da função do Lambda, que é um arquivo ZIP que contém o código e as dependências da função. Você também baixa o AWS IoT Greengrass Core SDK para incluí-lo no pacote como uma dependência.

1. No seu computador, copie o script do Python a seguir em um arquivo local chamado `lraTest.py`. Essa é a lógica do aplicativo para a função do Lambda.

```
# Demonstrates a simple use case of local resource access.
# This Lambda function writes a file test to a volume mounted inside
# the Lambda environment under destLRAtest. Then it reads the file and
# publishes the content to the AWS IoT LRAtest topic.

import sys
import greengrasssdk
import platform
import os
import logging

# Setup logging to stdout
logger = logging.getLogger(__name__)
logging.basicConfig(stream=sys.stdout, level=logging.DEBUG)

# Create a Greengrass Core SDK client.
client = greengrasssdk.client('iot-data')
volumePath = '/dest/LRAtest'

def function_handler(event, context):
    try:
        client.publish(topic='LRA/test', payload='Sent from AWS IoT Greengrass
 Core.')
        volumeInfo = os.stat(volumePath)
        client.publish(topic='LRA/test', payload=str(volumeInfo))
        with open(volumePath + '/test', 'a') as output:
            output.write('Successfully write to a file.')
        with open(volumePath + '/test', 'r') as myfile:
            data = myfile.read()
```

```
        client.publish(topic='LRA/test', payload=data)
    except Exception as e:
        logger.error('Failed to publish message: ' + repr(e))
    return
```

2. Na página de downloads do AWS IoT Greengrass Core SDK, baixe o AWS IoT Greengrass Core SDK para Python em seu computador.
3. Descompacte o pacote obtido por download para obter o SDK. O SDK é a pasta do `greengrasssdk`.
4. Comprima os seguintes itens em um arquivo chamado `lraTestLambda.zip`.

   - `lraTest.py`. Lógica do aplicativo.
   - `greengrasssdk`. Biblioteca necessária para todas as funções do Lambda do Python.

   O arquivo `lraTestLambda.zip` é o pacote de implantação de sua função do Lambda. Agora, você está pronto para criar uma função do Lambda e fazer upload do pacote de implantação.

Etapa 2: crie e publique uma função do Lambda

Nesta etapa, você usa o AWS Lambda console para criar uma função do Lambda e configurá-la para usar seu pacote de implantação. Depois, publique uma versão da função e crie um alias.

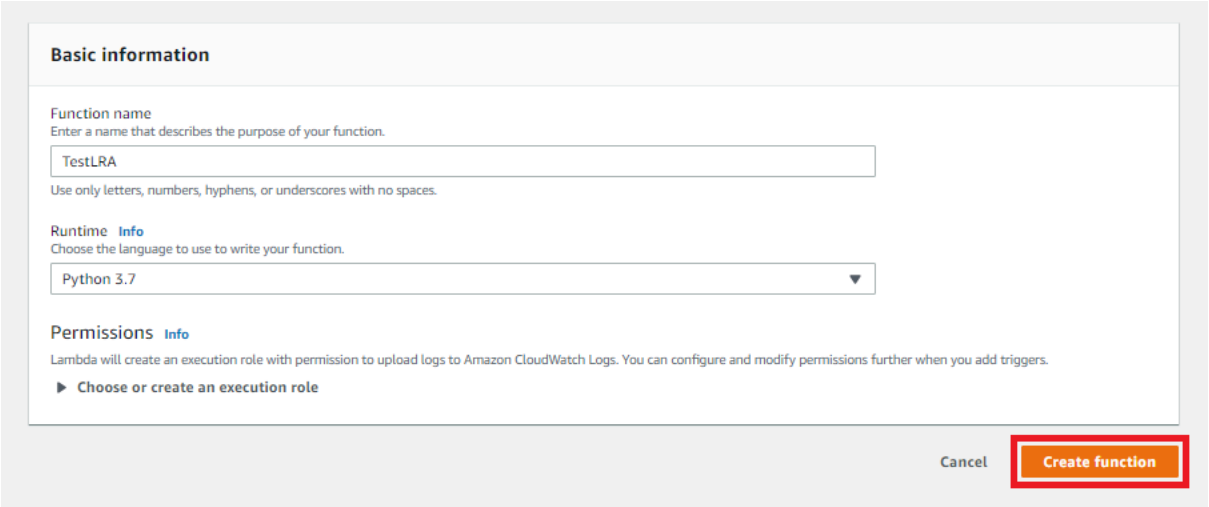
Primeiro, crie a função do Lambda.

1. No Console de gerenciamento da AWS, escolha Serviços e abra o AWS Lambda console.
2. Selecione Funções.
3. Selecione Criar função e, em seguida, selecione Criar do zero.
4. Na seção Basic information (Informações básicas), especifique os seguintes valores.

   a. Em Function name (Nome da função), insira **`TestLRA`**.

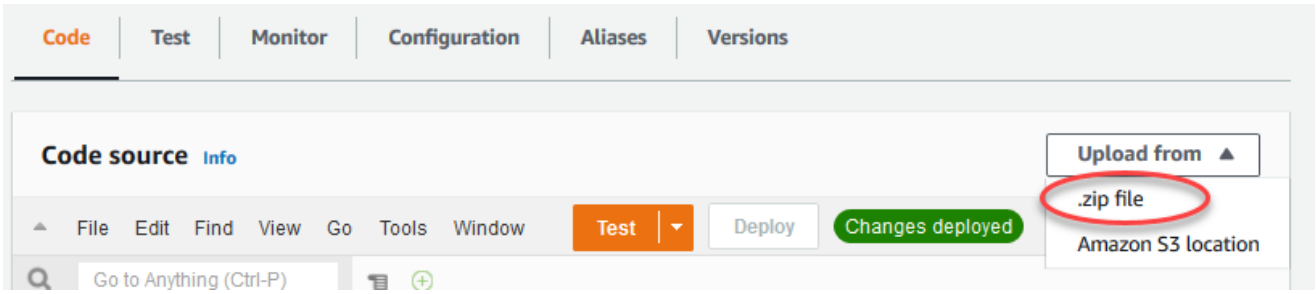
   b. Em Runtime (Tempo de execução), selecione Python 3.7.

   c. Em Permissões, mantenha a configuração padrão. Isso cria uma função de execução que concede permissões básicas do Lambda. Essa função não é usada por AWS IoT Greengrass.
5. Selecione Criar função.



6. Carregue o seu pacote de implantação da função do Lambda e registre o manipulador.

   a. Na guia Código, em Fonte do código, selecione Fazer upload a partir de. No menu suspenso, selecione o arquivo .zip.



   b. Selecione Upload e, em seguida, selecione seu pacote de implantação `lraTestLambda.zip`. Selecione Salvar.

   c. Na guia Código da função, em Configurações de runtime, selecione Editar e insira os valores a seguir.

      - Em Runtime (Tempo de execução), selecione Python 3.7.
      - Em Handler, insira lra Test.function _handler.

   d. Escolha Salvar.

> **Note**
>
> O botão Testar no AWS Lambda console não funciona com essa função. O SDK AWS IoT Greengrass Core não contém módulos necessários para executar suas

funções do Greengrass Lambda de forma independente no console. AWS Lambda Esses módulos (por exemplo, `greengrass_common`) são fornecidos às funções depois de serem implantados no núcleo do Greengrass.

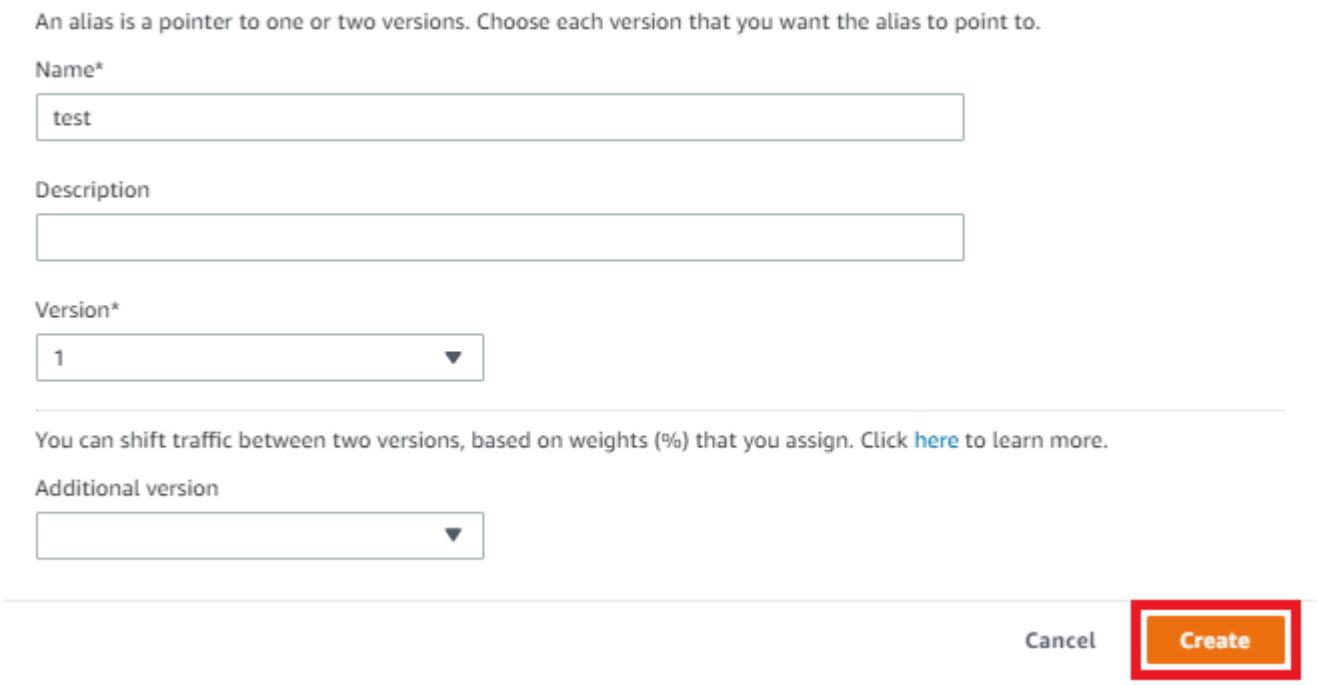
Em seguida, publique a primeira versão da sua função do Lambda. Em seguida, crie um alias para a versão.

Os grupos do Greengrass podem fazer referência a uma função do Lambda por alias (recomendado) ou por versão. Usar um alias facilita o gerenciamento de atualizações de código porque você não precisa alterar a tabela de assinaturas nem a definição do grupo ao atualizar a função do código. Em vez disso, você pode simplesmente apontar o alias para a nova versão da função.

7. No menu Ações, selecione Publish new version (Publicar nova versão).
8. Em Version description (Descrição da versão), insira **`First version`** e, em seguida, selecione Publish (Publicar).
9. Na página de configuração TestLRA: 1, em Ações, selecione Criar alias.
10. Na página Criar novo alias, em Nome, insira **`test`**. Em Version (Versão), insira 1.

> **Note**
>
> AWS IoT Greengrass não oferece suporte a aliases do Lambda para as versões $LATEST.

11. Selecione Create (Criar).



Agora, você pode adicionar a função do Lambda ao seu grupo do Greengrass.

Etapa 3: adicionar a função do Lambda ao grupo do Greengrass

Nesta etapa, você adiciona a função para o grupo e configura seu ciclo de vida da função.

Primeiro, adicione a função do Lambda ao seu grupo do Greengrass.

1. No painel de navegação do AWS IoT console, em Gerenciar, expanda dispositivos Greengrass e escolha Grupos (V1).
2. Selecione o grupo Greengrass onde você deseja adicionar a função do Lambda.
3. Na página de configuração do grupo, selecione a guia Funções do Lambda.
4. Na seção Minhas funções do Lambda, selecione Adicionar.
5. Na página Adicionar função do Lambda, selecione Função do Lambda. Selecione **`TestLRA`**.
6. Selecione a Versão da função do Lambda.
7. Na seção Configuração da função do Lambda, selecione Usuário e grupo do sistema e Conteinerização da função do Lambda.

Em seguida, configure o ciclo de vida da função do Lambda.

8. Para Tempo limite, selecione 30 seconds (30 segundos).

> **Important**
>
> As funções do Lambda que usam recursos locais (conforme descrito neste procedimento) deve ser executadas em um contêiner do Greengrass. Caso contrário, haverá falha na implantação se você tentar implantar a função. Para obter mais informações, consulte Conteinerização.

9. Na parte inferior da página, selecione Adicionar função do Lambda.

Etapa 4: Adicionar um recurso local a um grupo do Greengrass

Nesta etapa, você adicionará um recurso de volume local a um grupo do Greengrass e concederá à função acesso de leitura e gravação ao recurso. Um recurso local tem um escopo no nível do grupo. É possível conceder permissões para qualquer função do Lambda no grupo para acessar o recurso.

1. Na página de configuração do grupo, selecione a guia Recursos.
2. Na seção Recursos locais, selecione Adicionar.
3. Na página Adicionar um recurso local, use os seguintes valores.

   a. Em Resource Name (Nome do recurso), insira **`testDirectory`**.

   b. Em Tipo de recurso, selecione Volume.

   c. Em Caminho do dispositivo local, insira **`/src/LRAtest`**. Este caminho deve existir no sistema operacional do host.

      O caminho local do dispositivo é o caminho local absoluto do recurso no sistema de arquivos do dispositivo de núcleo. Esse local está fora do contêiner em que a função é executada. O caminho não pode começar com `/sys`.

   d. Para Destination path (Caminho de destino), insira **`/dest/LRAtest`**. Este caminho deve existir no sistema operacional do host.

      O caminho de destino é o caminho absoluto do recurso no namespace do Lambda. Esse local está dentro do contêiner em que a função é executada.

   e. Em Permissão de acesso ao arquivo do proprietário do grupo do sistema, selecione Adicionar automaticamente as permissões de grupo do sistema operacional do grupo Linux que tem o recurso.

A opção Proprietário do grupo do sistema e permissão de acesso a arquivos permite conceder permissões adicionais de acesso a arquivos para o processo do Lambda. Para obter mais informações, consulte Permissão de acesso a arquivo do proprietário do grupo.

4. Selecione Adicionar recurso. A página Resources exibe o novo recurso testDirectory.

Etapa 5: Adicionar assinaturas ao grupo do Greengrass

Nesta etapa, você adiciona duas assinaturas ao grupo do Greengrass. Essas assinaturas permitem a comunicação bidirecional entre a função Lambda e. AWS IoT

Primeiro, crie uma assinatura para a função do Lambda para enviar mensagens para a AWS IoT.

1. Na página de configuração do grupo, selecione a guia Assinaturas .
2. Selecione Adicionar.
3. Na página Criar uma assinatura, configure a origem e o destino, da seguinte forma:
   a. Em Tipo de origem, selecione Função do Lambda e, em seguida, TestLRA.
   b. Para Tipo de destino, selecione Serviço e, em seguida, IoT Cloud.
   c. Em Filtro de tópicos, insira **`LRA/test`** e, em seguida selecione Criar assinatura.
4. A página Subscriptions exibe a nova assinatura.

Em seguida, configure uma assinatura que invoque a função de. AWS IoT

5. Na página Subscriptions, selecione Add Subscription.
6. Na página Select your source and target, configure a origem e o destino, da seguinte forma:
   a. Em Tipo de origem, selecione Função do Lambda e, em seguida, selecione IoT Cloud.
   b. Em Tipo de destino, selecione Serviço e, em seguida, selecione TestLRA.
   c. Selecione Next (Próximo).
7. Na página Filter your data with a topic (Filtrar os dados com um tópico), em Topic filter (Filtro de tópico), insira **`invoke/LRAFunction`** e, em seguida, selecione Next (Próximo).
8. Selecione Finish. A página Subscriptions exibe ambas as assinaturas.

Etapa 6: implantar o AWS IoT Greengrass grupo

Nesta etapa, você implanta a versão atual da definição do grupo.

1. Certifique-se de que o AWS IoT Greengrass núcleo esteja funcionando. Execute os seguintes comandos no seu terminal do Raspberry Pi, conforme necessário.

   a. Para verificar se o daemon está em execução:

   ```
   ps aux | grep -E 'greengrass.*daemon'
   ```

   Se a saída contém uma entrada `root` para `/greengrass/ggc/packages/1.11.6/bin/daemon`, o daemon está em execução.

   > **Note**
   >
   > A versão no caminho depende da versão do software AWS IoT Greengrass Core instalada em seu dispositivo principal.

   b. Para iniciar o daemon:

   ```
   cd /greengrass/ggc/core/
   sudo ./greengrassd start
   ```

2. Na página de configuração do grupo, selecione Implantar.

   > **Note**
   >
   > A implantação falhará se você executar sua função do Lambda sem a conteinerização e tentar acessar recursos locais anexados.

3. Se solicitado, na guia Função do Lambda, em Funções do Lambda do sistema, selecione Detector de IP e, em seguida, Editar e, em seguida, Detectar automaticamente.

   Isso permite que os dispositivos adquiram automaticamente as informações de conectividade para o núcleo, como endereço IP, DNS e o número da porta. A detecção automática é recomendada, mas AWS IoT Greengrass também oferece suporte a endpoints especificados manualmente. Você só é solicitado pelo método de descoberta na primeira vez em que o grupo for implantado.

> **Note**
>
> Se solicitado, conceda permissão para criar a função de serviço do Greengrass e associá-la à sua Conta da AWS na atual. Região da AWS Essa função permite AWS IoT Greengrass acessar seus recursos em AWS serviços.

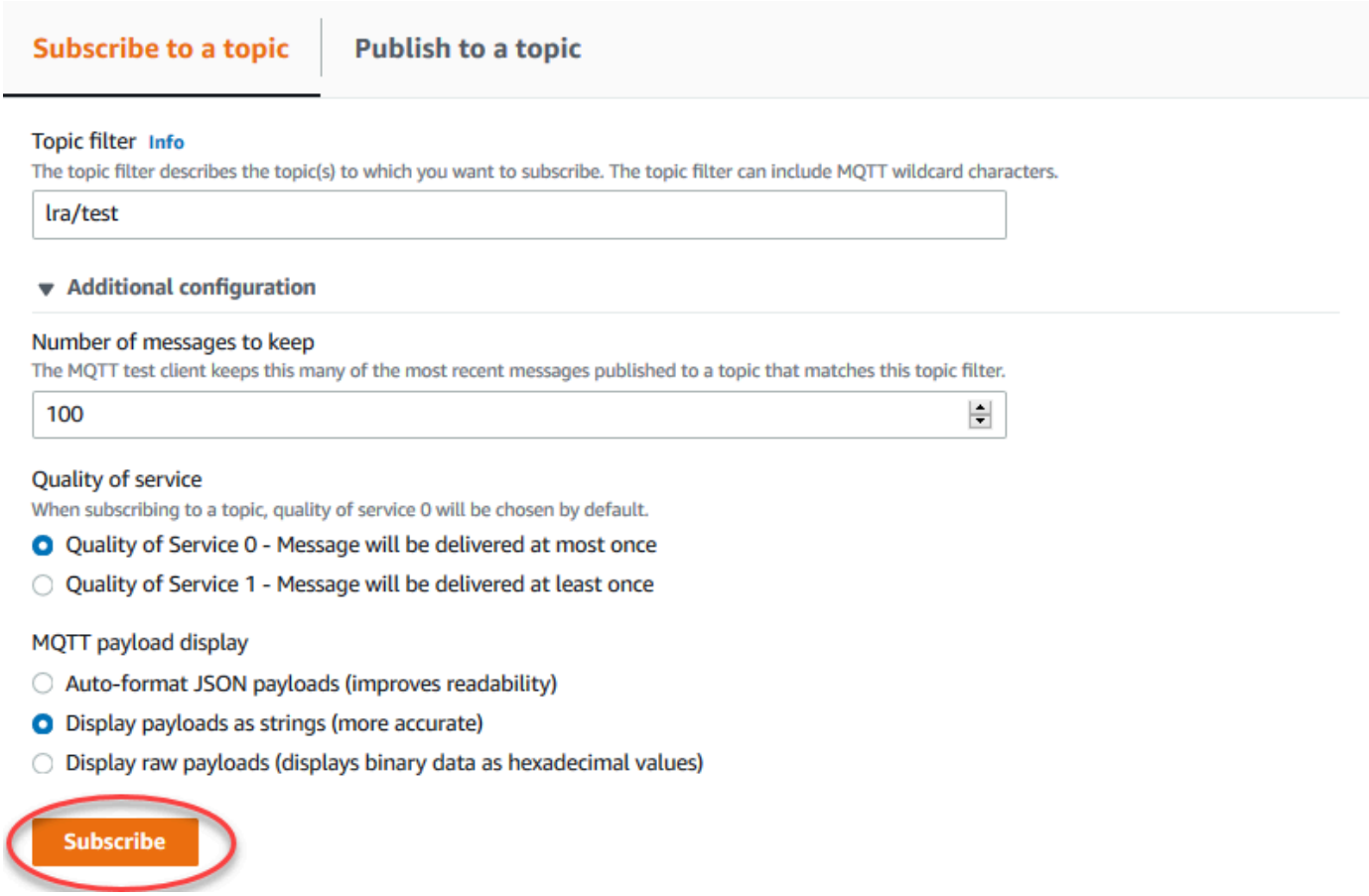
A página Deployments mostra a data e hora, ID de versão e status da implantação. Quando concluído, o status da implantação é Concluído.

Para obter ajuda sobre a solução de problemas, consulte Solução de problemas.

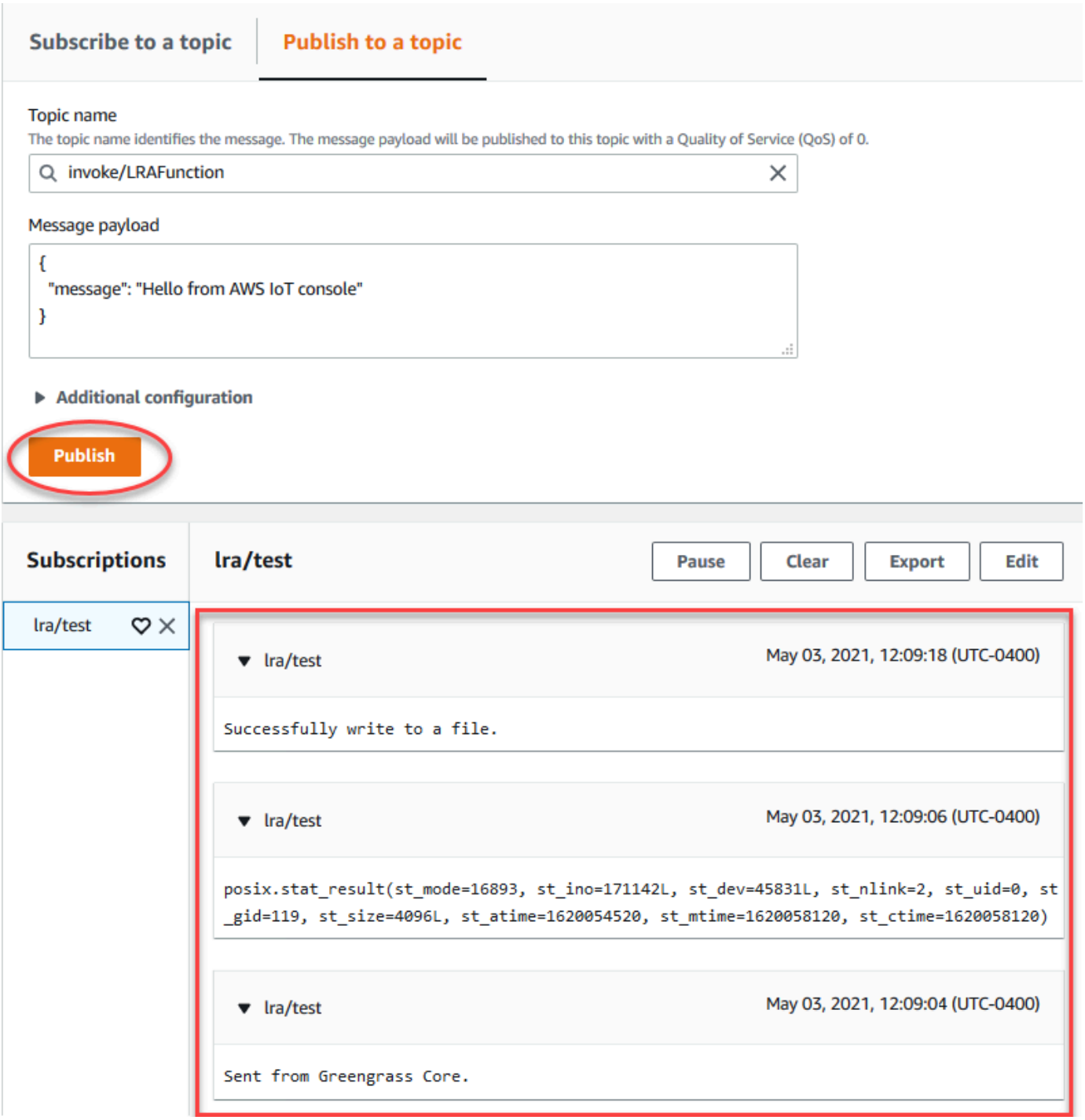
Testar o acesso aos recursos locais

Agora você pode verificar se o acesso aos recursos locais foi configurado corretamente. Para testar, inscreva-se no tópico `LRA/test` e publique no tópico `invoke/LRAFunction`. O teste será bem-sucedido se a função Lambda enviar a carga útil esperada para o. AWS IoT

1. No menu de navegação do AWS IoT console, em Teste, escolha Cliente de teste MQTT.
2. Em Inscrever-se em um tópico, em Filtro de tópicos, insira**`LRA/test`**.
3. Em Informações adicionais, para Exibição da carga útil do MQTT, selecione Exibir cargas úteis como strings.
4. Selecione Subscribe. Sua função do Lambda é publicada no LRA/test tópico.



5. Em Publicar em um tópico, em Nome do tópico, insira **`invoke/LRAFunction`** e, em seguida selecione Publicar para invocar sua função do Lambda. O teste é bem-sucedido se a página exibir as três cargas úteis de mensagens da função.



O arquivo de teste criado pela função do Lambda está no diretório `/src/LRAtest` no dispositivo de núcleo do Greengrass. Embora a função do Lambda grave em um arquivo no diretório de `/dest/LRAtest`, esse arquivo está visível apenas no namespace do Lamda. Você não pode vê-lo em um namespace Linux regular. Qualquer alteração no caminho de destino é refletida no caminho de origem no sistema de arquivos.

Para obter ajuda sobre a solução de problemas, consulte Solução de problemas.

Executar a inferência de machine learning

Esse recurso está disponível para AWS IoT Greengrass Core v1.6 ou posterior.

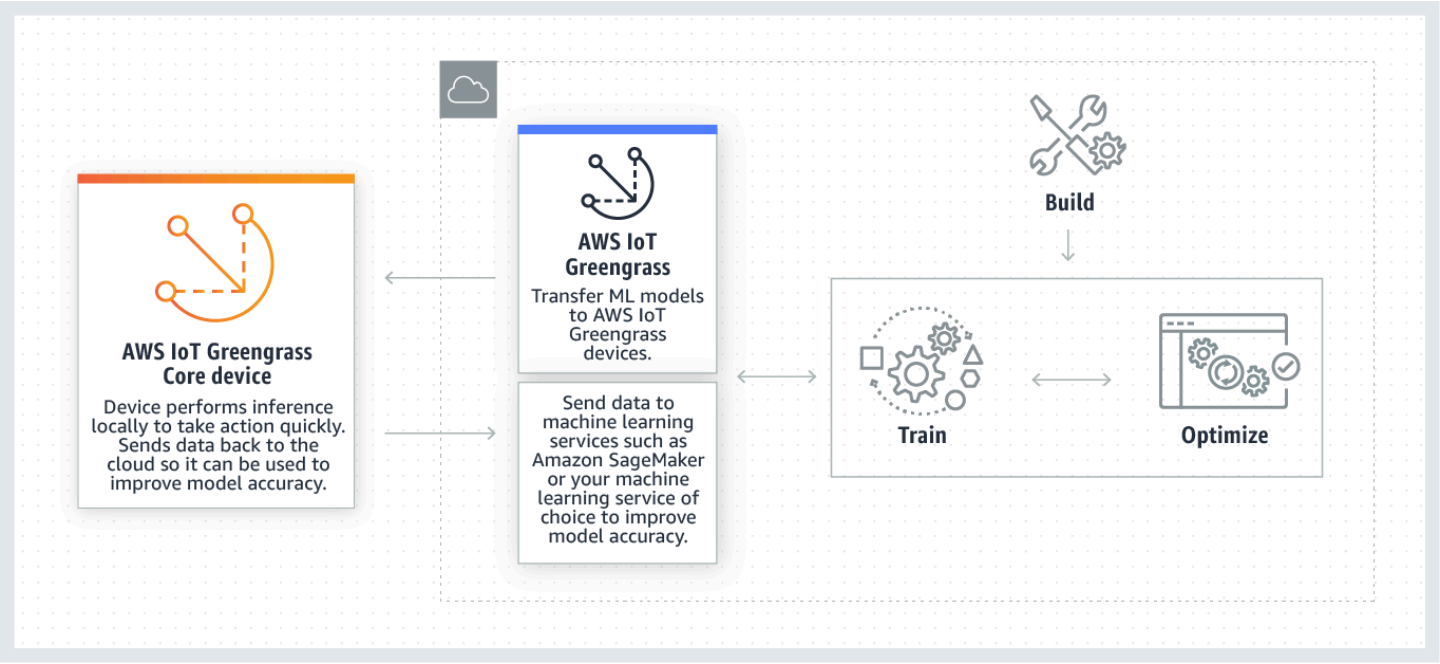
Com AWS IoT Greengrass, você pode realizar inferências de aprendizado de máquina (ML) na borda em dados gerados localmente usando modelos treinados em nuvem. Você se beneficia da baixa latência e da redução de custos na execução da inferência local, e ainda aproveita a capacidade de computação em nuvem para modelos de treinamento e processamento complexo.

Para começar a realizar a inferência local, consulte the section called “Como configurar a inferência de machine learning”.

Como AWS IoT Greengrass A inferência de ML funciona

Você pode treinar seus modelos de inferência em qualquer lugar, implantá-los localmente como recursos de machine learning em um grupo do Greengrass e, em seguida, acessá-los a partir das funções do Lambda do Greengrass. Por exemplo, você pode criar e treinar modelos de aprendizado profundo em SageMaker IA e implantá-los no seu núcleo do Greengrass. Em seguida, suas funções do Lambda poderão usar os modelos locais para realizar inferências nos dispositivos conectados e enviar novos dados de treinamento de volta para a nuvem.

O diagrama a seguir mostra o fluxo de trabalho AWS IoT Greengrass de inferência de ML.



AWS IoT Greengrass A inferência de ML simplifica cada etapa do fluxo de trabalho de ML, incluindo:

- Criação e implantação de protótipos de estrutura do ML.
- Acesso aos modelos treinados em nuvem e implementação em dispositivos de núcleo do Greengrass.
- Criação de aplicativos de inferência que podem acessar aceleradores de hardware (por exemplo, GPUs e FPGAs) como recursos locais.

Recursos de machine learning

Os recursos de aprendizado de máquina representam modelos de inferência treinados em nuvem que são implantados em um núcleo. AWS IoT Greengrass Para implantar recursos de machine learning, primeiro adicione os recursos a um grupo do Greengrass e defina como as funções do Lambda no grupo poderão acessá-los. Durante a implantação em grupo, AWS IoT Greengrass recupera os pacotes do modelo de origem da nuvem e os extrai para diretórios dentro do namespace de tempo de execução do Lambda. Em seguida, as funções do Lambda do Greengrass usam os modelos implantados localmente para realizar a inferência.

Para atualizar um modelo implantado localmente, primeiro atualize o modelo de origem (na nuvem) que corresponde ao recurso de machine learning e, em seguida, implemente o grupo. Durante a implantação, o AWS IoT Greengrass verifica se há alterações na origem. Se forem detectadas alterações, AWS IoT Greengrass atualizará o modelo local.

Fontes de modelo compatíveis

AWS IoT Greengrass oferece suporte a fontes de modelos de SageMaker IA e Amazon S3 para recursos de aprendizado de máquina.

Os requisitos a seguir se aplicam a fontes de modelo:

- Os buckets do S3 que armazenam suas fontes do modelo SageMaker AI e do Amazon S3 não devem ser criptografados usando. SSE-C Para buckets que usam criptografia do lado do servidor, a inferência de AWS IoT Greengrass ML atualmente oferece suporte somente às opções de criptografia or. SSE-S3 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.
- Os nomes dos buckets do S3 que armazenam suas fontes de modelos de SageMaker IA e Amazon S3 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.

- Service-level Região da AWS o suporte deve estar disponível tanto para a IA AWS IoT Greengrass quanto para a SageMaker IA. Atualmente, AWS IoT Greengrass oferece suporte a modelos de SageMaker IA nas seguintes regiões:
  - Leste dos EUA (Ohio)
  - Leste dos EUA (N. da Virgínia)
  - Oeste dos EUA (Oregon)
  - Ásia-Pacífico (Mumbai)
  - Ásia-Pacífico (Seul)
  - Ásia-Pacífico (Singapura)
  - Ásia-Pacífico (Sydney)
  - Ásia-Pacífico (Tóquio)
  - Europa (Frankfurt)
  - Europa (Irlanda)
  - Europa (Londres)
- AWS IoT Greengrass deve ter `read` permissão para a fonte do modelo, conforme descrito nas seções a seguir.

SageMaker AI

AWS IoT Greengrass oferece suporte a modelos que são salvos como trabalhos de treinamento de SageMaker IA. SageMaker A IA é um serviço de ML totalmente gerenciado que você pode usar para criar e treinar modelos usando algoritmos integrados ou personalizados. Para obter mais informações, consulte O que é SageMaker IA? no Guia do desenvolvedor de SageMaker IA.

Se você configurou seu ambiente de SageMaker IA criando um bucket cujo nome contém`sagemaker`, então AWS IoT Greengrass tem permissão suficiente para acessar seus trabalhos de treinamento de SageMaker IA. A política gerenciada `AWSGreengrassResourceAccessRolePolicy` permite o acesso a buckets cujo nome contém a string `sagemaker`. Esta política é anexada à função de serviço do Greengrass.

Caso contrário, você deverá conceder AWS IoT Greengrass `read` permissão ao bucket em que seu trabalho de treinamento está armazenado. Para fazer isso, incorpore a seguinte política em linha no perfil de serviço. Você pode listar vários ARNs do bucket.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket"
            ]
        }
    ]
}
```

Amazon S3

AWS IoT Greengrass oferece suporte a modelos que são armazenados no Amazon S3 como `tar.gz` ou `.zip` arquivos.

Para AWS IoT Greengrass habilitar o acesso a modelos que estão armazenados em buckets do Amazon S3, você deve conceder AWS IoT Greengrass `read` permissão para acessar os buckets fazendo o seguinte:

- Armazene seu modelo em um recipiente cujo nome contém `greengrass`.

  A política gerenciada `AWSGreengrassResourceAccessRolePolicy` permite o acesso a buckets cujo nome contém a string `greengrass`. Esta política é anexada à função de serviço do Greengrass.

- Incorpore uma política inline na função de serviço do Greengrass.

  Se o nome do seu intervalo não contiver `greengrass`, adicione a seguinte política inline à função de serviço. Você pode listar vários ARNs do bucket.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-demo-bucket"
            ]
        }
    ]
}
```

Para mais informações, consulte Incorporando políticas inline no Guia do usuário do IAM.

Requisitos

Os requisitos a seguir se aplicam para criar e usar recursos de machine learning:

- Você deve estar usando o AWS IoT Greengrass Core v1.6 ou posterior.
- User-defined As funções do Lambda podem realizar `read` nossas `read and write` operações no recurso. Permissões para outras operações não estão disponíveis. O modo de conteinerização das funções afiliadas do Lambda determina as permissões de acesso são definidas. Para obter mais informações, consulte the section called “Acesse os recursos de machine learning”.
- Você precisa fornecer o caminho completo do recurso no sistema operacional do dispositivo de núcleo.
- Um nome de recurso ou ID tem um comprimento máximo de 128 caracteres e deve usar o padrão `[a-zA-Z0-9:_-]+`.

Tempos de execução e bibliotecas para inferência de ML

Você pode usar os seguintes tempos de execução e bibliotecas de ML com AWS IoT Greengrass.

- Tempo de execução de aprendizado profundo do Amazon SageMaker Neo

- Apache MXNet
- TensorFlow

Esses tempos de execução e bibliotecas podem ser instalados nas plataformas NVIDIA Jetson TX2, Intel Atom e Raspberry Pi. Para baixar as informações, consulte the section called “Bibliotecas e tempos de execução de machine learning compatíveis”. Você pode instalá-los diretamente no dispositivo principal.

Leia as seguintes informações sobre compatibilidade e limitações.

SageMaker Tempo de execução de aprendizado profundo do AI Neo

Você pode usar o tempo de execução de aprendizado profundo do SageMaker AI Neo para realizar inferências com modelos otimizados de aprendizado de máquina em seus AWS IoT Greengrass dispositivos. Esses modelos são otimizados usando o compilador de aprendizado profundo SageMaker AI Neo para melhorar as velocidades de previsão de inferência de aprendizado de máquina. Para obter mais informações sobre otimização de modelos em SageMaker IA, consulte a documentação do SageMaker AI Neo.

> **Note**
>
> No momento, é possível otimizar modelos de machine learning usando o compilador de aprendizado profundo do Neo somente em Regiões específicas da Amazon Web Services. No entanto, você pode usar o tempo de execução de aprendizado profundo Neo com modelos otimizados em cada um em Região da AWS que o AWS IoT Greengrass núcleo é suportado. Para obter informações, consulte Como configurar a inferência otimizada de Machine Learning.

Versionamento do MXNet

O Apache MXNet atualmente não garante compatibilidade com versões futuras, portanto, os modelos que você treina usando versões posteriores da estrutura podem não funcionar corretamente em versões anteriores da estrutura. Para evitar conflitos entre os estágios de treinamento de modelo e a veiculação de modelo e fornecer uma experiência completa e consistente, use a mesma versão de estrutura do MXNet em ambos os estágios.

MXNet no Raspberry Pi

As funções do Lambda do Greengrass que acessam modelos locais do MXNet devem definir a seguinte variável de ambiente:

```
MXNET_ENGINE_TYPE=NativeEngine
```

É possível definir a variável de ambiente no código da função ou adicioná-la à configuração específica do grupo da função. Para um exemplo que a adiciona como uma configuração, consulte esta etapa.

> **Note**
>
> Para uso geral da estrutura do MXNet, como executar um exemplo de código de terceiros, a variável de ambiente deve ser configurada no Raspberry Pi.

TensorFlow limitações de serviço de modelos no Raspberry Pi

As recomendações a seguir para melhorar os resultados da inferência são baseadas em nossos testes com as bibliotecas Arm de TensorFlow 32 bits na plataforma Raspberry Pi. Estas recomendações são destinadas apenas para usuários avançados, sem garantias de qualquer tipo.

- Modelos treinados usando o formato de ponto de verificação deve ser "congelado" para o formato de buffer de protocolo antes do fornecimento. Para ver um exemplo, consulte a biblioteca de modelos de classificação de TensorFlow-Slim imagens.
- Não use as TF-Slim bibliotecas TF-Estimator e no código de treinamento ou de inferência. Em vez disso, use o arquivo `.pb` padrão de carregamento do modelo mostrado no exemplo a seguir.

```
graph = tf.Graph()
graph_def = tf.GraphDef()
graph_def.ParseFromString(pb_file.read())
with graph.as_default():
  tf.import_graph_def(graph_def)
```

> **Note**
>
> Para obter mais informações sobre as plataformas suportadas pelo TensorFlow, consulte Instalação TensorFlow na TensorFlow documentação.

Acesse os recursos de machine learning das funções do Lambda

User-defined As funções Lambda podem acessar recursos de aprendizado de máquina para executar inferência local no núcleo. AWS IoT Greengrass Um recurso de machine learning consiste no modelo treinado e outros artefatos que são obtidos por download para o dispositivo de núcleo.

Para permitir que uma função do Lambda acesse um recurso de machine learning no núcleo, você deve anexar o recurso à função do Lambda e definir permissões de acesso. O modo de conteinerização da função do Lambda afiliada (ou anexada) determina como você faz isso.

Acessar as permissões para recursos de machine learning

A partir AWS IoT Greengrass do Core v1.10.0, você pode definir um proprietário de recurso para um recurso de aprendizado de máquina. O proprietário do recurso representa o grupo de sistemas operacionais e as permissões que AWS IoT Greengrass usa para baixar os artefatos do recurso. Se um proprietário do recurso não estiver definido, os artefatos de recurso obtidos por download estarão acessíveis somente na raiz.

- Se funções Lambda não conteinerizadas acessarem um recurso de aprendizado de máquina, você deverá definir um proprietário do recurso, pois não há controle de permissão do contêiner. Non-containerized As funções Lambda podem herdar as permissões do proprietário do recurso e usá-las para acessar o recurso.

- Se apenas funções conteinerizadas do Lambda acessarem o recurso, recomendamos que você use permissões no nível da função em vez de definir um proprietário do recurso.

Propriedades do proprietário do recurso

Um proprietário do recurso especifica um proprietário do grupo e permissões de proprietário do grupo.

Proprietário do grupo. O ID do grupo (GID) de um grupo de SO Linux existente no dispositivo de núcleo. As permissões do grupo são adicionadas ao processo do Lambda. Especificamente, o GID é adicionado aos IDs suplementares de grupo da função do Lambda.

Se uma função do Lambda no grupo Greengrass estiver configurada para ser executada como o mesmo grupo de SO que o proprietário do recurso de um recurso de machine learning, o recurso deverá ser anexado à função do Lambda. Caso contrário, a implantação falhará porque essa configuração fornece permissões implícitas que a função Lambda pode usar para acessar o recurso AWS IoT Greengrass sem autorização. A verificação de validação da implantação será ignorada se a função do Lambda for executada como raiz (UID=0).

Recomendamos que você use um grupo de SO que não seja usado por outros recursos, funções do Lambda ou arquivos no núcleo do Greengrass. O uso de um grupo de SO compartilhado concede mais permissões de acesso às funções anexadas do Lambda do que elas precisam. Se você usar um grupo de SO compartilhado, uma função do Lambda anexada também deverá ser anexada a todos os recursos de machine learning que usam o grupo de SO compartilhado. Caso contrário, haverá falha na implantação.

Permissões de proprietário do grupo. A permissão somente leitura ou leitura e gravação a ser adicionada ao processo do Lambda.

Non-containerized As funções Lambda devem herdar essas permissões de acesso ao recurso. As funções conteinerizadas do Lambda podem herdar essas permissões no nível do recurso ou definir permissões no nível da função. Se elas definirem permissões no nível da função, as permissões deverão ser as mesmas ou mais restritivas do que as permissões no nível do recurso.

A tabela a seguir mostra as configurações de permissão de acesso compatíveis.

GGC v1.10 or later

Propriedade	Se apenas as funções conteinerizadas do Lambda acessarem o recurso	Se qualquer função não conteinerizada do Lambda acessar o recurso
Function-level propriedades		
Permissões (read/write)	Obrigatório, a menos que o recurso defina um proprietário do recurso. Se um proprietário do recurso estiver definido, as permissões no nível da função devem ser as mesmas ou mais restritivas do que as permissões de proprietário do recurso. Se apenas funções conteinerizadas do Lambda acessarem o recurso, recomendamos que você não defina um proprietário do recurso.	Non-containerized Funções do Lambda: Não suportado. Non-conta inerized As funções Lambda devem herdar permissões em nível de recurso. Funções conteinerizadas do Lambda: Opcional, mas devem ser as mesmas ou mais restritiv as do que as permissões no nível do recurso.
Resource-level propriedades		
Proprietário do recurso	Opcional (não recomendado).	Obrigatório.
Permissões (read/write)	Opcional (não recomendado).	Obrigatório.

GGC v1.9 or earlier

Propriedade	Se apenas as funções conteinerizadas do Lambda acessarem o recurso	Se qualquer função não conteinerizada do Lambda acessar o recurso
Function-level propriedades		
Permissões (read/write)	Obrigatório.	Sem compatibilidade.
Resource-level propriedades		
Proprietário do recurso	Sem compatibilidade.	Sem compatibilidade.
Permissões (read/write)	Sem compatibilidade.	Sem compatibilidade.

> **Note**
>
> Quando você usa a AWS IoT Greengrass API para configurar funções e recursos do Lambda, a `ResourceId` propriedade em nível de função também é necessária. A propriedade `ResourceId` anexa o recurso de machine learning à função do Lambda.

Definição das permissões de acesso às funções do Lambda (console)

No AWS IoT console, você define as permissões de acesso ao configurar um recurso de aprendizado de máquina ou anexar um a uma função Lambda.

Funções conteinerizadas do Lambda

Se apenas funções conteinerizadas do Lambda estiverem anexadas ao recurso de machine learning:

- Selecione Nenhum grupo de sistema como proprietário do recurso para o recurso de machine learning. Essa é a configuração recomendada quando apenas funções conteinerizadas do Lambda acessam o recurso de machine learning. Caso contrário, você pode conceder mais permissões de acesso às funções anexadas do Lambda do que elas precisam.

Non-containerized Funções Lambda (requer GGC v1.10 ou posterior)

Se alguma função não conteinerizada do Lambda estiver anexada ao recurso de machine learning:

- Especifique o ID do grupo do sistema (GID) a ser usado como proprietário do recurso para o recurso de machine learning. Selecione Specify system group and permissions (Especificar grupo do sistema e permissões) e insira o GID. Você pode usar o comando `getent group` no dispositivo de núcleo para pesquisar o ID de um grupo de sistemas.

- Escolha Read-only acesso ou acesso de leitura e gravação para as permissões do grupo Sistema.

Definindo as permissões de acesso para funções do Lambda (API)

Na AWS IoT Greengrass API, você define permissões para recursos de aprendizado de máquina na `ResourceAccessPolicy` propriedade da função Lambda ou na `OwnerSetting` propriedade do recurso.

Funções conteinerizadas do Lambda

Se apenas funções conteinerizadas do Lambda estiverem anexadas ao recurso de machine learning:

- Para funções conteinerizadas do Lambda, defina permissões de acesso na propriedade `Permission` da propriedade `ResourceAccessPolicies`. Por exemplo:

```
"Functions": [
    {
        "Id": "my-containerized-function",
        "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:function-
name:alias-or-version",
        "FunctionConfiguration": {
            "Environment": {
                "ResourceAccessPolicies": [
                    {
                        "ResourceId": "my-resource-id",
                        "Permission": "ro-or-rw"
                    }
```

```
                ]
            },
            "MemorySize": 512,
            "Pinned": true,
            "Timeout": 5
        }
    }
]
```

- Para recursos de machine learning, omita a propriedade `OwnerSetting`. Por exemplo:

```
"Resources": [
    {
        "Id": "my-resource-id",
        "Name": "my-resource-name",
        "ResourceDataContainer": {
            "S3MachineLearningModelResourceData": {
                "DestinationPath": "/local-destination-path",
                "S3Uri": "s3://uri-to-resource-package"
            }
        }
    }
]
```

Essa é a configuração recomendada quando apenas funções conteinerizadas do Lambda acessam o recurso de machine learning. Caso contrário, você pode conceder mais permissões de acesso às funções anexadas do Lambda do que elas precisam.

Non-containerized Funções Lambda (requer GGC v1.10 ou posterior)

Se alguma função não conteinerizada do Lambda estiver anexada ao recurso de machine learning:

- Para funções não conteinerizadas do Lambda, omita a propriedade `Permission` em `ResourceAccessPolicies`. Essa configuração é necessária e permite que a função herde a permissão no nível do recurso. Por exemplo:

```
"Functions": [
    {
        "Id": "my-non-containerized-function",
```

```
        "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:function-
name:alias-or-version",
        "FunctionConfiguration": {
            "Environment": {
                "Execution": {
                    "IsolationMode": "NoContainer",
                },
                "ResourceAccessPolicies": [
                    {
                        "ResourceId": "my-resource-id"
                    }
                ]
            },
            "Pinned": true,
            "Timeout": 5
        }
    }
]
```

- Para funções conteinerizadas do Lambda que também acessam o recurso de machine learning, omita a propriedade `Permission` em `ResourceAccessPolicies` ou defina uma permissão que seja a mesma ou mais restritiva que a permissão no nível do recurso. Por exemplo:

```
"Functions": [
    {
        "Id": "my-containerized-function",
        "FunctionArn": "arn:aws:lambda:us-west-2:123456789012:function:function-
name:alias-or-version",
        "FunctionConfiguration": {
            "Environment": {
                "ResourceAccessPolicies": [
                    {
                        "ResourceId": "my-resource-id",
                        "Permission": "ro-or-rw" // Optional, but cannot exceed
 the GroupPermission defined for the resource.
                    }
                ]
            },
            "MemorySize": 512,
            "Pinned": true,
            "Timeout": 5
        }
```

```
    }
]
```

- Para recursos de machine learning, defina a propriedade `OwnerSetting`, incluindo o filho `GroupOwner` e as propriedades `GroupPermission`. Por exemplo:

```
"Resources": [
    {
        "Id": "my-resource-id",
        "Name": "my-resource-name",
        "ResourceDataContainer": {
            "S3MachineLearningModelResourceData": {
                "DestinationPath": "/local-destination-path",
                "S3Uri": "s3://uri-to-resource-package",
                "OwnerSetting": {
                    "GroupOwner": "os-group-id",
                    "GroupPermission": "ro-or-rw"
                }
            }
        }
    }
]
```

Acessando os recursos de machine learning do código de função do Lambda

User-defined As funções Lambda usam interfaces de sistema operacional específicas da plataforma para acessar recursos de aprendizado de máquina em um dispositivo principal.

GGC v1.10 or later

Para funções conteinerizadas do Lambda, o recurso é montado dentro do contêiner Greengrass e disponibilizado no caminho de destino local definido para o recurso. Para funções Lambda não conteinerizadas, o recurso é vinculado a Lambda-specific um diretório de trabalho e passado para a variável de `AWS_GG_RESOURCE_PREFIX` ambiente no processo Lambda.

Para obter o caminho para os artefatos de um recurso de machine learning obtidos por download, as funções do Lambda acrescentam a variável de ambiente `AWS_GG_RESOURCE_PREFIX` ao caminho de destino local definido para o recurso. Para funções conteinerizadas do Lambda, o valor retornado é uma barra única (`/`).

```
resourcePath = os.getenv("AWS_GG_RESOURCE_PREFIX") + "/destination-path"
with open(resourcePath, 'r') as f:
    # load_model(f)
```

GGC v1.9 or earlier

Os artefatos de um recurso de machine learning obtidos por download estão localizados no caminho de destino local definido para o recurso. Apenas funções conteinerizadas do Lambda podem acessar recursos de machine learning no AWS IoT Greengrass Core v1.9 e versões anteriores.

```
resourcePath = "/local-destination-path"
with open(resourcePath, 'r') as f:
    # load_model(f)
```

A implementação do seu modelo de carregamento depende da sua biblioteca de ML.

Solução de problemas

Use as informações a seguir para ajudá-lo a solucionar problemas com o acesso aos recursos de machine learning.

Tópicos

- InvalidMLModelOwner - GroupOwnerSetting é fornecido no recurso do modelo ML, mas GroupOwner ou não GroupPermission está presente
- NoContainer a função não pode configurar a permissão ao anexar recursos do Machine Learning. <function-arn > se refere ao ID do recurso de aprendizado de máquina com permissão na política de < acesso a recursos>. < ro/rw >
- A função < function-arn > se refere ao < resource-id > do recurso de Machine Learning sem permissão em ambos e no recurso. ResourceAccessPolicy OwnerSetting
- A função < function-arn > se refere ao < resource-id do recurso do Machine Learning > com a permissão\ "rw\", enquanto a configuração do proprietário do recurso permite GroupPermission apenas\ "ro\".
- NoContainer A função < function-arn > se refere aos recursos do caminho de destino aninhado.
- O Lambda < function-arn > obtém acesso ao < resource-id do recurso compartilhando o mesmo ID > de proprietário do grupo

InvalidMLModelOwner - GroupOwnerSetting é fornecido no recurso do modelo ML, mas GroupOwner ou não GroupPermission está presente

Solução: você receberá esse erro se um recurso de aprendizado de máquina contiver o ResourceDownloadOwnerSettingobjeto, mas o necessário `GroupOwner` ou a `GroupPermission` propriedade não estiverem definidos. Para resolver esse problema, defina a propriedade ausente.

NoContainer a função não pode configurar a permissão ao anexar recursos do Machine Learning. <function-arn > se refere ao ID do recurso de aprendizado de máquina com permissão na política de < acesso a recursos>. < ro/rw >

Solução: você recebe esse erro se uma função Lambda não conteinerizada especificar permissões em nível de função para um recurso de aprendizado de máquina. Non-containerized as funções devem herdar as permissões do proprietário do recurso definidas no recurso de aprendizado de máquina. Para resolver esse problema, selecione herdar permissões de proprietário do recurso (console) ou rremover as permissões da política de acesso aos recursos da função do Lambda (API) .

A função < function-arn > se refere ao < resource-id > do recurso de Machine Learning sem permissão em ambos e no recurso. ResourceAccessPolicy OwnerSetting

Solução: você receberá esse erro se as permissões para o recurso de machine learning não estiverem configuradas para a função anexada do Lambda ou para o recurso. Para resolver esse problema, configure as permissões na ResourceAccessPolicypropriedade para a função Lambda ou na OwnerSettingpropriedade para o recurso.

A função < function-arn > se refere ao < resource-id do recurso do Machine Learning > com a permissão\ "rw\”, enquanto a configuração do proprietário do recurso permite GroupPermission apenas\ "ro\”.

Solução: você receberá esse erro se as permissões de acesso definidas para a função anexada do Lambda excederem as permissões de proprietário do recurso definidas para o recurso de machine

learning. Para resolver esse problema, defina permissões mais restritivas para a função do Lambda ou permissões menos restritivas para o proprietário do recurso.

NoContainer A função < function-arn > se refere aos recursos do caminho de destino aninhado.

Solução: você receberá esse erro se vários recursos de machine learning anexados a uma função não conteinerizada do Lambda utilizarem o mesmo caminho de destino ou um caminho de destino aninhado. Para resolver esse problema, especifique caminhos de destino separados para os recursos.

O Lambda < function-arn > obtém acesso ao < resource-id do recurso compartilhando o mesmo ID > de proprietário do grupo

Solução: você receberá esse erro em `runtime.log` se o mesmo grupo de sistemas operacionais for especificado como a identidade da função do Lambda Executar como e o proprietário do recurso de um recurso de machine learning, mas o recurso não estiver vinculado à função do Lambda. Essa configuração dá à função Lambda permissões implícitas que ela pode usar para acessar o recurso sem autorização. AWS IoT Greengrass

Para resolver esse problema, use um grupo de SO diferente para uma das propriedades ou anexe o recurso de machine learning à função do Lambda.

Consulte também

- Executar a inferência de machine learning
- the section called “Como configurar a inferência de machine learning”
- the section called “Como configurar a inferência otimizada de Machine Learning”
- AWS IoT Greengrass Version 1 API Reference

Como configurar a inferência de aprendizado de máquina usando o Console de gerenciamento da AWS

Para seguir as etapas deste tutorial, você precisa do AWS IoT Greengrass Core v1.10 ou posterior.

Você pode realizar inferência de machine learning (ML) localmente em um dispositivo de núcleo do Greengrass usando dados gerados localmente. Para obter mais informações, inclusive sobre os requisitos e restrições, consulte Executar a inferência de machine learning.

Este tutorial descreve como usar o Console de gerenciamento da AWS para configurar um grupo do Greengrass para executar um aplicativo de inferência Lambda que reconhece imagens de uma câmera localmente, sem enviar dados para a nuvem. O aplicativo de inferência acessa o módulo da câmera em um Raspberry Pi e executa a inferência usando o modelo de código aberto. SqueezeNet

O tutorial contém as seguintes etapas de nível elevado:

1. Configurar o Raspberry Pi
2. Instalar a estrutura de trabalho do MXNet.
3. Criar um pacote de modelo
4. Crie e publique uma função do Lambda
5. Adicionar a função do Lambda ao grupo
6. Adicionar recursos ao grupo
7. Adicionar uma assinatura ao grupo
8. Implantar o grupo
9. Testar o aplicativo

Pré-requisitos

Para concluir este tutorial, é necessário:

- Raspberry Pi 4 Model B ou Raspberry Pi 3 Model B/B +, configurado e configurado para uso com. AWS IoT Greengrass Para configurar seu Raspberry Pi com o AWS IoT Greengrass, execute o script Greengrass Device Setup ou conclua o Módulo 1 e o Módulo 2 do Conceitos básicos de AWS IoT Greengrass.

  > **Note**
  >
  > O Raspberry Pi pode exigir uma fonte de alimentação de 2,5 A para executar as estruturas de aprendizado profundo que normalmente são usadas para classificação de imagens. Uma fonte de alimentação com uma classificação mais baixa pode fazer com que o dispositivo seja reinicializado.

- Módulo de câmera Raspberry Pi V2 - 8 megapixels, 1080 p. Para obter informações sobre como configurar a câmera, consulte Conectar a câmera na documentação do Raspberry Pi.
- Um grupo do Greengrass e um núcleo do Greengrass. Para obter informações sobre como criar um grupo ou núcleo do Greengrass, consulte Conceitos básicos de AWS IoT Greengrass.

> **Note**
>
> Este tutorial usa um Raspberry Pi, mas AWS IoT Greengrass oferece suporte a outras plataformas, como Intel Atom e NVIDIA Jetson TX2. No exemplo para Jetson TX2, você pode usar imagens estáticas em vez de imagens transmitidas por uma câmera. Se estiver usando o exemplo do Jetson TX2, talvez seja necessário instalar o Python 3.6 em vez do Python 3.7. Para obter informações sobre como configurar seu dispositivo para que você possa instalar o software AWS IoT Greengrass Core, consultethe section called “Configurar outros dispositivos”.
>
> Para plataformas de terceiros que AWS IoT Greengrass não oferecem suporte, você deve executar sua função do Lambda no modo não conteinerizado. Para executar no modo sem contêiner, você deve executar sua função do Lambda como root. Para obter mais informações, consulte the section called “Considerações ao escolher a conteinerização de função do Lambda” e the section called “Definir a identidade de acesso padrão para as funções do Lambda em um grupo”.

Etapa 1: Configurar o Raspberry Pi

Nesta etapa, instale as atualizações no sistema operacional Raspbian, instale o software do módulo da câmera e as dependências do Python e ative a interface da câmera.

Execute os seguintes comandos no seu terminal do Raspberry Pi.

1. Instale as atualizações no Raspbian.

   ```
   sudo apt-get update
   sudo apt-get dist-upgrade
   ```

2. Instale a interface do `picamera` para o módulo da câmera e outras bibliotecas do Python que são necessárias para este tutorial.

```
sudo apt-get install -y python3-dev python3-setuptools python3-pip python3-picamera
```

Valide a instalação:

- Verifique se a instalação do Python 3.7 inclui o pip.

  ```
  python3 -m pip
  ```

  Se o pip não estiver instalado, faça download dele no site do pip e execute o seguinte comando.

  ```
  python3 get-pip.py
  ```

- Verifique se a versão do Python é 3.7 ou superior.

  ```
  python3 --version
  ```

  Se a saída listar uma versão anterior, execute o seguinte comando.

  ```
  sudo apt-get install -y python3.7-dev
  ```

- Verifique se o Setuptools e Picamera foram instalados com êxito.

  ```
  sudo -u ggc_user bash -c 'python3 -c "import setuptools"'
  sudo -u ggc_user bash -c 'python3 -c "import picamera"'
  ```

  Se a saída não contiver erros, a validação será bem-sucedida.

> **Note**
>
> Se o Python executável instalado no dispositivo for o `python3.7`, use `python3.7` em vez de `python3` para os comandos neste 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 Raspberry Pi.

```
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. Use o seguinte comando 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.

Etapa 2: Instalar a estrutura de trabalho do MXNet

Nesta etapa, instale as bibliotecas MXNet no seu Raspberry Pi.

1. Inicie sessão no seu Raspberry Pi remotamente.

   ```
   ssh pi@your-device-ip-address
   ```

2. Abra a documentação do MXNet, abra Installing MXNet e siga as instruções para instalar o MXNet no dispositivo.

   > **Note**
   >
   > Recomendamos instalar a versão 1.5.0 e criar o MXNet a partir da fonte para este tutorial para evitar conflitos de dispositivos.

3. Depois de instalar o MXNet, valide a seguinte configuração:

   - Verifique se a conta do sistema `ggc_user` pode usar a estrutura de trabalho do MXNet.

     ```
     sudo -u ggc_user bash -c 'python3 -c "import mxnet"'
     ```

   - Verifique se NumPy está instalado.

```
sudo -u ggc_user bash -c 'python3 -c "import numpy"'
```

Etapa 3: Criar um pacote de modelo do MXNet

Nesta etapa, crie um pacote de modelo que contenha uma amostra de modelo MXNet pré-treinado para carregar no Amazon Simple Storage Service (Amazon S3). AWS IoT Greengrass pode usar um pacote modelo do Amazon S3, desde que você use o formato tar.gz ou zip.

1. No seu computador, faça download do exemplo do MXNet para Raspberry Pi em the section called “Exemplos de machine learning”.
2. Não descompacte o arquivo `mxnet-py3-armv7l.tar.gz` obtido por download.
3. Navegue até o diretório `squeezenet`.

   ```
   cd path-to-downloaded-sample/mxnet-py3-armv7l/models/squeezenet
   ```

   O arquivo `squeezenet.zip` neste diretório é o pacote de modelo. Ele contém artefatos de modelo de código SqueezeNet aberto para um modelo de classificação de imagens. Posteriormente, esse pacote de modelo é carregado no Amazon S3.

Etapa 4: Crie e publique uma função do Lambda

Nesta etapa, crie um pacote de implantação da função do Lambda e uma função do Lambda. Depois, publique uma versão da função e crie um alias.

Primeiro, crie o pacote de implantação da função do Lambda.

1. No computador, navegue até o diretório `examples` no pacote de exemplo que você descompactou em the section called “Criar um pacote de modelo”.

   ```
   cd path-to-downloaded-sample/mxnet-py3-armv7l/examples
   ```

   O diretório `examples` contém código de função e dependências.

   - O `greengrassObjectClassification.py` é o código de inferência usado neste tutorial. Você pode usar esse código como modelo para criar sua própria função de inferência.
   - `greengrasssdk` é a versão 1.5.0 do SDK do AWS IoT Greengrass Core para Python.

> **Note**
>
> Se uma nova versão estiver disponível, você poderá fazer download dela e atualizar a versão do SDK no seu pacote de implantação. Para obter mais informações, consulte AWS IoT Greengrass Core SDK para Python on GitHub.

2. Compacte o conteúdo do diretório `examples` em um arquivo chamado `greengrassObjectClassification.zip`. Esse é o pacote de implantação.

```
zip -r greengrassObjectClassification.zip .
```

> **Note**
>
> Verifique se os arquivos `.py` e as dependências estão na raiz do diretório.

Depois, crie a função do Lambda .

3. No AWS IoT console, escolha Funções e Criar função.
4. Selecione Criar do zero e use os valores a seguir para criar a função:

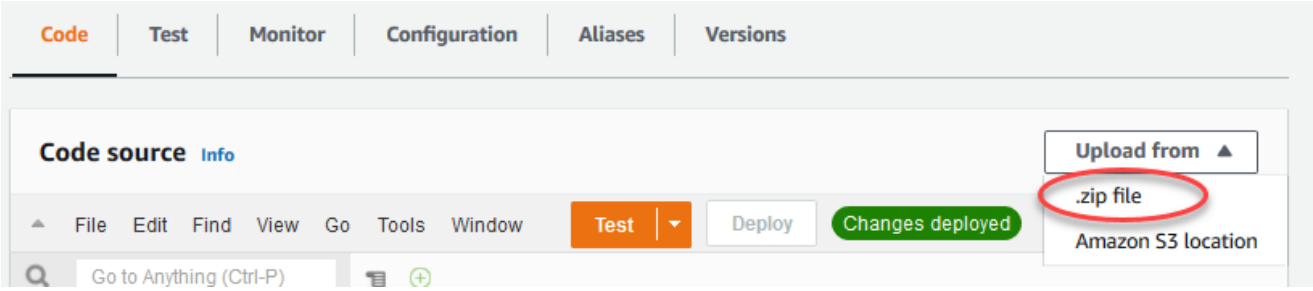
   - Em Function name (Nome da função), insira **`greengrassObjectClassification`**.
   - Em Runtime (Tempo de execução), selecione Python 3.7.

   Em Permissões, mantenha a configuração padrão. Isso cria uma função de execução que concede permissões básicas do Lambda. Essa função não é usada por AWS IoT Greengrass.
5. Selecione Criar função.

Agora, carregue o seu pacote de implantação da função do Lambda e registre o manipulador.

6. Selecione sua função do Lambda e carregue seu pacote de implantação da função do Lambda.

   a. Na guia Código, em Fonte do código, selecione Fazer upload a partir de. No menu suspenso, selecione o arquivo .zip.



b. Selecione Upload e, em seguida, selecione seu pacote de implantação `greengrassObjectClassification.zip`. Selecione Salvar.

c. Na guia Código da função, em Configurações de runtime, selecione Editar e insira os valores a seguir.

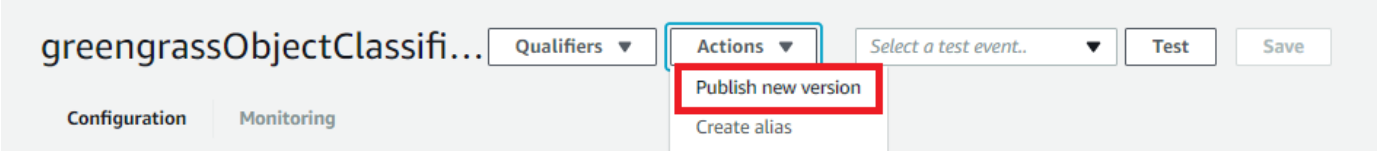
   - Em Runtime (Tempo de execução), selecione Python 3.7.
   - Em Handler (Manipulador), insira **`greengrassObjectClassification.function_handler`**.

   Selecione Salvar.

Em seguida, publique a primeira versão da sua função do Lambda. Em seguida, crie um alias para a versão.

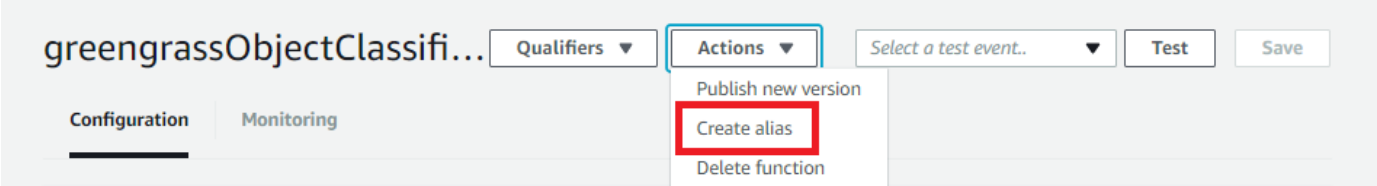
> **Note**
>
> Os grupos do Greengrass podem fazer referência a uma função do Lambda por alias (recomendado) ou por versão. Usar um alias facilita o gerenciamento de atualizações de código porque você não precisa alterar a tabela de assinaturas nem a definição do grupo ao atualizar a função do código. Em vez disso, você pode simplesmente apontar o alias para a nova versão da função.

7. No menu Actions, selecione Publish new version.



8. Em Version description (Descrição da versão), insira **`First version`** e, em seguida, selecione Publish (Publicar).

9. Na página de configuração greengrassObjectClassification: 1, no menu Ações, escolha Criar alias.



10. Na página Create a new alias, use os seguintes valores:

    - Em Nome, digite **`mlTest`**.
    - Em Version (Versão), insira **`1`**.

> **Note**
>
> AWS IoT Greengrass não oferece suporte a aliases do Lambda para as versões $LATEST.

11. Selecione Salvar.

Agora, adicione a função do Lambda ao seu grupo do Greengrass.

Etapa 5: adicionar a função do Lambda ao grupo do Greengrass

Nesta etapa, adicione a função do Lambda ao grupo e configura o ciclo de vida e as variáveis de ambiente.

Primeiro, adicione a função do Lambda ao seu grupo do Greengrass.

1. No painel de navegação do AWS IoT console, em Gerenciar, expanda dispositivos Greengrass e escolha Grupos (V1).
2. Na página de configuração do grupo, selecione a guia Funções do Lambda.
3. Na seção Minhas funções do Lambda, selecione Adicionar.
4. Para a função Lambda, escolha ObjectClassification greengrass.
5. Para a versão da função Lambda, escolha Alias:mlTest.

Em seguida, configure o ciclo de vida e as variáveis de ambiente da função do Lambda.

6. Na seção Configuração da função do Lambda, faça as atualizações a seguir.

> **Note**
>
> Recomendamos que você execute sua função do Lambda sem conteinerização, a menos que seu caso de negócios faça essa exigência. Isso ajuda a habilitar o acesso à GPU e à câmera do seu dispositivo sem configurar os recursos do dispositivo. Se você executar sem conteinerização, também deverá conceder acesso root às suas funções do AWS IoT Greengrass Lambda.

a. Para executar sem conteinerização:

- Para Usuário e grupo do sistema, selecione **`Another user ID/group ID`**. Para ID de usuário do sistema, insira **`0`**. Para ID do grupo do sistema, insira **`0`**.

  Isso permite que sua função do Lambda seja executada como root. Para obter mais informações sobre como executar como raiz, consulte the section called “Definir a identidade de acesso padrão para as funções do Lambda em um grupo”.

  > **Tip**
  >
  > Você também deve atualizar seu arquivo `config.json` para conceder acesso root à sua função do Lambda. Para o procedimento, consulte the section called “Executar uma função do Lambda como raiz”.

- Para conteinerização da função do Lambda, selecione Sem contêiner.

  Para obter mais informações sobre como executar sem conteinerização, consulte the section called “Considerações ao escolher a conteinerização de função do Lambda”.

- Em Timeout (Tempo limite), insira **`10 seconds`**.
- Para Fixado, selecione Verdadeiro.

  Para obter mais informações, consulte the section called “Configuração do ciclo de vida”.

b. Em vez disso, para executar no modo conteinerizado:

> **Note**
>
> Não recomendamos a execução no modo conteinerizado, a menos que seu caso de negócios faça essa exigência.

- Para Usuário e grupo do sistema, selecione Usar padrão de grupo.
- Para Conteinerização da função do Lambda, selecione Usar padrão de grupo.
- Em Memory limit (Limite de memória), insira **`96 MB`**.
- Em Timeout (Tempo limite), insira **`10 seconds`**.
- Para Fixado, selecione Verdadeiro.

Para obter mais informações, consulte the section called “Configuração do ciclo de vida”.

7. Em Environment variables (Variáveis de ambiente), crie um par de chave/valor. Um par de chave-valor é exigido por funções que interagem com modelos do MXNet em um Raspberry Pi.

   Na chave, use MXNET_ENGINE_TYPE. Para o valor, use NaiveEngine.

   > **Note**
   >
   > Se preferir, nas suas próprias funções do Lambda definidas pelo usuário, é possível definir a variável de ambiente no código da função.

8. Mantenha os valores padrão para todas as outras propriedades e selecione Adicionar função do Lambda.

Etapa 6: Adicionar recursos ao grupo do Greengrass

Nesta etapa, crie recursos para o módulo da câmera e o modelo de inferência de ML e afilie os recursos com a função do Lambda. Isso possibilita que a função do Lambda acesse os recursos no dispositivo básico.

> **Note**
>
> Se você executar no modo não conteinerizado, AWS IoT Greengrass poderá acessar a GPU e a câmera do seu dispositivo sem configurar esses recursos do dispositivo.

Primeiro, crie dois recursos de dispositivos locais para a câmera: um para memória compartilhada e outro para a interface do dispositivo. Para obter mais informações sobre acesso a recursos locais, consulte Acesso aos recursos locais com funções e conectores do Lambda.

1. Na página de configuração do grupo, selecione a guia Recursos.
2. Na seção Recursos locais, selecione Adicionar recurso local.
3. Na página Adicionar recurso local, use os seguintes valores:

   - Em Resource Name (Nome do recurso), insira **`videoCoreSharedMemory`**.
   - Em Resource type, selecione Device.
   - Em Caminho do dispositivo local, insira **`/dev/vcsm`**.

     O caminho do dispositivo é o caminho absoluto local do recurso do dispositivo. Este caminho só pode se referir a um dispositivo de caractere ou dispositivo de blocos em `/dev`.
   - Em Permissões do proprietário do grupo do sistema e de acesso a arquivos, selecione Adicionar automaticamente permissões de sistema de arquivos do grupo do sistema que é proprietário do recurso.

     A opção Proprietário do grupo do sistema e permissões de acesso ao arquivo permite conceder permissões adicionais de acesso a arquivos para o processo do Lambda. Para obter mais informações, consulte Permissão de acesso a arquivo do proprietário do grupo.
4. Em seguida, você adiciona um recurso de dispositivo local para a interface da câmera.
5. Selecione Adicionar recurso local.
6. Na página Adicionar recurso local, use os seguintes valores:

   - Em Resource Name (Nome do recurso), insira **`videoCoreInterface`**.
   - Em Resource type, selecione Device.
   - Em Caminho do dispositivo local, insira **`/dev/vchiq`**.
   - Em Permissões do proprietário do grupo do sistema e de acesso a arquivos, selecione Adicionar automaticamente permissões de sistema de arquivos do grupo do sistema que é proprietário do recurso.
7. Na parte inferior da página, selecione Adicionar recurso.

Agora, adicione o modelo de inferência como um recurso de machine learning. Esta etapa inclui o upload do pacote de modelo `squeezenet.zip` para o Amazon S3.

1. Na guia Recursos do seu grupo, na seção Machine Learning, selecione Adicionar recurso de machine learning.
2. Na página Adicionar recurso de machine learning, em Nome do recurso, insira **`squeezenet_model`**.
3. Em Fonte do modelo, selecione Usar um modelo armazenado no S3, como um modelo otimizado por meio do compilador de aprendizado profundo.
4. Para S3 URI, insira um caminho em que o bucket do S3 seja salvo.
5. Selecione Browse S3 (Navegar no S3). Isso abre uma nova guia no console do Amazon S3.
6. Na guia do console do Amazon S3, faça upload do arquivo `squeezenet.zip` em um bucket do S3. Para obter informações, consulte Como carregar arquivos e pastas em um bucket do S3? no Guia do usuário do Amazon Simple Storage Service.

   > **Note**
   >
   > Para que o bucket do S3 seja acessível, o nome do bucket deve conter a string **`greengrass`** e o bucket deve estar na mesma região em que você usa o AWS IoT Greengrass. Selecione um nome exclusivo (como **`greengrass-bucket-`*`user-id`*`-`*`epoch-time`***). Não use ponto (`.`) no nome do bucket.

7. Na guia do AWS IoT Greengrass console, localize e escolha seu bucket do S3. Localize o arquivo `squeezenet.zip` carregado e, em seguida selecione Select (Selecionar). Talvez seja necessário escolher Refresh (Atualizar) para atualizar a lista de buckets e arquivos disponíveis.
8. Para Destination path (Caminho de destino), insira **`/greengrass-machine-learning/mxnet/squeezenet`**.

   Este é o destino do modelo local no namespace de runtime do Lambda. Quando você implanta o grupo, AWS IoT Greengrass recupera o pacote do modelo de origem e, em seguida, extrai o conteúdo para o diretório especificado. O exemplo da função do Lambda para este tutorial já está configurado para usar este caminho (na variável `model_path`).
9. Em Proprietário do grupo do sistema e permissões de acesso ao arquivo, selecione Sem grupo do sistema.
10. Selecione Adicionar recurso.

Usando modelos treinados em SageMaker IA

Este tutorial usa um modelo armazenado no Amazon S3, mas você também pode usar modelos de SageMaker IA com facilidade. O AWS IoT Greengrass console tem integração de SageMaker IA integrada, então você não precisa fazer o upload manual desses modelos para o Amazon S3. Para ver os requisitos e limitações do uso de modelos de SageMaker IA, consultethe section called "Fontes de modelo compatíveis".

Para usar um modelo de SageMaker IA:

- Em Fonte do modelo, escolha Usar um modelo treinado em AWS SageMaker IA e, em seguida, escolha o nome do trabalho de treinamento do modelo.
- Em Caminho de destino, insira o caminho para o diretório em que sua função do Lambda procura o modelo.

Etapa 7: Adicionar uma assinatura ao grupo do Greengrass

Nesta etapa, adicione uma assinatura ao grupo. Essa assinatura permite que a função Lambda envie resultados de previsão AWS IoT publicando em um tópico do MQTT.

1. Na página de configuração do grupo, selecione a guia Assinaturas e, em seguida, Adicionar assinatura.
2. Na página de Detalhes da assinatura, configure a origem e o destino, da seguinte forma:
   a. Em Tipo de origem, escolha Função Lambda e, em seguida, escolha ObjectClassification greengrass.
   b. Em Tipo de destino, selecione Serviços e, em seguida, IoT Cloud.
3. Em Filtro de tópicos, insira **`hello/world`** e, em seguida, selecione Criar assinatura.

Etapa 8: Implantar o grupo do Greengrass

Nesta etapa, implante a versão atual da definição do grupo ao dispositivo de núcleo do Greengrass. A definição contém a função do Lambda, recursos e configurações de inscrição que você adicionou.

1. Certifique-se de que o AWS IoT Greengrass núcleo esteja funcionando. Execute os seguintes comandos no seu terminal do Raspberry Pi, conforme necessário.
   a. Para verificar se o daemon está em execução:

```
ps aux | grep -E 'greengrass.*daemon'
```

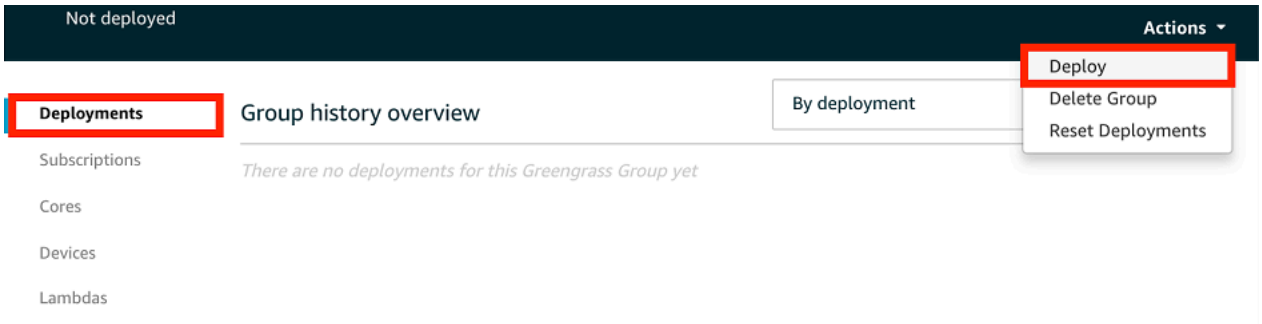
Se a saída contém uma entrada `root` para `/greengrass/ggc/packages/1.11.6/bin/daemon`, o daemon está em execução.

> **Note**
>
> A versão no caminho depende da versão do software AWS IoT Greengrass Core instalada em seu dispositivo principal.

b. Para iniciar o daemon:

```
cd /greengrass/ggc/core/
sudo ./greengrassd start
```

2. Na página de configuração do grupo, selecione Implantar.



3. Na guia Funções do Lambda, na seção Funções do sistema Lambda, selecione Detector de IP e, em seguida selecione Editar.
4. Na caixa de diálogo Editar configurações do detector IP, selecione Detectar e substituir automaticamente os endpoints do corretor MQTT.
5. Selecione Salvar.

Isso permite que os dispositivos adquiram automaticamente as informações de conectividade para o núcleo, como endereço IP, DNS e o número da porta. A detecção automática é recomendada, mas AWS IoT Greengrass também oferece suporte a endpoints especificados manualmente. Você só é solicitado pelo método de descoberta na primeira vez em que o grupo for implantado.

> **Note**
>
> Se solicitado, conceda permissão para criar a função de serviço do Greengrass e associá-la à sua Conta da AWS na atual. Região da AWS Essa função permite AWS IoT Greengrass acessar seus recursos em AWS serviços.

A página Deployments mostra a data e hora, ID de versão e status da implantação. Quando concluída, o status exibido para a implantação deve ser Concluída.

Para obter mais informações sobre implantações, consulte Implemente AWS IoT Greengrass grupos. Para obter ajuda sobre a solução de problemas, consulte Solução de problemas.

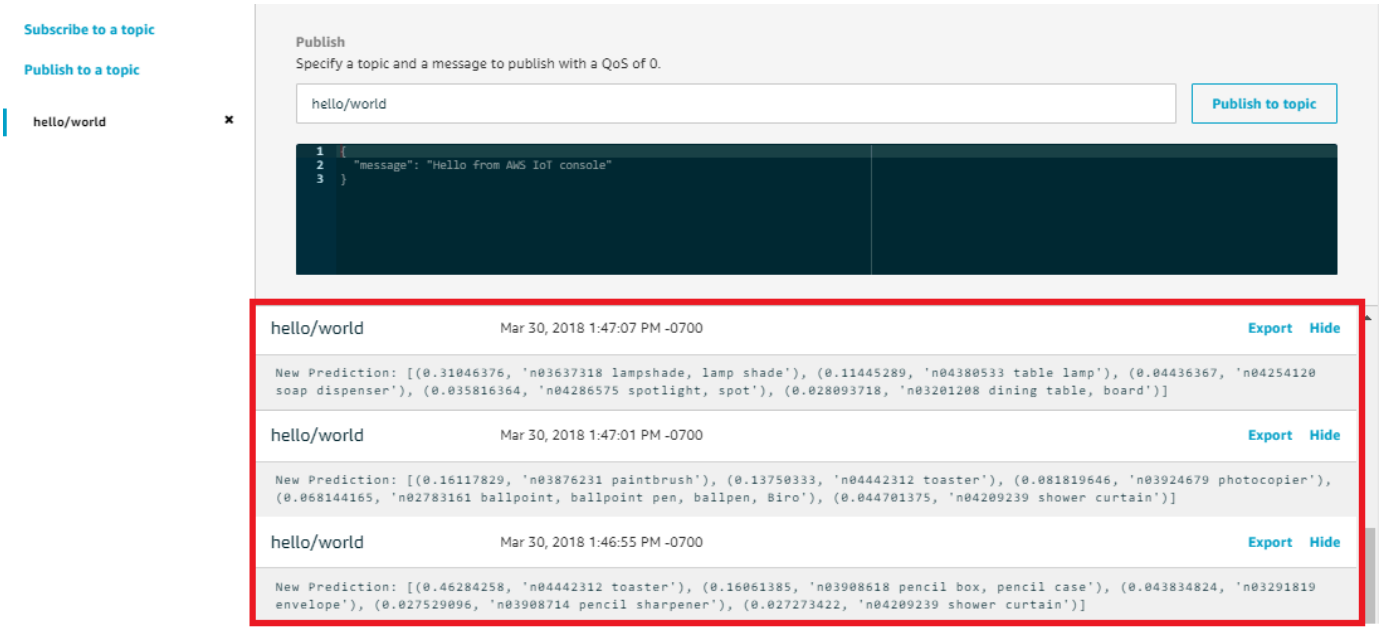
Etapa 9: Testar o aplicativo de inferência

Agora você pode verificar se a implementação foi configurada corretamente. Para testar, inscreva-se no tópico `hello/world` e visualize os resultados de previsões publicados pela função do Lambda.

> **Note**
>
> Se um monitor estiver conectado ao Raspberry Pi, o feed da câmera ao vivo será exibido em uma janela de visualização.

1. No AWS IoT console, em Teste, escolha Cliente de teste MQTT.
2. Para Subscriptions, use os seguintes valores:
   - Para o tópico de assinatura, use hello/world.
   - Em Configuração adicional, para Exibição da carga útil do MQTT, selecione Exibir cargas úteis como strings.
3. Selecione Subscribe.

   Se o teste for bem-sucedido, as mensagens da função do função do Lambda serão exibidas na parte inferior da página. Cada mensagem contém os cinco principais resultados de previsão da imagem, usando o formato: probabilidade, ID da classe prevista e nome da classe correspondente.



Solução de problemas AWS IoT Greengrass Inferência de ML

Se o teste não for bem-sucedido, você poderá tentar as etapas de solução de problemas a seguir. Execute os comandos no seu terminal Raspberry Pi.

Verificar logs de erros

1. Alterne para o usuário raiz e navegue até o diretório `log`. O acesso aos AWS IoT Greengrass registros requer permissões de root.

   ```
   sudo su
   cd /greengrass/ggc/var/log
   ```

2. No diretório `system`, verifique `runtime.log` ou `python_runtime.log`.

   No diretório `user/`*`region`*`/`*`account-id`*, verifique `greengrassObjectClassification.log`.

   Para obter mais informações, consulte the section called “Solução de problemas com logs”.

Erro "Unpacking" em runtime.log

Se `runtime.log` contiver um erro semelhante ao seguinte, verifique se o pacote de modelos de origem `tar.gz` tem um diretório pai.

```
Greengrass deployment error: unable to download the artifact model-arn: Error while
 processing.
Error while unpacking the file from /tmp/greengrass/artifacts/model-arn/path to /
greengrass/ggc/deployment/path/model-arn,
error: open /greengrass/ggc/deployment/path/model-arn/squeezenet/
squeezenet_v1.1-0000.params: no such file or directory
```

Se seu pacote não tiver um diretório pai que contenha os arquivos de modelo, use o comando a seguir para tentar empacotar o modelo novamente:

```
tar -zcvf model.tar.gz ./model
```

Por exemplo:

```
#$ tar -zcvf test.tar.gz ./test
./test
./test/some.file
./test/some.file2
./test/some.file3
```

> **Note**
>
> Não inclui caracteres `/*` finais neste comando.

Verifique se a função do Lambda é implantada com sucesso

1. Liste o conteúdo do Lambda implementado no diretório `/lambda`. Substitua os valores de espaço reservado antes de executar o comando.

   ```
   cd /greengrass/ggc/deployment/lambda/
   arn:aws:lambda:region:account:function:function-name:function-version
   ls -la
   ```

2. Verifique se o diretório contém o mesmo conteúdo do pacote de implantação `greengrassObjectClassification.zip` que você carregou no Etapa 4: Crie e publique uma função do Lambda.

Verifique se os arquivos `.py` e as dependências estão na raiz do diretório.

Verifique se o modelo de inferência é implantado com sucesso

1. Encontre o número de identificação de processo (PID) do runtime do Lambda:

   ```
   ps aux | grep 'lambda-function-name*'
   ```

   Na saída, o PID aparece na segunda coluna da linha para o processo de runtime do Lambda.

2. Digite o namespace de runtime do Lambda. Certifique-se de substituir o *pid* valor do espaço reservado antes de executar o comando.

   > **Note**
   >
   > Este diretório e seu conteúdo estão no namespace de runtime do Lambda para que não sejam visíveis em um namespace comum do Linux.

   ```
   sudo nsenter -t pid -m /bin/bash
   ```

3. Liste o conteúdo do diretório local que você especificou para o recurso de ML.

   ```
   cd /greengrass-machine-learning/mxnet/squeezenet/
   ls -ls
   ```

   Você deve ver os arquivos a seguir:

   ```
   32 -rw-r--r-- 1 ggc_user ggc_group   31675 Nov 18 15:19 synset.txt
   32 -rw-r--r-- 1 ggc_user ggc_group   28707 Nov 18 15:19 squeezenet_v1.1-symbol.json
   4832 -rw-r--r-- 1 ggc_user ggc_group 4945062 Nov 18 15:19
    squeezenet_v1.1-0000.params
   ```

Próximas etapas

Em seguida, explore outros aplicativos de inferência. AWS IoT Greengrass fornece outras funções do Lambda que você pode usar para testar a inferência local. Você pode encontrar o pacote de exemplos na pasta de bibliotecas pré-compiladas que você obteve por download the section called "Instalar a estrutura de trabalho do MXNet.".

Como configurar um Intel Atom

Para executar este tutorial em um dispositivo Intel Atom, você deve fornecer imagens de origem, configurar a função do Lambda e adicionar outro recurso de dispositivo local. Para usar a GPU para inferência, verifique se o software a seguir está instalado no dispositivo:

- OpenCL versão 1.0 ou posterior
- Python 3.7 e pip

> **Note**
>
> Se o dispositivo for pré-construído com o Python 3.6, você poderá criar um symlink para o Python 3.7. Para obter mais informações, consulte Step 2.

- NumPy
- OpenCV sobre rodas

1. Faça download de imagens PNG ou JPG estáticas para a função do Lambda para usá-las na classificação de imagens. O exemplo funciona melhor com arquivos de imagem pequenos.

   Salve os arquivos de imagem no diretório que contém o arquivo `greengrassObjectClassification.py` (ou em um subdiretório desse diretório). Isso está no pacote de implantação da função do Lambda do qual você fez upload em the section called "Crie e publique uma função do Lambda".

   > **Note**
   >
   > Se estiver usando AWS DeepLens, você pode usar a câmera integrada ou montar sua própria câmera para realizar inferências em imagens capturadas em vez de imagens estáticas. No entanto, recomendamos que você comece com imagens estáticas primeiro.

Se utilizar uma câmara, verifique se o pacote APT `awscam` está instalado e atualizado. Para mais informações, consulte Atualizar seu dispositivo AWS DeepLens no Guia do desenvolvedor AWS DeepLens .

2. Se você não estiver usando o Python 3.7, crie um symlink do Python 3.x para o Python 3.7. Isso configura seu dispositivo para usar o Python 3 com. AWS IoT Greengrass Execute o seguinte comando para localizar a instalação do Python:

```
which python3
```

Execute o comando a seguir para criar o symlink:

```
sudo ln -s path-to-python-3.x/python3.x path-to-python-3.7/python3.7
```

Reinicie o dispositivo.

3. Edite a configuração da função do Lambda. Siga o procedimento em the section called "Adicionar a função do Lambda ao grupo".

> **Note**
>
> Recomendamos que você execute sua função do Lambda sem conteinerização, a menos que seu caso de negócios faça essa exigência. Isso ajuda a habilitar o acesso à GPU e à câmera do seu dispositivo sem configurar os recursos do dispositivo. Se você executar sem conteinerização, também deverá conceder acesso root às suas funções do AWS IoT Greengrass Lambda.

a. Para executar sem conteinerização:

- Para Usuário e grupo do sistema, selecione **`Another user ID/group ID`**. Para ID de usuário do sistema, insira **`0`**. Para ID do grupo do sistema, insira **`0`**.

  Isso permite que sua função do Lambda seja executada como root. Para obter mais informações sobre como executar como raiz, consulte the section called "Definir a identidade de acesso padrão para as funções do Lambda em um grupo".

> **Tip**
>
> Você também deve atualizar seu arquivo `config.json` para conceder acesso root à sua função do Lambda. Para o procedimento, consulte the section called "Executar uma função do Lambda como raiz".

- Para Conteinerização da função do Lambda, selecione Sem contêiner.

  Para obter mais informações sobre como executar sem conteinerização, consulte the section called "Considerações ao escolher a conteinerização de função do Lambda".
- Atualize o valor de Tempo limite para 5 segundos. Isso garante que a solicitação não expire muito cedo. Leva alguns minutos para que a inferência seja executada após a configuração.
- Em Fixado, selecione Verdadeiro.
- Em Parâmetros adicionais, para Acesso de leitura ao diretório /sys), selecione Habilitado.
- Em Lambda lifecycle (Ciclo de vida Lambda), selecione Make this function long-lived and keep it running indefinitely (Definir esta função de longa duração e mantê-la em execução indefinidamente).

b. Em vez disso, para executar no modo conteinerizado:

> **Note**
>
> Não recomendamos a execução no modo conteinerizado, a menos que seu caso de negócios faça essa exigência.

- Atualize o valor de Tempo limite para 5 segundos. Isso garante que a solicitação não expire muito cedo. Leva alguns minutos para que a inferência seja executada após a configuração.
- Para Fixado, selecione Verdadeiro.
- Em Parâmetros adicionais, para Acesso de leitura ao diretório /sys, selecione Habilitado.

4. Se estiver executando no modo conteinerizado, adicione o recurso de dispositivo local necessário para conceder acesso à GPU do seu dispositivo.

> **Note**
>
> Se você executar no modo não conteinerizado, AWS IoT Greengrass poderá acessar a GPU do seu dispositivo sem configurar os recursos do dispositivo.

a. Na página de configuração do grupo, selecione a guia Recursos.

b. Selecione Adicionar recurso local.

c. Defina o recurso:

- Em Resource Name (Nome do recurso), insira **`renderD128`**.
- Em Tipo de recurso, selecione Dispositivo local.
- Em Device path (Caminho do dispositivo), insira **`/dev/dri/renderD128`**.
- Em Permissões do proprietário do grupo do sistema e de acesso a arquivos, selecione Adicionar automaticamente permissões de sistema de arquivos do grupo do sistema que é proprietário do recurso.
- Para Afiliações da função do Lambda, conceda a permissão Acesso para leitura e gravação para sua função Lambda.

Configuração NVIDIA Jetson TX2

Para executar este tutorial em um NVIDIA Jetson TX2, forneça imagens de origem e configure a função do Lambda. Se estiver usando a GPU, você também deverá adicionar recursos do dispositivo local.

1. Certifique-se de que seu dispositivo Jetson esteja configurado para que você possa instalar o software AWS IoT Greengrass Core. Para obter mais informações sobre como configurar seu projeto, consulte the section called “Configurar outros dispositivos”.
2. Abra a documentação do MXNet, vá para Installing MXNet on a Jetson e siga as instruções para instalar o MXNet no dispositivo Jetson.

> **Note**
>
> Se você quiser criar o MXNet do código-fonte, siga as instruções para criar a biblioteca compartilhada. Edite as seguintes configurações no seu arquivo `config.mk` para trabalhar com um dispositivo Jetson TX2:
>
> - Adicione `-gencode arch=compute-62, code=sm_62` à `CUDA_ARCH` configuração.
> - Ligue o CUDA.
>
> ```
> USE_CUDA = 1
> ```

3. Faça download de imagens PNG ou JPG estáticas para a função do Lambda para usá-las na classificação de imagens. O aplicativo funciona melhor com arquivos de imagem pequenos. Se preferir, é possível instrumentar uma câmera na placa Jetson para capturar as imagens de origem.

   Salve seus arquivos de imagem no diretório que contém o arquivo `greengrassObjectClassification.py`. Você também pode salvá-los em um subdiretório desse diretório. Esse diretório está no pacote de implantação da função do Lambda do qual você fez upload em the section called “Crie e publique uma função do Lambda”.

4. Crie um link simbólico do Python 3.7 para o Python 3.6 para usar o Python 3 com. AWS IoT Greengrass Execute o seguinte comando para localizar a instalação do Python:

   ```
   which python3
   ```

   Execute o comando a seguir para criar o symlink:

   ```
   sudo ln -s path-to-python-3.6/python3.6 path-to-python-3.7/python3.7
   ```

   Reinicie o dispositivo.

5. Verifique se a conta do sistema `ggc_user` pode usar a estrutura de trabalho do MXNet:

   ```
   “sudo -u ggc_user bash -c 'python3 -c "import mxnet"'
   ```

6. Edite a configuração da função do Lambda. Siga o procedimento em the section called “Adicionar a função do Lambda ao grupo”.

> **Note**
>
> Recomendamos que você execute sua função do Lambda sem conteinerização, a menos que seu caso de negócios faça essa exigência. Isso ajuda a habilitar o acesso à GPU e à câmera do seu dispositivo sem configurar os recursos do dispositivo. Se você executar sem conteinerização, também deverá conceder acesso root às suas funções do AWS IoT Greengrass Lambda.

a. Para executar sem conteinerização:

- Para Usuário e grupo do sistema, selecione **`Another user ID/group ID`**. Para ID de usuário do sistema, insira **`0`**. Para ID do grupo do sistema, insira **`0`**.

  Isso permite que sua função do Lambda seja executada como root. Para obter mais informações sobre como executar como raiz, consulte the section called “Definir a identidade de acesso padrão para as funções do Lambda em um grupo”.

  > **Tip**
  >
  > Você também deve atualizar seu arquivo `config.json` para conceder acesso root à sua função do Lambda. Para o procedimento, consulte the section called “Executar uma função do Lambda como raiz”.

- Para conteinerização da função do Lambda, selecione Sem contêiner.

  Para obter mais informações sobre como executar sem conteinerização, consulte the section called “Considerações ao escolher a conteinerização de função do Lambda”.

- Em Parâmetros adicionais, para Acesso de leitura ao diretório /sys, selecione Habilitado.
- Em Variáveis de ambiente, adicione os seguintes pares chave-valor à sua função do Lambda. Isso é configurado AWS IoT Greengrass para usar a estrutura MXNet.

Chave	Valor
PATH	/usr/local/cuda/bin: $PATH
MXNET_HOME	$HOME/mxnet/

Chave	Valor
PYTHONPATH	$MXNET_HOME/python: $PYTHONPATH
CUDA_HOME	/usr/local/cuda
LD_LIBRARY_PATH	$LD_LIBRARY_PATH:${CUDA_HOME}/lib64

b. Em vez disso, para executar no modo conteinerizado:

> **Note**
>
> Não recomendamos a execução no modo conteinerizado, a menos que seu caso de negócios faça essa exigência.

- Aumente o valor de Memory limit (Limite de memória). Use 500 MB para a CPU ou pelo menos 2000 MB para a GPU.
- Em Parâmetros adicionais, para Acesso de leitura ao diretório /sys, selecione Habilitado.
- Em Variáveis de ambiente, adicione os seguintes pares chave-valor à sua função do Lambda. Isso é configurado AWS IoT Greengrass para usar a estrutura MXNet.

Chave	Valor
PATH	/usr/local/cuda/bin: $PATH
MXNET_HOME	$HOME/mxnet/
PYTHONPATH	$MXNET_HOME/python: $PYTHONPATH
CUDA_HOME	/usr/local/cuda
LD_LIBRARY_PATH	$LD_LIBRARY_PATH:${CUDA_HOME}/lib64

7. Se estiver executando no modo conteinerizado, adicione os recursos de dispositivo local a seguir para conceder acesso à GPU do seu dispositivo. Siga o procedimento em the section called “Adicionar recursos ao grupo”.

   > **Note**
   >
   > Se você executar no modo não conteinerizado, AWS IoT Greengrass poderá acessar a GPU do seu dispositivo sem configurar os recursos do dispositivo.

   Para cada recurso:

   - Em Resource type, selecione Device.
   - Em Permissões do proprietário do grupo do sistema e de acesso a arquivos, selecione Adicionar automaticamente permissões de sistema de arquivos do grupo do sistema que é proprietário do recurso.

Nome	Caminho do dispositivo
nvhost-ctrl	/dev/nvhost-ctrl
nvhost-gpu	/dev/nvhost-gpu
nvhost-ctrl-gpu	/dev/nvhost-ctrl-gpu
nvhost-dbg-gpu	/dev/nvhost-dbg-gpu
nvhost-prof-gpu	/dev/nvhost-prof-gpu
nvmap	/dev/nvmap
nvhost-vic	/dev/nvhost-vitória
tegra_dc_ctrl	/dev/tegra_dc_ctrl

8. Se estiver executando no modo conteinerizado, adicione o recurso de volume local a seguir para conceder acesso à câmera do seu dispositivo. Siga o procedimento em the section called “Adicionar recursos ao grupo”.

> **Note**
>
> Se você executar no modo não conteinerizado, AWS IoT Greengrass poderá acessar a câmera do seu dispositivo sem configurar os recursos de volume.

- Em Tipo de recurso, selecione Volume.
- Em Permissões do proprietário do grupo do sistema e de acesso a arquivos, selecione Adicionar automaticamente permissões de sistema de arquivos do grupo do sistema que é proprietário do recurso.

Nome	Caminho de origem	Caminho de destino
shm	/dev/shm	/dev/shm
.tmp	/tmp	/tmp

Como configurar a inferência otimizada de aprendizado de máquina usando o Console de gerenciamento da AWS

Para seguir as etapas deste tutorial, você deve usar o AWS IoT Greengrass Core v1.10 ou posterior.

Você pode usar o compilador de aprendizado profundo SageMaker AI Neo para otimizar a eficiência de previsão dos modelos nativos de inferência de aprendizado de máquina nas estruturas Tensorflow, Apache MXNet PyTorch, ONNX e XGBoost para um espaço menor e um desempenho mais rápido. Em seguida, você pode baixar o modelo otimizado, instalar o tempo de execução de aprendizado profundo do SageMaker AI Neo e implantá-lo em seus AWS IoT Greengrass dispositivos para uma inferência mais rápida.

Este tutorial descreve como usar o Console de gerenciamento da AWS para configurar um grupo do Greengrass para executar um exemplo de inferência Lambda que reconheça imagens de uma câmera localmente, sem enviar dados para a nuvem. O exemplo de inferência acessa o módulo da câmera em um Raspberry Pi. Neste tutorial, você baixa um modelo pré-empacotado que é treinado

Resnet-50 e otimizado no compilador de aprendizado profundo Neo. Em seguida, você usa o modelo para realizar a classificação local de imagens em seu AWS IoT Greengrass dispositivo.

O tutorial contém as seguintes etapas de nível elevado:

1. Configurar o Raspberry Pi
2. Instalar o tempo de execução de aprendizagem profunda do Neo
3. Criar uma função do Lambda de inferência
4. Adicionar a função do Lambda ao grupo
5. Adicionar um recurso Neo-optimized modelo ao grupo
6. Adicionar um recurso de dispositivo de câmera ao grupo
7. Adicionar assinaturas ao grupo
8. Implantar o grupo
9. Testar o exemplo

Pré-requisitos

Para concluir este tutorial, é necessário:

- Raspberry Pi 4 Model B ou Raspberry Pi 3 Model B/B +, configurado e configurado para uso com. AWS IoT Greengrass Para configurar seu Raspberry Pi com o AWS IoT Greengrass, execute o script Greengrass Device Setup ou conclua o Módulo 1 e o Módulo 2 do Conceitos básicos de AWS IoT Greengrass.

  > **Note**
  >
  > O Raspberry Pi pode exigir uma fonte de alimentação de 2,5 A para executar as estruturas de aprendizado profundo que normalmente são usadas para classificação de imagens. Uma fonte de alimentação com uma classificação mais baixa pode fazer com que o dispositivo seja reinicializado.

- Módulo de câmera Raspberry Pi V2 - 8 megapixels, 1080 p. Para aprender a configurar a câmera, consulte Conectar a câmera na documentação do Raspberry Pi.
- Um grupo do Greengrass e um núcleo do Greengrass. Para saber como criar um grupo ou núcleo do Greengrass, consulte Conceitos básicos de AWS IoT Greengrass.

> **Note**
>
> Este tutorial usa um Raspberry Pi, mas AWS IoT Greengrass oferece suporte a outras plataformas, como Intel Atom e NVIDIA Jetson TX2. Se estiver usando o exemplo do Intel Atom, talvez seja necessário instalar o Python 3.6 em vez do Python 3.7. Para obter informações sobre como configurar seu dispositivo para que você possa instalar o software AWS IoT Greengrass Core, consultethe section called “Configurar outros dispositivos”. Para plataformas de terceiros que AWS IoT Greengrass não oferecem suporte, você deve executar sua função do Lambda no modo não conteinerizado. Para executar no modo sem contêiner, você deve executar sua função do Lambda como root. Para obter mais informações, consulte the section called “Considerações ao escolher a conteinerização de função do Lambda” e the section called “Definir a identidade de acesso padrão para as funções do Lambda em um grupo”.

Etapa 1: Configurar o Raspberry Pi

Nesta etapa, instale as atualizações no sistema operacional Raspbian, instale o software do módulo da câmera e as dependências do Python e ative a interface da câmera.

Execute os seguintes comandos no seu terminal do Raspberry Pi.

1. Instale as atualizações no Raspbian.

   ```
   sudo apt-get update
   sudo apt-get dist-upgrade
   ```

2. Instale a interface do `picamera` para o módulo da câmera e outras bibliotecas do Python que são necessárias para este tutorial.

   ```
   sudo apt-get install -y python3-dev python3-setuptools python3-pip python3-picamera
   ```

   Valide a instalação:

   - Verifique se a instalação do Python 3.7 inclui o pip.

     ```
     python3 -m pip
     ```

Se o pip não estiver instalado, faça download dele no site do pip e execute o seguinte comando.

```
python3 get-pip.py
```

- Verifique se a versão do Python é 3.7 ou superior.

```
python3 --version
```

Se a saída listar uma versão anterior, execute o seguinte comando.

```
sudo apt-get install -y python3.7-dev
```

- Verifique se o Setuptools e Picamera foram instalados com êxito.

```
sudo -u ggc_user bash -c 'python3 -c "import setuptools"'
sudo -u ggc_user bash -c 'python3 -c "import picamera"'
```

Se a saída não contiver erros, a validação será bem-sucedida.

> **Note**
>
> Se o Python executável instalado no dispositivo for o `python3.7`, use `python3.7` em vez de `python3` para os comandos neste 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 Raspberry Pi.

```
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. Use o seguinte comando 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.

Etapa 2: instalar o ambiente de execução de aprendizado profundo do Amazon SageMaker Neo

Nesta etapa, você baixa o Runtime de Aprendizado Profundo (DLR) e o instala no seu Raspberry Pi.

> **Note**
>
> Recomendamos instalar a versão 1.1.0 para este tutorial.

1. Inicie sessão no seu Raspberry Pi remotamente.

   ```
   ssh pi@your-device-ip-address
   ```

2. Abra a documentação do DLR, abra Installing DLR e localize o URL de roda para dispositivos Raspberry Pi. Depois, siga as instruções para instalar o DLR no dispositivo. Por exemplo, você pode usar o pip:

   ```
   pip3 install rasp3b-wheel-url
   ```

3. Depois de instalar o DLR, valide a seguinte configuração:

   - Verifique se a conta do sistema do `ggc_user` pode usar a biblioteca do DLR.

     ```
     sudo -u ggc_user bash -c 'python3 -c "import dlr"'
     ```

   - Verifique se NumPy está instalado.

     ```
     sudo -u ggc_user bash -c 'python3 -c "import numpy"'
     ```

Etapa 3: Criar uma função de inferência da função do Lambda

Nesta etapa, crie um pacote de implantação da função do Lambda e uma função do Lambda Depois, publique uma versão da função e crie um alias.

1. No computador, faça download do exemplo do DLR para Raspberry Pi em the section called "Exemplos de machine learning".
2. Não descompacte o arquivo `dlr-py3-armv7l.tar.gz` obtido por download.

   ```
   cd path-to-downloaded-sample
   tar -xvzf dlr-py3-armv7l.tar.gz
   ```

   O diretório `examples` no pacote de exemplo extraído contém o código e as dependências da função.

   - O `inference.py` é o código de inferência usado neste tutorial. Você pode usar esse código como modelo para criar sua própria função de inferência.
   - `greengrasssdk` é a versão 1.5.0 do SDK do AWS IoT Greengrass Core para Python.

     > **Note**
     >
     > Se uma nova versão estiver disponível, você poderá fazer download dela e atualizar a versão do SDK no seu pacote de implantação. Para obter mais informações, consulte AWS IoT Greengrass Core SDK para Python on GitHub.

3. Compacte o conteúdo do diretório `examples` em um arquivo chamado `optimizedImageClassification.zip`. Esse é o pacote de implantação.

   ```
   cd path-to-downloaded-sample/dlr-py3-armv7l/examples
   zip -r optimizedImageClassification.zip .
   ```

   O pacote de implantação contém seu código e as dependências da função. Isso inclui o código que invoca as APIs do Python do runtime de aprendizado profundo do Neo para realizar inferência com os modelos do compilador de aprendizado profundo do Neo.

> **Note**
>
> Verifique se os arquivos `.py` e as dependências estão na raiz do diretório.

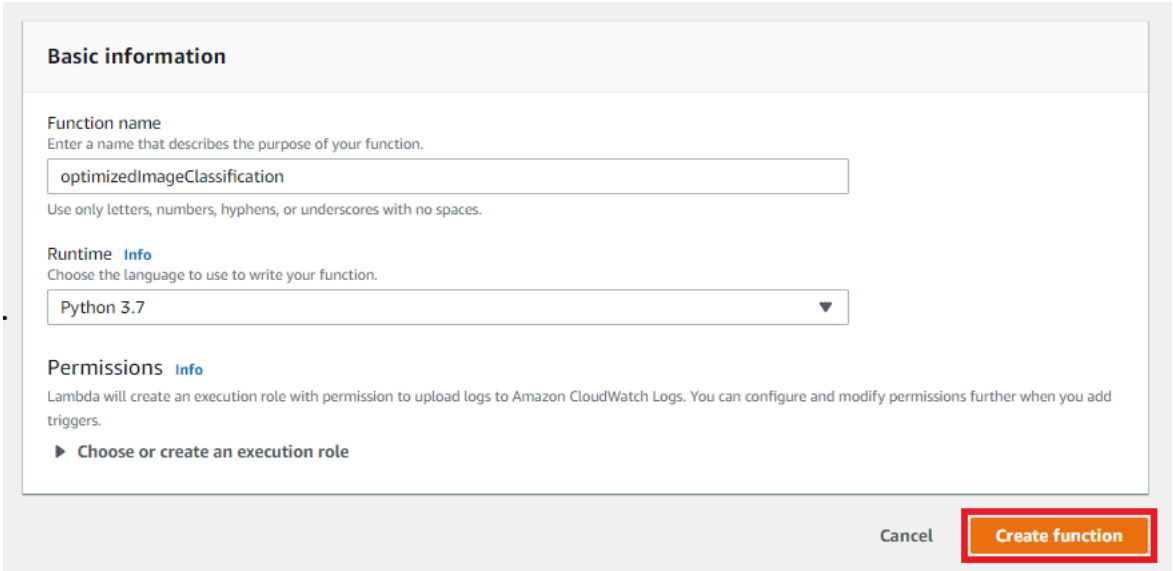
4. Agora, adicione a função do Lambda ao seu grupo do Greengrass.

   Na página do console do Lambda, selecione Funções e, em seguida, Criar função.

5. Selecione Criar do zero e use os valores a seguir para criar a função:

   - Em Function name (Nome da função), insira **`optimizedImageClassification`**.
   - Em Runtime (Tempo de execução), selecione Python 3.7.

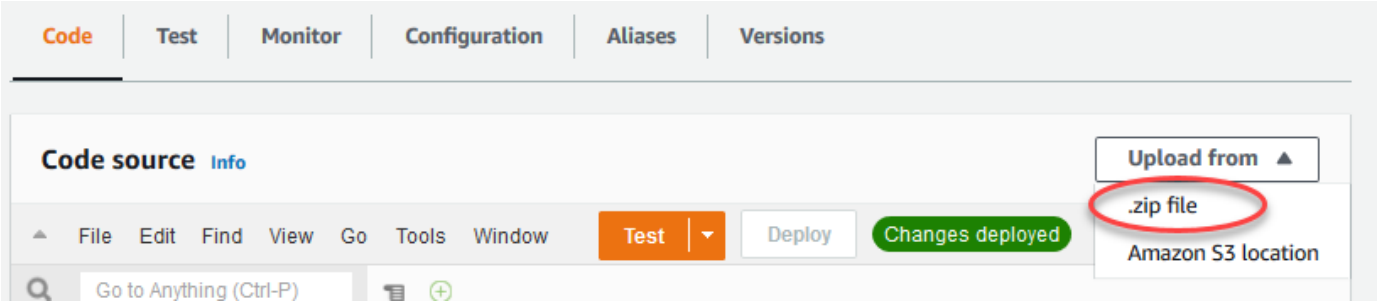
   Em Permissões, mantenha a configuração padrão. Isso cria uma função de execução que concede permissões básicas do Lambda. Essa função não é usada por AWS IoT Greengrass.



6. Selecione Criar função.

Agora, carregue o seu pacote de implantação da função do Lambda e registre o manipulador.

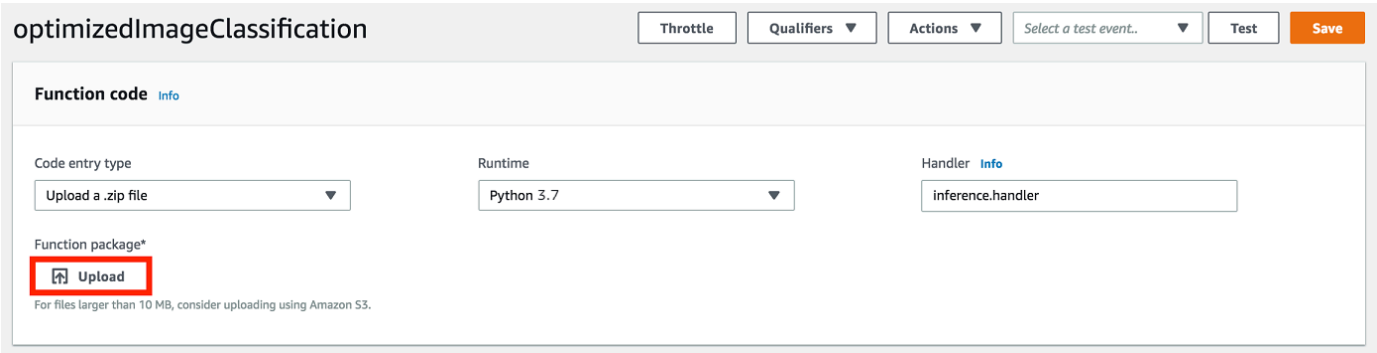
1. Na guia Código, em Fonte do código, selecione Fazer upload a partir de. No menu suspenso, selecione o arquivo .zip.



2. Selecione seu pacote de implantação `optimizedImageClassification.zip` e, em seguida, selecione Salvar.
3. Na guia Código da função, em Configurações de runtime, selecione Editar e insira os valores a seguir.

   - Em Runtime (Tempo de execução), selecione Python 3.7.
   - Em Handler (Manipulador), insira **`inference.handler`**.

   Selecione Salvar.

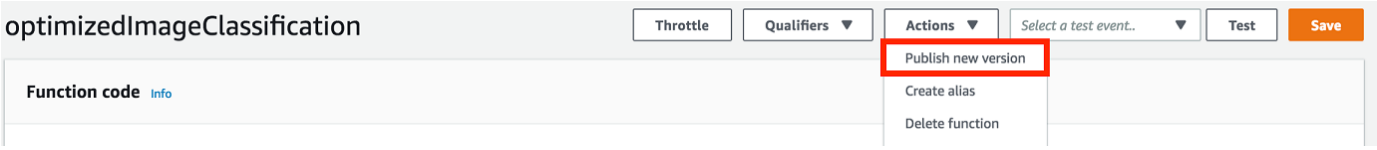


Em seguida, publique a primeira versão da sua função do Lambda. Em seguida, crie um alias para a versão.

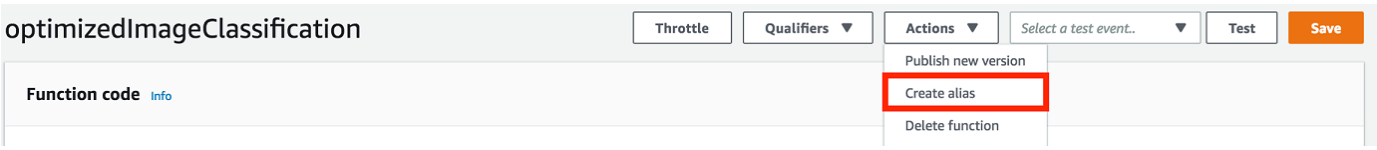
> **Note**
>
> Os grupos do Greengrass podem fazer referência a uma função do Lambda por alias (recomendado) ou por versão. Usar um alias facilita o gerenciamento de atualizações de código porque você não precisa alterar a tabela de assinaturas nem a definição do grupo ao

atualizar a função do código. Em vez disso, você pode simplesmente apontar o alias para a nova versão da função.

1. No menu Actions, selecione Publish new version.



2. Em Version description (Descrição da versão), insira **`First version`** e, em seguida, selecione Publish (Publicar).
3. Na página de configuração otimizadaImageClassification: 1, no menu Ações, escolha Criar alias.



4. Na página Create a new alias, use os seguintes valores:
   - Em Nome, digite **`mlTestOpt`**.
   - Em Version (Versão), insira **`1`**.

> **Note**
>
> AWS IoT Greengrass não oferece suporte a aliases do Lambda para as versões $LATEST.

5. Escolha Criar.

Agora, adicione a função do Lambda ao seu grupo do Greengrass.

Etapa 4: adicionar a função do Lambda ao grupo do Greengrass

Nesta etapa, adicione a função do Lambda ao grupo e configura seu ciclo de vida.

Primeiro, adicione a função do Lambda ao seu grupo do Greengrass.

1. No painel de navegação do AWS IoT console, em Gerenciar, expanda dispositivos Greengrass e escolha Grupos (V1).
2. Na página de configuração do grupo, selecione a guia funções do Lambda e, em seguida selecione Adicionar.
3. Escolha a função Lambda e selecione otimizado ImageClassification.
4. Na versão da função do Lambda, selecione o alias para a versão que você publicou.

Em seguida, configure o ciclo de vida da função do Lambda.

1. Na seção configuração da função do Lambda, faça as atualizações a seguir.

   > **Note**
   >
   > Recomendamos que você execute sua função do Lambda sem conteinerização, a menos que seu caso de negócios faça essa exigência. Isso ajuda a habilitar o acesso à GPU e à câmera do seu dispositivo sem configurar os recursos do dispositivo. Se você executar sem conteinerização, também deverá conceder acesso root às suas funções do AWS IoT Greengrass Lambda.

   a. Para executar sem conteinerização:

      - Para Usuário e grupo do sistema, selecione **`Another user ID/group ID`**. Para ID de usuário do sistema, insira **`0`**. Para ID do grupo do sistema, insira **`0`**.

        Isso permite que sua função do Lambda seja executada como root. Para obter mais informações sobre como executar como raiz, consulte the section called “Definir a identidade de acesso padrão para as funções do Lambda em um grupo”.

        > **Tip**
        >
        > Você também deve atualizar seu arquivo `config.json` para conceder acesso root à sua função do Lambda. Para o procedimento, consulte the section called “Executar uma função do Lambda como raiz”.

      - Para conteinerização da função do Lambda, selecione Sem contêiner.

Para obter mais informações sobre como executar sem conteinerização, consulte the section called “Considerações ao escolher a conteinerização de função do Lambda”.

- Em Timeout (Tempo limite), insira **`10 seconds`**.
- Para Fixado, selecione Verdadeiro.

  Para obter mais informações, consulte the section called “Configuração do ciclo de vida”.
- Em Parâmetro adicional, para Acesso de leitura ao diretório /sys, selecione Habilitado.

b. Em vez disso, para executar no modo conteinerizado:

> **Note**
>
> Não recomendamos a execução no modo conteinerizado, a menos que seu caso de negócios faça essa exigência.

- Para Usuário e grupo do sistema, selecione Usar padrão de grupo.
- Para Conteinerização da função do Lambda, selecione Usar padrão de grupo.
- Em Memory limit (Limite de memória), insira **`1024 MB`**.
- Em Timeout (Tempo limite), insira **`10 seconds`**.
- Para Fixado, selecione Verdadeiro.

  Para obter mais informações, consulte the section called “Configuração do ciclo de vida”.
- Em Parâmetros adicionais, para Acesso de leitura ao diretório /sys, selecione Habilitado.

2. Selecione Adicionar função do Lambda.

Etapa 5: adicionar um recurso de Neo-optimized modelo de SageMaker IA ao grupo Greengrass

Nesta etapa, crie um recurso para o modelo de inferência de ML otimizado e o carrega em um bucket do Amazon S3. Em seguida, localize o modelo carregado pelo Amazon S3 no AWS IoT Greengrass console e associe o recurso recém-criado à função Lambda. Isso possibilita que a função acesse os recursos no dispositivo básico.

1. No computador, navegue até o diretório `resnet50` no pacote de exemplo que você descompactou em the section called “Criar uma função do Lambda de inferência”.

> **Note**
>
> Se estiver usando o exemplo do NVIDIA Jetson, precisará usar o diretório `resnet18` no pacote de exemplo. Para obter mais informações, consulte the section called "Configuração NVIDIA Jetson TX2".

```
cd path-to-downloaded-sample/dlr-py3-armv7l/models/resnet50
```

Esse diretório contém artefatos de modelo pré-compilados para um modelo de classificação de imagens treinado com. Resnet-50

2. Compacte os arquivos dentro do diretório `resnet50` em um arquivo chamado `resnet50.zip`.

```
zip -r resnet50.zip .
```

3. Na página de configuração de grupo AWS IoT Greengrass do seu grupo, escolha a guia Recursos. Navegue até a seção Machine Learning e, em seguida selecione Adicionar recurso de machine learning. Na página Create a machine learning resource (Criar recurso de Machine Learning), em Resource name (Nome do recurso), insira **`resnet50_model`**.
4. Em Fonte do modelo, selecione Usar um modelo armazenado no S3, como um modelo otimizado por meio do compilador de aprendizado profundo.
5. Em URI do S3, selecione Procurar no S3.

> **Note**
>
> Atualmente, os modelos de SageMaker IA otimizados são armazenados automaticamente no Amazon S3. Você pode localizar seu modelo otimizado no bucket de seu Amazon S3 usando essa opção. Para obter mais informações sobre otimização de modelos em SageMaker IA, consulte a documentação do SageMaker AI Neo.

6. Selecione Upload a model (Fazer upload de um modelo).
7. Na guia do console do Amazon S3, carregue o arquivo zip em um bucket do Amazon S3. Para obter informações, consulte Como carregar arquivos e pastas em um bucket do S3? no Guia do usuário do Amazon Simple Storage Service.

> **Note**
>
> O nome do bucket deve conter a string **`greengrass`**. Selecione um nome exclusivo (como **`greengrass-dlr-bucket-`*`user-id`*`-`*`epoch-time`***). Não use ponto (`.`) no nome do bucket.

8. Na guia do AWS IoT Greengrass console, localize e escolha seu bucket Amazon S3. Localize o arquivo `resnet50.zip` carregado e, em seguida selecione Select (Selecionar). Talvez seja necessário atualizar a página para atualizar a lista de buckets e arquivos disponíveis.
9. Em Caminho de destino, insira **`/ml_model`**.



Este é o destino do modelo local no namespace de runtime do Lambda. Quando você implanta o grupo, AWS IoT Greengrass recupera o pacote do modelo de origem e, em seguida, extrai o conteúdo para o diretório especificado.

> **Note**
>
> Recomendamos que você use o caminho exato fornecido para o seu caminho local. O uso de um caminho de destino do modelo local diferente nesta etapa causa imprecisão em alguns comandos de solução de problemas fornecidos neste tutorial. Se você usar um caminho diferente, deverá configurar uma variável de ambiente `MODEL_PATH` que use o caminho exato fornecido aqui. Para obter informações sobre as variáveis de ambiente, consulte Variáveis de ambiente do AWS Lambda.

10. Se estiver executando no modo conteinerizado:
    a. Em Proprietário do grupo do sistema e permissões de acesso ao arquivo, selecione Especificar grupos e permissões do sistema.
    b. Escolha Read-only acesso e, em seguida, escolha Adicionar recurso.

Etapa 6: Adicionar um recurso de dispositivo de câmera a um grupo do Greengrass

Nesta etapa, crie um recurso para o módulo da câmera e afilie-o com a função do Lambda. Isso possibilita que a função do Lambda acesse o recurso no dispositivo básico.

> **Note**
>
> Se você executar no modo não conteinerizado, AWS IoT Greengrass poderá acessar a GPU e a câmera do seu dispositivo sem configurar esse recurso do dispositivo.

1. Na página de configuração do grupo, selecione Recursos.
2. Na guia Recursos locais, selecione Adicionar recurso local.
3. Na página Adicionar recurso local, use os seguintes valores:
   - Em Resource Name (Nome do recurso), insira **`videoCoreSharedMemory`**.
   - Em Resource type, selecione Device.
   - Em Caminho do dispositivo local, insira **`/dev/vcsm`**.

     O caminho do dispositivo é o caminho absoluto local do recurso do dispositivo. Este caminho só pode se referir a um dispositivo de caractere ou dispositivo de blocos em `/dev`.
   - Em Permissões do proprietário do grupo do sistema e de acesso a arquivos, selecione Adicionar automaticamente permissões de sistema de arquivos do grupo do sistema que é proprietário do recurso.

     A opção Group owner file access permission permite conceder permissões adicionais de acesso a arquivos para o processo do Lambda. Para obter mais informações, consulte Permissão de acesso a arquivo do proprietário do grupo.
4. Na parte inferior da página, selecione Adicionar recurso.
5. Na guia Recursos, crie outro recurso local escolhendo Adicionar e use os seguintes valores:
   - Em Resource Name (Nome do recurso), insira **`videoCoreInterface`**.
   - Em Resource type, selecione Device.
   - Em Caminho do dispositivo local, insira **`/dev/vchiq`**.

- Em Permissões do proprietário do grupo do sistema e de acesso a arquivos, selecione Adicionar automaticamente permissões de sistema de arquivos do grupo do sistema que é proprietário do recurso.

6. Selecione Adicionar recurso.

Etapa 7: Adicionar assinaturas ao grupo do Greengrass

Nesta etapa, adicione assinaturas ao grupo. Essas assinaturas permitem que a função Lambda envie resultados de previsão AWS IoT publicando em um tópico do MQTT.

1. Na página de configuração do grupo, selecione a guia Assinaturas e, em seguida, Adicionar assinatura.
2. Na página Criar uma assinatura, configure a origem e o destino, da seguinte forma:

   a. Em Tipo de origem, escolha Função Lambda e, em seguida, escolha otimizado ImageClassification.

   b. Em Tipo de destino, selecione Serviço e então IoT Cloud.

   c. Em Filtro de tópicos, insira **`/resnet-50/predictions`** e, em seguida, selecione Criar assinatura.

3. Adicione uma segunda assinatura. Selecione a guia Assinaturas, selecione Adicionar assinatura e configure a origem e o destino, da seguinte forma:

   a. Em Tipo de origem, selecione Serviços e em seguida IoT Cloud.

   b. Em Tipo de destino, escolha Função Lambda e, em seguida, escolha otimizado ImageClassification.

   c. Em Filtro de tópicos, insira **`/resnet-50/test`** e, em seguida, selecione Criar assinatura.

Etapa 8: Implantar o grupo do Greengrass

Nesta etapa, implante a versão atual da definição do grupo ao dispositivo de núcleo do Greengrass. A definição contém a função do Lambda, recursos e configurações de inscrição que você adicionou.

1. Verifique se o AWS IoT Greengrass núcleo está funcionando. Execute os seguintes comandos no seu terminal do Raspberry Pi, conforme necessário.

   a. Para verificar se o daemon está em execução:

```
ps aux | grep -E 'greengrass.*daemon'
```

Se a saída contém uma entrada `root` para `/greengrass/ggc/packages/latest-core-version/bin/daemon`, o daemon está em execução.

b. Para iniciar o daemon:

```
cd /greengrass/ggc/core/
sudo ./greengrassd start
```

2. Na página de configuração do grupo, selecione Implantar.
3. Na guia Funções do Lambda, selecione Detector de IP e selecione Editar.
4. Na caixa de diálogo Editar configurações do detector de IP, selecione Detectar e substituir automaticamente os endpoints de atendente MQTT e selecione Salvar.

   Isso permite que os dispositivos adquiram automaticamente as informações de conectividade para o núcleo, como endereço IP, DNS e o número da porta. A detecção automática é recomendada, mas AWS IoT Greengrass também oferece suporte a endpoints especificados manualmente. Você só é solicitado pelo método de descoberta na primeira vez em que o grupo for implantado.

   > **Note**
   >
   > Se solicitado, conceda permissão para criar a função de serviço do Greengrass e associá-la à sua Conta da AWS na atual. Região da AWS Essa função permite AWS IoT Greengrass acessar seus recursos em AWS serviços.

   A página Deployments mostra a data e hora, ID de versão e status da implantação. Quando concluída, o status exibido para a implantação deve ser Concluída.

   Para obter mais informações sobre implantações, consulte Implemente AWS IoT Greengrass grupos. Para obter ajuda sobre a solução de problemas, consulte Solução de problemas.

Testar o exemplo de inferência

Agora você pode verificar se a implementação foi configurada corretamente. Para testar, inscreva-se no tópico `/resnet-50/predictions` e publique qualquer mensagem no tópico `/resnet-50/test`. Isso aciona a função do Lambda para tirar uma foto com seu Raspberry Pi e executar inferência na imagem capturada.

> **Note**
>
> Se estiver usando o exemplo do NVIDIA Jetson, use os tópicos `resnet-18/predictions` e `resnet-18/test`.

> **Note**
>
> Se um monitor estiver conectado ao Raspberry Pi, o feed da câmera ao vivo será exibido em uma janela de visualização.

1. Na página inicial do AWS IoT console, em Teste, escolha Cliente de teste MQTT.
2. Em Assinaturas, selecione Inscrever-se em um tópico. Use os seguintes valores. Deixe as opções restantes em seus valores padrão.
   - Em Subscription topic (Tópico de assinatura), insira **`/resnet-50/predictions`**.
   - Em Configuração adicional, para exibição da carga útil do MQTT, selecione Exibir cargas úteis como strings.
3. Selecione Subscribe.
4. Selecione Publicar em um tópico, insira **`/resnet-50/test`** como o Nome do tópico e selecione Publicar.
5. Se o teste for bem-sucedido, a mensagem publicada faz com que a câmera do Raspberry Pi capture uma imagem. Uma mensagem da função do Lambda é exibida na parte inferior da página. A mensagem contém o resultado de previsão da imagem, usando o formato: nome da classe prevista, probabilidade e pico de utilização da memória.

Como configurar um Intel Atom

Para executar este tutorial em um dispositivo Intel Atom, você deve fornecer imagens de origem, configurar a função do Lambda e adicionar outro recurso de dispositivo local. Para usar a GPU para inferência, verifique se o software a seguir está instalado no dispositivo:

- OpenCL versão 1.0 ou posterior
- Python 3.7 e pip
- NumPy
- OpenCV sobre rodas

1. Faça download de imagens PNG ou JPG estáticas para a função do Lambda para usá-las na classificação de imagens. O exemplo funciona melhor com arquivos de imagem pequenos.

   Salve os arquivos de imagem no diretório que contém o arquivo `inference.py` (ou em um subdiretório desse diretório). Isso está no pacote de implantação da função do Lambda do qual você fez upload em the section called “Criar uma função do Lambda de inferência”.

   > **Note**
   >
   > Se estiver usando AWS DeepLens, você pode usar a câmera integrada ou montar sua própria câmera para realizar inferências em imagens capturadas em vez de imagens estáticas. No entanto, recomendamos que você comece com imagens estáticas primeiro.
   > Se utilizar uma câmara, verifique se o pacote APT `awscam` está instalado e atualizado. Para mais informações, consulte Atualizar seu AWS DeepLens dispositivo no AWS DeepLens Guia do desenvolvedor.

2. Edite a configuração da função do Lambda. Siga o procedimento em the section called “Adicionar a função do Lambda ao grupo”.

   > **Note**
   >
   > Recomendamos que você execute sua função do Lambda sem conteinerização, a menos que seu caso de negócios faça essa exigência. Isso ajuda a habilitar o acesso à GPU e à câmera do seu dispositivo sem configurar os recursos do dispositivo. Se você

executar sem conteinerização, também deverá conceder acesso root às suas funções do AWS IoT Greengrass Lambda.

a. Para executar sem conteinerização:

- Para Usuário e grupo do sistema, selecione **`Another user ID/group ID`**. Para ID de usuário do sistema, insira **`0`**. Para ID do grupo do sistema, insira **`0`**.

  Isso permite que sua função do Lambda seja executada como root. Para obter mais informações sobre como executar como raiz, consulte the section called “Definir a identidade de acesso padrão para as funções do Lambda em um grupo”.

  > **Tip**
  >
  > Você também deve atualizar seu arquivo `config.json` para conceder acesso root à sua função do Lambda. Para o procedimento, consulte the section called “Executar uma função do Lambda como raiz”.

- Para conteinerização da função do Lambda, selecione Sem contêiner.

  Para obter mais informações sobre como executar sem conteinerização, consulte the section called “Considerações ao escolher a conteinerização de função do Lambda”.

- Aumente o valor de Timeout (Tempo limite) para 2 minutos. Isso garante que a solicitação não expire muito cedo. Leva alguns minutos para que a inferência seja executada após a configuração.
- Para Fixado, selecione Verdadeiro.
- Em Parâmetros adicionais, para Acesso de leitura ao diretório /sys, selecione Habilitado.

b. Em vez disso, para executar no modo conteinerizado:

> **Note**
>
> Não recomendamos a execução no modo conteinerizado, a menos que seu caso de negócios faça essa exigência.

- Aumente o valor de Memory limit (Limite de memória) para 3000 MB.

- Aumente o valor de Timeout (Tempo limite) para 2 minutos. Isso garante que a solicitação não expire muito cedo. Leva alguns minutos para que a inferência seja executada após a configuração.
- Para Fixado, selecione Verdadeiro.
- Em Parâmetros adicionais, para Acesso de leitura ao diretório /sys, selecione Habilitado.

3. Adicione seu recurso de Neo-optimized modelo ao grupo. Carregue os recursos do modelo no diretório `resnet50` do pacote de amostra que você descompactou em the section called “Criar uma função do Lambda de inferência”. Esse diretório contém artefatos de modelo pré-compilados para um modelo de classificação de imagens treinado com. Resnet-50 Use o procedimento em the section called “Adicionar um recurso Neo-optimized modelo ao grupo” com as seguintes atualizações.

   - Compacte os arquivos dentro do diretório `resnet50` em um arquivo chamado `resnet50.zip`.
   - Na página Create a machine learning resource (Criar recurso de Machine Learning), em Resource name (Nome do recurso), insira **`resnet50_model`**.
   - Carregue o arquivo `resnet50.zip`.

4. Se estiver executando no modo conteinerizado, adicione o recurso de dispositivo local necessário para conceder acesso à GPU do seu dispositivo.

   > **Note**
   >
   > Se você executar no modo não conteinerizado, AWS IoT Greengrass poderá acessar a GPU do seu dispositivo sem configurar os recursos do dispositivo.

   a. Na página de configuração do grupo, selecione a guia Recursos.
   b. Na seção Recursos locais, selecione Adicionar recurso local.
   c. Defina o recurso:

      - Em Resource Name (Nome do recurso), insira **`renderD128`**.
      - Em Resource type, selecione Device.
      - Em Caminho do dispositivo local, insira **`/dev/dri/renderD128`**.

- Em Permissões do proprietário do grupo do sistema e de acesso a arquivos, selecione Adicionar automaticamente permissões de sistema de arquivos do grupo do sistema que é proprietário do recurso.

Configuração NVIDIA Jetson TX2

Para executar este tutorial em um NVIDIA Jetson TX2, forneça imagens de origem, configure a função do Lambda e adicione mais recursos de dispositivo local.

1. Certifique-se de que seu dispositivo Jetson esteja configurado para que você possa instalar o software AWS IoT Greengrass Core e usar a GPU para inferência. Para obter mais informações sobre como configurar seu projeto, consulte the section called “Configurar outros dispositivos”. Para usar a GPU para inferência em um NVIDIA Jetson TX2, você deve instalar o CUDA 10.0 e cuDNN 7.0 no dispositivo quando você fizer a imagem da sua placa com Jetpack 4.3.
2. Faça download de imagens PNG ou JPG estáticas para a função do Lambda para usá-las na classificação de imagens. O exemplo funciona melhor com arquivos de imagem pequenos.

   Salve seus arquivos de imagem no diretório que contém o arquivo `inference.py`. Você também pode salvá-los em um subdiretório desse diretório. Esse diretório está no pacote de implantação da função do Lambda do qual você fez upload em the section called “Criar uma função do Lambda de inferência”.

   > **Note**
   >
   > Em vez disso, você pode optar por instrumentar uma câmera na placa Jetson para capturar as imagens de origem. No entanto, recomendamos que você comece com imagens estáticas primeiro.

3. Edite a configuração da função do Lambda. Siga o procedimento em the section called “Adicionar a função do Lambda ao grupo”.

   > **Note**
   >
   > Recomendamos que você execute sua função do Lambda sem conteinerização, a menos que seu caso de negócios faça essa exigência. Isso ajuda a habilitar o acesso à GPU e à câmera do seu dispositivo sem configurar os recursos do dispositivo. Se você

executar sem conteinerização, também deverá conceder acesso root às suas funções do AWS IoT Greengrass Lambda.

a. Para executar sem conteinerização:

- Para Executar como, selecione **`Another user ID/group ID`**. Em UID, insira **`0`**. Para GUID, insira **`0`**.

  Isso permite que sua função do Lambda seja executada como root. Para obter mais informações sobre como executar como raiz, consulte the section called “Definir a identidade de acesso padrão para as funções do Lambda em um grupo”.

  > **Tip**
  >
  > Você também deve atualizar seu arquivo `config.json` para conceder acesso root à sua função do Lambda. Para o procedimento, consulte the section called “Executar uma função do Lambda como raiz”.

- Para conteinerização da função do Lambda, selecione Sem contêiner.

  Para obter mais informações sobre como executar sem conteinerização, consulte the section called “Considerações ao escolher a conteinerização de função do Lambda”.

- Aumente o valor de Timeout (Tempo limite) para 5 minutos. Isso garante que a solicitação não expire muito cedo. Leva alguns minutos para que a inferência seja executada após a configuração.
- Para Fixado, selecione Verdadeiro.
- Em Parâmetros adicionais, para Acesso de leitura ao diretório /sys, selecione Habilitado.

b. Em vez disso, para executar no modo conteinerizado:

> **Note**
>
> Não recomendamos a execução no modo conteinerizado, a menos que seu caso de negócios faça essa exigência.

- Aumente o valor de Memory limit (Limite de memória). Para usar o modelo fornecido no modo de GPU, use 2000 MB.
- Aumente o valor de Timeout (Tempo limite) para 5 minutos. Isso garante que a solicitação não expire muito cedo. Leva alguns minutos para que a inferência seja executada após a configuração.
- Para Fixado, selecione Verdadeiro.
- Em Parâmetros adicionais, para Acesso de leitura ao diretório /sys, selecione Habilitado.

4. Adicione seu recurso de Neo-optimized modelo ao grupo. Carregue os recursos do modelo no diretório `resnet18` do pacote de amostra que você descompactou em the section called “Criar uma função do Lambda de inferência”. Esse diretório contém artefatos de modelo pré-compilados para um modelo de classificação de imagens treinado com. Resnet-18 Use o procedimento em the section called “Adicionar um recurso Neo-optimized modelo ao grupo” com as seguintes atualizações.

   - Compacte os arquivos dentro do diretório `resnet18` em um arquivo chamado `resnet18.zip`.
   - Na página Create a machine learning resource (Criar recurso de Machine Learning), em Resource name (Nome do recurso), insira **`resnet18_model`**.
   - Carregue o arquivo `resnet18.zip`.

5. Se estiver executando no modo conteinerizado, adicione os recursos de dispositivo local necessários para conceder acesso à GPU do seu dispositivo.

   > **Note**
   >
   > Se você executar no modo não conteinerizado, AWS IoT Greengrass poderá acessar a GPU do seu dispositivo sem configurar os recursos do dispositivo.

   a. Na página de configuração do grupo, selecione a guia Recursos.

   b. Na seção Recursos locais, selecione Adicionar recurso local.

   c. Defina cada recurso:

      - Em Resource name (Nome do recurso) e Device path (Caminho do dispositivo), use os valores na tabela a seguir. Crie um recurso de dispositivo para cada linha na tabela.

- Em Resource type, selecione Device.
- Em Permissões do proprietário do grupo do sistema e de acesso a arquivos, selecione Adicionar automaticamente permissões de sistema de arquivos do grupo do sistema que é proprietário do recurso.

Nome	Caminho do dispositivo
nvhost-ctrl	/dev/nvhost-ctrl
nvhost-gpu	/dev/nvhost-gpu
nvhost-ctrl-gpu	/dev/nvhost-ctrl-gpu
nvhost-dbg-gpu	/dev/nvhost-dbg-gpu
nvhost-prof-gpu	/dev/nvhost-prof-gpu
nvmap	/dev/nvmap
nvhost-vic	/dev/nvhost-vitória
tegra_dc_ctrl	/dev/tegra_dc_ctrl

6. Se estiver executando no modo conteinerizado, adicione o recurso de volume local a seguir para conceder acesso à câmera do seu dispositivo. Siga o procedimento em the section called “Adicionar um recurso Neo-optimized modelo ao grupo”.

> **Note**
>
> Se você executar no modo não conteinerizado, AWS IoT Greengrass poderá acessar a câmera do seu dispositivo sem configurar os recursos do dispositivo.

- Em Tipo de recurso, selecione Volume.
- Em Permissões do proprietário do grupo do sistema e de acesso a arquivos, selecione Adicionar automaticamente permissões de sistema de arquivos do grupo do sistema que é proprietário do recurso.

Nome	Caminho de origem	Caminho de destino
shm	/dev/shm	/dev/shm
.tmp	/tmp	/tmp

7. Atualize suas assinaturas de grupo para usar o diretório correto. Use o procedimento em the section called “Adicionar assinaturas ao grupo” com as seguintes atualizações.
   - Para o primeiro filtro de tópico, digite **`/resnet-18/predictions`**.
   - Para o segundo filtro de tópico, digite **`/resnet-18/test`**.
8. Atualize suas assinaturas de teste para usar o diretório correto. Use o procedimento em the section called “Testar o exemplo” com as seguintes atualizações.
   - Em Assinaturas, selecione Inscrever-se em um tópico. Em Subscription topic (Tópico de assinatura), insira **`/resnet-18/predictions`**.
   - Na página `/resnet-18/predictions`, especifique o tópico `/resnet-18/test` no qual publicar.

Solução de problemas AWS IoT Greengrass Inferência de ML

Se o teste não for bem-sucedido, você poderá tentar as etapas de solução de problemas a seguir. Execute os comandos no seu terminal Raspberry Pi.

Verificar logs de erros

1. Alterne para o usuário raiz e navegue até o diretório `log`. O acesso aos AWS IoT Greengrass registros requer permissões de root.

```
sudo su
cd /greengrass/ggc/var/log
```

2. Verifique `runtime.log` em busca de algum erro.

```
cat system/runtime.log | grep 'ERROR'
```

Você também pode verificar em seu log se há erros na sua função do Lambda definida pelo usuário:

```
cat user/your-region/your-account-id/lambda-function-name.log | grep 'ERROR'
```

Para obter mais informações, consulte the section called “Solução de problemas com logs”.

Verifique se a função do Lambda foi implantada com êxito

1. Liste o conteúdo do Lambda implementado no diretório `/lambda`. Substitua os valores de espaço reservado antes de executar o comando.

   ```
   cd /greengrass/ggc/deployment/lambda/
   arn:aws:lambda:region:account:function:function-name:function-version
   ls -la
   ```

2. Verifique se o diretório contém o mesmo conteúdo do pacote de implantação `optimizedImageClassification.zip` que você carregou no Etapa 3: Criar uma função de inferência da função do Lambda.

   Verifique se os arquivos `.py` e as dependências estão na raiz do diretório.

Verifique se o modelo de inferência foi implantado com sucesso

1. Encontre o número de identificação de processo (PID) do runtime do Lambda:

   ```
   ps aux | grep lambda-function-name
   ```

   Na saída, o PID aparece na segunda coluna da linha para o processo de runtime do Lambda.

2. Digite o namespace de runtime do Lambda. Certifique-se de substituir o *pid* valor do espaço reservado antes de executar o comando.

> **Note**
>
> Este diretório e seu conteúdo estão no namespace de runtime do Lambda para que não sejam visíveis em um namespace comum do Linux.

```
sudo nsenter -t pid -m /bin/bash
```

3. Liste o conteúdo do diretório local que você especificou para o recurso de ML.

> **Note**
>
> Se o caminho do recurso de ML for diferente de `ml_model`, você deverá substituí-lo aqui.

```
cd /ml_model
ls -ls
```

Você deve ver os arquivos a seguir:

```
    56 -rw-r--r-- 1 ggc_user ggc_group     56703 Oct 29 20:07 model.json
196152 -rw-r--r-- 1 ggc_user ggc_group 200855043 Oct 29 20:08 model.params
   256 -rw-r--r-- 1 ggc_user ggc_group    261848 Oct 29 20:07 model.so
    32 -rw-r--r-- 1 ggc_user ggc_group     30564 Oct 29 20:08 synset.txt
```

A função Lambda não consegue encontrar **/dev/dri/renderD128**

Isso poderá ocorrer se o OpenCL não puder se conectar aos dispositivos de GPU de que precisa. Você deve criar recursos de dispositivo para os dispositivos necessários para a função do lambda.

Próximas etapas

Em seguida, explore outros modelos otimizados. Para obter informações, consulte a documentação do SageMaker AI Neo.

Gerencie fluxos de dados no AWS IoT Greengrass core

AWS IoT Greengrass O gerenciador de fluxos torna mais fácil e confiável transferir dados de IoT de alto volume para o. Nuvem AWS O gerenciador de fluxos processa fluxos de dados localmente e os exporta para o Nuvem AWS automaticamente. Esse recurso se integra a cenários periféricos comuns, como inferência de aprendizado de máquina (ML), em que os dados são processados e analisados localmente antes de serem exportados para os destinos de armazenamento Nuvem AWS ou locais.

O gerenciador de fluxo simplifica o desenvolvimento de aplicativos. Seus aplicativos para IoT podem usar um mecanismo padronizado para processar fluxos de alto volume e gerenciar políticas de retenção de dados locais em vez de criar uma funcionalidade personalizada do gerenciamento de fluxo. Os aplicativos para IoT podem ler e gravar em fluxos. Eles podem definir políticas para o tipo de armazenamento, tamanho e retenção de dados por estilhaço para controlar como o gerenciador de fluxo processa e exporta fluxos.

O gerenciador de fluxo foi projetado para funcionar em ambientes de conectividade intermitente ou limitada. Você pode definir o uso da largura de banda, o comportamento de tempo limite e como os dados do fluxo são manipulados quando o núcleo é conectado ou desconectado. Para dados críticos, você pode definir prioridades a fim de controlar a ordem em que os fluxos são exportados para a Nuvem AWS.

Você pode configurar exportações automáticas para o Nuvem AWS para armazenamento ou processamento e análise adicionais. O Stream Manager suporta a exportação para os seguintes Nuvem AWS destinos.

- Fluxos no 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 de redução de mapas. Para obter mais informações, consulte O que é o Amazon Kinesis Data Streams? no Guia do desenvolvedor do Amazon Kinesis.
- Propriedades de ativos em AWS IoT SiteWise. AWS IoT SiteWise permite coletar, organizar e analisar dados de equipamentos industriais em grande escala. Para obter mais informações, consulte O que é AWS IoT SiteWise? no Guia do AWS IoT SiteWise usuário.
- Objetos no 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.

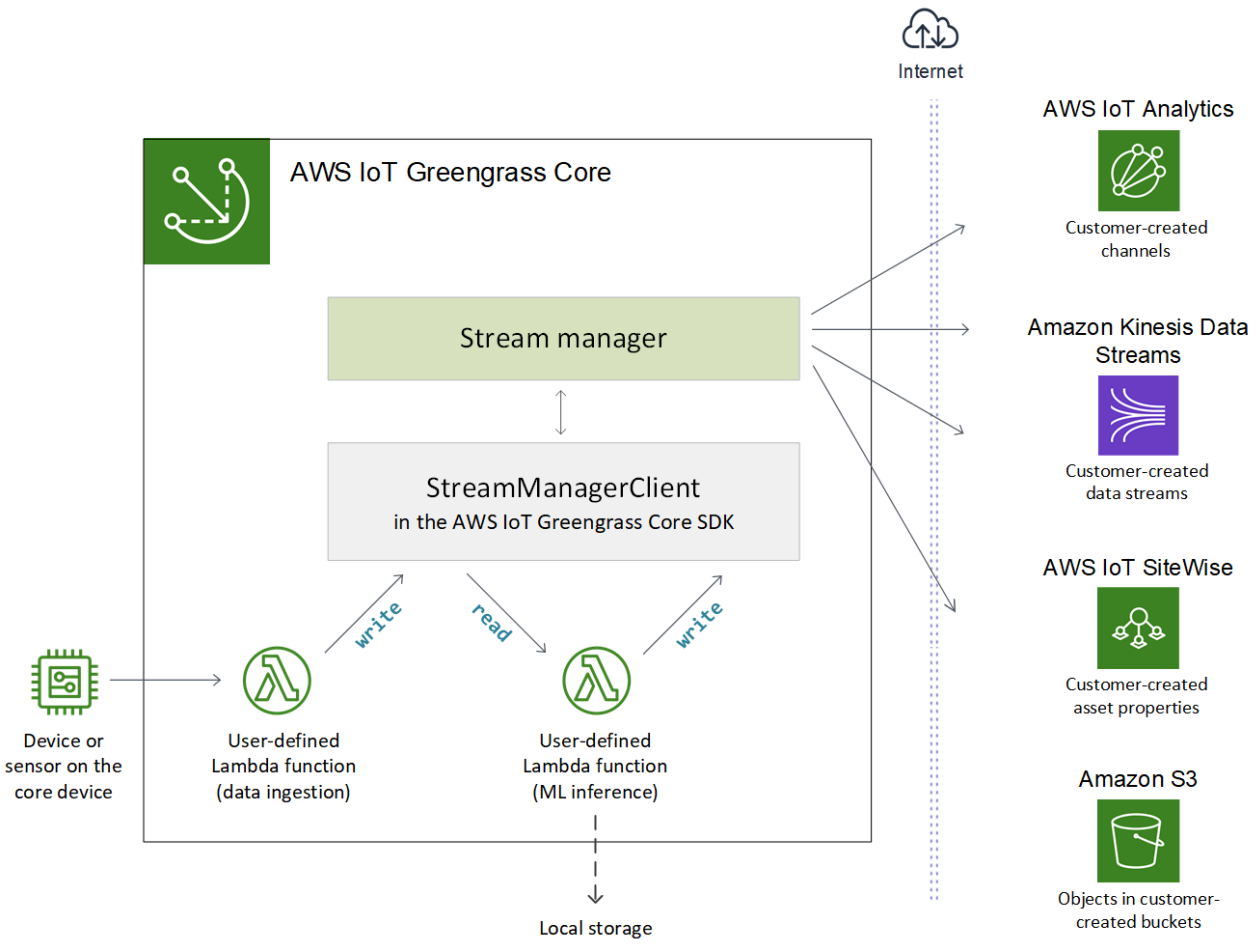
> **Note**
>
> AWS IoT Analytics foi descontinuado em 15 de dezembro de 2025 e não é mais um destino de exportação suportado. Se você tiver streams configurados com destinos de `IoTAnalyticsConfig` exportação, remova-os para evitar tentativas contínuas de exportação fracassadas. Para obter mais informações, consulte the section called “AWS IoT Analytics canais (descontinuados)”.

Fluxo de trabalho do gerenciamento de streams

Seus aplicativos de IoT interagem com o gerenciador de streams por meio do AWS IoT Greengrass Core SDK. Em um fluxo de trabalho simples, uma função do Lambda definida pelo usuário em execução no núcleo do Greengrass consome dados da IoT, como métricas de temperatura e pressão de séries temporais. A função Lambda pode filtrar ou compactar os dados e, em seguida, chamar o AWS IoT Greengrass Core SDK para gravar os dados em um stream no gerenciador de streams. O gerenciador de streams pode exportar o stream para o Nuvem AWS automaticamente, com base nas políticas definidas para o stream. User-defined As funções do Lambda também podem enviar dados diretamente para bancos de dados locais ou repositórios de armazenamento.

Os aplicativos para IoT podem incluir várias funções do Lambda definidas pelo usuário para leitura e gravação em fluxos. Essas funções locais do Lambda podem ler e gravar fluxos de modo a filtrar, agregar e analisar dados localmente. Isso torna possível responder rapidamente a eventos locais e extrair informações valiosas antes que os dados sejam transferidos do núcleo para destinos locais ou na nuvem.

Um fluxo de trabalho de exemplo é mostrado no diagrama a seguir.



Para usar o gerenciador de fluxo, comece configurando os parâmetros do gerenciador de fluxo para definir as configurações de runtime em nível de grupo que se aplicam a todos os fluxos no núcleo do Greengrass. Essas configurações personalizáveis permitem controlar como o gerenciador de fluxo armazena, processa e exporta fluxos com base nas necessidades do seu negócios e nas restrições do ambiente. Para obter mais informações, consulte the section called “Configurar o gerenciador de fluxo”.

Depois de configurar o gerenciador de fluxo, você pode criar e implantar seus aplicativos de IoT. Normalmente, essas são funções do Lambda definidas pelo usuário que são usadas `StreamManagerClient` no AWS IoT Greengrass Core SDK para criar e interagir com streams. Durante a criação do fluxo, a função do Lambda define políticas por fluxo, como destinos de exportação, prioridade e persistência. Para obter mais informações, incluindo trechos de código para operações `StreamManagerClient`, consulte the section called “Use StreamManagerClient para trabalhar com streams”.

Para tutoriais que configuram um fluxo de trabalho simples, consulte the section called “Exportar streams de dados (console)” ou the section called “Exportar streams de dados (CLI)”.

Requisitos

Os seguintes requisitos são aplicados para usar o gerenciador de fluxo:

- Você deve usar o software AWS IoT Greengrass Core v1.10 ou posterior, com o gerenciador de stream ativado. Para obter mais informações, consulte the section called “Configurar o gerenciador de fluxo”.

  O gerenciador de streams não é suportado em OpenWrt distribuições.
- O Java 8 Runtime (JDK 8) deve ser instalado no núcleo.
  - Para Debian-based distribuições (incluindo Raspbian) ou Ubuntu-based distribuições, execute o seguinte comando:

    ```
    sudo apt install openjdk-8-jdk
    ```
  - Para Hat-based distribuições Red (incluindo Amazon Linux), execute o seguinte comando:

    ```
    sudo yum install java-1.8.0-openjdk
    ```

    Para obter mais informações, consulte Como fazer download e instalar pacotes OpenJDK pré-compilados na documentação do OpenJDK.

- O gerenciador de streams requer um mínimo de 70 MB de RAM, além do software AWS IoT Greengrass Core básico. O requisito de memória total depende da sua workload.

- User-defined As funções do Lambda devem usar o AWS IoT Greengrass Core SDK para interagir com o gerenciador de streams. O AWS IoT Greengrass Core SDK está disponível em vários idiomas, mas somente as versões a seguir oferecem suporte às operações do gerenciador de streams:
  - SDK do Java (v1.4.0 ou posterior)
  - SDK do Python (v1.5.0 ou posterior)
  - Node.js SDK (v1.6.0 ou posterior)

Faça download da versão do SDK que corresponde ao runtime da função do Lambda e a inclua no pacote de implantação da função do Lambda.

> **Note**
>
> O SDK AWS IoT Greengrass Core para Python requer o Python 3.7 ou posterior e tem outras dependências de pacotes. Para obter mais informações, consulte Crie um pacote de implantação da função do Lambda (console) ou Crie um pacote de implantação da função do Lambda (CLI).

- Se você definir destinos de Nuvem AWS exportação para um stream, deverá criar seus alvos de exportação e conceder permissões de acesso na função de grupo do Greengrass. Dependendo do destino, outros requisitos também podem ser aplicados. Para obter mais informações, consulte:
  - the section called “AWS IoT Analytics canais (descontinuados)”
  - the section called “Amazon Kinesis Data Streams”
  - the section called “AWS IoT SiteWise propriedades do ativo”
  - the section called “Objetos do Amazon S3”

  Você é responsável por manter esses Nuvem AWS recursos.

Segurança de dados

Ao usar o gerenciador de fluxo, esteja ciente das seguintes considerações de segurança.

Segurança de dados locais

AWS IoT Greengrass não criptografa dados de fluxo em repouso ou em trânsito localmente entre componentes no dispositivo principal.

- Dados em repouso. Os dados de fluxo são armazenados localmente em um diretório de armazenamento o núcleo do Greengrass. Para segurança de dados, AWS IoT Greengrass depende das permissões de arquivo Unix e da criptografia de disco inteiro, se ativada. É possível usar o parâmetro STREAM_MANAGER_STORE_ROOT_DIR opcional para especificar o diretório de armazenamento. Se você alterar esse parâmetro posteriormente para usar um diretório de armazenamento diferente, AWS IoT Greengrass não excluirá o diretório de armazenamento anterior nem seu conteúdo.

- Dados em trânsito localmente. AWS IoT Greengrass não criptografa dados de stream em trânsito local no núcleo entre fontes de dados, funções do Lambda, o AWS IoT Greengrass Core SDK e o gerenciador de streams.

- Dados em trânsito para Nuvem AWS o. Os fluxos de dados exportados pelo gerenciador de fluxos Nuvem AWS usam criptografia de cliente de AWS serviço padrão com Transport Layer Security (TLS).

Para obter mais informações, consulte the section called “Criptografia de dados”.

Autenticação de cliente

Os clientes do Stream Manager usam o AWS IoT Greengrass Core SDK para se comunicarem com o stream manager. Quando a autenticação do cliente está ativada, apenas as funções do Lambda no grupo do Greengrass podem interagir com fluxos no gerenciador de fluxo. Quando a autenticação do cliente está desabilitada, qualquer processo em execução no núcleo do Greengrass (como contêineres do Docker) pode interagir com fluxos no gerenciador de fluxo. 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 no console ou na AWS IoT Greengrass API. As alterações entrarão em vigor após a implantação do grupo.

	Habilitado	Desabilitado
Valor do parâmetro	`true` (padrão e recomendado)	`false`
Clientes permitidos	User-defined Funções do Lambda no grupo Greengrass	User-defined Funções do Lambda no grupo Greengrass Outros processos em execução no dispositivo de Greengrass nucleus

Consulte também

- the section called “Configurar o gerenciador de fluxo”
- the section called “Use StreamManagerClient para trabalhar com streams”
- the section called “Exporte configurações para suporte Nuvem AWS destinos”
- the section called “Exportar streams de dados (console)”
- the section called “Exportar streams de dados (CLI)”

Configurar AWS IoT Greengrass gerenciador de stream

No AWS IoT Greengrass núcleo, 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 configurações de tempo de execução no nível do grupo. Essas configurações se aplicam a todos os streams no núcleo do Greengrass. Você pode usar o AWS IoT console ou a AWS IoT Greengrass API para definir as configurações do gerenciador de streams. As alterações entrarão em vigor após a implantação do grupo.

> **Note**
>
> Após configurar o gerenciador de fluxo, você pode criar e implantar aplicativos de IoT que são executados no núcleo do Greengrass e interagem com o gerenciador de fluxo. Esses aplicativos de IoT geralmente são funções do Lambda definidas pelo usuário. Para obter mais informações, consulte the section called “Use StreamManagerClient para trabalhar com streams”.

Parâmetros do gerenciador de fluxo

O gerenciador de fluxo fornece os seguintes parâmetros que permitem definir configurações no nível do grupo. Todos os parâmetros são opcionais.

Diretório de armazenamento

Nome do parâmetro: `STREAM_MANAGER_STORE_ROOT_DIR`

O caminho absoluto do diretório local usado para armazenar fluxos. Esse valor deve começar com uma barra (por exemplo, `/data`).

Para obter informações sobre como proteger dados de fluxo, consulte the section called "Segurança de dados locais".

Versão mínima AWS IoT Greengrass do núcleo: 1.10.0

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`.

Versão mínima AWS IoT Greengrass do núcleo: 1.10.0

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 AWS IoT Greengrass principal. Esse parâmetro determina quais clientes podem chamar o SDK do AWS IoT Greengrass Core para trabalhar com fluxos. Para obter mais informações, consulte the section called "Autenticação de cliente".

Os valores válidos são `true` ou `false`. O padrão é `true` (recomendado).

- `true`. Permite apenas funções do Lambda do Greengrass como clientes. Os clientes da função Lambda usam protocolos AWS IoT Greengrass principais internos para se autenticar com o AWS IoT Greengrass SDK principal.
- `false`. Permite que qualquer processo executado no AWS IoT Greengrass núcleo seja um cliente. Não defina como `false` a menos que seu caso de negócios exija. Por exemplo, defina esse valor como `false` somente se os processos que não são do dispositivo central precisarem se comunicar diretamente com o gerenciador de fluxo, como contêineres do Docker em execução no núcleo.

Versão mínima AWS IoT Greengrass do núcleo: 1.10.0

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.

Versão mínima AWS IoT Greengrass do núcleo: 1.10.0

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, será possível 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, portanto, você deve tentar limitar o número de fluxos de exportação em dispositivos restritos.

Versão mínima AWS IoT Greengrass do núcleo: 1.10.0

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.

Versão mínima AWS IoT Greengrass do núcleo: 1.10.0

Read-only diretórios de arquivos de entrada

Nome do parâmetro: `STREAM_MANAGER_READ_ONLY_DIRS`

Uma lista separada por vírgulas de caminhos absolutos para os diretórios fora do sistema de arquivos raiz que armazenam arquivos de entrada. O gerenciador de fluxo lê e carrega os arquivos no Amazon S3 e monta os diretórios como somente leitura. Para obter mais informações, sobre como exportar para o Amazon S3, consulte the section called “Objetos do Amazon S3”.

Use esse parâmetro somente se as condições a seguir forem verdadeiras:

- O diretório de arquivos de entrada de um fluxo que exporta para o Amazon S3 está em um dos seguintes locais:
  - Uma partição diferente do sistema de arquivos raiz.
  - Em `/tmp` no sistema de arquivos raiz.
- A conteinerização padrão do grupo Greengrass é o contêiner Greengrass.

Valor de exemplo: `/mnt/directory-1,/mnt/directory-2,/tmp`

Versão mínima AWS IoT Greengrass do núcleo: 1.11.0

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. User-defined As funções Lambda definem esse limite quando criam um stream que exporta para o Amazon S3. O limite padrão é 5 MB.

Versão mínima AWS IoT Greengrass do núcleo: 1.11.0

Definir configurações do gerenciador de fluxo (console)

Você pode usar o AWS IoT console para as seguintes tarefas de gerenciamento:

- Verificar se o gerenciador de fluxo está habilitado
- Habilitar ou desabilitar o gerenciador de fluxo durante a criação do grupo
- Habilitar ou desabilitar o gerenciador de fluxo para um grupo existente
- Alterar configurações do gerenciador de fluxo

As alterações entram em vigor após a implantação do grupo do Greengrass. Para ver um tutorial que mostra como implantar um grupo do Greengrass que contém uma função do Lambda que interage com o gerenciador de fluxo, consulte the section called “Exportar streams de dados (console)”.

> **Note**
>
> Quando você usa o console para habilitar o gerenciador de fluxo e implantar o grupo, o limite de memória para o gerenciador de fluxo é definido como 4194304 KB (4 GB), por padrão. É recomendável definir o tamanho da memória para pelo menos 128000 KB.

Para verificar se o gerenciador de fluxo está habilitado (console)

1. No painel de navegação do AWS IoT console, em Gerenciar, expanda dispositivos Greengrass e escolha Grupos (V1).
2. Selecione o grupo de destino.
3. Selecione a guia Funções do Lambda.
4. Em Funções do Lambda do sistema, selecione Gerenciador de fluxo e selecione Editar.
5. Verifique o status ativado ou desativado. Todas as configurações personalizadas do gerenciador de fluxo que estão definidas também são exibidas.

Para habilitar ou desabilitar o gerenciador de fluxo durante a criação do grupo (console)

1. No painel de navegação do AWS IoT console, em Gerenciar, expanda dispositivos Greengrass e escolha Grupos (V1).
2. Selecione Create Group. Sua selecione na próxima página determina a configuração do gerenciador de fluxo para o grupo.
3. Vá até a seção Nomeie seu grupo e selecione as páginas de Núcleo do Greengrass.
4. Selecione Criar grupo.
5. Na página de configuração do grupo, selecione a guia Funções do Lambda, selecione Gerenciador de fluxo e selecione Editar.

- Para habilitar o gerenciador de fluxo com configurações padrão, selecione Habilitar com definições padrão.

- Para habilitar o gerenciador de fluxo com configurações personalizadas, selecione Customize settings (Personalizar configurações).
  1. Na página Configurar gerenciador de fluxo, selecione Habilitar.
  2. Em Custom settings (Configurações personalizadas), insira os valores para os parâmetros do gerenciador de fluxo. Para obter mais informações, consulte the section called "Parâmetros do gerenciador de fluxo". Deixe os campos vazios para AWS IoT Greengrass permitir o uso de seus valores padrão.

- Para desativar o gerenciador de fluxo, selecione Desativar.
  1. Na página Configure stream manager (Configurar gerenciador de fluxo) selecione Disable (Desabilitar).

6. Selecione Salvar.
7. Continue nas páginas restantes para criar seu grupo.
8. Na página Dispositivos cliente, faça download dos recursos de segurança, reveja as informações e, em seguida selecione Concluir.

> **Note**
>
> Quando o gerenciador de fluxo estiver habilitado, instale o Java 8 Runtime no dispositivo de núcleo antes de implantar o grupo.

Para habilitar ou desabilitar o gerenciador de fluxo para um grupo existente (console)

1. No painel de navegação do AWS IoT console, em Gerenciar, expanda dispositivos Greengrass e escolha Grupos (V1).
2. Selecione o grupo de destino.

3. Selecione a guia Funções do Lambda.
4. Em Funções do Lambda do sistema, selecione Gerenciador de fluxo e selecione Editar.
5. Verifique o status ativado ou desativado. Todas as configurações personalizadas do gerenciador de fluxo que estão definidas também são exibidas.

Para alterar as configurações do gerenciador de fluxo (console)

1. No painel de navegação do AWS IoT console, em Gerenciar, expanda dispositivos Greengrass e escolha Grupos (V1).
2. Selecione o grupo de destino.
3. Selecione a guia Funções do Lambda.
4. Em Funções do Lambda do sistema, selecione Gerenciador de fluxo e selecione Editar.
5. Verifique o status ativado ou desativado. Todas as configurações personalizadas do gerenciador de fluxo que estão definidas também são exibidas.
6. Selecione Salvar.

Definir configurações do gerenciador de fluxo (CLI)

No AWS CLI, use a função `GGStreamManager` Lambda do sistema para configurar o gerenciador de streams. As funções do System Lambda são componentes do software AWS IoT Greengrass Core. Para o gerenciador de fluxo e algumas outras funções do Lambda do sistema, você pode configurar a funcionalidade do Greengrass gerenciando os objetos `Function` e `FunctionDefinitionVersion` correspondentes no grupo Greengrass. Para obter mais informações, consulte the section called “Visão geral do modelo de objeto de grupo”.

É possível usar a API para as seguintes tarefas de gerenciamento. Os exemplos nesta seção mostram como usar o AWS CLI, mas você também pode chamar a AWS IoT Greengrass API diretamente ou usar um AWS SDK.

- Verificar se o gerenciador de fluxo está habilitado
- Habilitar, desabilitar ou definir configurações do gerenciador de fluxo

As alterações entrarão em vigor após a implantação do grupo. Para ver um tutorial que mostra como implantar um grupo do Greengrass com uma função do Lambda que interage com o gerenciador de fluxo, consulte the section called “Exportar streams de dados (CLI)”.

> **Tip**
>
> Para analisar se o gerenciador de fluxo está habilitado e em execução, você pode executar o comando a seguir em um terminal no seu dispositivo de núcleo.
>
> ```
> ps aux | grep -i 'streammanager'
> ```

Para verificar se o gerenciador de fluxo está habilitado (CLI)

O gerenciador de fluxo será habilitado se a versão de definição de função implantada incluir a função do Lambda do sistema `GGStreamManager`. Para verificar, faça o seguinte;

1. Obtenha os IDs do grupo do Greengrass de destino e a versão do grupo. Esse procedimento pressupõe que esse seja o grupo e a versão mais recente do grupo. A consulta a seguir retorna o grupo criado mais recentemente.

   ```
   aws greengrass list-groups --query "reverse(sort_by(Groups, &CreationTimestamp))
   [0]"
   ```

   Ou é possível consultar por nome. Os nomes de grupo não precisam ser exclusivos, portanto, vários grupos podem ser retornados.

   ```
   aws greengrass list-groups --query "Groups[?Name=='MyGroup']"
   ```

   > **Note**
   >
   > Você também pode encontrar esses valores no AWS IoT console. O ID do grupo é exibido na página Settings (Configurações) do grupo. Os IDs de versão do grupo são exibidos na guia Implantações do grupo.

2. Copie os valores `Id` e `LatestVersion` do grupo de destino na saída.

3. Obtenha a versão do grupo mais recente.

   - Substitua *group-id* pelo `Id` que você copiou.
   - Substitua *latest-group-version-id* pelo `LatestVersion` que você copiou.

   ```
   aws greengrass get-group-version \
   --group-id group-id \
   --group-version-id latest-group-version-id
   ```

4. Em `FunctionDefinitionVersionArn`, na saída, copie os IDs de definição de função e versão de definição de função.

   - O ID de definição de função é o GUID que segue o segmento `functions` no nome do recurso da Amazon (ARN).
   - O ID de versão de definição de função é o GUID que segue o segmento `versions` no ARN.

   ```
   arn:aws:greengrass:us-west-2:123456789012:/greengrass/definition/
   functions/function-definition-id/versions/function-definition-version-id
   ```

5. Obtenha o ID da versão de definição de função.

   - *function-definition-id*Substitua pelo ID da definição da função.
   - *function-definition-version-id*Substitua pelo ID da versão da definição da função.

   ```
   aws greengrass get-function-definition-version \
   --function-definition-id function-definition-id \
   --function-definition-version-id function-definition-version-id
   ```

Se a matriz `functions` na saída incluir a função `GGStreamManager`, o gerenciador de fluxo será habilitado. Quaisquer variáveis de ambiente definidas para a função representam configurações personalizadas para o gerenciador de fluxo.

Habilitar, desabilitar ou definir as configurações do gerenciador de fluxo (CLI)

No AWS CLI, use a função `GGStreamManager` Lambda do sistema para configurar o gerenciador de streams. As alterações entrarão em vigor após a implantação do grupo.

- Para habilitar o gerenciador de fluxo, inclua `GGStreamManager` na matriz `functions` da sua versão de definição de função. Para definir configurações personalizadas, defina variáveis de ambiente para os parâmetros correspondentes do gerenciador de fluxo.
- Para desabilitar o gerenciador de fluxo, remova `GGStreamManager` da matriz `functions` da sua versão de definição de função.

gerenciador de fluxo com configurações padrão

O exemplo de configuração a seguir habilita o gerenciador de fluxo com configurações padrão. Ele define o ID da função arbitrária como `streamManager`.

```
{
    "FunctionArn": "arn:aws:lambda:::function:GGStreamManager:1",
    "FunctionConfiguration": {
        "MemorySize": 4194304,
        "Pinned": true,
        "Timeout": 3
    },
    "Id": "streamManager"
}
```

> **Note**
>
> Para as propriedades `FunctionConfiguration`, talvez você saiba o seguinte:
>
> - `MemorySize` está definido para 4194304 KB (4 GB) com as configurações padrão. Sempre há a possibilidade de alterar esse valor. É recomendável definir o `MemorySize` para pelo menos 128000 KB.
> - `Pinned` deve ser definido como `true`.
> - O `Timeout` é exigido pela versão de definição de função, mas o `GGStreamManager` não o utiliza.

gerenciador de fluxo com configurações personalizadas

O exemplo de configuração a seguir habilita o gerenciador de fluxo com os valores personalizados para o diretório de armazenamento, a porta do servidor e os parâmetros de tamanho do grupo de threads.

```
{
    "FunctionArn": "arn:aws:lambda:::function:GGStreamManager:1",
    "FunctionConfiguration": {
        "Environment": {
            "Variables": {
                "STREAM_MANAGER_STORE_ROOT_DIR": "/data",
                "STREAM_MANAGER_SERVER_PORT": "1234",
                "STREAM_MANAGER_EXPORTER_THREAD_POOL_SIZE": "4"
            }
        },
        "MemorySize": 4194304,
        "Pinned": true,
        "Timeout": 3
    },
    "Id": "streamManager"
}
```

AWS IoT Greengrass usa valores padrão para parâmetros do gerenciador de fluxo que não são especificados como variáveis de ambiente.

Gerenciador de fluxo com configurações personalizadas para exportações do Amazon S3

O exemplo de configuração a seguir habilita o gerenciador de fluxo com valores personalizados para o diretório de upload e com parâmetros mínimos de tamanho de upload de várias partes.

```
{
    "FunctionArn": "arn:aws:lambda:::function:GGStreamManager:1",
    "FunctionConfiguration": {
        "Environment": {
            "Variables": {
                "STREAM_MANAGER_READ_ONLY_DIRS": "/mnt/directory-1,/mnt/
directory-2,/tmp",

 "STREAM_MANAGER_EXPORTER_S3_DESTINATION_MULTIPART_UPLOAD_MIN_PART_SIZE_BYTES":
 "10485760"
            }
        },
        "MemorySize": 4194304,
        "Pinned": true,
        "Timeout": 3
    },
    "Id": "streamManager"
}
```

Habilitar, desabilitar ou definir as configurações do gerenciador de fluxo (CLI)

1. Obtenha os IDs do grupo do Greengrass de destino e a versão do grupo. Esse procedimento pressupõe que esse seja o grupo e a versão mais recente do grupo. A consulta a seguir retorna o grupo criado mais recentemente.

   ```
   aws greengrass list-groups --query "reverse(sort_by(Groups, &CreationTimestamp))
   [0]"
   ```

   Ou é possível consultar por nome. Os nomes de grupo não precisam ser exclusivos, portanto, vários grupos podem ser retornados.

   ```
   aws greengrass list-groups --query "Groups[?Name=='MyGroup']"
   ```

   > **Note**
   >
   > Você também pode encontrar esses valores no AWS IoT console. O ID do grupo é exibido na página Settings (Configurações) do grupo. Os IDs de versão do grupo são exibidos na guia Implantações do grupo.

2. Copie os valores `Id` e `LatestVersion` do grupo de destino na saída.
3. Obtenha a versão do grupo mais recente.

   - Substitua *`group-id`* pelo `Id` que você copiou.
   - Substitua *`latest-group-version-id`* pelo `LatestVersion` que você copiou.

   ```
   aws greengrass get-group-version \
   --group-id group-id \
   --group-version-id latest-group-version-id
   ```

4. Copie o ARN `CoreDefinitionVersionArn` e todos os outros ARNs da versão da saída, exceto `FunctionDefinitionVersionArn`. Você usa esses valores ao criar uma nova versão do grupo.
5. No `FunctionDefinitionVersionArn` na saída, copie o ID da definição de função. O ID é o GUID que segue o segmento `functions` no ARN, conforme mostrado no exemplo a seguir.

```
arn:aws:greengrass:us-west-2:123456789012:/greengrass/
definition/functions/bcfc6b49-beb0-4396-b703-6dEXAMPLEcu5/
versions/0f7337b4-922b-45c5-856f-1aEXAMPLEsf6
```

> **Note**
>
> Ou você pode criar uma definição de função executando o comando `create-function-definition` e copiar o ID da saída.

6. Adicione uma versão de definição de função à definição da função.

   - *`function-definition-id`*Substitua pelo `Id` que você copiou para a definição da função.
   - Na matriz `functions`, inclua todas as outras funções que você deseja disponibilizar no núcleo do Greengrass. Você pode usar o comando `get-function-definition-version` para obter a lista de funções existentes.

   Habilitar o gerenciador de fluxo com configurações padrão

   O exemplo a seguir habilita o gerenciador de fluxo, incluindo a função `GGStreamManager`, na matriz `functions`. Este exemplo usa valores padrão para os parâmetros do gerenciador de fluxo.

```
aws greengrass create-function-definition-version \
--function-definition-id function-definition-id \
--functions '[
        {
            "FunctionArn": "arn:aws:lambda:::function:GGStreamManager:1",
            "FunctionConfiguration": {
                "MemorySize":  4194304,
                "Pinned": true,
                "Timeout": 3
            },
            "Id": "streamManager"
        },
        {
            "FunctionArn": "arn:aws:lambda:us-
west-2:123456789012:function:MyLambdaFunction:MyAlias",
```

```
            "FunctionConfiguration": {
                "Executable": "myLambdaFunction.function_handler",
                "MemorySize": 16000,
                "Pinned": true,
                "Timeout": 5
            },
            "Id": "myLambdaFunction"
        },
        ... more user-defined functions
    ]
}'
```

> **Note**
>
> A função `myLambdaFunction` nos exemplos representa uma de suas funções do Lambda definidas pelo usuário.

Habilitar o gerenciador de fluxo com configurações personalizadas

O exemplo a seguir habilita o gerenciador de fluxo, incluindo a função `GGStreamManager`, na matriz `functions`. Todas as configurações do gerenciador de fluxo são opcionais, a menos que você queira alterar os valores padrão. Este exemplo mostra como usar variáveis de ambiente para definir valores personalizados.

```
aws greengrass create-function-definition-version \
--function-definition-id function-definition-id \
--functions '[
        {
            "FunctionArn": "arn:aws:lambda:::function:GGStreamManager:1",
            "FunctionConfiguration": {
                "Environment": {
                    "Variables": {
                        "STREAM_MANAGER_STORE_ROOT_DIR": "/data",
                        "STREAM_MANAGER_SERVER_PORT": "1234",
                        "STREAM_MANAGER_EXPORTER_THREAD_POOL_SIZE": "4"
                    }
                },
                "MemorySize":  4194304,
                "Pinned": true,
                "Timeout": 3
```

```
            },
            "Id": "streamManager"
        },
        {
            "FunctionArn": "arn:aws:lambda:us-
west-2:123456789012:function:MyLambdaFunction:MyAlias",
            "FunctionConfiguration": {
                "Executable": "myLambdaFunction.function_handler",
                "MemorySize": 16000,
                "Pinned": true,
                "Timeout": 5
            },
            "Id": "myLambdaFunction"
        },
        ... more user-defined functions
    ]
}'
```

> **Note**
>
> Para as propriedades `FunctionConfiguration`, talvez você saiba o seguinte:
>
> - `MemorySize` está definido para 4194304 KB (4 GB) com as configurações padrão. Sempre há a possibilidade de alterar esse valor. É recomendável definir o `MemorySize` para pelo menos 128000 KB.
> - `Pinned` deve ser definido como `true`.
> - O `Timeout` é exigido pela versão de definição de função, mas o `GGStreamManager` não o utiliza.

Desabilitar o gerenciador de fluxo

O exemplo a seguir omite a função `GGStreamManager`, que desabilita o gerenciador de fluxo.

```
aws greengrass create-function-definition-version \
--function-definition-id function-definition-id \
--functions '[
        {
            "FunctionArn": "arn:aws:lambda:us-
west-2:123456789012:function:MyLambdaFunction:MyAlias",
```

```
            "FunctionConfiguration": {
                "Executable": "myLambdaFunction.function_handler",
                "MemorySize": 16000,
                "Pinned": true,
                "Timeout": 5
            },
            "Id": "myLambdaFunction"
        },
        ... more user-defined functions
    ]
}'
```

> **Note**
>
> Se não quiser implantar nenhuma função do Lambda, você poderá omitir completamente a versão de definição de função.

7. Copie o `Arn` da versão da definição de função da saída.
8. Crie uma versão do grupo que contém a função do Lambda definida pelo sistema.

   - *`group-id`*Substitua `Id` pelo para o grupo.
   - *`core-definition-version-arn`*Substitua pelo `CoreDefinitionVersionArn` que você copiou da versão mais recente do grupo.
   - *`function-definition-version-arn`*Substitua pelo `Arn` que você copiou para a nova versão de definição de função.
   - Substitua os ARNs de outros componentes do grupo (por exemplo, `SubscriptionDefinitionVersionArn` ou `DeviceDefinitionVersionArn`) que você copiou na versão do grupo mais recente.
   - Remova todos os parâmetros inutilizados. Por exemplo, remova a versão `--resource-definition-version-arn` caso a versão do grupo não contenha recursos.

```
aws greengrass create-group-version \
--group-id group-id \
--core-definition-version-arn core-definition-version-arn \
--function-definition-version-arn function-definition-version-arn \
--device-definition-version-arn device-definition-version-arn \
--logger-definition-version-arn logger-definition-version-arn \
--resource-definition-version-arn resource-definition-version-arn \
```

```
--subscription-definition-version-arn subscription-definition-version-arn
```

9. Copie a `Version` da saída. Este é o ID da nova versão do grupo.
10. Implante o grupo com a nova versão do grupo.

    - *group-id*Substitua pelo `Id` que você copiou para o grupo.
    - *group-version-id*Substitua pelo `Version` que você copiou para a nova versão do grupo.

```
aws greengrass create-deployment \
--group-id group-id \
--group-version-id group-version-id \
--deployment-type NewDeployment
```

Siga este procedimento se quiser editar as configurações do gerenciador de fluxo posteriormente. Assegure-se de criar uma versão de definição de função que inclua a função do `GGStreamManager` com a configuração atualizada. A versão do grupo deve fazer referência a todos os ARNs da versão do componente que você deseja implantar no núcleo. As alterações entrarão em vigor após a implantação do grupo.

Consulte também

- Gerenciamento de fluxos de dados
- the section called “Use StreamManagerClient para trabalhar com streams”
- the section called “Exporte configurações para suporte Nuvem AWS destinos”
- the section called “Exportar streams de dados (console)”
- the section called “Exportar streams de dados (CLI)”

Use StreamManagerClient para trabalhar com streams

User-defined As funções Lambda executadas no AWS IoT Greengrass núcleo podem usar o `StreamManagerClient` objeto no SDK AWS IoT Greengrass principal para criar fluxos no gerenciador de fluxos e depois interagir com os fluxos. Quando uma função Lambda cria um fluxo, ela define os Nuvem AWS destinos, a priorização e outras políticas de exportação e retenção de dados para o fluxo. Para enviar dados para o gerenciador de fluxo, as funções do Lambda 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 gerenciador de fluxo são funções do Lambda definidas pelo usuário. Se o seu caso de negócios exigir, você também pode permitir que os processos não Lambda sendo executados no núcleo do Greengrass (por exemplo, um contêiner do Docker) interajam com o gerenciador de fluxo. Para obter 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. Para tutoriais que incluem uma função do Lambda em Python completa, consulte the section called “Exportar streams de dados (console)” ou the section called “Exportar streams de dados (CLI)”.

Você deve instanciar sua função do Lambda `StreamManagerClient` fora do manipulador de funções. 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 fluxos”
- 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 fluxo, uma função do Lambda definida pelo usuário chama o método create e passa em 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, Kinesis Data AWS IoT SiteWise Streams e Amazon S3, destinos para exportações automáticas. Para obter mais informações, consulte the section called “Exporte configurações para suporte Nuvem AWS destinos”.
- 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 AWS IoT SiteWise . O gerenciador de fluxo exporta mensagens quando qualquer condição é atendida.
- Time-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 Node.js SDK
- 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 fluxo é criado, suas funções do Lambda podem anexar mensagens ao fluxo para enviar dados para exportação e ler mensagens do fluxo para processamento local. O número de fluxos criados depende dos seus recursos de hardware e caso de negócios. Uma estratégia é criar um stream para cada canal de destino no stream AWS IoT Analytics de dados do Kinesis, embora você possa definir vários destinos para um stream. Um fluxo tem longa duração.

Requisitos

Essa operação tem os seguintes requisitos:

- Versão mínima AWS IoT Greengrass do núcleo: 1.10.0
- Versão mínima AWS IoT Greengrass do SDK principal: Python: 1.5.0 | Java: 1.4.0 |: 1.6.0 Node.js

> **Note**
>
> A criação de streams com um destino de exportação AWS IoT SiteWise ou do Amazon S3 tem os seguintes requisitos:
>
> - Versão mínima AWS IoT Greengrass do núcleo: 1.11.0
> - Versão mínima AWS IoT Greengrass do SDK principal: Python: 1.6.0 | Java: 1.5.0 |: 1.7.0 Node.js

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 =
 GreengrassClientBuilder.streamManagerClient().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)
                                    .withS3TaskExecutor(null)
                    )

    );
} catch (StreamManagerException e) {
    // Properly handle exception.
}
```

Referência do Java SDK: criar | MessageStream MessageStreamDefinition

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)
                        .withS3TaskExecutor(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.
```

```
});
```

Node.js Referência do SDK: criar | MessageStream MessageStreamDefinition

Para obter mais informações sobre como configurar destinos de exportação, consulte the section called “Exporte configurações para suporte Nuvem AWS destinos”.

Anexar mensagem

Para enviar dados ao gerenciador de fluxo para exportação, suas funções do Lambda anexam os dados ao fluxo 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 AWS IoT Greengrass do núcleo: 1.10.0
- Versão mínima AWS IoT Greengrass do SDK principal: Python: 1.5.0 | Java: 1.4.0 |: 1.6.0 Node.js

> **Note**
>
> Anexar mensagens a um destino de exportação AWS IoT SiteWise ou ao Amazon S3 tem os seguintes requisitos:
>
> - Versão mínima AWS IoT Greengrass do núcleo: 1.11.0
> - Versão mínima AWS IoT Greengrass do SDK principal: Python: 1.6.0 | Java: 1.5.0 |: 1.7.0 Node.js

Exemplos

AWS IoT Analytics ou destinos de exportação 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, suas funções Lambda acrescentam um blob de dados.

Esse snippet tem os seguintes requisitos:

- Versão mínima AWS IoT Greengrass do núcleo: 1.10.0
- Versão mínima AWS IoT Greengrass do SDK principal: Python: 1.5.0 | Java: 1.4.0 |: 1.6.0 Node.js

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 =
 GreengrassClientBuilder.streamManagerClient().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.
});
```

Node.js Referência do SDK: appendMessage

AWS IoT SiteWise destinos de exportação

O snippet a seguir anexa uma mensagem ao fluxo chamado `StreamName`. Para AWS IoT SiteWise destinos, suas funções do Lambda acrescentam um objeto serializado. `PutAssetPropertyValueEntry` Para obter mais informações, consulte the section called "Exportando para AWS IoT SiteWise".

> **Note**
>
> Quando você envia dados para AWS IoT SiteWise, seus dados devem atender aos requisitos da `BatchPutAssetPropertyValue` ação. Para obter mais informações, consulte BatchPutAssetPropertyValue na Referência de APIs do AWS IoT SiteWise .

Esse snippet tem os seguintes requisitos:

- Versão mínima AWS IoT Greengrass do núcleo: 1.11.0
- Versão mínima AWS IoT Greengrass do SDK principal: Python: 1.6.0 | Java: 1.5.0 |: 1.7.0 Node.js

Python

```
client = StreamManagerClient()

try:
    # SiteWise requires unique timestamps in all messages. 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",
 data=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. 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.
});
```

Node.js Referência do SDK: 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, suas funções do Lambda anexam um objeto `S3ExportTaskDefinition` serializado que contém informações sobre o arquivo de entrada de origem e o objeto do Amazon S3 de destino. Se o objeto especificado não existir, o gerenciador de fluxo criará o objeto para você. Para obter mais informações, consulte the section called “Exportar para o Amazon S3”.

Esse snippet tem os seguintes requisitos:

- Versão mínima AWS IoT Greengrass do núcleo: 1.11.0
- Versão mínima AWS IoT Greengrass do SDK principal: Python: 1.6.0 | Java: 1.5.0 |: 1.7.0 Node.js

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",
 data=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 | S3ExportTaskDefinition

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 | S3ExportTaskDefinition

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.
});
```

Node.js Referência do SDK: appendMessage | S3ExportTaskDefinition

Ler Mensagens

Ler mensagens de um fluxo.

Requisitos

Essa operação tem os seguintes requisitos:

- Versão mínima AWS IoT Greengrass do núcleo: 1.10.0
- Versão mínima AWS IoT Greengrass do SDK principal: Python: 1.5.0 | Java: 1.4.0 |: 1.6.0 Node.js

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 =
 GreengrassClientBuilder.streamManagerClient().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.
});
```

Node.js Referência do SDK: readMessages | ReadMessagesOptions

Listar fluxos

Obtenha a lista de fluxos no gerenciador de fluxos.

Requisitos

Essa operação tem os seguintes requisitos:

- Versão mínima AWS IoT Greengrass do núcleo: 1.10.0
- Versão mínima AWS IoT Greengrass do SDK principal: Python: 1.5.0 | Java: 1.4.0 |: 1.6.0 Node.js

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 =
 GreengrassClientBuilder.streamManagerClient().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.
});
```

Node.js Referência do SDK: 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 AWS IoT Greengrass do núcleo: 1.10.0
- Versão mínima AWS IoT Greengrass do SDK principal: Python: 1.5.0 | Java: 1.4.0 |: 1.6.0 Node.js

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 =
 GreengrassClientBuilder.streamManagerClient().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: descrever MessageStream

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.
});
```

Node.js Referência do SDK: descrever MessageStream

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 Nuvem AWS destino.
- 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.

Suas funções do Lambda seguem esse processo de alto nível para atualizar um fluxo:

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 AWS IoT Greengrass do núcleo: 1.11.0
- Versão mínima AWS IoT Greengrass do SDK principal: Python: 1.6.0 | Java: 1.5.0 |: 1.7.0 Node.js

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: atualização | MessageStream MessageStreamDefinition

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.
});
```

Node.js Referência do SDK: atualização | MessageStream MessageStreamDefinition

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 AWS IoT Greengrass do núcleo: 1.10.0
- Versão mínima AWS IoT Greengrass do SDK principal: Python: 1.5.0 | Java: 1.4.0 |: 1.6.0 Node.js

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: excluir MessageStream

Java

```
try (final StreamManagerClient client =
 GreengrassClientBuilder.streamManagerClient().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.
```

```
});
```

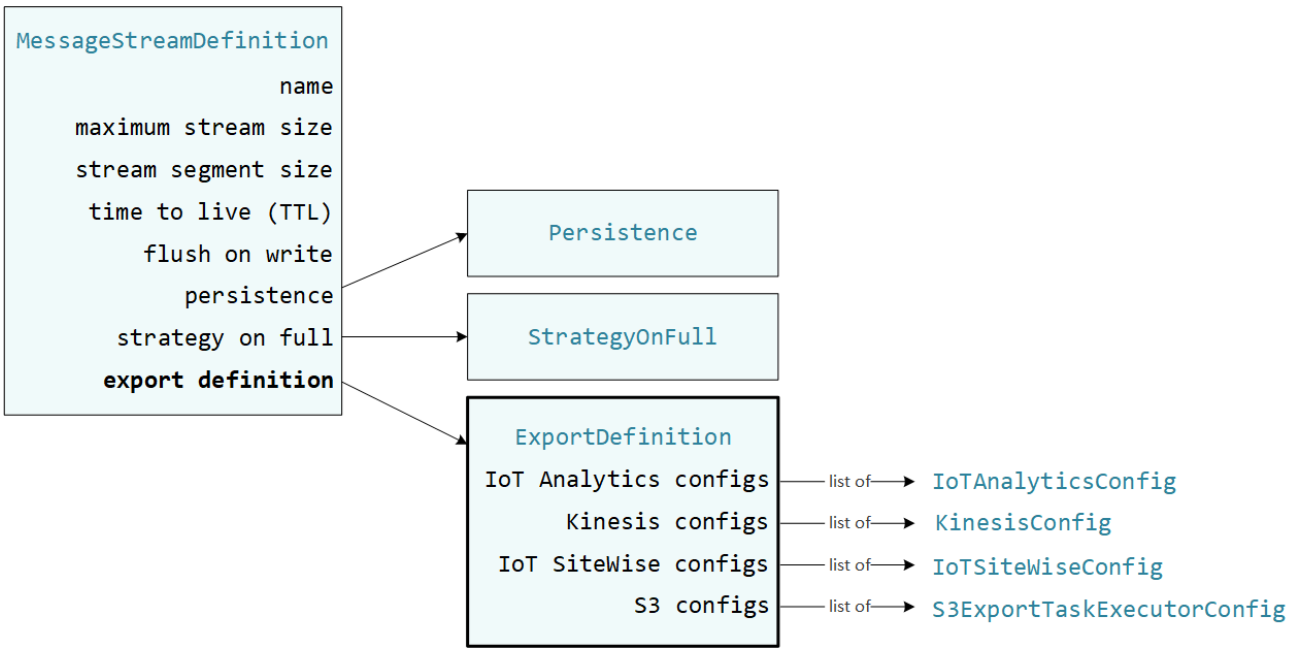
Node.js Referência do SDK: excluir MessageStream

Consulte também

- Gerenciamento de fluxos de dados
- the section called “Configurar o gerenciador de fluxo”
- the section called “Exporte configurações para suporte Nuvem AWS destinos”
- the section called “Exportar streams de dados (console)”
- the section called “Exportar streams de dados (CLI)”
- `StreamManagerClient`na referência do SDK AWS IoT Greengrass principal:
  - Python
  - Java
  - Node.js

Exporte configurações para suporte Nuvem AWS destinos

User-defined As funções do Lambda são usadas `StreamManagerClient` no AWS IoT Greengrass Core SDK para interagir com o gerenciador de streams. Quando uma função do Lambda cria um fluxo ou atualiza um fluxo, ela passa um objeto `MessageStreamDefinition` 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.



É possível 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 stream para dois streams de dados do Kinesis e um bucket do Amazon S3.

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.

Nuvem AWS Destinos compatíveis

- AWS IoT Analytics canais (descontinuados)
- Amazon Kinesis Data Streams
- AWS IoT SiteWise propriedades do ativo
- Objetos do Amazon S3

Você é responsável por manter esses Nuvem AWS recursos.

AWS IoT Analytics canais (descontinuados)

> **AWS IoT Analytics descontinuado**
>
> AWS IoT Analytics foi descontinuado em 15 de dezembro de 2025. As exportações do gerenciador de streams para AWS IoT Analytics canais não funcionam mais. Se você tiver streams configurados com destinos de `IoTAnalyticsConfig` exportação, remova-os para evitar tentativas contínuas de exportação fracassadas.

Para remover as configurações de AWS IoT Analytics exportação de um stream existente, atualize sua definição de stream para omitir a `IoTAnalyticsConfig` entrada. Como alternativa, exclua e recrie o fluxo sem a configuração de AWS IoT Analytics exportação.

Considere migrar para um dos outros destinos de exportação compatíveis, como the section called "Amazon Kinesis Data Streams" outhe section called "Objetos do Amazon S3".

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 de redução de mapas. Para obter mais informações, consulte O que é o Amazon Kinesis Data Streams? no Guia do desenvolvedor do Amazon Kinesis.

No AWS IoT Greengrass Core SDK, suas funções do Lambda usam o `KinesisConfig` 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 Node.js SDK

Requisitos

Esse destino de exportação tem os seguintes requisitos:

- Os streams de destino no Kinesis Data Streams devem estar no mesmo grupo do Conta da AWS Região da AWS Greengrass.
- O the section called “Função do grupo do Greengrass.” deve conceder a permissão `kinesis:PutRecords` para os fluxos de dados de destino. Por exemplo:

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kinesis:PutRecords"
            ],
            "Resource": [
            "arn:aws:kinesis:us-east-1:123456789012:stream/stream_1_name",
    "arn:aws:kinesis:us-east-1:123456789012:stream/stream_2_name"
            ]
        }
    ]
}
```

  É possível 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 fluxo que exporte para o Kinesis Data Streams, suas funções do Lambda criam um fluxo 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 suas funções do Lambda recebem dados de dispositivos, elas acrescentam mensagens que contêm um blob de dados ao fluxo 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.

AWS IoT SiteWise propriedades do ativo

O gerenciador de streams suporta exportações automáticas para AWS IoT SiteWise o. AWS IoT SiteWise permite coletar, organizar e analisar dados de equipamentos industriais em grande escala. Para obter mais informações, consulte O que é AWS IoT SiteWise? no Guia do AWS IoT SiteWise usuário.

No AWS IoT Greengrass Core SDK, suas funções do Lambda usam o `IoTSiteWiseConfig` para definir a configuração de exportação para esse tipo de destino. Para mais informações, consulte a referência do SDK para seu idioma de destino:

- IoTSiteWiseConfigno SDK do Python
- IoTSiteWiseConfigno Java SDK
- IoTSiteWiseConfigno Node.js SDK

> **Note**
>
> AWS também fornece othe section called “IoT SiteWise”, que é uma solução pré-criada que você pode usar com OPC-UA fontes.

Requisitos

Esse destino de exportação tem os seguintes requisitos:

- As propriedades do ativo de destino em AWS IoT SiteWise devem estar no mesmo grupo Conta da AWS e Região da AWS pertencer ao grupo Greengrass.

  > **Note**
  >
  > Para ver a lista de regiões que AWS IoT SiteWise oferecem suporte, consulte AWS IoT SiteWise endpoints e cotas na Referência AWS geral.

- O the section called “Função do grupo do Greengrass.” 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.

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
             "Effect": "Allow",
             "Action": "iotsitewise:BatchPutAssetPropertyValue",
             "Resource": "*",
             "Condition": {
                 "StringLike": {
                     "iotsitewise:assetHierarchyPath": [
                         "/root node asset ID",
                         "/root node asset ID/*"
                     ]
                 }
             }
        }
    ]
}
```

  É possível 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 importantes sobre segurança, consulte a BatchPutAssetPropertyValue autorização no Guia AWS IoT SiteWise do usuário.

Exportando para AWS IoT SiteWise

Para criar um stream que exporte para AWS IoT SiteWise, suas funções do Lambda criam um stream 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 suas funções do Lambda recebem dados de propriedades de ativos de dispositivos, elas anexam mensagens que contêm os dados ao fluxo de destino. As mensagens são JSON-serialized `PutAssetPropertyValueEntry` objetos que contêm valores de propriedade para uma ou mais propriedades do ativo. Para obter mais informações, consulte Anexar mensagem para destinos de exportação do AWS IoT SiteWise .

> **Note**
>
> Quando você envia dados para AWS IoT SiteWise, seus dados devem atender aos requisitos da `BatchPutAssetPropertyValue` ação. Para obter mais informações, consulte BatchPutAssetPropertyValue na Referência de APIs 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 fluxo e a lógica da função do Lambda 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, suas funções do Lambda podem agrupar os pontos de dados de timestamp-quality-value (TQV) recebidos para uma única propriedade do ativo antes de anexar os dados ao fluxo. 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. É possível 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 AWS IoT Greengrass Core SDK, suas funções do Lambda usam o `S3ExportTaskExecutorConfig` para definir a configuração de exportação para esse tipo de destino. Para mais informações, consulte a referência do SDK para seu idioma de destino:

- S3ExportTaskExecutorConfigno SDK do Python
- S3ExportTaskExecutorConfigno Java SDK
- S3ExportTaskExecutorConfigno Node.js SDK

Requisitos

Esse destino de exportação tem os seguintes requisitos:

- Os buckets do Amazon S3 de destino devem estar no mesmo grupo Conta da AWS do Greengrass.
- Se a conteinerização padrão para o grupo do Greengrass for contêiner do Greengrass, você deverá definir o parâmetro STREAM_MANAGER_READ_ONLY_DIRS para usar um diretório de arquivos de entrada que esteja sob `/tmp` ou não faça parte do sistema de arquivos raiz.
- Se uma função do Lambda em execução no modo de contêiner do Greengrass gravar arquivos de entrada no diretório de arquivos de entrada, você deverá criar um recurso de volume local para o diretório e montar o diretório no contêiner com permissões de gravação. Isso garante que os arquivos sejam gravados no sistema de arquivos raiz e sejam visíveis fora do contêiner. Para obter mais informações, consulte Acesso aos recursos locais.
- O the section called “Função do grupo do Greengrass.” deve conceder as permissões a seguir para os buckets de destino. Por exemplo:

  JSON

```
{
    "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/*"
            ]
        }
    ]
}
```

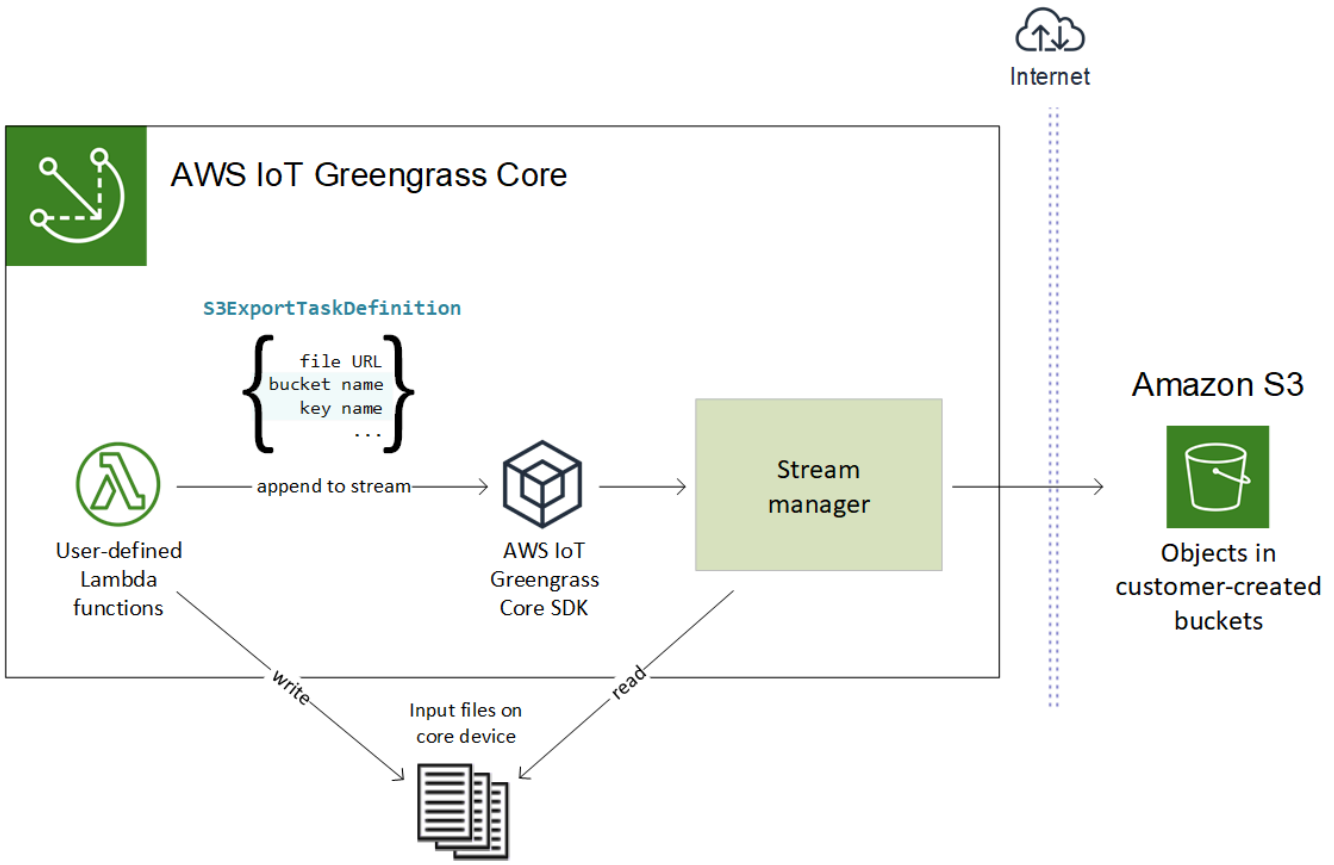
É possível conceder acesso granular ou condicional aos recursos, por exemplo, usando um esquema de nomeação * curinga. Para obter mais informações, consulte Adicionando e removendo políticas do IAM no Guia do usuário do IAM.

Exportar para o Amazon S3

Para criar um fluxo que exporte para o Amazon S3, suas funções do Lambda usam o objeto `S3ExportTaskExecutorConfig` 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, suas funções do Lambda anexam uma tarefa de exportação ao fluxo 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 balde alvo já deve existir em seu Conta da AWS. Se um objeto para a chave especificada não existir, o gerenciador de fluxo criará o objeto para você.

Esse fluxo de alto nível é mostrado no diagrama a seguir.



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 do Amazon S3 de destino podem incluir DateTimeFormatter strings Java válidas em `!{timestamp:value}` espaços reservados. É possível 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 obter 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 as funções do Lambda 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.

   > **Note**
   >
   > Para determinar se você deve configurar o acesso ao diretório de arquivos de entrada, consulte o parâmetro STREAM_MANAGER_READ_ONLY_DIRS.
   > O processo no qual o gerenciador de fluxo é executado herda todas as permissões do sistema de arquivos da identidade de acesso padrão do grupo. O gerenciador de fluxo deve ter permissão para acessar os arquivos de entrada. Você pode usar o comando `chmod(1)` para alterar a permissão dos arquivos, se necessário.

2. Uma função do Lambda verifica o diretório e anexa uma tarefa de exportação ao fluxo de destino quando um novo arquivo é criado. A tarefa é um JSON-serialized `S3ExportTaskDefinition` objeto que especifica a URL do arquivo de entrada, o bucket e a chave do Amazon S3 de destino e os 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 balde alvo já deve existir em seu Conta da AWS. Se um objeto para a chave especificada não existir, o gerenciador de fluxo criará o objeto para você.
4. A função do Lambda lê mensagens de um fluxo de status para monitorar o status da exportação. Depois que as tarefas de exportação forem concluídas, a função do Lambda poderá excluir os arquivos de entrada correspondentes. Para obter 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. Suas funções do Lambda devem criar um fluxo de status e, em seguida, 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. É possível 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. Isso faz com que o gerenciador de fluxo grave 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 com suporte 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 as funções do Lambda podem usar um fluxo de status para monitorar o status de exportação.

1. Conforme descrito no fluxo de trabalho anterior, uma função do Lambda 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. Uma função do Lambda lê mensagens sequencialmente do fluxo de status e, em seguida, filtra as mensagens com base no nome do fluxo e na ID da tarefa ou com base em uma propriedade da tarefa de exportação do contexto da mensagem. Por exemplo, a função do Lambda pode filtrar pela URL do arquivo de entrada da tarefa de exportação, que é representada pelo objeto `S3ExportTaskDefinition` 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 a vida útil (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 interrompido retorna um aviso.

3. Depois que as tarefas de exportação forem concluídas, a função do Lambda poderá excluir os arquivos de entrada correspondentes.

O exemplo a seguir mostra como uma função do Lambda pode ler e processar mensagens de status.

Python

```
import time
from greengrasssdk.stream_manager import (
    ReadMessagesOptions,
    Status,
    StatusConfig,
    StatusLevel,
    StatusMessage,
    StreamManagerClient,
)
from greengrasssdk.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.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 =
 GreengrassClientBuilder.streamManagerClient().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 Java SDK: readMessages | StatusMessage

Node.js

```
const {
    StreamManagerClient, ReadMessagesOptions,
    Status, StatusConfig, StatusLevel, StatusMessage,
    util,
} = require('aws-greengrass-core-sdk').StreamManager;

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) => {
```

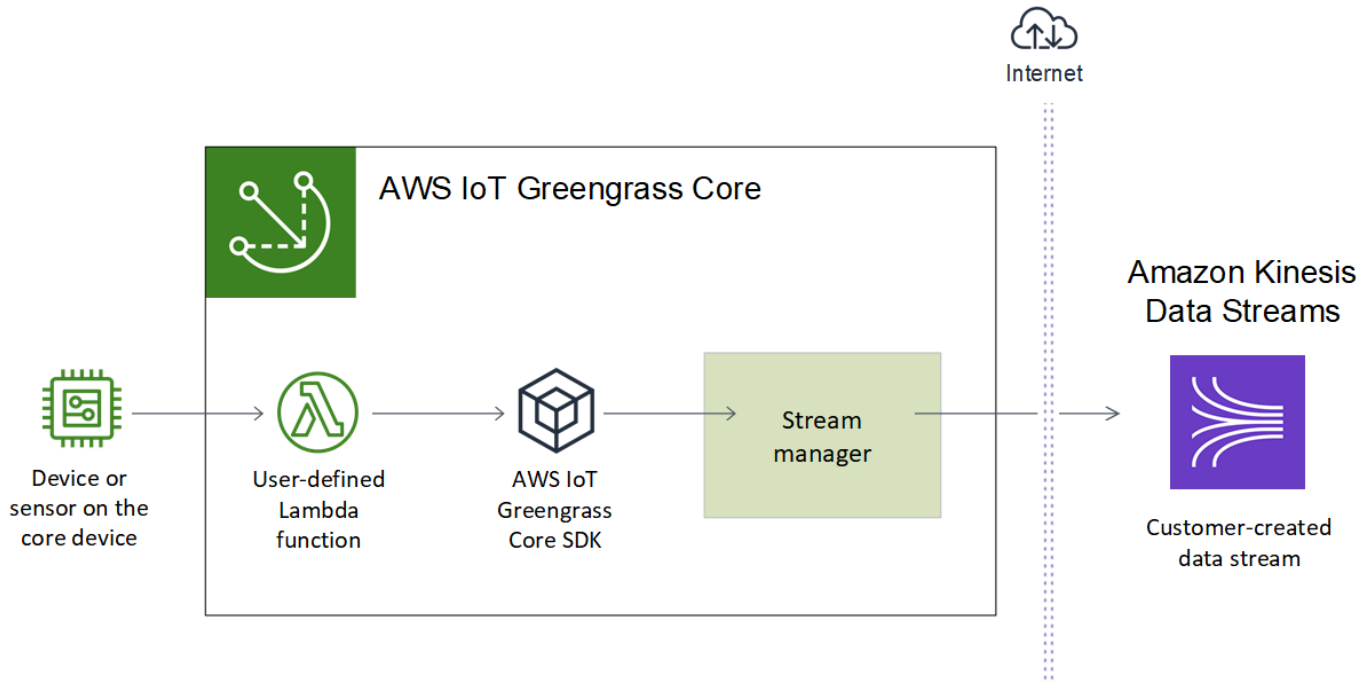
```
                    // Deserialize the status message first.
                    const statusMessage =
 util.deserializeJsonBytesToObj(message.payload, StatusMessage);
                    // Check the status of the status message. If the status is
 'Success', the file was successfully uploaded to 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.
});
```

Node.js Referência do SDK: readMessages | StatusMessage

Exportar fluxos de dados para o Nuvem AWS (console)

Este tutorial mostra como usar o AWS IoT console para configurar e implantar um AWS IoT Greengrass grupo com o gerenciador de stream ativado. O grupo contém uma função do Lambda definida pelo usuário que grava em um fluxo no gerenciador de fluxo e é exportado automaticamente para a Nuvem AWS.

O gerenciador de fluxo torna mais eficientes e confiáveis a ingestão, o processamento e a exportação de fluxos de dados de alto volume. Neste tutorial, você criará uma função do Lambda `TransferStream` que consome dados de IoT. A função Lambda usa o AWS IoT Greengrass Core SDK para criar um stream no gerenciador de streams e depois ler e gravar nele. Em seguida, o gerenciador de fluxo exporta o fluxo para o Kinesis Data Streams. O diagrama a seguir mostra esse fluxo de trabalho.



O foco deste tutorial é mostrar como as funções do Lambda definidas pelo usuário usam o `StreamManagerClient` objeto no AWS IoT Greengrass Core SDK para interagir com o gerenciador de streams. Para simplificar, a função do Lambda em Python que você cria para este tutorial gera dados simulados do dispositivo.

Pré-requisitos

Para concluir este tutorial, é necessário:

- Um grupo do Greengrass e um núcleo do Greengrass (versão 1.10 ou posterior). Para obter informações sobre como criar um grupo e um núcleo do Greengrass, consulte Começando com AWS IoT Greengrass. O tutorial de introdução também inclui etapas para instalar o software AWS IoT Greengrass Core.

  > **Note**
  >
  > O gerenciador de streams não é suportado em OpenWrt distribuições.

- O Java 8 Runtime (JDK 8) instalado no dispositivo de núcleo.
  - Para Debian-based distribuições (incluindo Raspbian) ou Ubuntu-based distribuições, execute o seguinte comando:

    ```
    sudo apt install openjdk-8-jdk
    ```

  - Para Hat-based distribuições Red (incluindo Amazon Linux), execute o seguinte comando:

    ```
    sudo yum install java-1.8.0-openjdk
    ```

    Para obter mais informações, consulte Como fazer download e instalar pacotes OpenJDK pré-compilados na documentação do OpenJDK.
- AWS IoT Greengrass SDK principal para Python v1.5.0 ou posterior. Para usar `StreamManagerClient` no SDK do AWS IoT Greengrass Core para Python, você deve:
  - Instalar o Python 3.7 ou posterior no dispositivo de núcleo.
  - Incluir o SDK e suas dependências em seu pacote de implantação da função do Lambda. As instruções são fornecidas neste tutorial.

  > **Tip**
  >
  > Você pode usar o `StreamManagerClient` com Java ou NodeJS. Por exemplo, código, consulte o AWS IoT Greengrass Core SDK para Java e o AWS IoT Greengrass Core SDK para Node.js on. GitHub

- Um stream de destino chamado **`MyKinesisStream`** criado no Amazon Kinesis Data Streams da Região da AWS mesma forma que seu grupo do Greengrass. Para obter mais informações, consulte Criar um fluxo no Guia do desenvolvedor do Amazon Kinesis.

> **Note**
>
> Neste tutorial, o gerenciador de fluxo exporta dados para o Kinesis Data Streams, o que resulta em cobranças em sua Conta da AWS. Para obter informações sobre a definição de preços, consulte Definição de preço do Kinesis Data Streams.
> Para evitar incorrer em cobranças, você pode executar este tutorial sem criar um fluxo de dados do Kinesis. Nesse caso, verifique os logs para confirmar se o gerenciador de fluxo tentou exportar o fluxo para o Kinesis Data Streams.

- Uma política do IAM adicionada à the section called “Função do grupo do Greengrass.” que permite a ação `kinesis:PutRecords` no fluxo de dados de destino, conforme mostrado no exemplo a seguir:

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kinesis:PutRecords"
            ],
            "Resource": [
            "arn:aws:kinesis:us-east-1:123456789012:stream/MyKinesisStream"
            ]
        }
    ]
}
```

O tutorial contém as seguintes etapas de nível elevado:

1. Crie um pacote de implantação para a função do Lambda
2. Criar uma função do Lambda
3. Adicionar uma função ao grupo
4. Habilitar o gerenciador de fluxo
5. Configurar o registro em log local

6. Implantar o grupo
7. Teste a aplicação

O tutorial levará aproximadamente 20 minutos para ser concluído.

Etapa 1: crie um pacote de implantação para a função do Lambda

Nesta etapa, você cria um pacote de implantação da função do Lambda que contém o código e as dependências da função do Python. Faça upload desse pacote posteriormente, ao criar a função do Lambda no AWS Lambda. A função Lambda usa o AWS IoT Greengrass Core SDK para criar e interagir com streams locais.

> **Note**
>
> As funções do Lambda definidas pelo usuário devem usar o SDK do AWS IoT Greengrass Core para interagir com o gerenciador de fluxo. Para obter mais informações sobre os requisitos para o gerenciador de fluxo do Greengrass, consulte os requisitos do gerenciador de fluxo do Greengrass.

1. Baixe o SDK do AWS IoT Greengrass Core para Python v1.5.0 ou posterior.
2. Descompacte o pacote obtido por download para obter o SDK. O SDK é a pasta do `greengrasssdk`.
3. Instale as dependências do pacote para serem incluídas com o SDK no pacote de implantação da função do Lambda.
   1. Navegue até o diretório SDK que contém o arquivo `requirements.txt`. Esse arquivo lista as dependências.
   2. Instale as dependências do SDK. Por exemplo, execute o seguinte comando `pip` para instalar as dependências no diretório atual:

      ```
      pip install --target . -r requirements.txt
      ```
4. Salve a seguinte função do código Python em um arquivo local denominado `transfer_stream.py`.

> **Tip**
>
> Por exemplo, código que usa Java e NodeJS, consulte o AWS IoT Greengrass Core SDK para Java e o AWS IoT Greengrass Core SDK para on. Node.js GitHub

```
import asyncio
import logging
import random
import time

from greengrasssdk.stream_manager import (
    ExportDefinition,
    KinesisConfig,
    MessageStreamDefinition,
    ReadMessagesOptions,
    ResourceNotFoundException,
    StrategyOnFull,
    StreamManagerClient,
)


# This example creates a local stream named "SomeStream".
# It starts writing data into that stream and then stream manager automatically
 exports
# the data to a customer-created Kinesis data stream named "MyKinesisStream".
# This example runs forever until the program is stopped.

# The size of the local stream on disk will not exceed the default (which is 256
 MB).
# Any data appended after the stream reaches the size limit continues to be
 appended, and
# stream manager deletes the oldest data until the total stream size is back under
 256 MB.
# The Kinesis data stream in the cloud has no such bound, so all the data from this
 script is
# uploaded to Kinesis and you will be charged for that usage.


def main(logger):
    try:
```

```
        stream_name = "SomeStream"
        kinesis_stream_name = "MyKinesisStream"

        # Create a client for the StreamManager
        client = StreamManagerClient()

        # Try deleting the stream (if it exists) so that we have a fresh start
        try:
            client.delete_message_stream(stream_name=stream_name)
        except ResourceNotFoundException:
            pass

        exports = ExportDefinition(
            kinesis=[KinesisConfig(identifier="KinesisExport" + stream_name,
kinesis_stream_name=kinesis_stream_name)]
        )
        client.create_message_stream(
            MessageStreamDefinition(
                name=stream_name,
strategy_on_full=StrategyOnFull.OverwriteOldestData, export_definition=exports
            )
        )

        # Append two messages and print their sequence numbers
        logger.info(
            "Successfully appended message to stream with sequence number %d",
            client.append_message(stream_name, "ABCDEFGHIJKLMNO".encode("utf-8")),
        )
        logger.info(
            "Successfully appended message to stream with sequence number %d",
            client.append_message(stream_name, "PQRSTUVWXYZ".encode("utf-8")),
        )

        # Try reading the two messages we just appended and print them out
        logger.info(
            "Successfully read 2 messages: %s",
            client.read_messages(stream_name,
ReadMessagesOptions(min_message_count=2, read_timeout_millis=1000)),
        )

        logger.info("Now going to start writing random integers between 0 and 1000
to the stream")
        # Now start putting in random data between 0 and 1000 to emulate device
sensor input
```

```
        while True:
            logger.debug("Appending new random integer to stream")
            client.append_message(stream_name, random.randint(0,
 1000).to_bytes(length=4, signed=True, byteorder="big"))
            time.sleep(1)

    except asyncio.TimeoutError:
        logger.exception("Timed out while executing")
    except Exception:
        logger.exception("Exception while running")


def function_handler(event, context):
    return


logging.basicConfig(level=logging.INFO)
# Start up this sample code
main(logger=logging.getLogger())
```

5. Comprima os seguintes itens em um arquivo denominado `transfer_stream_python.zip`. Esse é o pacote de implantação de sua função do Lambda.

   - transfer_stream.py. Lógica do aplicativo.
   - greengrasssdk. Biblioteca necessária para funções Python do Lambda do Greengrass que publicam mensagens MQTT.

     ???As operações do gerenciador de streams estão disponíveis na versão 1.5.0 ou posterior do AWS IoT Greengrass Core SDK para Python.
   - As dependências que você instalou para o AWS IoT Greengrass Core SDK para Python (por exemplo, os `cbor2` diretórios).

   Ao criar o arquivo `zip`, inclua apenas esses itens, não a pasta que contém os arquivos.

Etapa 2: Criar uma função do Lambda

Nesta etapa, você usa o AWS Lambda console para criar uma função do Lambda e configurá-la para usar seu pacote de implantação. Depois, publique uma versão da função e crie um alias.

1. Primeiro, crie a função do Lambda.

a. No Console de gerenciamento da AWS, escolha Serviços e abra o AWS Lambda console.

b. Selecione Criar função e, em seguida, selecione Criar do zero.

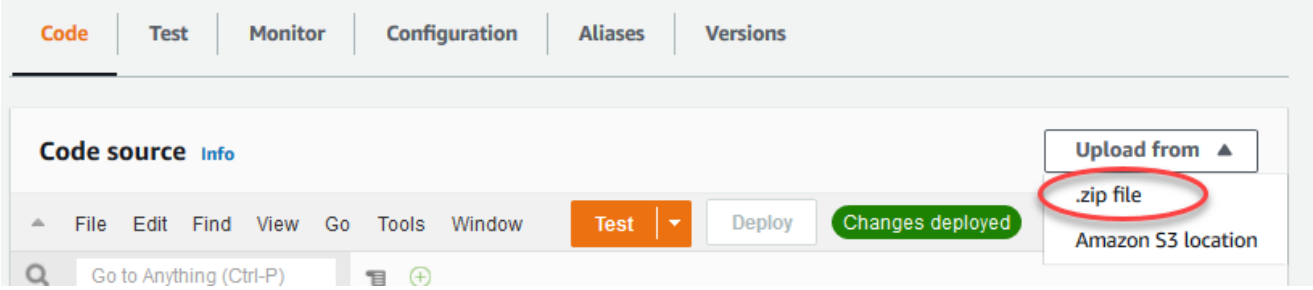
c. Na seção Basic information (Informações básicas), use os seguintes valores:

- Em Function name (Nome da função), insira **`TransferStream`**.
- Em Runtime (Tempo de execução), selecione Python 3.7.
- Em Permissões, mantenha a configuração padrão. Isso cria uma função de execução que concede permissões básicas do Lambda. Essa função não é usada por AWS IoT Greengrass.

d. Na parte inferior da página, selecione Create function.

2. Em seguida, registre o manipulador e faça upload do seu pacote de implantação da função do Lambda.

a. Na guia Código, em Fonte do código, selecione Fazer upload a partir de. No menu suspenso, selecione o arquivo .zip.



b. Selecione Upload e, em seguida, selecione seu pacote de implantação `transfer_stream_python.zip`. Selecione Salvar.

c. Na guia Código da função, em Configurações de runtime, selecione Editar e insira os valores a seguir.

- Em Runtime (Tempo de execução), selecione Python 3.7.
- Em Handler (Manipulador), insira **`transfer_stream.function_handler`**.

d. Escolha Salvar.

> **Note**
>
> O botão Testar no AWS Lambda console não funciona com essa função. O SDK AWS IoT Greengrass Core não contém módulos necessários para executar suas funções do Greengrass Lambda de forma independente no console. AWS Lambda

Esses módulos (por exemplo, `greengrass_common`) são fornecidos às funções depois de serem implantados no núcleo do Greengrass.

3. Agora, publique a primeira versão da sua função do Lambda e crie um alias para a versão.

> **Note**
>
> Os grupos do Greengrass podem fazer referência a uma função do Lambda por alias (recomendado) ou por versão. Usar um alias facilita o gerenciamento de atualizações de código porque você não precisa alterar a tabela de assinaturas nem a definição do grupo ao atualizar a função do código. Em vez disso, você pode simplesmente apontar o alias para a nova versão da função.

   a. No menu Actions, selecione Publish new version.
   b. Em Version description (Descrição da versão), insira **`First version`** e, em seguida, selecione Publish (Publicar).
   c. Na página de configuração TransferStream: 1, no menu Ações, escolha Criar alias.
   d. Na página Create a new alias, use os seguintes valores:
      - Em Nome, digite **`GG_TransferStream`**.
      - Em Version, selecione 1.

      > **Note**
      >
      > AWS IoT Greengrass não oferece suporte a aliases do Lambda para as versões $LATEST.

   e. Selecione Create (Criar).

Agora você está pronto para adicionar a função do Lambda ao seu grupo do Greengrass.

Etapa 3: Adicionar uma função do Lambda ao grupo do Greengrass

Nesta etapa, você adiciona a função do Lambda ao grupo e configura o ciclo de vida e as variáveis de ambiente. Para obter mais informações, consulte the section called “Controlando a execução da função do Lambda do Greengrass”.

1. No painel de navegação do AWS IoT console, em Gerenciar, expanda dispositivos Greengrass e escolha Grupos (V1).
2. Selecione o grupo de destino.
3. Na página de configuração do grupo, selecione a guia Funções do Lambda.
4. Em Minhas funções do Lambda, selecione Adicionar.
5. Na página Adicionar função Lambda, selecione a função do Lambda para sua função do Lambda.
6. Para a versão Lambda, escolha Alias:GG _ TransferStream.

   Agora, configure propriedades que determinem o comportamento da função do Lambda no grupo do Greengrass.
7. Na seção Configuração da função do Lambda, faça as seguintes alterações:

   - Defina o Memory limit (Limite de memória) como 32 MB.
   - Para Fixado, selecione Verdadeiro.

   > **Note**
   >
   > Uma função Lambda de longa duração (ou fixada) é iniciada automaticamente após o AWS IoT Greengrass início e continua sendo executada em seu próprio contêiner. Isso contrasta com uma função do Lambda sob demanda, que é iniciada quando invocada e interrompida quando não há tarefas a serem executadas. Para obter mais informações, consulte the section called “Configuração do ciclo de vida”.
8. Selecione Adicionar função do Lambda.

Etapa 4: Habilitar o gerenciador de fluxo

Nesta etapa, verifique se o gerenciador de fluxo está habilitado.

1. Na página de configuração do grupo, selecione a guia Funções do Lambda.
2. Em Funções do Lambda do sistema, selecione Gerenciador de fluxo e verifique o status. Se estiver desabilitado, selecione Edit (Editar). Em seguida, selecione Enable (Habilitar) e Salvar. Você pode usar as configurações de parâmetro padrão para este tutorial. Para obter mais informações, consulte the section called “Configurar o gerenciador de fluxo”.

> **Note**
>
> Quando você usa o console para habilitar o gerenciador de fluxo e implantar o grupo, o limite de memória para o gerenciador de fluxo é definido como 4194304 KB (4 GB), por padrão. É recomendável definir o tamanho da memória para pelo menos 128000 KB.

Etapa 5: Configurar o registro em log local

Nesta etapa, você configura componentes AWS IoT Greengrass do sistema, funções do Lambda definidas pelo usuário e conectores no grupo para gravar registros no sistema de arquivos do dispositivo principal. Você pode usar logs para solucionar quaisquer problemas que possa encontrar. Para obter mais informações, consulte the section called “Monitoramento com logs do AWS IoT Greengrass ”.

1. Em Local logs configuration (Configuração dos logs locais), verifique se o log local está configurado.
2. Se os logs não estiverem configurados para os componentes do sistema do Greengrass ou as funções do Lambda definidas pelo usuário, selecione Editar.
3. Selecione Nível de log das funções do Lambda e o Nível de log do sistema Greengrass.
4. Mantenha os valores padrão para o nível de registro em log e o limite de espaço em disco e selecione Salvar.

Etapa 6: Implantar o grupo do Greengrass

Implante o grupo no dispositivo do núcleo.

1. Certifique-se de que o AWS IoT Greengrass núcleo esteja funcionando. Execute os seguintes comandos no seu terminal do Raspberry Pi, conforme necessário.

a. Para verificar se o daemon está em execução:

```
ps aux | grep -E 'greengrass.*daemon'
```

Se a saída contém uma entrada `root` para `/greengrass/ggc/packages/`*`ggc-version`*`/bin/daemon`, o daemon está em execução.

> **Note**
>
> A versão no caminho depende da versão do software AWS IoT Greengrass Core instalada em seu dispositivo principal.

b. Para iniciar o daemon:

```
cd /greengrass/ggc/core/
sudo ./greengrassd start
```

2. Na página de configuração do grupo, selecione Implantar.
3. a. Na guia Funções do Lambda, na seção Funções do Lambda do sistema, selecione Detector de IP e selecione Editar.

   b. Na caixa de diálogo Editar configurações do detector IP, selecione Detectar e substituir automaticamente os endpoints do corretor MQTT.

   c. Selecione Salvar.

   Isso permite que os dispositivos adquiram automaticamente as informações de conectividade para o núcleo, como endereço IP, DNS e o número da porta. A detecção automática é recomendada, mas AWS IoT Greengrass também oferece suporte a endpoints especificados manualmente. Você só é solicitado pelo método de descoberta na primeira vez em que o grupo for implantado.

   > **Note**
   >
   > Se solicitado, conceda permissão para criar a função de serviço do Greengrass e associá-la à sua Conta da AWS na atual. Região da AWS Essa função permite AWS IoT Greengrass acessar seus recursos em AWS serviços.

A página Deployments mostra a data e hora, ID de versão e status da implantação. Quando concluída, o status exibido para a implantação deve ser Concluída.

Para obter ajuda sobre a solução de problemas, consulte Solução de problemas.

Etapa 7: Testar o aplicativo

A função do Lambda do `TransferStream` gera dados simulados do dispositivo. Ela grava dados em um fluxo que o gerenciador de fluxo exporta para o fluxo de dados do Kinesis de destino.

1. No console do Amazon Kinesis, em Streams de dados do Kinesis, escolha MyKinesisStream.

> **Note**
>
> Se você executou o tutorial sem um fluxo de dados do Kinesis de destino, verifique o arquivo de log do gerenciador de fluxo (`GGStreamManager`). Se ele contiver `export stream MyKinesisStream doesn't exist` em uma mensagem de erro, o teste será bem-sucedido. Esse erro significa que o serviço tentou exportar para o fluxo, mas o fluxo não existe.

2. Na página MyKinesisStream, escolha Monitoramento. Se o teste for bem-sucedido, você verá os dados nos gráficos Put Records (Inserir registros) . Dependendo da sua conexão, pode demorar um minuto até que os dados sejam exibidos.

> **Important**
>
> Ao terminar o teste, exclua o fluxo de dados do Kinesis para evitar mais cobranças. Ou execute o comando a seguir para interromper o daemon do Greengrass. Isso impedirá que o núcleo envie mensagens até que você esteja pronto para dar continuidade aos testes.
>
> ```
> cd /greengrass/ggc/core/
> sudo ./greengrassd stop
> ```

3. Remova a função TransferStream Lambda do núcleo.

a. No painel de navegação do AWS IoT console, em Gerenciar, expanda dispositivos Greengrass e escolha Grupos (V1).
b. Em Grupos do Greengrass, selecione seu grupo.
c. Na página Lambdas, escolha as elipses (...) para a TransferStream função e, em seguida, escolha Remover função.
d. Em Actions (Ações), selecione Deploy (Implantar).

Para exibir informações de registro ou solucionar problemas com fluxos, verifique os logs das funções `TransferStream` e `GGStreamManager`. Você deve ter `root` permissões para ler AWS IoT Greengrass os registros no sistema de arquivos.

- `TransferStream` grava entradas de log em *`greengrass-root`*`/ggc/var/log/user/`*`region`*`/`*`account-id`*`/TransferStream.log`.
- `GGStreamManager` grava entradas de log em *`greengrass-root`*`/ggc/var/log/system/GGStreamManager.log`.

Se precisar de mais informações sobre a solução de problemas, você pode definir o nível de registro dos logs do usuário do Lambda como logs de depuração e, em seguida, implantar o grupo novamente.

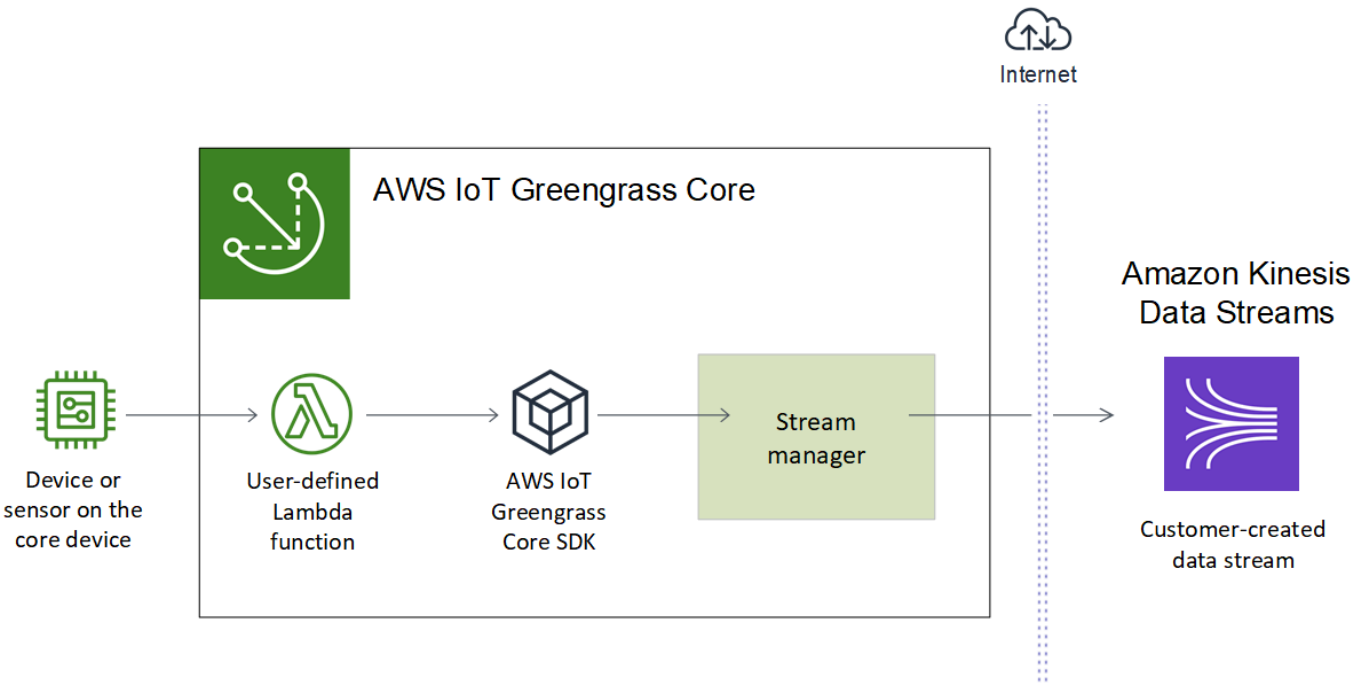
Consulte também

- Gerenciamento de fluxos de dados
- the section called “Configurar o gerenciador de fluxo”
- the section called “Use StreamManagerClient para trabalhar com streams”
- the section called “Exporte configurações para suporte Nuvem AWS destinos”
- the section called “Exportar streams de dados (CLI)”

Exportar fluxos de dados para o Nuvem AWS (CLIPE)

Este tutorial mostra como usar o para configurar e AWS CLI implantar um AWS IoT Greengrass grupo com o gerenciador de stream ativado. O grupo contém uma função do Lambda definida pelo usuário que grava em um fluxo no gerenciador de fluxo e é exportado automaticamente para a Nuvem AWS.

O gerenciador de fluxo torna mais eficientes e confiáveis a ingestão, o processamento e a exportação de fluxos de dados de alto volume. Neste tutorial, você criará uma função do Lambda `TransferStream` que consome dados de IoT. A função Lambda usa o AWS IoT Greengrass Core SDK para criar um stream no gerenciador de streams e depois ler e gravar nele. Em seguida, o gerenciador de fluxo exporta o fluxo para o Kinesis Data Streams. O diagrama a seguir mostra esse fluxo de trabalho.



O foco deste tutorial é mostrar como as funções do Lambda definidas pelo usuário usam o `StreamManagerClient` objeto no AWS IoT Greengrass Core SDK para interagir com o gerenciador de streams. Para simplificar, a função do Lambda em Python que você cria para este tutorial gera dados simulados do dispositivo.

Quando você usa a AWS IoT Greengrass API, que inclui os comandos do Greengrass no AWS CLI, para criar um grupo, o gerenciador de streams é desativado por padrão. Para habilitar o gerenciador de fluxo em seu núcleo, crie uma versão de definição de função que inclua a função do Lambda do sistema `GGStreamManager` e uma versão de grupo que faça referência à nova versão de definição de função. Depois, implante o grupo.

Pré-requisitos

Para concluir este tutorial, é necessário:

- Um grupo do Greengrass e um núcleo do Greengrass (versão 1.10 ou posterior). Para obter informações sobre como criar um grupo e um núcleo do Greengrass, consulte Começando com AWS IoT Greengrass. O tutorial de introdução também inclui etapas para instalar o software AWS IoT Greengrass Core.

  > **Note**
  >
  > O gerenciador de streams não é suportado em OpenWrt distribuições.

- O Java 8 Runtime (JDK 8) instalado no dispositivo de núcleo.
  - Para Debian-based distribuições (incluindo Raspbian) ou Ubuntu-based distribuições, execute o seguinte comando:

    ```
    sudo apt install openjdk-8-jdk
    ```

  - Para Hat-based distribuições Red (incluindo Amazon Linux), execute o seguinte comando:

    ```
    sudo yum install java-1.8.0-openjdk
    ```

    Para obter mais informações, consulte Como fazer download e instalar pacotes OpenJDK pré-compilados na documentação do OpenJDK.
- AWS IoT Greengrass SDK principal para Python v1.5.0 ou posterior. Para usar `StreamManagerClient` no SDK do AWS IoT Greengrass Core para Python, você deve:
  - Instalar o Python 3.7 ou posterior no dispositivo de núcleo.
  - Incluir o SDK e suas dependências em seu pacote de implantação da função do Lambda. As instruções são fornecidas neste tutorial.

  > **Tip**
  >
  > Você pode usar o `StreamManagerClient` com Java ou NodeJS. Por exemplo, código, consulte o AWS IoT Greengrass Core SDK para Java e o AWS IoT Greengrass Core SDK para Node.js on. GitHub

- Um stream de destino chamado **`MyKinesisStream`** criado no Amazon Kinesis Data Streams da Região da AWS mesma forma que seu grupo do Greengrass. Para obter mais informações, consulte Criar um fluxo no Guia do desenvolvedor do Amazon Kinesis.

> **Note**
>
> Neste tutorial, o gerenciador de fluxo exporta dados para o Kinesis Data Streams, o que resulta em cobranças em sua Conta da AWS. Para obter informações sobre a definição de preços, consulte Definição de preço do Kinesis Data Streams.
> Para evitar incorrer em cobranças, você pode executar este tutorial sem criar um fluxo de dados do Kinesis. Nesse caso, verifique os logs para confirmar se o gerenciador de fluxo tentou exportar o fluxo para o Kinesis Data Streams.

- Uma política do IAM adicionada à the section called “Função do grupo do Greengrass.” que permite a ação `kinesis:PutRecords` no fluxo de dados de destino, conforme mostrado no exemplo a seguir:

  JSON

  ```
  {
      "Version":"2012-10-17",
      "Statement": [
          {
              "Effect": "Allow",
              "Action": [
                  "kinesis:PutRecords"
              ],
              "Resource": [
              "arn:aws:kinesis:us-east-1:123456789012:stream/MyKinesisStream"
              ]
          }
      ]
  }
  ```

- O AWS CLI instalado e configurado em seu computador. Para obter mais informações, consulte o Instalando o AWS Command Line Interface e Configurando o AWS CLI no Guia do usuário da AWS Command Line Interface .

  Os comandos de exemplo neste tutorial foram escritos para Linux e outros Unix-based sistemas. Se você estiver usando o Windows, consulte Especificação de valores de parâmetros para a interface da linha de AWS comando para obter mais informações sobre diferenças na sintaxe.

Se o comando contém uma string JSON, o tutorial fornece um exemplo que tem o JSON em uma única linha. Em alguns sistemas, pode ser mais eficiente editar e executar comandos usando esse formato.

O tutorial contém as seguintes etapas de nível elevado:

1. Crie um pacote de implantação para a função do Lambda
2. Criar uma função do Lambda
3. Criar uma definição e uma versão de função
4. Criar uma definição e versão do logger
5. Obter o ARN da sua versão de definição de núcleo
6. Criar uma versão de grupo
7. Crie um implantação do .
8. Teste a aplicação

O tutorial levará aproximadamente 30 minutos para ser concluído.

Etapa 1: crie um pacote de implantação para a função do Lambda

Nesta etapa, você cria um pacote de implantação da função do Lambda que contém o código e as dependências da função do Python. Faça upload desse pacote posteriormente, ao criar a função do Lambda no AWS Lambda. A função Lambda usa o AWS IoT Greengrass Core SDK para criar e interagir com streams locais.

> **Note**
>
> As funções do Lambda definidas pelo usuário devem usar o SDK do AWS IoT Greengrass Core para interagir com o gerenciador de fluxo. Para obter mais informações sobre os requisitos para o gerenciador de fluxo do Greengrass, consulte os requisitos do gerenciador de fluxo do Greengrass.

1. Baixe o SDK do AWS IoT Greengrass Core para Python v1.5.0 ou posterior.

2. Descompacte o pacote obtido por download para obter o SDK. O SDK é a pasta do `greengrasssdk`.
3. Instale as dependências do pacote para serem incluídas com o SDK no pacote de implantação da função do Lambda.

   1. Navegue até o diretório SDK que contém o arquivo `requirements.txt`. Esse arquivo lista as dependências.
   2. Instale as dependências do SDK. Por exemplo, execute o seguinte comando `pip` para instalar as dependências no diretório atual:

      ```
      pip install --target . -r requirements.txt
      ```

4. Salve a seguinte função do código Python em um arquivo local denominado `transfer_stream.py`.

   > **Tip**
   >
   > Por exemplo, código que usa Java e NodeJS, consulte o AWS IoT Greengrass Core SDK para Java e o AWS IoT Greengrass Core SDK para on. Node.js GitHub

   ```
   import asyncio
   import logging
   import random
   import time

   from greengrasssdk.stream_manager import (
       ExportDefinition,
       KinesisConfig,
       MessageStreamDefinition,
       ReadMessagesOptions,
       ResourceNotFoundException,
       StrategyOnFull,
       StreamManagerClient,
   )


   # This example creates a local stream named "SomeStream".
   # It starts writing data into that stream and then stream manager automatically
    exports
   ```

```
# the data to a customer-created Kinesis data stream named "MyKinesisStream".
# This example runs forever until the program is stopped.

# The size of the local stream on disk will not exceed the default (which is 256
 MB).
# Any data appended after the stream reaches the size limit continues to be
 appended, and
# stream manager deletes the oldest data until the total stream size is back under
 256 MB.
# The Kinesis data stream in the cloud has no such bound, so all the data from this
 script is
# uploaded to Kinesis and you will be charged for that usage.


def main(logger):
    try:
        stream_name = "SomeStream"
        kinesis_stream_name = "MyKinesisStream"

        # Create a client for the StreamManager
        client = StreamManagerClient()

        # Try deleting the stream (if it exists) so that we have a fresh start
        try:
            client.delete_message_stream(stream_name=stream_name)
        except ResourceNotFoundException:
            pass

        exports = ExportDefinition(
            kinesis=[KinesisConfig(identifier="KinesisExport" + stream_name,
 kinesis_stream_name=kinesis_stream_name)]
        )
        client.create_message_stream(
            MessageStreamDefinition(
                name=stream_name,
 strategy_on_full=StrategyOnFull.OverwriteOldestData, export_definition=exports
            )
        )

        # Append two messages and print their sequence numbers
        logger.info(
            "Successfully appended message to stream with sequence number %d",
            client.append_message(stream_name, "ABCDEFGHIJKLMNO".encode("utf-8")),
        )
```

```
        logger.info(
            "Successfully appended message to stream with sequence number %d",
            client.append_message(stream_name, "PQRSTUVWXYZ".encode("utf-8")),
        )

        # Try reading the two messages we just appended and print them out
        logger.info(
            "Successfully read 2 messages: %s",
            client.read_messages(stream_name,
 ReadMessagesOptions(min_message_count=2, read_timeout_millis=1000)),
        )

        logger.info("Now going to start writing random integers between 0 and 1000
 to the stream")
        # Now start putting in random data between 0 and 1000 to emulate device
 sensor input
        while True:
            logger.debug("Appending new random integer to stream")
            client.append_message(stream_name, random.randint(0,
 1000).to_bytes(length=4, signed=True, byteorder="big"))
            time.sleep(1)

    except asyncio.TimeoutError:
        logger.exception("Timed out while executing")
    except Exception:
        logger.exception("Exception while running")


def function_handler(event, context):
    return


logging.basicConfig(level=logging.INFO)
# Start up this sample code
main(logger=logging.getLogger())
```

5. Comprima os seguintes itens em um arquivo denominado `transfer_stream_python.zip`. Esse é o pacote de implantação de sua função do Lambda.

   - transfer_stream.py. Lógica do aplicativo.
   - greengrasssdk. Biblioteca necessária para funções Python do Lambda do Greengrass que publicam mensagens MQTT.

???As operações do gerenciador de streams estão disponíveis na versão 1.5.0 ou posterior do AWS IoT Greengrass Core SDK para Python.

- As dependências que você instalou para o AWS IoT Greengrass Core SDK para Python (por exemplo, os `cbor2` diretórios).

Ao criar o arquivo `zip`, inclua apenas esses itens, não a pasta que contém os arquivos.

Etapa 2: Criar uma função do Lambda

1. Crie um perfil do IAM para você transmitir no ARN da função ao criar a função.

   JSON Expanded

   ```
   aws iam create-role --role-name Lambda_empty --assume-role-policy '{
       "Version": "2012-10-17",
       "Statement": [
           {
               "Effect": "Allow",
               "Principal": {
                   "Service": "lambda.amazonaws.com"
               },
              "Action": "sts:AssumeRole"
           }
       ]
   }'
   ```

   JSON Single-line

   ```
   aws iam create-role --role-name Lambda_empty --assume-role-policy '{"Version":
    "2012-10-17",        "Statement": [{"Effect": "Allow", "Principal": {"Service":
    "lambda.amazonaws.com"},"Action": "sts:AssumeRole"}]}'
   ```

> **Note**
>
> AWS IoT Greengrass não usa essa função porque as permissões para suas funções do Greengrass Lambda são especificadas na função de grupo do Greengrass. Neste tutorial, você cria uma função vazia.

2. Copie a `Arn` da saída.
3. Use a AWS Lambda API para criar a `TransferStream` função. O comando a seguir pressupõe que o arquivo zip esteja no diretório atual.

   - Substitua *`role-arn`* pelo `Arn` que você copiou.

```
aws lambda create-function \
--function-name TransferStream \
--zip-file fileb://transfer_stream_python.zip \
--role role-arn \
--handler transfer_stream.function_handler \
--runtime python3.7
```

4. Publique uma versão da função.

```
aws lambda publish-version --function-name TransferStream --description 'First
 version'
```

5. Crie um alias para a versão publicada.

   Os grupos do Greengrass podem fazer referência a uma função do Lambda por alias (recomendado) ou por versão. Usar um alias facilita o gerenciamento de atualizações de código porque você não precisa alterar a tabela de assinaturas nem a definição do grupo ao atualizar a função do código. Em vez disso, você pode simplesmente apontar o alias para a nova versão da função.

```
aws lambda create-alias --function-name TransferStream --name GG_TransferStream --
function-version 1
```

> **Note**
>
> AWS IoT Greengrass não oferece suporte a aliases do Lambda para as versões $LATEST.

6. Copie a `AliasArn` da saída. Você usa esse valor ao configurar a função para AWS IoT Greengrass.

Agora você está pronto para configurar a função para AWS IoT Greengrass.

Etapa 3: Criar uma definição e uma versão de função

Essa etapa cria uma versão de definição de função que faz referência à função do Lambda do sistema `GGStreamManager` e à sua função do Lambda `TransferStream` definida pelo usuário. Para ativar o gerenciador de stream ao usar a AWS IoT Greengrass API, sua versão de definição de função deve incluir a `GGStreamManager` função.

1. Crie uma definição de função com uma versão inicial que contenha as funções do Lambda definidas pelo sistema e pelo usuário.

   A versão de definição a seguir ativa o gerenciador de fluxo com configurações de parâmetros padrão. Para definir configurações personalizadas, você deve definir variáveis de ambiente para os parâmetros correspondentes do gerenciador de fluxo. Para obter um exemplo, consultethe section called “Habilitar, desabilitar ou definir configurações do gerenciador de fluxo”. AWS IoT Greengrass usa as configurações padrão para parâmetros que são omitidos. `MemorySize`deveria ser pelo menos`128000`. `Pinned`deve ser configurado como`true`.

   > **Note**
   >
   > Uma função Lambda de longa duração (ou fixada) é iniciada automaticamente após o AWS IoT Greengrass início e continua sendo executada em seu próprio contêiner. Isso contrasta com uma função do Lambda sob demanda, que é iniciada quando invocada e interrompida quando não há tarefas a serem executadas. Para obter mais informações, consulte the section called “Configuração do ciclo de vida”.

   - *`arbitrary-function-id`*Substitua por um nome para a função, como**`stream-manager`**.

- *alias-arn*Substitua pelo `AliasArn` que você copiou ao criar o alias para a função `TransferStream` Lambda.

JSON expanded

```
aws greengrass create-function-definition --name MyGreengrassFunctions --
initial-version '{
    "Functions": [
        {
            "Id": "arbitrary-function-id",
            "FunctionArn": "arn:aws:lambda:::function:GGStreamManager:1",
            "FunctionConfiguration": {
                "MemorySize": 128000,
                "Pinned": true,
                "Timeout": 3
            }
        },
        {
            "Id": "TransferStreamFunction",
            "FunctionArn": "alias-arn",
            "FunctionConfiguration": {
                "Executable": "transfer_stream.function_handler",
                "MemorySize": 16000,
                "Pinned": true,
                "Timeout": 5
            }
        }
    ]
}'
```

JSON single

```
aws greengrass create-function-definition \
--name MyGreengrassFunctions \
--initial-version '{"Functions": [{"Id": "arbitrary-function-
id","FunctionArn": "arn:aws:lambda:::function:GGStreamManager:1",
 "FunctionConfiguration": {"Environment": {"Variables":
{"STREAM_MANAGER_STORE_ROOT_DIR": "/data","STREAM_MANAGER_SERVER_PORT":
 "1234","STREAM_MANAGER_EXPORTER_MAX_BANDWIDTH": "20000"}},"MemorySize":
 128000,"Pinned": true,"Timeout": 3}},{"Id": "TransferStreamFunction",
```

```
"FunctionArn": "alias-arn", "FunctionConfiguration": {"Executable":
"transfer_stream.function_handler", "MemorySize": 16000,"Pinned":
true,"Timeout": 5}}]}'
```

> **Note**
>
> O `Timeout` é exigido pela versão de definição de função, mas o `GGStreamManager` não o utiliza. Para obter mais informações sobre `Timeout` e outras configurações em nível de grupo, consulte the section called “Controlando a execução da função do Lambda do Greengrass”.

2. Copie a `LatestVersionArn` da saída. Você usa esse valor para adicionar a versão de definição da função à versão de grupo que você implanta no núcleo.

Etapa 4: Criar uma definição e versão do logger

Defina as configurações de registro do grupo. Neste tutorial, você configura componentes AWS IoT Greengrass do sistema, funções do Lambda definidas pelo usuário e conectores para gravar registros no sistema de arquivos do dispositivo principal. Você pode usar logs para solucionar quaisquer problemas que possa encontrar. Para obter mais informações, consulte the section called “Monitoramento com logs do AWS IoT Greengrass ”.

1. Crie uma definição de logger que inclua uma versão inicial.

   JSON Expanded

```
aws greengrass create-logger-definition --name "LoggingConfigs" --initial-
version '{
    "Loggers": [
        {
            "Id": "1",
            "Component": "GreengrassSystem",
            "Level": "INFO",
            "Space": 10240,
            "Type": "FileSystem"
        },
        {
            "Id": "2",
            "Component": "Lambda",
```

```
            "Level": "INFO",
            "Space": 10240,
            "Type": "FileSystem"
        }
    ]
}'
```

JSON Single-line

```
aws greengrass create-logger-definition \
    --name "LoggingConfigs" \
    --initial-version '{"Loggers":
[{"Id":"1","Component":"GreengrassSystem","Level":"INFO","Space":10240,"Type":"FileSyste
{"Id":"2","Component":"Lambda","Level":"INFO","Space":10240,"Type":"FileSystem"}]}'
```

2. Copie a `LatestVersionArn` da definição de logger da saída. Use esse valor para adicionar a versão de definição de logger à versão de grupo implantada no núcleo.

Etapa 5: Obter o ARN da sua versão de definição de núcleo

Obtenha o ARN da versão de definição de núcleo para adicionar à sua nova versão de grupo. Para implantar uma versão de grupo, ela deve fazer referência a uma versão de definição de núcleo que contenha exatamente um núcleo.

1. Obtenha os IDs do grupo do Greengrass de destino e a versão do grupo. Esse procedimento pressupõe que esse seja o grupo e a versão mais recente do grupo. A consulta a seguir retorna o grupo criado mais recentemente.

   ```
   aws greengrass list-groups --query "reverse(sort_by(Groups, &CreationTimestamp))
   [0]"
   ```

   Ou é possível consultar por nome. Os nomes de grupo não precisam ser exclusivos, portanto, vários grupos podem ser retornados.

   ```
   aws greengrass list-groups --query "Groups[?Name=='MyGroup']"
   ```

> **Note**
>
> Você também pode encontrar esses valores no AWS IoT console. O ID do grupo é exibido na página Settings (Configurações) do grupo. Os IDs de versão do grupo são exibidos na guia Implantações do grupo.

2. Copie da saída o `Id` do grupo de destino. Você usa isso para obter a versão de definição de núcleo e ao implantar o grupo.
3. Copie a `LatestVersion` da saída, que é o ID da última versão adicionada ao grupo. Você usa isso para obter a versão de definição do núcleo.
4. Obtenha o ARN da versão de definição de núcleo:

   a. Obtenha a versão do grupo.

   - *group-id*Substitua pelo `Id` que você copiou para o grupo.
   - *group-version-id*Substitua pelo `LatestVersion` que você copiou para o grupo.

   ```
   aws greengrass get-group-version \
   --group-id group-id \
   --group-version-id group-version-id
   ```

   b. Copie a `CoreDefinitionVersionArn` da saída. Use esse valor para adicionar a versão de definição à versão de grupo implantada no núcleo.

Etapa 6: Criar uma versão de grupo

Agora, você está pronto para criar uma versão de grupo que contém todas as entidades que você deseja implantar. Faça isso criando uma versão de grupo que faz referência à versão de destino de cada componente. Neste tutorial, você inclui uma versão de definição de núcleo, uma versão de definição de função e uma versão de definição de logger.

1. Criar uma versão de grupo.

   - *group-id*Substitua pelo `Id` que você copiou para o grupo.
   - *core-definition-version-arn*Substitua pela `CoreDefinitionVersionArn` que você copiou para a versão de definição principal.

- *function-definition-version-arn*Substitua pelo `LatestVersionArn` que você copiou para sua nova versão de definição de função.
- *logger-definition-version-arn*Substitua pelo `LatestVersionArn` que você copiou para sua nova versão de definição do registrador.

```
aws greengrass create-group-version \
--group-id group-id \
--core-definition-version-arn core-definition-version-arn \
--function-definition-version-arn function-definition-version-arn \
--logger-definition-version-arn logger-definition-version-arn
```

2. Copie a `Version` da saída. Este é o ID da nova versão do grupo.

Etapa 7: Criar uma implantação

Implante o grupo no dispositivo do núcleo.

1. Verifique se o AWS IoT Greengrass núcleo está funcionando. Execute os seguintes comandos no seu terminal do Raspberry Pi, conforme necessário.

   a. Para verificar se o daemon está em execução:

   ```
   ps aux | grep -E 'greengrass.*daemon'
   ```

   Se a saída contém uma entrada `root` para `/greengrass/ggc/packages/`*ggc-version*`/bin/daemon`, o daemon está em execução.

   > **Note**
   >
   > A versão no caminho depende da versão do software AWS IoT Greengrass Core instalada em seu dispositivo principal.

   b. Para iniciar o daemon:

   ```
   cd /greengrass/ggc/core/
   sudo ./greengrassd start
   ```

2. Crie um implantação do .

- *group-id*Substitua pelo `Id` que você copiou para o grupo.
- *group-version-id*Substitua pelo `Version` que você copiou para a nova versão do grupo.

```
aws greengrass create-deployment \
--deployment-type NewDeployment \
--group-id group-id \
--group-version-id group-version-id
```

3. Copie a `DeploymentId` da saída.
4. Obtenha o status de implantação.

   - *group-id*Substitua pelo `Id` que você copiou para o grupo.
   - *deployment-id*Substitua pelo `DeploymentId` que você copiou para a implantação.

   ```
   aws greengrass get-deployment-status \
   --group-id group-id \
   --deployment-id deployment-id
   ```

   Se o status for `Success`, a implantação foi bem-sucedida. Para obter ajuda sobre a solução de problemas, consulte Solução de problemas.

Etapa 8: Testar o aplicativo

A função do Lambda do `TransferStream` gera dados simulados do dispositivo. Ela grava dados em um fluxo que o gerenciador de fluxo exporta para o fluxo de dados do Kinesis de destino.

1. No console do Amazon Kinesis, em Streams de dados do Kinesis, escolha. MyKinesisStream

   > **Note**
   >
   > Se você executou o tutorial sem um fluxo de dados do Kinesis de destino, verifique o arquivo de log do gerenciador de fluxo (`GGStreamManager`). Se ele contiver `export stream MyKinesisStream doesn't exist` em uma mensagem de erro, o teste será bem-sucedido. Esse erro significa que o serviço tentou exportar para o fluxo, mas o fluxo não existe.

2. Na MyKinesisStream página, escolha Monitoramento. Se o teste for bem-sucedido, você verá os dados nos gráficos Put Records (Inserir registros) . Dependendo da sua conexão, pode demorar um minuto até que os dados sejam exibidos.

   > **Important**
   >
   > Ao terminar o teste, exclua o fluxo de dados do Kinesis para evitar mais cobranças. Ou execute o comando a seguir para interromper o daemon do Greengrass. Isso impedirá que o núcleo envie mensagens até que você esteja pronto para dar continuidade aos testes.
   >
   > ```
   > cd /greengrass/ggc/core/
   > sudo ./greengrassd stop
   > ```

3. Remova a função TransferStream Lambda do núcleo.

   a. Siga the section called “Criar uma versão de grupo” para criar uma nova versão de grupo, mas remova a opção `--function-definition-version-arn` no comando `create-group-version`. Ou crie uma versão de definição de função que não inclua a função TransferStream Lambda.

      > **Note**
      >
      > Ao omitir a função do Lambda do sistema `GGStreamManager` da versão do grupo implantado, desabilite o gerenciamento de fluxo no núcleo.

   b. Siga the section called “Crie um implantação do .” para implantar a nova versão do grupo.

Para exibir informações de registro ou solucionar problemas com fluxos, verifique os logs das funções `TransferStream` e `GGStreamManager`. Você deve ter `root` permissões para ler AWS IoT Greengrass os registros no sistema de arquivos.

- `TransferStream` grava entradas de log em *`greengrass-root`*`/ggc/var/log/user/`*`region`*`/`*`account-id`*`/TransferStream.log`.
- `GGStreamManager` grava entradas de log em *`greengrass-root`*`/ggc/var/log/system/GGStreamManager.log`.

Se precisar de mais informações sobre a resolução de problemas, você pode definir o nível de registro do `Lambda` como `DEBUG` e criar e implantar uma nova versão de grupo.

Consulte também

- Gerenciamento de fluxos de dados
- the section called “Use StreamManagerClient para trabalhar com streams”
- the section called “Exporte configurações para suporte Nuvem AWS destinos”
- the section called “Configurar o gerenciador de fluxo”
- the section called “Exportar streams de dados (console)”
- AWS Identity and Access Management Comandos (IAM) na Referência de AWS CLI Comandos
- AWS Lambda comandos na Referência de AWS CLI Comandos
- AWS IoT Greengrass comandos na Referência de AWS CLI Comandos

Implemente segredos no AWS IoT Greengrass core

Esse recurso está disponível para AWS IoT Greengrass Core v1.7 e versões posteriores.

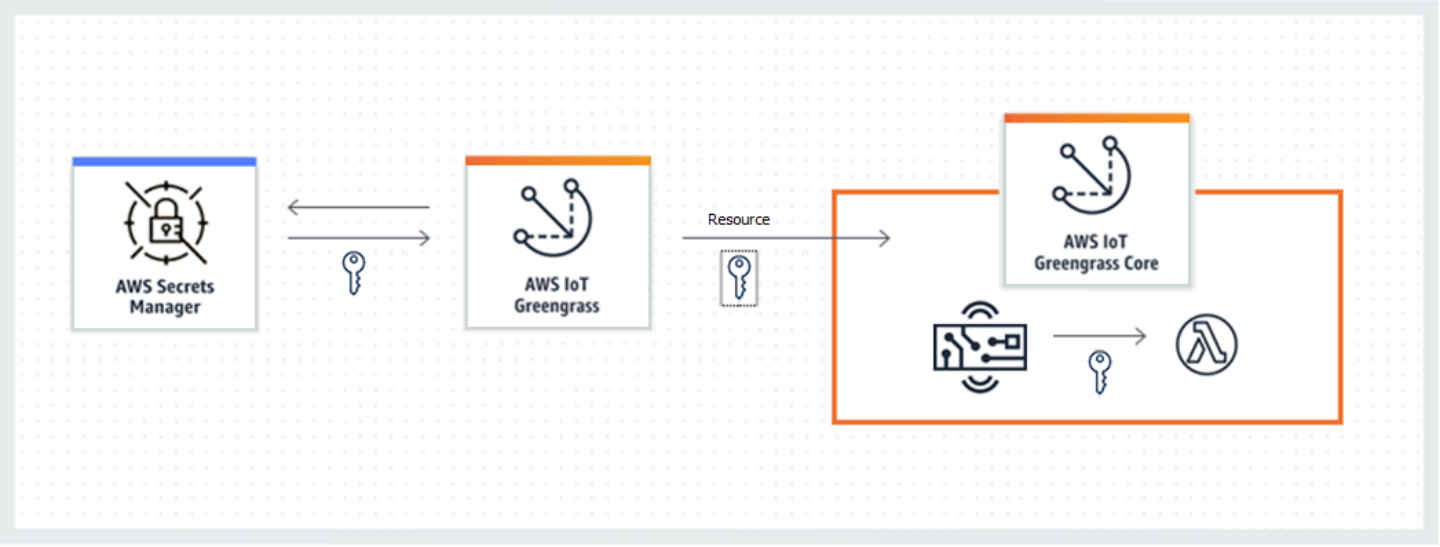
AWS IoT Greengrass permite que você se autentique com serviços e aplicativos de dispositivos Greengrass sem codificar senhas, tokens ou outros segredos.

AWS Secrets Manager é um serviço que você pode usar para armazenar e gerenciar seus segredos com segurança na nuvem. AWS IoT Greengrass estende o Secrets Manager aos dispositivos principais do Greengrass, para que seus conectores e funções do Lambda possam usar segredos locais para interagir com serviços e aplicativos. Por exemplo, o conector Twilio Notifications usa um token de autenticação armazenado localmente.

Para integrar um segredo a um grupo do Greengrass, crie um recurso de grupo que faça referência ao segredo do Secrets Manager. Esse recurso de segredo faz referência ao segredo na nuvem pelo ARN. Para saber como criar, gerenciar e usar recursos de segredos, consulte the section called “Trabalhar com recursos de segredos”.

AWS IoT Greengrass criptografa seus segredos enquanto estão em trânsito e em repouso. Durante a implantação em grupo, AWS IoT Greengrass busca o segredo do Secrets Manager e cria uma cópia criptografada local no núcleo do Greengrass. Após alternar os segredos na nuvem do Secrets Manager, implante o grupo novamente para propagar os valores atualizados para o núcleo.

O diagrama a seguir mostra o processo de alto nível de implantação de um segredo no núcleo. Os segredos são criptografados em repouso e em trânsito.



Usar AWS IoT Greengrass para armazenar seus segredos localmente oferece as seguintes vantagens:

- Dissociação do código (não codificado). Oferece suporte a credenciais gerenciadas centralmente e ajuda a proteger dados confidenciais contra o risco de comprometimento.
- Disponível para cenários offline. Conectores e funções podem acessar serviços e software locais com segurança quando desconectados da Internet.
- Acesso controlado a segredos. Somente conectores e funções autorizados no grupo podem acessar seus segredos. O AWS IoT Greengrass usa criptografia de chave privada para proteger os segredos. Os segredos são criptografados em repouso e em trânsito. Para obter mais informações, consulte the section called “Criptografia de segredos”.
- Alternância controlada. Após alternar seus segredos no Secrets Manager, implante o grupo do Greengrass novamente para atualizar as cópias locais dos segredos. Para obter mais informações, consulte the section called “Criar e gerenciar segredos”.

> **Important**
>
> AWS IoT Greengrass não atualiza automaticamente os valores dos segredos locais após a rotação das versões em nuvem. Para atualizar valores locais, você deve implantar o grupo novamente.

Criptografia de segredos

AWS IoT Greengrass criptografa segredos em trânsito e em repouso.

> **Important**
>
> Certifique-se de que suas funções do Lambda definidas pelo usuário tratem os segredos com segurança e não registrem em log nenhum dado confidencial armazenado no segredo. Para obter mais informações, consulte Mitigate the Risks of Logging and Debugging Your Lambda Function no Guia do usuário do AWS Secrets Manager . Embora essa documentação se refira especificamente às funções de alternância, a recomendação também se aplica às funções do Lambda do Greengrass.

Criptografia em trânsito

AWS IoT Greengrass usa Transport Layer Security (TLS) para criptografar todas as comunicações pela Internet e pela rede local. Isso protege segredos em trânsito, o que ocorre quando segredos são recuperados do Secrets Manager e implantados no núcleo. Para pacotes de criptografia do TLS com suporte, consulte the section called “Suporte a pacotes de criptografia TLS”.

Criptografia em repouso

AWS IoT Greengrass usa a chave privada especificada em `config.json` para criptografar os segredos armazenados no núcleo. Por esse motivo, o armazenamento seguro da chave privada é essencial para proteger segredos locais. No modelo de responsabilidade AWS compartilhada, é responsabilidade do cliente garantir o armazenamento seguro da chave privada no dispositivo principal.

AWS IoT Greengrass suporta dois modos de armazenamento de chaves privadas:

- Uso dos módulos de segurança de hardware. Para obter mais informações, consulte the section called “Integração de segurança de hardware”.

  > **Note**
  >
  > Atualmente, AWS IoT Greengrass oferece suporte somente ao mecanismo de preenchimento PKCS #1 v1.5 para criptografia e decodificação de segredos locais ao usar chaves privadas baseadas em hardware. Se você estiver seguindo as instruções fornecidas pelo fornecedor para gerar manualmente chaves privadas baseadas em hardware, certifique-se de escolher PKCS #1 v1.5. AWS IoT Greengrass não oferece suporte ao Optimal Asymmetric Encryption Padding (OAEP).

- Uso de permissões do sistema de arquivos (padrão).

A chave privada é usada para proteger a chave de dados, que é usada para criptografar segredos locais. A chave de dados é alternada com cada grupo de implantação.

O AWS IoT Greengrass núcleo é a única entidade que tem acesso à chave privada. Os conectores do Greengrass ou funções do Lambda que são afiliados a um recurso de segredo obtêm o valor do segredo do núcleo.

Requisitos

Estes são os requisitos para suporte de segredo local:

- Você deve estar usando o AWS IoT Greengrass Core v1.7 ou posterior.
- Para obter os valores dos segredos locais, suas funções do Lambda definidas pelo usuário devem usar o AWS IoT Greengrass Core SDK v1.3.0 ou posterior.
- A chave privada usada para criptografia de segredos locais deve ser especificada no arquivo de configuração do Greengrass. Por padrão, AWS IoT Greengrass usa a chave privada principal armazenada no sistema de arquivos. Para fornecer sua própria chave privada, consulte the section called “Especificar a chave privada para criptografia de segredos”. Somente o tipo de chave RSA tem suporte.

  > **Note**
  >
  > Atualmente, AWS IoT Greengrass oferece suporte somente ao mecanismo de preenchimento PKCS #1 v1.5 para criptografia e decodificação de segredos locais ao usar chaves privadas baseadas em hardware. Se você estiver seguindo as instruções fornecidas pelo fornecedor para gerar manualmente chaves privadas baseadas em hardware, certifique-se de escolher PKCS #1 v1.5. AWS IoT Greengrass não oferece suporte ao Optimal Asymmetric Encryption Padding (OAEP).

- AWS IoT Greengrass deve ter permissão para obter seus valores secretos. Isso permite AWS IoT Greengrass buscar os valores durante a implantação do grupo. Se você estiver usando a função de serviço padrão do Greengrass, então AWS IoT Greengrass já tem acesso a segredos com nomes que começam com greengrass-. Para personalizar o acesso, consulte the section called “Permitir AWS IoT Greengrass para obter valores secretos”.

  > **Note**
  >
  > Recomendamos que você use essa convenção de nomenclatura para identificar os segredos que AWS IoT Greengrass podem ser acessados, mesmo que você personalize as permissões. O console usa permissões diferentes para ler seus segredos, então é possível que você possa selecionar segredos no console que AWS IoT Greengrass não tenham permissão para buscá-los. O uso de uma convenção de nomenclatura pode ajudar a evitar um conflito de permissões, resultando em um erro de implantação.

Especificar a chave privada para criptografia de segredos

Neste procedimento, você fornece o caminho para uma chave privada que é usada para criptografia de segredos locais. Essa deve ser uma chave RSA com tamanho mínimo de 2048 bits. Para obter mais informações sobre chaves privadas usadas no AWS IoT Greengrass núcleo, consultethe section called “Entidades principais de segurança”.

AWS IoT Greengrass suporta dois modos de armazenamento de chaves privadas: baseado em hardware ou baseado em sistema de arquivos (padrão). Para obter mais informações, consulte the section called “Criptografia de segredos”.

Siga este procedimento somente se você deseja alterar a configuração padrão, que usa a chave privada do núcleo no sistema de arquivos. Estas etapas foram criadas pressupondo que você criou seu grupo e núcleo, conforme descrito no Módulo 2 do tutorial Conceitos básicos.

1. Abra o arquivo `config.json` que está localizado no diretório */greengrass-root*/`config`.

   > **Note**
   >
   > *greengrass-root*representa o caminho em que o software AWS IoT Greengrass Core está instalado em seu dispositivo. Normalmente, esse é o diretório `/greengrass`.

2. No objeto `crypto.principals.SecretsManager`, para a propriedade `privateKeyPath`, insira o caminho da chave privada:

   - Se a chave privada estiver armazenada no sistema de arquivos, especifique o caminho absoluto para a chave . Por exemplo:

     ```
     "SecretsManager" : {
       "privateKeyPath" : "file:///somepath/hash.private.key"
     }
     ```

   - Se a chave privada for armazenada em um módulo de segurança de hardware (HSM), especifique o caminho usando o esquema de URI RFC 7512 PKCS#11. Por exemplo:

     ```
     "SecretsManager" : {
       "privateKeyPath" : "pkcs11:object=private-key-label;type=private"
     }
     ```

Para obter mais informações, consulte the section called “Configuração de segurança do hardware”.

> **Note**
>
> Atualmente, AWS IoT Greengrass oferece suporte somente ao mecanismo de preenchimento PKCS #1 v1.5 para criptografia e decodificação de segredos locais ao usar chaves privadas baseadas em hardware. Se você estiver seguindo as instruções fornecidas pelo fornecedor para gerar manualmente chaves privadas baseadas em hardware, certifique-se de escolher PKCS #1 v1.5. AWS IoT Greengrass não oferece suporte ao Optimal Asymmetric Encryption Padding (OAEP).

Permitir AWS IoT Greengrass para obter valores secretos

Neste procedimento, você adiciona uma política embutida à função de serviço do Greengrass que permite AWS IoT Greengrass obter os valores de seus segredos.

Siga esse procedimento somente se quiser conceder permissões AWS IoT Greengrass personalizadas aos seus segredos ou se sua função de serviço do Greengrass não incluir a política `AWSGreengrassResourceAccessRolePolicy` gerenciada. `AWSGreengrassResourceAccessRolePolicy`concede acesso a segredos com nomes que começam com greengrass-.

1. Execute o seguinte comando da CLI para obter o ARN da função de serviço do Greengrass:

   ```
   aws greengrass get-service-role-for-account --region region
   ```

   O ARN retornado contém o nome da função.

   ```
   {
     "AssociatedAt": "time-stamp",
     "RoleArn": "arn:aws:iam::account-id:role/service-role/role-name"
   }
   ```

   Você usará o ARN ou nome na etapa a seguir.

2. Adicione uma política em linha que permita a ação `secretsmanager:GetSecretValue`. Para obter informações, consulte Adicionar e remover políticas do IAM no Guia do usuário do IAM.

   Você pode conceder acesso granular listando segredos explicitamente ou usando um esquema de nomenclatura do caractere curinga * ou você pode conceder acesso condicional a segredos marcados ou versionados. Por exemplo, a política a seguir AWS IoT Greengrass permite ler somente os segredos especificados.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetSecretValue"
            ],
            "Resource": [
            "arn:aws:secretsmanager:us-east-1:123456789012:secret:greengrass-
SecretA-abc",
    "arn:aws:secretsmanager:us-east-1:123456789012:secret:greengrass-SecretB-
xyz"
            ]
        }
    ]
}
```

> **Note**
>
> Se você usa uma AWS KMS chave gerenciada pelo cliente para criptografar segredos, sua função de serviço do Greengrass também deve permitir a ação. `kms:Decrypt`

Para obter mais informações sobre políticas do IAM para o Secrets Manager, consulte Controle de acesso e autenticação para o AWS Secrets Manager e Ações, recursos e chaves de contexto que podem ser usados em uma política do IAM ou em uma política de segredos para o AWS Secrets Manager no Guia do usuário do AWS Secrets Manager .

Consulte também

- O que é AWS Secrets Manager? no Guia do AWS Secrets Manager usuário
- PKCS #1: RSA Encryption versão 1.5

Como trabalhar com recursos de segredos

AWS IoT Greengrass usa recursos secretos para integrar segredos de AWS Secrets Manager um grupo do Greengrass. Um recurso de segredo é uma referência a um segredo do Secrets Manager. Para obter mais informações, consulte Implantar segredos no núcleo.

No dispositivo AWS IoT Greengrass principal, os conectores e as funções do Lambda podem usar o recurso secreto para se autenticar com serviços e aplicativos, sem codificar senhas, tokens ou outras credenciais.

Criar e gerenciar segredos

Em um grupo do Greengrass, um recurso de segredo faz referência ao ARN de um segredo do Secrets Manager. Quando o recurso de segredo é implantado no núcleo, o valor do segredo é criptografado e disponibilizado para conectores e funções do Lambda afiliados. Para obter mais informações, consulte the section called “Criptografia de segredos”.

Você pode usar o Secrets Manager para criar e gerenciar as versões na nuvem de seus segredos. Você usa AWS IoT Greengrass para criar, gerenciar e implantar seus recursos secretos.

> **Important**
>
> Recomendamos que você siga a melhor prática de alternar seus segredos no Secrets Manager. Em seguida, implante o grupo do Greengrass para atualizar as cópias locais dos seus segredos. Para obter mais informações, consulte Como alternar seus AWS Secrets Manager segredos no Guia do AWS Secrets Manager usuário.

Para disponibilizar um segredo no núcleo do Greengrass

1. Crie um segredo no Secrets Manager. Essa é a versão na nuvem de seu segredo, que é armazenado e gerenciado centralmente no Secrets Manager. As tarefas de gerenciamento incluem a alternância de valores de segredos e a aplicação de políticas de recursos.

2. Crie um recurso secreto em AWS IoT Greengrass. Esse grupo é um tipo de recurso de grupo que faz referência ao segredo na nuvem pelo ARN. Você pode fazer referência a um segredo somente uma vez por grupo.
3. Configure seu conector ou sua função do Lambda. Você deve afiliar o recurso a uma função ou a um conector especificando parâmetros ou propriedades correspondentes. Isso permite que o usuário obtenha o valor do recurso de segredo implantado localmente. Para obter mais informações, consulte the section called “Usar segredos locais”.
4. Implante o grupo do Greengrass. Durante a implantação, o AWS IoT Greengrass obtém o valor do segredo na nuvem e cria (ou atualiza) o segredo local no núcleo.

O Secrets Manager registra um evento a AWS CloudTrail cada vez que AWS IoT Greengrass recupera um valor secreto. AWS IoT Greengrass não registra nenhum evento relacionado à implantação ou ao uso de segredos locais. Para obter mais informações sobre o registro do Secrets Manager, consulte Monitorar o uso de seus AWS Secrets Manager segredos no Guia AWS Secrets Manager do usuário.

Incluir rótulos de preparação em recursos de segredos

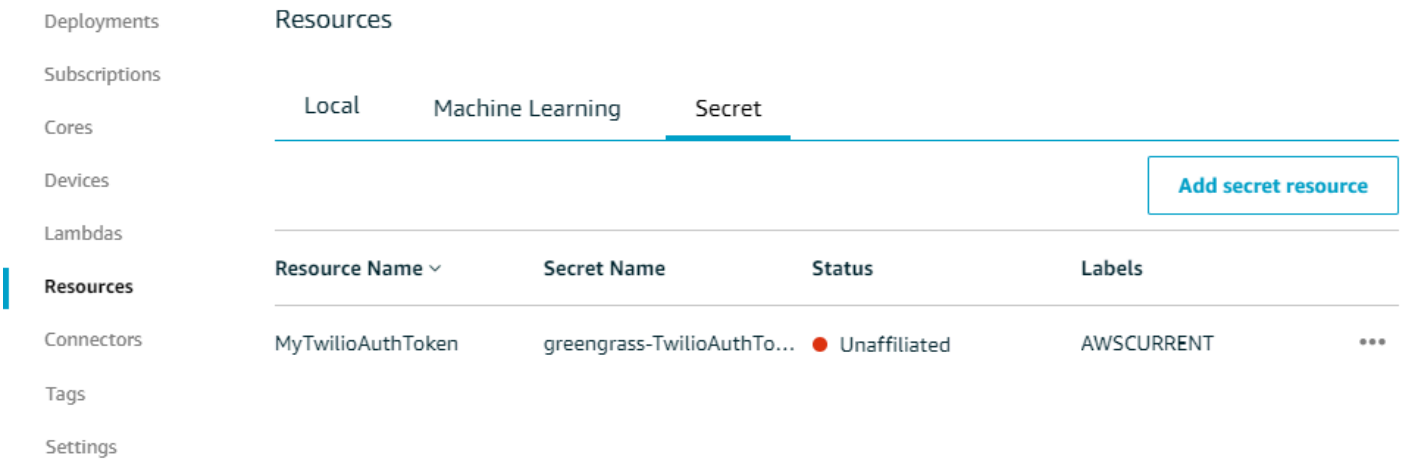
O Secrets Manager usa rótulos de preparação para identificar versões específicas de um valor de segredo. Os rótulos de teste podem ser definidos pelo sistema ou pelo usuário. O Secrets Manager atribui o rótulo `AWSCURRENT` à versão mais recente do valor do segredo. Os rótulos de preparação geralmente são usados para gerenciar a alternância de segredos. Para obter mais informações sobre o versionamento de segredos do Secrets Manager, consulte Termos e conceitos importantes do AWS Secrets Manager no Guia do usuário do AWS Secrets Manager .

Os recursos de segredos sempre incluem o rótulo de preparação `AWSCURRENT` e podem incluir outros rótulos de preparação se necessários para uma função do Lambda ou um conector. Durante a implantação em grupo, AWS IoT Greengrass recupera os valores dos rótulos de teste que são referenciados no grupo e, em seguida, cria ou atualiza os valores correspondentes no núcleo.

Criar e gerenciar recursos de segredos (console)

Criar recursos de segredos (console)

No AWS IoT Greengrass console, você cria e gerencia recursos secretos na guia Segredos na página Recursos do grupo. Para obter tutoriais que criam um recurso de segredo e o adicionam a um grupo, consulte the section called “Como criar um recurso de segredo (console)” e the section called “Conceitos básicos de conectores (console)”.



> **Note**
>
> Como alternativa, o console permite que você crie um segredo e um recurso de segredo ao configurar um conector ou uma função do Lambda. Você pode fazer isso na página Configurar parâmetros do conector ou na página Recursos da função do Lambda.

Gerenciar um recurso de segredo (console)

As tarefas de gerenciamento dos recursos de segredos em seu grupo do Greengrass incluem adicionar recursos de segredos ao grupo, remover recursos de segredos do grupo e alterar o conjunto de rótulos de preparação incluídos em um recurso de segredo.

Se você apontar para um segredo diferente do Secrets Manager, também deverá editar conectores que usem o segredo:

1. Na página de configuração do grupo, selecione Connectors (Conectores).
2. No menu de contexto do conector, selecione Edit (Editar).
3. A página Edit parameters (Editar parâmetros) exibe uma mensagem para informá-lo de que o ARN do segredo foi alterado. Para confirmar a alteração, selecione Save (Salvar).

Se você excluir um segredo no Secrets Manager, remova o recurso de segredode segredo correspondente do grupo e dos conectores e funções do Lambda que fazem referência a ele. Caso contrário, durante a implantação do grupo, AWS IoT Greengrass retornará um erro informando que o segredo não pode ser encontrado. Além disso, atualize o código da função do Lambda conforme necessário.

Criar e gerenciar recursos de segredos (CLI)

Criar recursos de segredos (CLI)

Na AWS IoT Greengrass API, um segredo é um tipo de recurso de grupo. O exemplo a seguir cria uma definição de recurso com uma versão inicial que inclui um recurso de segredo chamado `MySecretResource`. Para obter um tutorial que cria um recurso de segredo e o adiciona a uma versão de grupo, consulte the section called “Conceitos básicos de conectores (CLI)”.

O recurso de segredo faz referência ao ARN do segredo do Secrets Manager correspondente e inclui dois rótulos de preparação, além de `AWSCURRENT`, que é sempre incluído.

```
aws greengrass create-resource-definition --name MyGreengrassResources --initial-
version '{
    "Resources": [
        {
            "Id": "my-resource-id",
            "Name": "MySecretResource",
            "ResourceDataContainer": {
                "SecretsManagerSecretResourceData": {
                    "ARN": "arn:aws:secretsmanager:us-
west-2:123456789012:secret:greengrass-SomeSecret-KUj89s",
                    "AdditionalStagingLabelsToDownload": [
                        "Label1",
                        "Label2"
                    ]
                }
            }
        }
    ]
}'
```

Gerenciar recursos de segredos (CLI)

As tarefas de gerenciamento dos recursos de segredos em seu grupo do Greengrass incluem adicionar recursos de segredos ao grupo, remover recursos de segredos do grupo e alterar o conjunto de rótulos de preparação incluídos em um recurso de segredo.

Na AWS IoT Greengrass API, essas mudanças são implementadas usando versões.

A AWS IoT Greengrass API usa versões para gerenciar grupos. As versões são imutáveis, portanto, para adicionar ou alterar componentes do grupo — por exemplo, os dispositivos cliente, as funções e

os recursos do grupo — você deve criar versões de componentes novos ou atualizados. Em seguida, crie e implante uma versão do grupo que contenha a versão de destino de cada componente. Para saber mais sobre os grupos, consulte the section called “AWS IoT Greengrass groups”.

Por exemplo, para alterar o conjunto de rótulos de preparação para um recurso de segredo:

1. Crie uma versão de definição de recurso que contenha o recurso de segredo atualizado. O exemplo a seguir adiciona um terceiro rótulo de preparação ao recurso de segredo da seção anterior.

   > **Note**
   >
   > Para adicionar mais recursos à versão, inclua-os na matriz `Resources`.

   ```
   aws greengrass create-resource-definition --name MyGreengrassResources --initial-
   version '{
       "Resources": [
           {
               "Id": "my-resource-id",
               "Name": "MySecretResource",
               "ResourceDataContainer": {
                   "SecretsManagerSecretResourceData": {
                       "ARN": "arn:aws:secretsmanager:us-
   west-2:123456789012:secret:greengrass-SomeSecret-KUj89s",
                       "AdditionalStagingLabelsToDownload": [
                           "Label1",
                           "Label2",
                           "Label3"
                       ]
                   }
               }
           }
       ]
   }'
   ```

2. Se o ID do recurso de segredo for alterado, atualize os conectores e as funções que usam o recurso de segredo. Nas novas versões, atualize o parâmetro ou a propriedade que corresponde ao ID de recurso. Se o ARN do segredo for alterado, você também deverá atualizar o parâmetro correspondente para quaisquer conectores que usem o segredo.

> **Note**
>
> O ID do recurso é um identificador arbitrário fornecido pelo cliente.

3. Crie um grupo de destino que contenha a versão de destino de cada componente que você deseja enviar para o núcleo.
4. Implante a versão de grupo.

Para obter um tutorial que mostra como criar e implantar recursos de segredos, conectores e funções, consulte the section called “Conceitos básicos de conectores (CLI)”.

Se você excluir um segredo no Secrets Manager, remova o recurso de segredode segredo correspondente do grupo e dos conectores e funções do Lambda que fazem referência a ele. Caso contrário, durante a implantação do grupo, AWS IoT Greengrass retornará um erro informando que o segredo não pode ser encontrado. Além disso, atualize o código da função do Lambda conforme necessário. Você pode remover um segredo local implantando uma versão de definição de recurso que não contenha o recurso de segredo correspondente.

Usar segredos locais em conectores e funções do Lambda

As funções do Lambda e os conectores do Greengrass usam segredos locais para interagir com serviços e aplicativos. O valor `AWSCURRENT` é usado por padrão, mas os valores dos outros rótulos de preparação incluídos no recurso de segredo também estão disponíveis.

Conectores e funções devem ser configurados para que possam acessar segredos locais. Isso afilia o recurso de segredo ao conector ou à função.

Conectores

Se um conector exige acesso a um segredo local, ele fornece parâmetros que você configura com as informações necessárias para acessar o segredo.

- Para saber como fazer isso no AWS IoT Greengrass console, consultethe section called “Conceitos básicos de conectores (console)”.
- Para saber como fazer isso com a AWS IoT Greengrass CLI, consultethe section called “Conceitos básicos de conectores (CLI)”.

Para obter informações sobre os requisitos para conectores individuais, consulte the section called “Conectores do Greengrass fornecidos pela AWS”.

A lógica para acessar e usar o segredo está integrada ao conector.

Funções do Lambda

Para permitir que uma função do Lambda do Greengrass acesse um segredo local, configure as propriedades da função.

- Para saber como fazer isso no AWS IoT Greengrass console, consultethe section called “Como criar um recurso de segredo (console)”.
- Para fazer isso na AWS IoT Greengrass API, você fornece as seguintes informações na `ResourceAccessPolicies` propriedade.
  - `ResourceId`: o ID do recurso de segredo no grupo do Greengrass. Esse é o recurso que faz referência ao ARN do segredo do Secrets Manager correspondente.
  - `Permission`: o tipo de acesso que a função tem ao recurso. Somente a permissão `ro` (somente leitura) tem suporte para recursos de segredos.

  O exemplo a seguir cria uma função do Lambda que pode acessar o recurso de segredo `MyApiKey`.

```
aws greengrass create-function-definition --name MyGreengrassFunctions --initial-
version '{
    "Functions": [
        {
            "Id": "MyLambdaFunction",
            "FunctionArn": "arn:aws:lambda:us-
west-2:123456789012:function:myFunction:1",
            "FunctionConfiguration": {
                "Pinned": false,
                "MemorySize": 16384,
                "Timeout": 10,
                "Environment": {
                    "ResourceAccessPolicies": [
                        {
                            "ResourceId": "MyApiKey",
                            "Permission": "ro"
                        }
                    ],
                    "AccessSysfs": true
                }
            }
        }
    ]
```

```
}'
```

Para acessar segredos locais em tempo de execução, as funções do Greengrass Lambda chamam a `get_secret_value` função do `secretsmanager` cliente no AWS IoT Greengrass Core SDK (v1.3.0 ou posterior).

O exemplo a seguir mostra como usar o SDK AWS IoT Greengrass Core para Python para obter um segredo. Ele transmite o nome do segredo para a função `get_secret_value`. O `SecretId` pode ser o nome ou o ARN do segredo do Secrets Manager (e não o recurso de segredo).

```
import greengrasssdk

secrets_client = greengrasssdk.client("secretsmanager")
secret_name = "greengrass-MySecret-abc"


def function_handler(event, context):
    response = secrets_client.get_secret_value(SecretId=secret_name)
    secret = response.get("SecretString")
```

Para segredos do tipo texto, a função `get_secret_value` retorna uma string. Para segredos do tipo binário, ela retorna uma string codificada em base64.

> **Important**
>
> Certifique-se de que suas funções do Lambda definidas pelo usuário tratem os segredos com segurança e não registrem em log nenhum dado confidencial armazenado no segredo. Para obter mais informações, consulte Mitigate the Risks of Logging and Debugging Your Lambda Function no Guia do usuário do AWS Secrets Manager . Embora essa documentação se refira especificamente às funções de alternância, a recomendação também se aplica às funções do Lambda do Greengrass.

O valor atual do segredo é retornado por padrão. Essa é a versão à qual o rótulo de preparação `AWSCURRENT` está anexado. Para acessar uma versão diferente, transmita o nome do rótulo de preparação correspondente para o argumento `VersionStage` opcional. Por exemplo:

```
import greengrasssdk

secrets_client = greengrasssdk.client("secretsmanager")
secret_name = "greengrass-TestSecret"
secret_version = "MyTargetLabel"


# Get the value of a specific secret version
def function_handler(event, context):
    response = secrets_client.get_secret_value(
        SecretId=secret_name, VersionStage=secret_version
    )
    secret = response.get("SecretString")
```

Para obter outro exemplo de função que chame `get_secret_value`, consulte Crie um pacote de implantação para a função do Lambda.

Como criar um recurso de segredo (console)

Esse recurso está disponível para AWS IoT Greengrass Core v1.7 e versões posteriores.

Este tutorial mostra como usar o Console de gerenciamento da AWS para adicionar um recurso secreto a um grupo do Greengrass. Um recurso secreto é uma referência a um segredo de AWS Secrets Manager. Para obter mais informações, consulte Implantar segredos no núcleo.

No dispositivo AWS IoT Greengrass principal, os conectores e as funções do Lambda podem usar o recurso secreto para se autenticar com serviços e aplicativos, sem codificar senhas, tokens ou outras credenciais.

Neste tutorial, você começa criando um segredo no AWS Secrets Manager console. Em seguida, no console do AWS IoT Greengrass , você adiciona um recurso de segredo a um grupo do Greengrass na página Recursos do grupo. Esse recurso de segredo faz referência ao segredo do Secrets

Manager. Posteriormente, você anexará o recurso de segredode segredo a uma função do Lambda, o que permite à função obter o valor do segredo local.

> **Note**
>
> Como alternativa, o console permite que você crie um segredo e um recurso de segredo ao configurar um conector ou uma função do Lambda. Você pode fazer isso na página Configurar parâmetros do conector ou na página Recursos da função do Lambda. Somente os conectores que contêm parâmetros para segredos podem acessar segredos. Para obter um tutorial que mostra como o conector de notificações Twilio usa um token de autenticação armazenamento localmente, consulte the section called “Conceitos básicos de conectores (console)”.

O tutorial contém as seguintes etapas de nível elevado:

1. Crie um segredo do Secrets Manager
2. Adicionar um recurso de segredo a um grupo
3. Crie um pacote de implantação para a função do Lambda
4. Criar uma função do Lambda
5. Adicionar a função ao grupo
6. Anexar o recurso de segredo à função
7. Adicionar assinaturas ao grupo
8. Implantar o grupo
9. the section called “Testar a função do Lambda”

O tutorial levará aproximadamente 20 minutos para ser concluído.

Pré-requisitos

Para concluir este tutorial, é necessário:

- Um grupo do Greengrass e um núcleo do Greengrass (versão 1.7 ou posterior). Para saber como criar um grupo e núcleo Greengrass, consulte Começando com AWS IoT Greengrass. O tutorial de introdução também inclui etapas para instalar o software AWS IoT Greengrass Core.

- AWS IoT Greengrass deve ser configurado para oferecer suporte a segredos locais. Para obter mais informações, consulte os Requisitos de segredos.

> **Note**
>
> Este requisito inclui permitir o acesso aos seus segredos do Secrets Manager. Se você estiver usando a função de serviço padrão do Greengrass, o Greengrass terá permissão para obter os valores de segredos com nomes que começam com greengrass-.

- Para obter os valores dos segredos locais, suas funções do Lambda definidas pelo usuário devem usar o AWS IoT Greengrass Core SDK v1.3.0 ou posterior.

Etapa 1: Criar um segredo do Secrets Manager

Nesta etapa, você usa o AWS Secrets Manager console para criar um segredo.

1. Faça login no console do AWS Secrets Manager.

   > **Note**
   >
   > Para obter mais informações sobre esse processo, consulte Etapa 1: Criar e armazenar seu segredo no AWS Secrets Manager no Guia do usuário do AWS Secrets Manager .

2. Selecione Armazenar um novo segredo.
3. Em Escolher tipo de segredo, selecione Outro tipo de segredo.
4. Em Specify the key/value pairs to be stored for this secret (Especificar os pares de chave/valor a serem armazenados para este segredo):
   - Em Chave, digite **`test`**.
   - Em Valor, insira **`abcdefghi`**.
5. Mantenha aws/secretsmanager selecionado para a chave de criptografia e, em seguida, escolha Avançar.

   > **Note**
   >
   > Você não será cobrado AWS KMS se usar a chave AWS gerenciada padrão que o Secrets Manager cria em sua conta.

6. Em Secret name, digite **`greengrass-TestSecret`** e selecione Next.

> **Note**
>
> Por padrão, a função de serviço do Greengrass permite AWS IoT Greengrass obter o valor dos segredos com nomes que começam com greengrass-. Para obter mais informações, consulte os requisitos de segredos.

7. A alternância não é necessária neste tutorial, portanto, selecione Desabilitar a alternância automática e, em seguida, Avançar.
8. Na página Review (Revisar), revise as configurações e selecione Store (Armazenar).

   Em seguida, crie um recurso de segredo no seu grupo do Greengrass que faz referência ao segredo.

Etapa 2: Adicionar um recurso de segredo a um grupo do Greengrass

Nesta etapa, você configura um recurso de grupo que faz referência ao segredo do Secrets Manager.

1. No painel de navegação do AWS IoT console, em Gerenciar, expanda dispositivos Greengrass e escolha Grupos (V1).
2. Selecione o grupo ao qual você deseja adicionar o recurso de segredo.
3. Na página de configuração do grupo, selecione a guia Recursos e, em seguida, role para baixo até a seção Segredos. A seção Segredos exibe os recursos de segredos que pertencem ao grupo. Você pode adicionar, editar e remover recursos de segredos nessa seção.

> **Note**
>
> Como alternativa, o console permite que você crie um segredo e um recurso de segredo ao configurar um conector ou uma função do Lambda. Você pode fazer isso na página Configurar parâmetros do conector ou na página Recursos da função do Lambda.

4. Selecione Adicionar na seção Segredos.
5. Na página Adicionar um recurso de segredo, insira **`MyTestSecret`** em Nome do recurso.
6. Em Segredo, escolha greengrass-. TestSecret

7. Na seção Selecionar rótulos (Opcional), o rótulo de preparação AWSCURRENT representa a versão mais recente do segredo. Esse rótulo é sempre incluído em um recurso de segredo.

   > **Note**
   >
   > Somente o rótulo AWSCURRENT é necessário neste tutorial. Você também pode incluir rótulos que a função do Lambda ou o connector exige.

8. Selecione Adicionar recurso.

Etapa 3: crie um pacote de implantação para a função do Lambda

Para criar uma função do Lambda, você deve, primeiro, criar um pacote de implantação da função do Lambda que contenha o código da função e as dependências. As funções do Lambda do Greengrass exigem o SDK do AWS IoT Greengrass Core para tarefas como comunicação com mensagens MQTT no ambiente de núcleo e acesso a segredos locais. Este tutorial cria uma função Python, então você usa a versão Python do SDK no pacote de implantação.

> **Note**
>
> Para obter os valores dos segredos locais, suas funções do Lambda definidas pelo usuário devem usar o AWS IoT Greengrass Core SDK v1.3.0 ou posterior.

1. Na página de downloads do AWS IoT Greengrass Core SDK, baixe o AWS IoT Greengrass Core SDK para Python em seu computador.
2. Descompacte o pacote obtido por download para obter o SDK. O SDK é a pasta do `greengrasssdk`.
3. Salve a seguinte função do código Python em um arquivo local denominado `secret_test.py`.

```
import greengrasssdk

secrets_client = greengrasssdk.client("secretsmanager")
iot_client = greengrasssdk.client("iot-data")
secret_name = "greengrass-TestSecret"
send_topic = "secrets/output"
```

```
def function_handler(event, context):
    """
    Gets a secret and publishes a message to indicate whether the secret was
    successfully retrieved.
    """
    response = secrets_client.get_secret_value(SecretId=secret_name)
    secret_value = response.get("SecretString")
    message = (
        f"Failed to retrieve secret {secret_name}."
        if secret_value is None
        else f"Successfully retrieved secret {secret_name}."
    )
    iot_client.publish(topic=send_topic, payload=message)
    print("Published: " + message)
```

A função `get_secret_value` oferece suporte ao nome ou ARN do segredo do Secrets Manager para o valor `SecretId`. Esse exemplo usa o nome do segredo. Para este exemplo, segredo, AWS IoT Greengrass retorna o par chave-valor:. `{"test":"abcdefghi"}`

> **Important**
>
> Certifique-se de que suas funções do Lambda definidas pelo usuário tratem os segredos com segurança e não registrem em log nenhum dado confidencial armazenado no segredo. Para obter mais informações, consulte Mitigate the Risks of Logging and Debugging Your Lambda Function no Guia do usuário do AWS Secrets Manager . Embora essa documentação se refira especificamente às funções de alternância, a recomendação também se aplica às funções do Lambda do Greengrass.

4. Comprima os seguintes itens em um arquivo denominado `secret_test_python.zip`. Quando você criar o arquivo ZIP, inclua apenas o código e suas dependências, e não a pasta que contém os arquivos.

   - secret_test.py. Lógica do aplicativo.
   - greengrasssdk. Biblioteca necessária para todas as funções Python do Lambda do Greengrass.

   Esse é o pacote de implantação de sua função do Lambda.

Etapa 4: Criar uma função do Lambda

Nesta etapa, você usa o AWS Lambda console para criar uma função do Lambda e configurá-la para usar seu pacote de implantação. Depois, publique uma versão da função e crie um alias.

1. Primeiro, crie a função do Lambda.

   a. No Console de gerenciamento da AWS, escolha Serviços e abra o AWS Lambda console.

   b. Selecione Criar função e, em seguida, selecione Criar do zero.

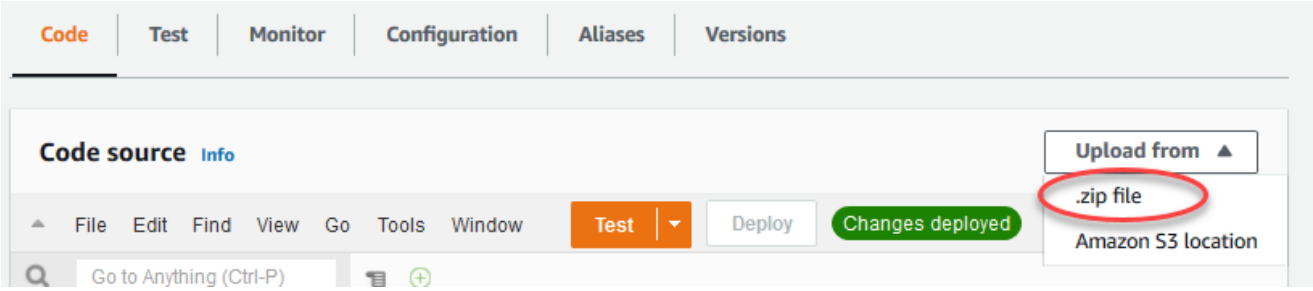
   c. Na seção Basic information (Informações básicas), use os seguintes valores:

      - Em Function name (Nome da função), insira **`SecretTest`**.
      - Em Runtime (Tempo de execução), selecione Python 3.7.
      - Em Permissões, mantenha a configuração padrão. Isso cria uma função de execução que concede permissões básicas do Lambda. Essa função não é usada por AWS IoT Greengrass.

   d. Na parte inferior da página, selecione Create function.

2. Em seguida, registre o manipulador e faça upload do seu pacote de implantação da função do Lambda.

   a. Na guia Código, em Fonte do código, selecione Fazer upload a partir de. No menu suspenso, selecione o arquivo .zip.



   b. Selecione Upload e, em seguida, selecione seu pacote de implantação `secret_test_python.zip`. Selecione Salvar.

   c. Na guia Código da função, em Configurações de runtime, selecione Editar e insira os valores a seguir.

      - Em Runtime (Tempo de execução), selecione Python 3.7.
      - Em Handler (Manipulador), insira **`secret_test.function_handler`**.

   d. Escolha Salvar.

> **Note**
>
> O botão Testar no AWS Lambda console não funciona com essa função. O SDK AWS IoT Greengrass Core não contém módulos necessários para executar suas funções do Greengrass Lambda de forma independente no console. AWS Lambda Esses módulos (por exemplo, `greengrass_common`) são fornecidos às funções depois de serem implantados no núcleo do Greengrass.

3. Agora, publique a primeira versão da sua função do Lambda e crie um alias para a versão.

> **Note**
>
> Os grupos do Greengrass podem fazer referência a uma função do Lambda por alias (recomendado) ou por versão. Usar um alias facilita o gerenciamento de atualizações de código porque você não precisa alterar a tabela de assinaturas nem a definição do grupo ao atualizar a função do código. Em vez disso, você pode simplesmente apontar o alias para a nova versão da função.

   a. No menu Actions, selecione Publish new version.

   b. Em Version description (Descrição da versão), insira **`First version`** e, em seguida, selecione Publish (Publicar).

   c. Na página de configuração SecretTest: 1, no menu Ações, escolha Criar alias.

   d. Na página Create a new alias, use os seguintes valores:

      - Em Nome, digite **`GG_SecretTest`**.
      - Em Version, selecione 1.

      > **Note**
      >
      > AWS IoT Greengrass não oferece suporte a aliases do Lambda para as versões $LATEST.

   e. Selecione Create (Criar).

Agora, você está pronto para adicionar a função do Lambda ao seu grupo do Greengrass e anexar o recurso de segredo.

Etapa 5: adicionar a função do Lambda ao grupo do Greengrass

Nesta etapa, você adiciona a função do Lambda ao grupo do Greengrass no console do AWS IoT .

1. Na página de configuração do grupo, selecione a guia Funções do Lambda.
2. Na seção Minhas funções do Lambda, selecione Adicionar.
3. Para a função Lambda, escolha SecretTest.
4. Para a Versão da função do Lambda, selecione o alias para a versão que você publicou.

Em seguida, configure o ciclo de vida da função do Lambda.

1. Na seção configuração da função do Lambda, faça as atualizações a seguir.

   > **Note**
   >
   > Recomendamos que você execute sua função do Lambda sem conteinerização, a menos que seu caso de negócios faça essa exigência. Isso ajuda a habilitar o acesso à GPU e à câmera do seu dispositivo sem configurar os recursos do dispositivo. Se você executar sem conteinerização, também deverá conceder acesso root às suas funções do AWS IoT Greengrass Lambda.

   a. Para executar sem conteinerização:

      - Para Usuário e grupo do sistema, selecione **`Another user ID/group ID`**. Para ID de usuário do sistema, insira **`0`**. Para ID do grupo do sistema, insira **`0`**.

        Isso permite que sua função do Lambda seja executada como root. Para obter mais informações sobre como executar como raiz, consulte the section called “Definir a identidade de acesso padrão para as funções do Lambda em um grupo”.

> **Tip**
>
> Você também deve atualizar seu arquivo `config.json` para conceder acesso root à sua função do Lambda. Para o procedimento, consulte the section called "Executar uma função do Lambda como raiz".

- Para conteinerização da função do Lambda, selecione Sem contêiner.

  Para obter mais informações sobre como executar sem conteinerização, consulte the section called "Considerações ao escolher a conteinerização de função do Lambda".
- Em Timeout (Tempo limite), insira **`10 seconds`**.
- Para Fixado, selecione Verdadeiro.

  Para obter mais informações, consulte the section called "Configuração do ciclo de vida".
- Em Parâmetro adicional, para Acesso de leitura ao diretório /sys, selecione Habilitado.

b. Em vez disso, para executar no modo conteinerizado:

> **Note**
>
> Não recomendamos a execução no modo conteinerizado, a menos que seu caso de negócios faça essa exigência.

- Para Usuário e grupo do sistema, selecione Usar padrão de grupo.
- Para Conteinerização da função do Lambda, selecione Usar padrão de grupo.
- Em Memory limit (Limite de memória), insira **`1024 MB`**.
- Em Timeout (Tempo limite), insira **`10 seconds`**.
- Para Fixado, selecione Verdadeiro.

  Para obter mais informações, consulte the section called "Configuração do ciclo de vida".
- Em Parâmetros adicionais, para Acesso de leitura ao diretório /sys, selecione Habilitado.

2. Selecione Adicionar função do Lambda.

Em seguida, associe o recurso de segredo à função.

Etapa 6: anexar o recurso de segredo à função do Lambda

Nesta etapa, você associa o recurso de segredo à função do função do Lambda em seu grupo do Greengrass. Isso associa o recurso à função, o que permite à função obter o valor do segredo local.

1. Na página de configuração do grupo, selecione a guia Funções do Lambda.
2. Escolha a SecretTest função.
3. Na página de detalhes da função, selecione Recursos.
4. Role até a seção Segredos e selecione Associar.
5. Escolha e MyTestSecret, em seguida, escolha Associar.

Etapa 7: Adicionar assinaturas ao grupo do Greengrass

Nesta etapa, você adiciona assinaturas que permitem AWS IoT e a função Lambda trocar mensagens. Uma assinatura permite AWS IoT invocar a função e outra permite que a função envie dados de saída para AWS IoT.

1. Na página de configuração do grupo, selecione a guia Assinaturas e, em seguida, Adicionar assinatura.
2. Crie uma assinatura que AWS IoT permita publicar mensagens na função.

   Na página de configuração do grupo, selecione a guia Assinaturas e, em seguida, Adicionar assinatura.
3. Na página Criar uma assinatura, configure a origem e o destino, da seguinte forma:

   a. Em Tipo de origem, selecione Função do Lambda e, em seguida, selecione IoT Cloud.

   b. Em Tipo de destino, escolha Serviço e, em seguida, escolha SecretTest.

   c. Em Filtro de tópicos, insira **`secrets/input`** e, em seguida, selecione Criar assinatura.
4. Adicione uma segunda assinatura. Selecione a guia Assinaturas, selecione Adicionar assinatura e configure a origem e o destino, da seguinte forma:

   a. Em Tipo de origem, escolha Serviços e, em seguida, escolha SecretTest.

   b. Em Tipo de destino, selecione Função do Lambda e, em seguida, selecione IoT Cloud.

   c. Em Filtro de tópicos, insira **`secrets/output`** e, em seguida, selecione Criar assinatura.

Etapa 8: Implantar o grupo do Greengrass

Implante o grupo no dispositivo do núcleo. Durante a implantação, AWS IoT Greengrass busca o valor do segredo no Secrets Manager e cria uma cópia criptografada local no núcleo.

1. Verifique se o AWS IoT Greengrass núcleo está funcionando. Execute os seguintes comandos no seu terminal do Raspberry Pi, conforme necessário.

   a. Para verificar se o daemon está em execução:

   ```
   ps aux | grep -E 'greengrass.*daemon'
   ```

   Se a saída contém uma entrada `root` para `/greengrass/ggc/packages/`*`ggc-version`*`/bin/daemon`, o daemon está em execução.

   > **Note**
   >
   > A versão no caminho depende da versão do software AWS IoT Greengrass Core instalada em seu dispositivo principal.

   b. Para iniciar o daemon:

   ```
   cd /greengrass/ggc/core/
   sudo ./greengrassd start
   ```

2. Na página de configuração do grupo, selecione Implantar.

3. a. Na guia Funções do Lambda, na seção Funções do Lambda do sistema, selecione Detector de IP e selecione Editar.

   b. Na caixa de diálogo Editar configurações do detector IP, selecione Detectar e substituir automaticamente os endpoints do corretor MQTT.

   c. Selecione Salvar.

   Isso permite que os dispositivos adquiram automaticamente as informações de conectividade para o núcleo, como endereço IP, DNS e o número da porta. A detecção automática é recomendada, mas AWS IoT Greengrass também oferece suporte a endpoints especificados manualmente. Você só é solicitado pelo método de descoberta na primeira vez em que o grupo for implantado.

> **Note**
>
> Se solicitado, conceda permissão para criar a função de serviço do Greengrass e associá-la à sua Conta da AWS na atual. Região da AWS Essa função permite AWS IoT Greengrass acessar seus recursos em AWS serviços.

A página Deployments mostra a data e hora, ID de versão e status da implantação. Quando concluída, o status exibido para a implantação deve ser Concluída.

Para obter ajuda sobre a solução de problemas, consulte Solução de problemas.

Testar a função do Lambda

1. Na página inicial do AWS IoT console, escolha Testar.
2. Em Inscrever-se no tópico, use os seguintes valores e, em seguida, selecione Inscrever-se.

Propriedade	Valor
Tópico de assinatura	secrets/output
Exibição da carga útil do MQTT	Exibir cargas úteis como strings

3. Para Publicar no tópico, use os valores a seguir e, em seguida selecione Publicar para invocar a função.

Propriedade	Valor
Tópico	secrets/input
Mensagem	Mantenha a mensagem padrão. A publicaçã o de uma mensagem invoca a função do Lambda, mas a função neste tutorial não processa o corpo da mensagem.

Se for bem-sucedida, a função publicará uma mensagem de "Êxito".

Consulte também

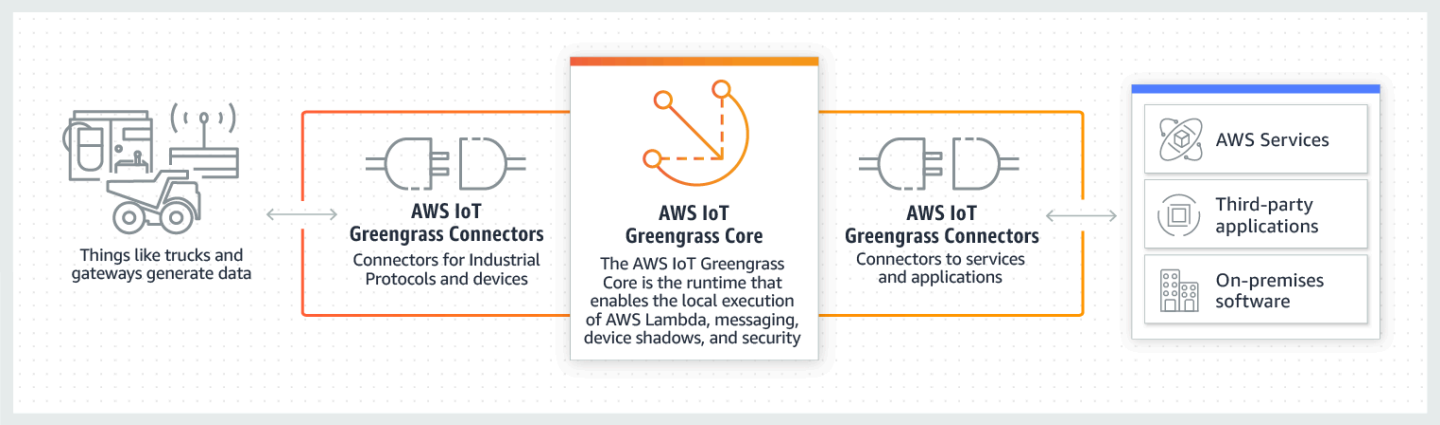
- Implantar segredos no núcleo

Integrar a serviços e protocolos usando conectores do Greengrass

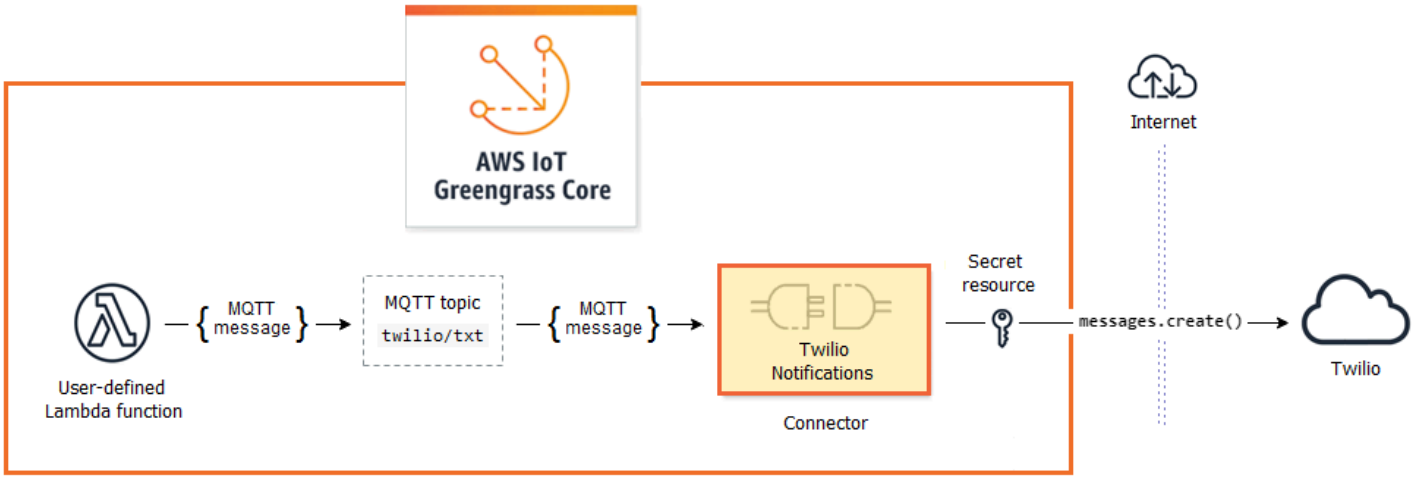
Esse recurso está disponível para AWS IoT Greengrass Core v1.7 e versões posteriores.

Os conectores AWS IoT Greengrass internos são módulos pré-construídos que tornam mais eficiente a interação com a infraestrutura local AWS, protocolos de dispositivos e outros serviços em nuvem. Usando os conectores, você pode perder menos tempo aprendendo novos protocolos e APIs e mais tempo se concentrando na lógica que é importante para a sua empresa.

O diagrama a seguir mostram onde os conectores podem se encaixar no cenário AWS IoT Greengrass do .



Muitos conectores usam mensagens MQTT para se comunicar com dispositivos clientes e funções do Greengrass Lambda no grupo ou com o serviço AWS IoT paralelo local. No exemplo a seguir, o conector de notificações do Twilio recebe mensagens MQTT de uma função Lambda definida pelo usuário, usa uma referência local de um segredo e chama a API do AWS Secrets Manager Twilio.



Para ver tutoriais que criam essa solução, consulte the section called “Conceitos básicos de conectores (console)” e the section called “Conceitos básicos de conectores (CLI)”.

Os conectores do Greengrass podem ajudar você a ampliar os recursos do dispositivo ou a criar dispositivos de finalidade única. Ao usar conectores, você pode:

- Implementar a lógica de negócios reutilizáveis.
- Interaja com serviços locais e na nuvem, incluindo AWS serviços de terceiros.
- Ingerir e o processar dados do dispositivo.
- Habilitar as chamadas de um dispositivo para outro usando assinaturas de tópicos MQTT e funções do Lambda definidas pelo usuário.

AWS fornece um conjunto de conectores do Greengrass que simplificam as interações com serviços e fontes de dados comuns. Esses módulos pré-compilados habilitam cenários para registro e diagnóstico, reabastecimento, processamento de dados industriais, além de alarmes e mensagens. Para obter mais informações, consulte the section called “Conectores do Greengrass fornecidos pela AWS”.

Requisitos

Para usar conectores, lembre-se dos seguintes pontos:

- Cada conector que você usa possui requisitos aos quais você deve atender. Esses requisitos podem incluir a versão mínima do software AWS IoT Greengrass Core, pré-requisitos do

dispositivo, permissões e limites necessários. Para obter mais informações, consulte the section called “Conectores do Greengrass fornecidos pela AWS”.

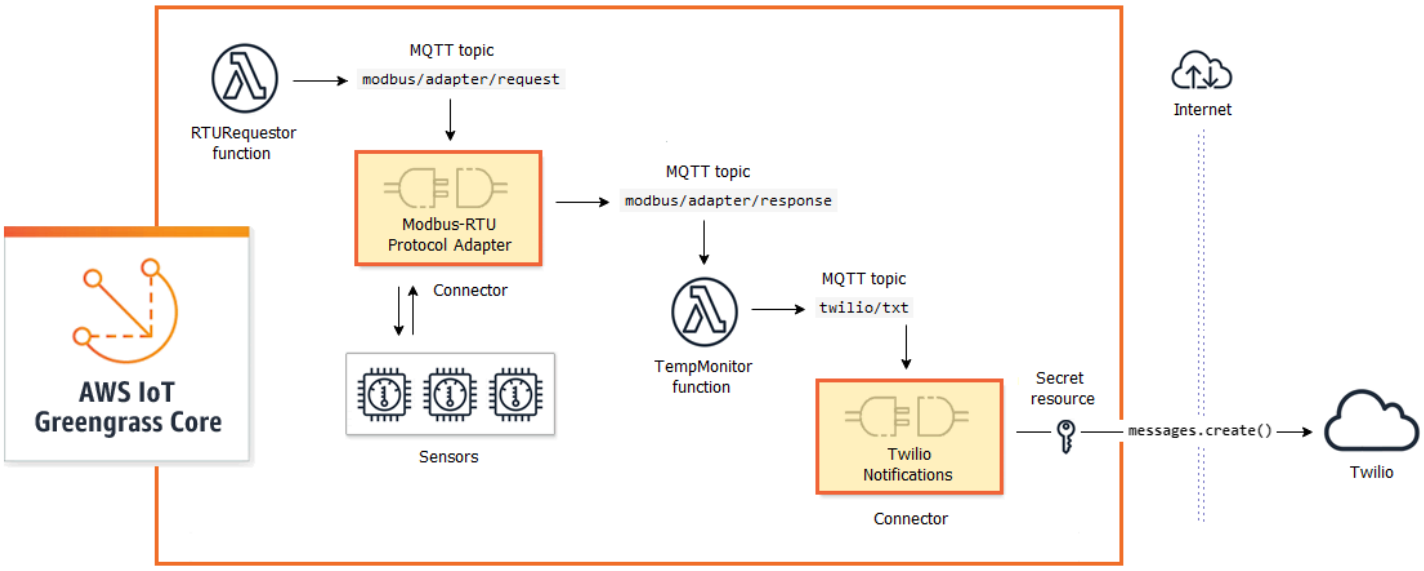
- Um grupo do Greengrass pode conter apenas uma instância configurada de um determinado conector. Porém, você pode a usar a instância em várias assinaturas. Para obter mais informações, consulte the section called “Parâmetros de configuração”.
- Quando a conteinerização padrão para o grupo do Greengrass é definida como Sem contêiner, os conectores do grupo devem ser executados sem conteinerização. Para localizar conectores compatíveis com o modo Sem contêiner consulte the section called “Conectores do Greengrass fornecidos pela AWS”.

Usar conectores do Greengrass

Um conector é um tipo de componente de grupo. Assim como outros componentes do grupo, como dispositivos cliente e funções do Lambda definidas pelo usuário, você adiciona conectores aos grupos, define suas configurações e os implanta no AWS IoT Greengrass núcleo. Os conectores são executados no ambiente do núcleo.

Você pode implantar alguns conectores como aplicativos autônomos simples. Por exemplo, o conector Device Defender lê as métricas do sistema no dispositivo de núcleo e as envia para o AWS IoT Device Defender para análise.

Você pode adicionar outros conectores como blocos de construção em soluções maiores. O exemplo de solução a seguir usa o conector do adaptador de Modbus-RTU protocolo para processar mensagens dos sensores e o conector de notificações do Twilio para iniciar as mensagens do Twilio.



As soluções geralmente incluem funções do Lambda definidas pelo usuário que permanecem ao lado dos conectores e processam os dados que o conector envia ou recebe. Neste exemplo, a TempMonitor função recebe dados do adaptador de Modbus-RTU protocolo, executa alguma lógica de negócios e, em seguida, envia dados para as notificações do Twilio.

Para criar e implantar uma solução, siga este processo geral:

1. Mapeie o fluxo de dados de alto nível. Identifique as fontes de dados, os canais de dados, os serviços, os protocolos e os recursos com os quais você precisa trabalhar. No solução de exemplo, isso inclui os dados sobre o protocolo Modbus RTU, a porta serial Modbus física e o Twilio.
2. Identifique os conectores para incluir na solução e adicione-os ao grupo. A solução de exemplo usa Adaptador de Modbus-RTU Protocolo e Notificações Twilio. Para ajudá-lo a encontrar os conectores que se aplicam ao seu cenário, bem como para saber mais sobre seus requisitos individuais, consulte the section called “Conectores do Greengrass fornecidos pela AWS”.
3. Identifique se as funções do Lambda definidas pelo usuário, os dispositivos cliente, ou os recursos são necessários, então, crie-os e adicione-os ao grupo. Isso pode incluir funções que contêm a lógica de negócios ou processam dados em um formato exigido por outra entidade na solução. O exemplo de solução usa funções para enviar solicitações do Modbus RTU e iniciar as notificações do Twilio. Ela também inclui um recurso de dispositivo local para a porta serial Modbus RTU e um recurso de segredo para o token de autenticação do Twilio.

> **Note**
>
> Recursos secretos fazem referência a senhas, tokens e outros segredos do AWS Secrets Manager. Os segredos podem ser usados por conectores e funções do Lambda para a autenticação nos serviços e aplicativos. Por padrão, AWS IoT Greengrass pode acessar segredos com nomes que começam com "greengrass-". Para obter mais informações, consulte Implantar segredos no núcleo.

4. Crie assinaturas que permitem que as entidades na solução troquem mensagens MQTT. Se um conector é usado em uma assinatura, o conector e a mensagem de origem ou de destino devem usar a sintaxe de tópico predefinida compatível com o conector. Para obter mais informações, consulte the section called "Entradas e saídas".
5. Implante o grupo no núcleo do Greengrass.

Para obter informações sobre como criar e implantar um conector, consulte os seguintes tutoriais:

- the section called "Conceitos básicos de conectores (console)"
- the section called "Conceitos básicos de conectores (CLI)"

Parâmetros de configuração

Muitos conectores fornecem parâmetros que permitem personalizar o comportamento ou a saída. Esses parâmetros são usados durante a inicialização, no tempo de execução ou em outros momentos no ciclo de vida do conector.

Os tipos de parâmetro e o uso variam de acordo com o conector. Por exemplo, o conector SNS tem um parâmetro que configura o tópico padrão do SNS e o Device Defender tem um parâmetro que configura a taxa de amostragem de dados.

Uma versão de grupo pode conter vários conectores, mas apenas a instância de determinado conector por vez. Isso significa que cada conector no grupo pode ter somente uma configuração ativa. Porém, a instância do conector pode ser usada em várias assinaturas no grupo. Por exemplo, é possível criar assinaturas que permitem que muitos dispositivos enviem dados ao conector do Kinesis Firehose.

Parâmetros usados para acessar recursos de grupo

Os conectores do Greengrass usam recursos de grupo para acessar o sistema de arquivos, as portas, os periféricos e outros recursos locais no dispositivo de núcleo. Se um conector exige acesso a um recurso de grupo, ele fornece parâmetros de configuração relacionados.

Recursos de grupo incluem:

- Recursos locais. Os diretórios, os arquivos, as portas, os pins e os periféricos que estão presentes no dispositivo de núcleo do Greengrass.
- Recursos de machine learning. Os modelos de machine learning formados na nuvem e implantados no núcleo para inferência local.
- Recursos secretos. Cópias locais e criptografadas de senhas, chaves, tokens ou texto arbitrário do AWS Secrets Manager. Os conectores podem acessar esses segredos locais com segurança e usá-los para autenticar para serviços ou infraestrutura local.

Por exemplo, os parâmetros do Device Defender habilitam o acesso a métricas do sistema no diretório `/proc` do host e os parâmetros das notificações do Twilio habilitam o acesso a um token de autenticação do Twilio armazenado localmente.

Atualizar parâmetros de conector

Os parâmetros são configurados quando o conector é adicionado a um grupo do Greengrass. É possível alterar valores de parâmetro após o conector ser adicionado.

- No console: na página de configurações do grupo, abra Connectors (Conectores) e, no menu de contexto do conector, selecione Edit (Editar).

  > **Note**
  >
  > Se o conector usar um recurso de segredo que foi alterado posteriormente para fazer referência a um segredo diferente, você deverá editar os parâmetros do conector e confirmar a alteração.

- Na API: crie outra versão do conector que defina a nova configuração.

  A AWS IoT Greengrass API usa versões para gerenciar grupos. As versões são imutáveis, portanto, para adicionar ou alterar componentes do grupo — por exemplo, os dispositivos

cliente, as funções e os recursos do grupo — você deve criar versões de componentes novos ou atualizados. Em seguida, crie e implante uma versão do grupo que contenha a versão de destino de cada componente.

Depois de fazer alterações na configuração do conector, você deverá implantar o grupo para propagar as alterações para o núcleo.

Entradas e saídas

Muitos conectores do Greengrass podem se comunicar com outras entidades pelo envio e recebimento de mensagens MQTT. A comunicação MQTT é controlada por assinaturas que permitem que um conector troque dados com funções do Lambda, dispositivos cliente e outros conectores no grupo Greengrass ou com AWS IoT o serviço paralelo local. Para permitir essa comunicação, você deve criar assinaturas no grupo ao qual o conector pertence. Para obter mais informações, consulte the section called “Assinaturas gerenciadas no fluxo de trabalho de mensagens MQTT”.

Os conectores podem ser editores de mensagens, assinantes de mensagens ou ambos. Cada conector define os tópicos MQTT ao qual ele publica ou o qual assina. Esses tópicos predefinidos devem ser usados em assinaturas nas quais o conector é uma mensagem de origem ou de destino. Para obter tutoriais que incluem etapas de configuração de assinaturas para um conector, consulte the section called “Conceitos básicos de conectores (console)” e the section called “Conceitos básicos de conectores (CLI)”.

> **Note**
>
> Muitos conectores também têm modos integrados de comunicação para interagir com serviços locais ou de nuvem. Isso varia de acordo com o conector e pode exigir que você configure parâmetros ou adicione permissões à função do grupo. Para obter mais informações sobre requisitos de conector, consulte the section called “Conectores do Greengrass fornecidos pela AWS”.

Tópicos de entrada

A maioria dos conectores recebe dados de entrada em tópicos MQTT. Alguns conectores assinam vários tópicos para dados de entrada. Por exemplo, o conector do Serial Stream oferece suporte a dois tópicos:

- `serial/+/read/#`
- `serial/+/write/#`

Para esse conector, as solicitações de leitura e gravação são enviadas para o tópico correspondente. Ao criar assinaturas, certifique-se de usar o tópico que se alinhe à sua implementação.

Os caracteres + e # nos exemplos anteriores são curingas. Esses curingas permitem que os assinantes recebam mensagens sobre diversos tópicos e que os editores personalizem os tópicos em que publicam.

- O curinga + pode aparecer em qualquer lugar na hierarquia de tópicos. Ele pode ser substituído por um item de hierarquia.

  Como exemplo, para o tópico `sensor/+/input`, as mensagens podem ser publicadas em tópicos `sensor/id-123/input`, mas não em `sensor/group-a/id-123/input`.

- O curinga # pode aparecer somente no final da hierarquia de tópicos. Ele pode ser substituído por zero ou mais itens de hierarquia.

  Como exemplo, para o tópico `sensor/#`, as mensagens podem ser publicadas em `sensor/`, `sensor/id-123` e `sensor/group-a/id-123`, mas não em `sensor`.

Caracteres curinga são válidos apenas ao assinar tópicos. As mensagens não podem ser publicadas em tópicos que contêm caracteres curinga. Verifique a documentação do conector para obter mais informações sobre seus requisitos de tópico de entrada ou saída. Para obter mais informações, consulte the section called “Conectores do Greengrass fornecidos pela AWS”.

Suporte a conteinerização

Por padrão, a maioria dos conectores é executada no núcleo do Greengrass, em um ambiente de tempo de execução isolado gerenciado pelo AWS IoT Greengrass. Esses ambientes de tempo de execução, chamados de contêineres, fornecem isolamento entre os conectores e o sistema de host, o que oferece mais segurança para o host e para o conector.

No entanto, essa conteinerização do Greengrass não é suportada em alguns ambientes, como quando você executa AWS IoT Greengrass em um contêiner Docker ou em kernels Linux mais antigos sem cgroups. Nesses ambientes, os conectores devem ser executados no modo Sem

contêiner. Para localizar conectores compatíveis com o modo Sem contêiner consulte the section called “Conectores do Greengrass fornecidos pela AWS”. Alguns conectores são executados nesse modo de forma nativa, outros permitem que você defina o modo de isolamento.

Também é possível definir o modo de isolamento como Sem contêiner em ambientes compatíveis com a conteinerização do Greengrass, mas recomendamos usar o modo de contêiner do Greengrass quando possível.

> **Note**
>
> A configuração padrão de conteinerização para o grupo do Greengrass não se aplica aos conectores.

Atualizar a versões do conector

Os provedores do conector podem lançar novas versões de um conector que adicionem atributos, corrijam problemas ou melhorem o desempenho. Para obter informações sobre as versões disponíveis e alterações relacionadas, consulte a documentação de cada conector.

No AWS IoT console, você pode verificar se há novas versões para os conectores em seu grupo do Greengrass.

1. No painel de navegação do AWS IoT console, em Gerenciar, expanda dispositivos Greengrass e escolha Grupos (V1).
2. Em Grupos do Greengrass, selecione seu grupo.
3. Selecione Conectores para exibir os conectores no grupo.

   Se o conector tiver uma nova versão, um botão Available (Disponível) será exibido na coluna Upgrade (Atualizar) .
4. Como atualizar a versão do conector:

   a. Na página Conectores, na coluna Upgrade (Atualizar), selecione Available (Disponível). A página Upgrade connector (Atualizar conector) é aberta e exibe as definições de parâmetros atuais, se aplicável.

      Selecione a nova versão do conector, defina os parâmetros conforme necessário e, em seguida selecione Upgrade (Atualizar).

b. Na página Subscriptions (Assinaturas), adicione novas assinaturas no grupo para substituir as que usam o conector como origem ou destino. Remova, então, as assinaturas antigas.

   As assinaturas fazem referência aos conectores por versão, portanto, elas se tornam inválidos se você alterar a versão do conector no grupo.

c. No menu Ações selecione Implantar para implantar as alterações no núcleo.

Para atualizar um conector da AWS IoT Greengrass API, crie e implante uma versão de grupo que inclua o conector e as assinaturas atualizados. Use o mesmo processo de adicionar um conector a um grupo. Para obter etapas detalhadas que mostram como usar o AWS CLI para configurar e implantar um exemplo de conector de notificações do Twilio, consulte. the section called “Conceitos básicos de conectores (CLI)”

Registro em log para conectores

Os conectores do Greengrass contém funções do Lambda que gravam eventos e erros em logs do Greengrass. Dependendo das configurações do seu grupo, os registros são gravados CloudWatch nos Registros, no sistema de arquivos local ou em ambos. Os logs de conectores incluem o ARN da função correspondente. O exemplo de ARN a seguir é do conector do Kinesis Firehose:

```
arn:aws:lambda:aws-region:account-id:function:KinesisFirehoseClient:1
```

A configuração de registro de log padrão grava logs informativos no sistema de arquivos usando a seguinte estrutura de diretório:

```
greengrass-root/ggc/var/log/user/region/aws/function-name.log
```

Para obter mais informações sobre o registro em log do Greengrass, consulte the section called “Monitoramento com logs do AWS IoT Greengrass ”.

AWS Conectores do Greengrass fornecidos pela

AWS fornece os seguintes conectores que suportam AWS IoT Greengrass cenários comuns. Para obter mais informações sobre o funcionamento dos conectores, consulte a seguinte documentação:

- Integrar a serviços e protocolos usando conectores

- Conceitos básicos de conectores (console) ou Conceitos básicos de conectores (CLI)

Conector	Description	Runtimes compatíveis do Lambda	Compatível com o modo Sem contêiner
CloudWatch Métricas	Publica métricas personalizadas na Amazon CloudWatch.	• Python 3.8 * • Python 3.7 • Python 2.7	Sim
Device Defender	Envia métricas do sistema para AWS IoT Device Defender.	• Python 3.8 * • Python 3.7 • Python 2.7	Não
Implantação de aplicativos Docker	Executa um arquivo Compose do Docker para iniciar um aplicativo Docker no dispositivo de núcleo.	• Python 3.8 • Python 3.7	Sim
IoT Analytics (descontinuado)	Envia dados de dispositivos e sensores para AWS IoT Analytics o.	• Python 3.8 * • Python 3.7 • Python 2.7	Sim
Adaptador do protocolo IP Ethernet IoT	Coleta dados dos Ethernet/IP dispositivos.	• Java 8	Sim
IoT SiteWise	Envia dados de dispositivos e sensores para propriedades de ativos em AWS IoT SiteWise.	• Java 8	Sim
Kinesis Firehose	Envia dados aos fluxos de entrega do Amazon Data Firehose.	• Python 3.8 * • Python 3.7	Sim

Conector	Description	Runtimes compatíveis do Lambda	Compatível com o modo Sem contêiner
		• Python 2.7	
Feedback do ML	Publica a entrada do modelo de machine learning na nuvem e a saída em um tópico MQTT.	• Python 3.8 • Python 3.7	Não
Classificação de imagens do ML	Executa um serviço de inferência de classificação de imagem local. Esse conector fornece versões para várias plataformas.	• Python 3.8 * • Python 3.7 • Python 2.7	Não
Detecção de objetos do ML	Executa um serviço de inferência de detecção de objetos locais. Esse conector fornece versões para várias plataformas.	• Python 3.8 • Python 3.7	Não
Modbus-RTU Adaptador de protocolo	Envia solicitações para dispositivos Modbus RTU.	• Python 3.8 * • Python 3.7 • Python 2.7	Não
Modbus-TCP Adaptador de protocolo	Coleta dados de dispositivos ModbusTCP.	• Java 8	Sim
Raspberry Pi GPIO	Controla pins GPIO em um dispositivo de núcleo Raspberry Pi.	• Python 3.8 * • Python 3.7 • Python 2.7	Não

Conector	Description	Runtimes compatíveis do Lambda	Compatível com o modo Sem contêiner
Fluxo serial	Faz leituras e gravações em uma porta serial no dispositivo de núcleo.	• Python 3.8 * • Python 3.7 • Python 2.7	Não
ServiceNow MetricBase Integration	Publica métricas de séries temporais em ServiceNow MetricBase.	• Python 3.8 * • Python 3.7 • Python 2.7	Sim
SNS	Envia mensagens a um tópico do Amazon SNS.	• Python 3.8 * • Python 3.7 • Python 2.7	Sim
Integração Splunk	Publica dados no HEC do Splunk.	• Python 3.8 * • Python 3.7 • Python 2.7	Sim
Notificações Twilio	Inicia uma mensagem de texto ou de voz do Twilio.	• Python 3.8 * • Python 3.7 • Python 2.7	Sim

* Para usar os runtimes do Python 3.8, você deve criar um symblink da pasta de instalação padrão do Python 3.7 para os binários instalados do Python 3.8. Para obter mais informações, consulte os requisitos específicos do conector.

> **Note**
>
> Recomendamos que você atualize as versões do conector do Python 2.7 para o Python 3.7. O suporte contínuo para conectores Python 2.7 depende do suporte ao AWS Lambda tempo de execução. Para obter mais informações, consulte Política de suporte do Runtime no Guia do desenvolvedor do AWS Lambda .

CloudWatch Conector de métricas

O conector CloudWatch Metrics publica métricas personalizadas dos dispositivos Greengrass na Amazon. CloudWatch O conector fornece uma infraestrutura centralizada para publicação de CloudWatch métricas, que você pode usar para monitorar e analisar o ambiente principal do Greengrass e atuar em eventos locais. Para obter mais informações, consulte Usando CloudWatch métricas da Amazon no Guia CloudWatch do usuário da Amazon.

Esse conector recebe dados de métrica como mensagens MQTT. O conector agrupa métricas que estão no mesmo namespace e as publica em intervalos regulares. CloudWatch

Esse conector tem as seguintes versões.

Versão	ARN
5	`arn:aws:greengrass:region::/connectors/CloudWatchMetrics/versions/5`
4	`arn:aws:greengrass:region::/connectors/CloudWatchMetrics/versions/4`
3	`arn:aws:greengrass:region::/connectors/CloudWatchMetrics/versions/3`
2	`arn:aws:greengrass:region::/connectors/CloudWatchMetrics/versions/2`

Versão	ARN
1	`arn:aws:greengrass:  region::/connectors/CloudWatchMetrics/versions/1`

Para obter informações sobre alterações de versão, consulte o Changelog.

Requisitos

Esse conector tem os seguintes requisitos:

Version 3 - 5

- AWS IoT Greengrass Software principal v1.9.3 ou posterior.
- Python, versão 3.7 ou 3.8, instalado no dispositivo de núcleo e adicionado à variável de ambiente PATH.

  > **Note**
  >
  > Para usar o Python 3.8, execute o comando a seguir para criar um symblink da pasta de instalação padrão do Python 3.7 para os binários instalados do Python 3.8.
  >
  > ```
  > sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
  > ```
  >
  > Isso configura seu dispositivo para atender ao requisito Python para AWS IoT Greengrass.

- A função de grupo do Greengrass configurada para permitir a ação `cloudwatch:PutMetricData`, conforme mostrado no seguinte exemplo de política do AWS Identity and Access Management (IAM).

  JSON

  ```
  {
      "Version":"2012-10-17",
      "Statement": [
          {
              "Sid": "Stmt1528133056761",
  ```

```
            "Action": [
                "cloudwatch:PutMetricData"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
}
```

Para o requisito de função de grupo, você deve configurar a função para conceder as permissões necessárias e certificar-se de que a função tenha sido adicionada ao grupo. Para acessar mais informações, consulte the section called “Gerenciar a função de grupo (console)” ou the section called “Gerenciar a função de grupo (CLI)”.

Para obter mais informações sobre CloudWatch permissões, consulte a referência de CloudWatch permissões da Amazon no Guia do usuário do IAM.

Versions 1 - 2

- AWS IoT Greengrass Software principal v1.7 ou posterior.
- Python versão 2.7 instalado no dispositivo de núcleo e adicionado à variável de ambiente PATH.
- A função de grupo do Greengrass configurada para permitir a ação `cloudwatch:PutMetricData`, conforme mostrado no seguinte exemplo de política do AWS Identity and Access Management (IAM).

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1528133056761",
            "Action": [
                "cloudwatch:PutMetricData"
            ],
            "Effect": "Allow",
            "Resource": "*"
        }
    ]
```

```
}
```

Para o requisito de função de grupo, você deve configurar a função para conceder as permissões necessárias e certificar-se de que a função tenha sido adicionada ao grupo. Para acessar mais informações, consulte the section called “Gerenciar a função de grupo (console)” ou the section called “Gerenciar a função de grupo (CLI)”.

Para obter mais informações sobre CloudWatch permissões, consulte a referência de CloudWatch permissões da Amazon no Guia do usuário do IAM.

Parâmetros do conector

Esse conector oferece os seguintes parâmetros:

Versions 4 - 5

`PublishInterval`

O número máximo de segundos de espera antes de publicar métricas em lote para um determinado namespace. O valor máximo é 900. Para configurar o conector a fim de publicar métricas conforme são recebidas (sem agrupamento em lotes), especifique 0.

O conector publica CloudWatch após receber 20 métricas no mesmo namespace ou após o intervalo especificado.

> **Note**
>
> O conector não garante a ordem de publicação de eventos.

Nome de exibição no AWS IoT console: intervalo de publicação

Obrigatório: `true`

Digite: `string`

Valores válidos: `0 - 900`

Padrão válido: `[0-9]|[1-9]\d|[1-9]\d\d|900`

`PublishRegion`

O Região da AWS para publicar CloudWatch métricas em. Esse valor substitui a região padrão de métricas do Greengrass. É necessário apenas ao publicar métricas entre regiões.

Nome de exibição no AWS IoT console: região de publicação

Obrigatório: `false`

Digite: `string`

Padrão válido: `^$|([a-z]{2}-[a-z]+-\d{1})`

`MemorySize`

A memória (em KB) de alocação no conector.

Nome de exibição no AWS IoT console: Tamanho da memória

Obrigatório: `true`

Digite: `string`

Padrão válido: `^[0-9]+$`

`MaxMetricsToRetain`

O número máximo de métricas em todos os namespaces para salvar na memória antes que sejam substituídas por novas métricas. O valor mínimo é 2000.

Esse limite aplica-se quando não há conexão com a Internet, e o conector começa a armazenar as métricas em buffer para publicar posteriormente. Quando o buffer está cheio, as métricas mais antigas são substituídas por novas métricas. As métricas em um determinado namespace são substituídas apenas por métricas no mesmo namespace.

> **Note**
>
> As métricas não são salvas se o processo de host do conector é interrompido. Por exemplo, essa interrupção pode ocorrer durante a implantação do grupo ou quando o dispositivo é reiniciado.

Nome de exibição no AWS IoT console: Métricas máximas a serem retidas

Obrigatório: `true`

Digite: `string`

Padrão válido: `^([2-9]\d{3}|[1-9]\d{4,})$`

`IsolationMode`

O modo de conteinerização para este conector. O padrão é`GreengrassContainer`, o que significa que o conector é executado em um ambiente de execução isolado dentro do AWS IoT Greengrass contêiner.

> **Note**
>
> A configuração padrão de conteinerização para o grupo não se aplica aos conectores.

Nome de exibição no AWS IoT console: modo de isolamento de contêiner

Obrigatório: `false`

Digite: `string`

Valores válidos: `GreengrassContainer` ou `NoContainer`

Padrão válido: `^NoContainer$|^GreengrassContainer$`

Versions 1 - 3

`PublishInterval`

O número máximo de segundos de espera antes de publicar métricas em lote para um determinado namespace. O valor máximo é 900. Para configurar o conector a fim de publicar métricas conforme são recebidas (sem agrupamento em lotes), especifique 0.

O conector publica CloudWatch após receber 20 métricas no mesmo namespace ou após o intervalo especificado.

> **Note**
>
> O conector não garante a ordem de publicação de eventos.

Nome de exibição no AWS IoT console: intervalo de publicação

Obrigatório: `true`

Digite: `string`

Valores válidos: `0 - 900`

Padrão válido: `[0-9]|[1-9]\d|[1-9]\d\d|900`

`PublishRegion`

O Região da AWS para publicar CloudWatch métricas em. Esse valor substitui a região padrão de métricas do Greengrass. É necessário apenas ao publicar métricas entre regiões.

Nome de exibição no AWS IoT console: região de publicação

Obrigatório: `false`

Digite: `string`

Padrão válido: `^$|([a-z]{2}-[a-z]+-\d{1})`

`MemorySize`

A memória (em KB) de alocação no conector.

Nome de exibição no AWS IoT console: Tamanho da memória

Obrigatório: `true`

Digite: `string`

Padrão válido: `^[0-9]+$`

`MaxMetricsToRetain`

O número máximo de métricas em todos os namespaces para salvar na memória antes que sejam substituídas por novas métricas. O valor mínimo é 2000.

Esse limite aplica-se quando não há conexão com a Internet, e o conector começa a armazenar as métricas em buffer para publicar posteriormente. Quando o buffer está cheio, as métricas mais antigas são substituídas por novas métricas. As métricas em um determinado namespace são substituídas apenas por métricas no mesmo namespace.

> **Note**
>
> As métricas não são salvas se o processo de host do conector é interrompido. Por exemplo, essa interrupção pode ocorrer durante a implantação do grupo ou quando o dispositivo é reiniciado.

Nome de exibição no AWS IoT console: Métricas máximas a serem retidas

Obrigatório: `true`

Digite: `string`

Padrão válido: `^([2-9]\d{3}|[1-9]\d{4,})$`

Exemplo de criação de conector (AWS CLI)

O comando CLI a seguir cria um `ConnectorDefinition` com uma versão inicial que contém o conector CloudWatch Metrics.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MyCloudWatchMetricsConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/CloudWatchMetrics/
versions/4",
            "Parameters": {
                "PublishInterval" : "600",
                "PublishRegion" : "us-west-2",
                "MemorySize" : "16",
                "MaxMetricsToRetain" : "2500",
                "IsolationMode" : "GreengrassContainer"
            }
        }
    ]
}'
```

No AWS IoT Greengrass console, você pode adicionar um conector na página Conectores do grupo. Para obter mais informações, consulte the section called “Conceitos básicos de conectores (console)”.

Dados de entrada

Esse conector aceita métricas em um tópico do MQTT e publica as métricas em. CloudWatch As mensagens de entrada devem estar no formato JSON.

Filtro de tópico na assinatura

`cloudwatch/metric/put`

Propriedades de mensagens

`request`

Informações sobre a métrica nesta mensagem.

O objeto de solicitação contém os dados de métrica para publicação no CloudWatch. Os valores de métrica devem atender às especificações de API `PutMetricData`. Apenas o `namespace`, `metricData.metricName` e as propriedades `metricData.value` são necessários.

Obrigatório: `true`

Tipo: `object` que inclui as seguintes propriedades:

`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/`.

Obrigatório: `true`

Digite: `string`

Padrão válido: `[^:].*`

`metricData`

Os dados para a métrica.

Obrigatório: `true`

Tipo: `object` que inclui as seguintes propriedades:

`metricName`

O nome da métrica.

Obrigatório: `true`

Digite: `string`

`dimensions`

As dimensões associadas com a métrica. As dimensões fornecem mais informações sobre a métrica e seus dados. Uma métrica pode definir até 10 dimensões.

Esse conector inclui, automaticamente, uma dimensão chamada `coreName`, em que o valor é o nome do núcleo.

Obrigatório: `false`

Tipo: `array` de objetos de dimensão que incluem as seguintes propriedades:

`name`

O nome da dimensão.

Obrigatório: `false`

Digite: `string`

`value`

O valor da dimensão.

Obrigatório: `false`

Digite: `string`

`timestamp`

O horário em que os dados da métrica foram recebidos, expresso em segundos desde `Jan 1, 1970 00:00:00 UTC`. Se esse valor for omitido, o conector usará o horário em que ele recebeu a mensagem.

Obrigatório: `false`

Digite: `timestamp`

> **Note**
>
> Se você usar entre as versões 1 e 4 desse conector, 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.

`value`

O valor para a métrica.

> **Note**
>
> CloudWatch rejeita valores muito pequenos ou muito grandes. Os valores devem estar no intervalo de `8.515920e-109` a `1.174271e+108` (Base 10) ou `2e-360` a `2e360` (Base 2). Os valores especiais (por exemplo, `NaN`, `+Infinity`, `-Infinity`) não são compatíveis.

Obrigatório: `true`

Digite: `double`

`unit`

A unidade da métrica.

Obrigatório: `false`

Digite: `string`

Valores válidos: `Seconds, Microseconds, Milliseconds, Bytes, Kilobytes, Megabytes, Gigabytes, Terabytes, Bits, Kilobits, Megabits, Gigabits, Terabits, Percent, Count, Bytes/Second, Kilobytes/Second, Megabytes/Second, Gigabytes/Second, Terabytes/`

```
Second, Bits/Second, Kilobits/Second, Megabits/Second, Gigabits/
Second, Terabits/Second, Count/Second, None
```

Limites

Todos os limites impostos pela CloudWatch `PutMetricData`API se aplicam às métricas ao usar esse conector. É especialmente importante observar os limites a seguir:

- Limite de 40 KB na carga da API
- 20 métricas por solicitação de API
- 150 transações por segundo (TPS) para a API `PutMetricData`

Para obter mais informações, consulte CloudWatch os limites no Guia CloudWatch do usuário da Amazon.

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

O conector publica informações de status como dados de saída em um tópico MQTT.

Filtro de tópico na assinatura

```
cloudwatch/metric/put/status
```

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"
    }
}
```

Exemplo de resultado: falha

```
{
   "response" : {
        "namespace": "Greengrass",
        "error": "InvalidInputException",
        "error_message":"cw metric is invalid",
        "status":"fail"
   }
}
```

> **Note**
>
> Se o conector detectar um erro que pode ser repetido (por exemplo, erros de conexão), ele tentará publicar novamente no próximo lote.

Exemplo de uso

Use as seguintes etapas de alto nível para configurar um exemplo de função do Lambda Python 3.7 que pode ser usado para testar o conector.

> **Note**
>
> - Se você usar outros tempos de execução do Python, poderá criar um link simbólico para Python3.x o Python 3.7.

- Os tópicos Conceitos básicos de conectores (console) e Conceitos básicos de conectores (CLI) contêm etapas detalhadas que mostram como configurar e implantar um exemplo do conector de notificações do Twilio.

1. Certifique-se de cumprir os requisitos para o conector.

   Para o requisito de função de grupo, você deve configurar a função para conceder as permissões necessárias e certificar-se de que a função tenha sido adicionada ao grupo. Para acessar mais informações, consulte the section called “Gerenciar a função de grupo (console)” ou the section called “Gerenciar a função de grupo (CLI)”.
2. Crie e publique uma função do Lambda que envie dados de entrada para o conector.

   Salve o código de exemplo como arquivo PY. Baixe e descompacte o SDK do AWS IoT Greengrass Core para Python. Crie então um pacote zip que contenha o arquivo PY e a pasta `greengrasssdk` no nível raiz. Este pacote zip é o pacote de implantação que você transfere por upload para o AWS Lambda.

   Depois de criar a função do Lambda Python 3.7, publique uma versão de função e crie um alias.
3. Configure o grupo do Greengrass.

   a. Adicione a função do Lambda pelo seu alias (recomendado). Configure o ciclo de vida do Lambda como de longa duração (ou `"Pinned": true` na CLI).

   b. Adicione o conector e configure seus parâmetros.

   c. Adicione assinaturas que permitam que o conector receba dados de entrada e envie dados de saída em filtros de tópico compatíveis.

      - Defina a função do Lambda como origem, o conector como destino e use um filtro de tópico de entrada compatível.
      - Defina o conector como origem, o AWS IoT Core como destino, e use um filtro de tópico de saída compatível. Você usa essa assinatura para visualizar mensagens de status no AWS IoT console.
4. Implante o grupo.
5. No AWS IoT console, na página Teste, inscreva-se no tópico de dados de saída para ver as mensagens de status do conector. A função de exemplo do Lambda é de longa duração e começa a enviar mensagens imediatamente após o grupo ser implantado.

Ao finalizar o teste, você pode definir o ciclo de vida do Lambda como sob demanda (ou `"Pinned": false` na CLI) e implantar o grupo. Isso impede o envio de mensagens pela função.

Exemplo

O exemplo a seguir da função do Lambda envia uma mensagem de entrada para o conector.

```
import greengrasssdk
import time
import json

iot_client = greengrasssdk.client('iot-data')
send_topic = 'cloudwatch/metric/put'

def create_request_with_all_fields():
    return  {
        "request": {
            "namespace": "Greengrass_CW_Connector",
            "metricData": {
                "metricName": "Count1",
                "dimensions": [
                    {
                        "name": "test",
                        "value": "test"
                    }
                ],
                "value": 1,
                "unit": "Seconds",
                "timestamp": time.time()
            }
        }
    }

def publish_basic_message():
    messageToPublish = create_request_with_all_fields()
    print("Message To Publish: ", messageToPublish)
    iot_client.publish(topic=send_topic,
        payload=json.dumps(messageToPublish))

publish_basic_message()
```

```
def lambda_handler(event, context):
    return
```

Licenças

O conector CloudWatch Metrics inclui os seguintes terceiros software/licensing:

- AWS SDK para 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 conector é liberado de acordo com o Contrato de licença de software do Greengrass Core.

Changelog

A tabela a seguir descreve as alterações em cada versão do conector.

Versão	Alterações
5	Correção para adicionar suporte para timestamps duplicados nos dados de entrada.
4	Adicionado o parâmetro `IsolationMode` para configurar o modo de conteinerização para o conector.
3	Atualização do runtime do Lambda para Python 3.7, o que altera o requisito de runtime.
2	Corrija para reduzir o registro excessivo.
1	Versão inicial.

Um grupo do Greengrass só pode conter uma versão do conector por vez. Para obter informações sobre como fazer upgrade de uma versão do conector, consulte the section called “Atualizar a versões do conector”.

Consulte também

- Integrar a serviços e protocolos usando conectores
- the section called “Conceitos básicos de conectores (console)”
- the section called “Conceitos básicos de conectores (CLI)”
- Usando CloudWatch métricas da Amazon no Guia do CloudWatch usuário da Amazon
- PutMetricDatana Amazon CloudWatch API Reference

Conector Device Defender

O conector Device Defender notifica os administradores sobre alterações no estado de um dispositivo de núcleo do Greengrass. Isso pode ajudar a identificar comportamento incomum e, assim, indicar um dispositivo comprometido.

Esse conector lê as métricas do sistema do `/proc` diretório no dispositivo principal e, em seguida, publica as métricas no. AWS IoT Device Defender Para obter detalhes de relatórios de métricas, consulte Especificação da documentação de métricas do dispositivo no Guia do desenvolvedor do AWS IoT .

Esse conector tem as seguintes versões.

Versão	ARN
3	`arn:aws:greengrass:  region::/connectors/DeviceDefender/versions/3`
2	`arn:aws:greengrass:  region::/connectors/DeviceDefender/versions/2`

Versão	ARN
1	`arn:aws:greengrass:  region::/connectors/DeviceDefender/versions/1`

Para obter informações sobre alterações de versão, consulte o Changelog.

Requisitos

Esse conector tem os seguintes requisitos:

Version 3

- AWS IoT Greengrass Software principal v1.9.3 ou posterior.
- Python, versão 3.7 ou 3.8, instalado no dispositivo de núcleo e adicionado à variável de ambiente PATH.

  > **Note**
  >
  > Para usar o Python 3.8, execute o comando a seguir para criar um symblink da pasta de instalação padrão do Python 3.7 para os binários instalados do Python 3.8.
  >
  > ```
  > sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
  > ```
  >
  > Isso configura seu dispositivo para atender ao requisito Python para AWS IoT Greengrass.

- AWS IoT Device Defender configurado para usar o recurso Detectar para acompanhar as violações. Para obter mais informações, consulte Detectar no Guia do desenvolvedor do AWS IoT .
- Um recurso de volume local no grupo do Greengrass que aponta para o diretório `/proc`. O recurso deverá usar as seguintes propriedades:
  - Caminho de origem: `/proc`
  - Caminho de destino: `/host_proc` (ou um valor que corresponda ao padrão válido)
  - AutoAddGroupOwner: `true`

- A biblioteca psutil instalada no núcleo do Greengrass. A versão 5.7.0 é a versão mais recente verificada para trabalhar com o conector.
- A biblioteca cbor instalada no núcleo do Greengrass. A versão 1.0.0 é a versão mais recente verificada para trabalhar com o conector.

Versions 1 - 2

- AWS IoT Greengrass Software principal v1.7 ou posterior.
- Python versão 2.7 instalado no dispositivo de núcleo e adicionado à variável de ambiente PATH.
- AWS IoT Device Defender configurado para usar o recurso Detectar para acompanhar as violações. Para obter mais informações, consulte Detectar no Guia do desenvolvedor do AWS IoT .
- Um recurso de volume local no grupo do Greengrass que aponta para o diretório `/proc`. O recurso deverá usar as seguintes propriedades:
  - Caminho de origem: `/proc`
  - Caminho de destino: `/host_proc` (ou um valor que corresponda ao padrão válido)
  - AutoAddGroupOwner: `true`
- A biblioteca psutil instalada no núcleo do Greengrass.
- A biblioteca cbor instalada no núcleo do Greengrass.

Parâmetros do conector

Esse conector oferece os seguintes parâmetros:

`SampleIntervalSeconds`

O número de segundos entre cada ciclo de coleta de métricas e geração de relatórios. O valor mínimo é 300 segundos (5 minutos).

Nome de exibição no AWS IoT console: intervalo de geração de relatórios de métricas

Obrigatório: `true`

Digite: `string`

Padrão válido: `^[0-9]*(?:3[0-9][0-9]|[4-9][0-9]{2}|[1-9][0-9]{3,})$`

`ProcDestinationPath-ResourceId`

O ID do recurso de volume `/proc`.

> **Note**
>
> Esse conector recebe acesso somente leitura ao recurso.

Nome de exibição no AWS IoT console: Recurso para o diretório /proc

Obrigatório: `true`

Digite: `string`

Padrão válido: `[a-zA-Z0-9_-]+`

`ProcDestinationPath`

O caminho de destino do recurso de volume `/proc`.

Nome de exibição no AWS IoT console: caminho de destino do recurso /proc

Obrigatório: `true`

Digite: `string`

Padrão válido: `\/[a-zA-Z0-9_-]+`

Exemplo de criação de conector (AWS CLI)

O seguinte comando da CLI cria um `ConnectorDefinition` com uma versão inicial que contém o conector Device Defender.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MyDeviceDefenderConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/DeviceDefender/
versions/3",
            "Parameters": {
                "SampleIntervalSeconds": "600",
                "ProcDestinationPath": "/host_proc",
```

```
                "ProcDestinationPath-ResourceId": "my-proc-resource"
            }
        }
    ]
}'
```

> **Note**
>
> A função do Lambda nesse conector tem um ciclo de vida longo.

No AWS IoT Greengrass console, você pode adicionar um conector na página Conectores do grupo. Para obter mais informações, consulte the section called “Conceitos básicos de conectores (console)”.

Dados de entrada

Esse conector não aceita mensagens MQTT como dados de entrada.

Dados de saída

Esse conector publica métricas de segurança AWS IoT Device Defender como dados de saída.

Filtro de tópico na assinatura

`$aws/things/+/defender/metrics/json`

> **Note**
>
> Essa é a sintaxe do tópico que AWS IoT Device Defender espera. O conector substitui o curinga + pelo nome do dispositivo (por exemplo, `$aws/things/thing-name/defender/metrics/json`).

Exemplo de saída

Para obter detalhes de relatórios de métricas, consulte Especificação da documentação de métricas do dispositivo no Guia do desenvolvedor do AWS IoT .

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

Licenças

Esse conector é liberado de acordo com o Contrato de licença de software do Greengrass Core.

Changelog

A tabela a seguir descreve as alterações em cada versão do conector.

Versão	Alterações
3	Atualização do runtime do Lambda para Python 3.7, o que altera o requisito de runtime.
2	Corrija para reduzir o registro excessivo.
1	Versão inicial.

Um grupo do Greengrass só pode conter uma versão do conector por vez. Para obter informações sobre como fazer upgrade de uma versão do conector, consulte the section called “Atualizar a versões do conector”.

Consulte também

- Integrar a serviços e protocolos usando conectores

- the section called “Conceitos básicos de conectores (console)”
- the section called “Conceitos básicos de conectores (CLI)”
- Device Defender no Guia do desenvolvedor AWS IoT

Conector de implantação de aplicativo do Docker

O conector de implantação do aplicativo Greengrass Docker facilita a execução de suas imagens do Docker em um núcleo. AWS IoT Greengrass O conector usa o Compose do Docker para iniciar um aplicativo Docker de vários contêineres de um arquivo `docker-compose.yml`. Especificamente, o conector executa comandos `docker-compose` para gerenciar contêineres do Docker em um dispositivo de núcleo único. Para obter mais informações, consulte Visão geral do Compose do Docker na documentação do Docker. O conector pode acessar imagens do Docker armazenadas em registros de contêiner do Docker, como o Amazon Elastic Container Registry (Amazon ECR), Hub do Docker e registros confiáveis privados do Docker.

Depois de implantar o grupo do Greengrass, o conector extrai as imagens mais recentes e inicia os contêineres do Docker. Ele executa o `docker-compose pull` e o comando `docker-compose up`. Depois, o conector publica o status do comando no tópico MQTT de saída. Ele também registra informações de status sobre a execução de contêineres do Docker. Isso possibilita que você monitore seus registros de aplicativos na Amazon CloudWatch. Para obter mais informações, consulte the section called “Monitoramento com logs do AWS IoT Greengrass ”. O conector também inicia contêineres do Docker sempre que o daemon do Greengrass é reiniciado. O número de contêineres do Docker que podem ser executados no núcleo depende do seu hardware.

Os contêineres do Docker são executados fora do domínio Greengrass no dispositivo de núcleo, portanto não podem acessar a IPC (Comunicação entre processos) do núcleo. No entanto, você pode configurar alguns canais de comunicação com componentes do Greengrass, como funções locais do Lambda. Para obter mais informações, consulte the section called “Comunicar-se com os contêineres do Docker”.

Você pode usar o conector para cenários como hospedagem de um servidor web ou servidor MySQL em seu dispositivo de núcleo. Os serviços locais em seus aplicativos Docker podem se comunicar entre si, com outros processos no ambiente local e com serviços em nuvem. Por exemplo, você pode executar um servidor Web no núcleo que envia solicitações de funções do Lambda para um serviço da Web na nuvem.

Este conector é executado no modo de isolamento Sem contêiner para que você possa implantá-lo em um grupo do Greengrass que é executado sem a conteinerização do Greengrass.

Esse conector tem as seguintes versões.

Versão	ARN
7	`arn:aws:greengrass:region::/connectors/DockerApplicationDeployment/versions/7`
6	`arn:aws:greengrass:region::/connectors/DockerApplicationDeployment/versions/6`
5	`arn:aws:greengrass:region::/connectors/DockerApplicationDeployment/versions/5`
4	`arn:aws:greengrass:region::/connectors/DockerApplicationDeployment/versions/4`
3	`arn:aws:greengrass:region::/connectors/DockerApplicationDeployment/versions/3`
2	`arn:aws:greengrass:region::/connectors/DockerApplicationDeployment/versions/2`
1	`arn:aws:greengrass:region::/connectors/DockerApplicationDeployment/versions/1`

Para obter informações sobre alterações de versão, consulte o Changelog.

Requisitos

Esse conector tem os seguintes requisitos:

- AWS IoT Greengrass Software principal v1.10 ou posterior.

> **Note**
>
> Esse conector não é suportado em OpenWrt distribuições.

- Python, versão 3.7 ou 3.8, instalado no dispositivo de núcleo e adicionado à variável de ambiente PATH.

  > **Note**
  >
  > Para usar o Python 3.8, execute o comando a seguir para criar um symblink da pasta de instalação padrão do Python 3.7 para os binários instalados do Python 3.8.
  >
  > ```
  > sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
  > ```
  >
  > Isso configura seu dispositivo para atender ao requisito Python para AWS IoT Greengrass.

- O mínimo de 36 MB de RAM no núcleo do Greengrass para que o conector monitore contêineres do Docker em execução. O requisito total de memória depende do número de contêineres do Docker executados no núcleo.
- Docker Engine 1.9.1 ou versão posterior instalado no núcleo do Greengrass. A versão 19.0.3 é a versão mais recente verificada para trabalhar com o conector.

  O executável `docker` deve estar no diretório `/usr/bin` ou `/usr/local/bin`.

  > **Important**
  >
  > Recomendamos que você instale um armazenamento de credenciais para proteger as cópias locais das suas credenciais do Docker. Para obter mais informações, consulte the section called “Observações de segurança”.

  Para obter informações sobre como instalar o Docker nas distribuições do Amazon Linux, consulte Noções básicas do Docker para o Amazon ECS no Guia do desenvolvedor do Elastic Container da Amazon Service.
- Compose do Docker instalado no núcleo do Greengrass. O executável `docker-compose` deve estar no diretório `/usr/bin` ou `/usr/local/bin`.

As versões a seguir do Docker Compose são verificadas para trabalhar com o conector.

Versão do conector	Versão verificada do Docker Compose
7	1.25.4
6	1.25.4
5	1.25.4
4	1.25.4
3	1.25.4
2	1.25.1
1	1.24.1

- Um único arquivo do Docker Compose (por exemplo, `docker-compose.yml`), armazenado no Amazon Simple Storage Service (Amazon S3). O formato deve ser compatível com a versão do Compose do Docker instalada no núcleo. Você deve testar o arquivo antes de usá-lo no núcleo. Se você editar o arquivo depois de implantar o grupo do Greengrass, será necessário reimplantar o grupo para atualizar sua cópia local no núcleo.
- Um usuário Linux com permissão para chamar o daemon do Docker local e gravar no diretório que armazena a cópia local do arquivo Compose. Para obter mais informações, consulte Configurar o usuário do Docker no núcleo.
- A função de grupo do Greengrass configurada para permitir a ação `s3:GetObject` no bucket do S3 que contém o arquivo Compose. Essa permissão é exibida no exemplo de política do IAM a seguir.

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowAccessToComposeFileS3Bucket",
            "Action": [
                "s3:GetObject",
```

```
                "s3:GetObjectVersion"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:s3:::bucket-name/*"
        }
    ]
}
```

> **Note**
>
> Se o bucket do S3 estiver habilitado para versionamento, a função deve ser configurada para permitir a ação `s3:GetObjectVersion` também. Para obter mais informações, consulte Usando versionamento no Guia do usuário do Amazon Simple Storage Service.

Para o requisito de função de grupo, você deve configurar a função para conceder as permissões necessárias e certificar-se de que a função tenha sido adicionada ao grupo. Para acessar mais informações, consulte the section called “Gerenciar a função de grupo (console)” ou the section called “Gerenciar a função de grupo (CLI)”.

- Se o arquivo Docker Compose fizer referência a uma imagem do Docker armazenada no Amazon ECR, a função de grupo do Greengrass deverá estar configurada para permitir o seguinte:
  - As ações `ecr:GetDownloadUrlForLayer` e `ecr:BatchGetImage` em seus repositórios do Amazon ECR que contêm as imagens do Docker.
  - A ação `ecr:GetAuthorizationToken` em seus recursos.

  Os repositórios devem estar no mesmo Conta da AWS e Região da AWS no conector.

> **Important**
>
> As permissões na função de grupo podem ser assumidas por todos os conectores e funções do Lambda no grupo do Greengrass. Para obter mais informações, consulte the section called “Observações de segurança”.

Essas permissões são exibidas na política de exemplo a seguir.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "AllowGetEcrRepositories",
            "Effect": "Allow",
            "Action": [
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage"
            ],
            "Resource": [
                "arn:aws:ecr:us-east-1:123456789012:repository/repository-name"
            ]
        },
        {
            "Sid": "AllowGetEcrAuthToken",
            "Effect": "Allow",
            "Action": "ecr:GetAuthorizationToken",
            "Resource": "*"
        }
    ]
}
```

Para obter mais informações, consulte Exemplos de políticas do repositório do Amazon ECR no Guia do usuário do Amazon ECR.

Para o requisito de função de grupo, você deve configurar a função para conceder as permissões necessárias e certificar-se de que a função tenha sido adicionada ao grupo. Para acessar mais informações, consulte the section called “Gerenciar a função de grupo (console)” ou the section called “Gerenciar a função de grupo (CLI)”.

- Se o arquivo Compose do Docker fizer referência a uma imagem do Docker do AWS Marketplace, o conector também terá os seguintes requisitos:
  - Você deve estar inscrito em produtos de AWS Marketplace contêiner. Para obter mais informações, consulte Encontrar e assinar produtos de contêiner no Guia de assinantes do AWS Marketplace .

  - AWS IoT Greengrass devem ser configurados para oferecer suporte a segredos locais, conforme descrito em Requisitos de segredos. O conector usa esse recurso apenas para recuperar seus segredos AWS Secrets Manager, não para armazená-los.
  - Você deve criar um segredo no Secrets Manager para cada AWS Marketplace registro que armazena uma imagem do Docker referenciada em seu arquivo Compose. Para obter mais informações, consulte the section called “Acessar imagens do Docker de repositórios privados”.
- Se o arquivo Compose do Docker fizer referência a uma imagem do Docker de repositórios privados em registros que não sejam o Amazon ECR, como o Hub do Docker, o conector também terá os seguintes requisitos:
  - AWS IoT Greengrass devem ser configurados para oferecer suporte a segredos locais, conforme descrito em Requisitos de segredos. O conector usa esse recurso apenas para recuperar seus segredos AWS Secrets Manager, não para armazená-los.
  - Você deve criar um segredo no Secrets Manager para cada repositório privado que armazena uma imagem do Docker referenciada no arquivo Compose. Para obter mais informações, consulte the section called “Acessar imagens do Docker de repositórios privados”.
- O daemon do Docker deve estar em execução quando você implantar um grupo do Greengrass que contenha esse conector.

Acessar imagens do Docker de repositórios privados

Se você usar credenciais para acessar suas imagens do Docker, será necessário permitir que o conector as acesse. A maneira de fazer isso dependerá de onde a imagem do Docker estiver localizada.

Para imagens do Docker armazenadas no Amazon ECR, você concede permissão para obter seu token de autorização na função de grupo do Greengrass. Para obter mais informações, consulte the section called “Requisitos”.

Para imagens do Docker armazenadas em outros repositórios ou registros privados, você deve criar um segredo AWS Secrets Manager para armazenar suas informações de login. Isso inclui imagens do Docker nas quais você se inscreveu. AWS Marketplace Crie um segredo para cada repositório. Ao atualizar seus segredos no Secrets Manager, as alterações serão propagadas para o núcleo da próxima vez que você implantar o grupo.

> **Note**
>
> O Secrets Manager é um serviço que você pode usar para armazenar e gerenciar com segurança suas credenciais, chaves e outros segredos na Nuvem AWS. Para obter mais informações, consulte O que é AWS Secrets Manager? no Guia do AWS Secrets Manager usuário.

Cada segredo deve conter as seguintes chaves:

Chave	Valor
`username`	O nome de usuário usado para acessar o repositório ou registro.
`password`	A senha usada para acessar o repositório ou registro.
`registryUrl`	O endpoint do registro. Isso deve corresponder ao URL de registro correspondente no arquivo Compose.

> **Note**
>
> AWS IoT Greengrass Para permitir o acesso a um segredo por padrão, o nome do segredo deve começar com greengrass-. Caso contrário, sua função de serviço do Greengrass deverá conceder acesso. Para obter mais informações, consulte the section called “Permitir AWS IoT Greengrass para obter valores secretos”.

Para obter informações de login para imagens do Docker no AWS Marketplace

1. Obtenha sua senha para imagens do Docker AWS Marketplace usando o `aws ecr get-login-password` comando. Para obter mais informações, consulte get-login-password na Referência de comandos da AWS CLI .

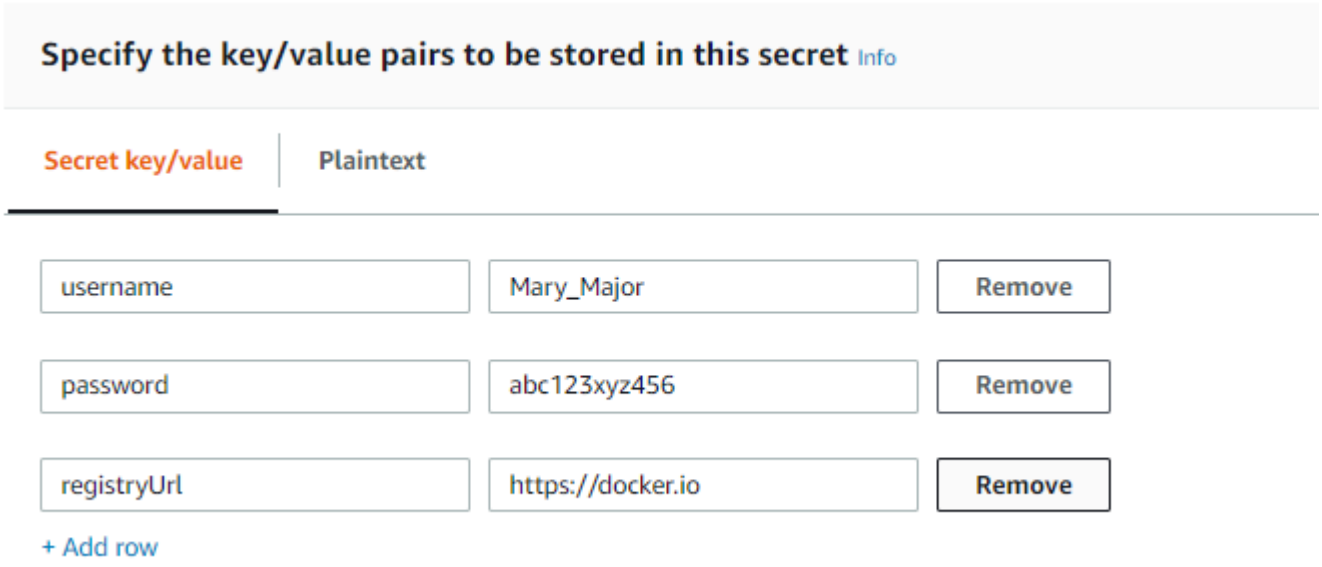
   ```
   aws ecr get-login-password
   ```

2. Recupere o URL do registro para a imagem do Docker. Abra o site do AWS Marketplace e abra a página de lançamento do produto em contêiner. Em Imagens do contêiner, selecione Exibir detalhes da imagem do contêiner para localizar o nome do usuário e o URL do registro.

Use o nome de usuário, a senha e o URL do registro recuperados para criar um segredo para cada AWS Marketplace registro que armazena imagens do Docker referenciadas em seu arquivo Compose.

Como criar segredos (console)

No AWS Secrets Manager console, escolha Outro tipo de segredos. Em Specify the key/value pairs to be stored for this secret (Especificar os pares de chave/valor a serem armazenados para este segredo), adicione as linhas para `username`, `password` e `registryUrl`: Para obter mais informações, consulte Criar um segredo básico no Guia do usuário do AWS Secrets Manager .



Como criar segredos (CLI)

No AWS CLI, use o `create-secret` comando Secrets Manager, conforme mostrado no exemplo a seguir. Para obter mais informações, consulte create-secret na AWS CLI Command Reference.

```
aws secretsmanager create-secret --name greengrass-MySecret --secret-string
 [{"username":"Mary_Major"},{"password":"abc123xyz456"},{"registryUrl":"https://
docker.io"}]
```

> **Important**
>
> Você é responsável por proteger o diretório `DockerComposeFileDestinationPath` que armazena seu arquivo Compose do Docker e as credenciais para suas imagens do Docker de repositórios privados. Para obter mais informações, consulte the section called "Observações de segurança".

Parâmetros

Esse conector oferece os seguintes parâmetros:

Version 7

`DockerComposeFileS3Bucket`

O nome do bucket do S3 que contém o arquivo Compose do Docker. Ao criar o bucket, certifique-se de seguir as regras para nomes de bucket descritas no Guia do usuário do Amazon Simple Storage Service.

Nome de exibição no AWS IoT console: arquivo Docker Compose no S3

> **Note**
>
> No console, a propriedade Docker Compose file in S3 (Arquivo Compose do Docker no S3) combina os parâmetros `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` e `DockerComposeFileS3Version`.

Obrigatório: `true`

Digite: `string`

Padrão válido `[a-zA-Z0-9\\-\\.]{3,63}`

`DockerComposeFileS3Key`

A chave de objeto do arquivo do Docker Compose no Amazon S3. Para obter mais informações, incluindo diretrizes de nomenclatura das chaves de objeto, consulte Metadados e chaves de objetos no Guia do usuário do Amazon Simple Storage Service.

> **Note**
>
> No console, a propriedade Docker Compose file in S3 (Arquivo Compose do Docker no S3) combina os parâmetros `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` e `DockerComposeFileS3Version`.

Obrigatório: `true`

Digite: `string`

Padrão válido `.+`

`DockerComposeFileS3Version`

A versão de objeto do arquivo do Docker Compose no Amazon S3. Para obter mais informações, incluindo diretrizes de nomenclatura das chaves de objeto, consulte Uso do versionamento no Guia do usuário do Amazon Simple Storage Service.

> **Note**
>
> No console, a propriedade Docker Compose file in S3 (Arquivo Compose do Docker no S3) combina os parâmetros `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` e `DockerComposeFileS3Version`.

Obrigatório: `false`

Digite: `string`

Padrão válido `.+`

`DockerComposeFileDestinationPath`

O caminho absoluto do diretório local usado para armazenar uma cópia do arquivo Compose do Docker. Esse deve ser um diretório existente. O usuário especificado para `DockerUserId` deve ter permissão para criar um arquivo neste diretório. Para obter mais informações, consulte the section called “Configurar o usuário do Docker no núcleo”.

> **Important**
>
> Este diretório armazena seu arquivo Compose do Docker e as credenciais para suas imagens do Docker de repositórios privados. Você é responsável por proteger este diretório. Para obter mais informações, consulte the section called “Observações de segurança”.

Nome de exibição no AWS IoT console: caminho do diretório para o arquivo Compose local

Obrigatório: `true`

Digite: `string`

Padrão válido `\/.*\/?`

Exemplo: `/home/username/myCompose`

`DockerUserId`

O UID do usuário Linux com o qual o conector é executado. Esse usuário deve pertencer ao grupo Linux `docker` no dispositivo de núcleo e ter permissões de gravação no diretório `DockerComposeFileDestinationPath`. Para obter mais informações, consulte Configurar o usuário do Docker no núcleo.

> **Note**
>
> Convém evitar a execução como raiz, a menos que isso seja absolutamente necessário. Se você especificar o usuário raiz, é necessário permitir que as funções do Lambda sejam executadas como raiz no núcleo AWS IoT Greengrass . Para obter mais informações, consulte the section called “Executar uma função do Lambda como raiz”.

Nome de exibição no AWS IoT console: ID de usuário do Docker

Obrigatório: `false`

Digite: `string`

Padrão válido: `^[0-9]{1,5}$`

`AWSSecretsArnList`

Os nomes de recursos da Amazon (ARNs) dos segredos AWS Secrets Manager que contêm as informações de login usadas para acessar suas imagens do Docker em repositórios privados. Para obter mais informações, consulte the section called “Acessar imagens do Docker de repositórios privados”.

Nome de exibição no AWS IoT console: Credenciais para repositórios privados

Obrigatório:`false`. Esse parâmetro é necessário para acessar imagens do Docker armazenadas em repositórios privados.

Tipo: `array` de `string`

Padrão válido: `[( ?,? ?"(arn:(aws(-[a-z]+)):secretsmanager:[a-z0-9-]+:[0-9]{12}:secret:([a-zA-Z0-9\]+/)[a-zA-Z0-9/_+=,.@-]+-[a-zA-Z0-9]+)")]`

`DockerContainerStatusLogFrequency`

A frequência (em segundos) na qual o conector registra informações de status sobre os contêineres do Docker em execução no núcleo. O padrão é 300 segundos (5 minutos).

Nome de exibição no AWS IoT console: Frequência de registro

Obrigatório: `false`

Digite: `string`

Padrão válido: `^[1-9]{1}[0-9]{0,3}$`

`ForceDeploy`

Indica se é preciso forçar a implantação do Docker em caso de falha devido à limpeza inadequada da última implantação. O valor padrão é `False`.

Nome de exibição no AWS IoT console: Forçar implantação

Obrigatório: `false`

Digite: `string`

Padrão válido: `^(true|false)$`

`DockerPullBeforeUp`

Indica se o implantador deve ser executado `docker-compose pull` antes de executar o `docker-compose up` para um comportamento pull-down-up. O valor padrão é `True`.

Nome de exibição no AWS IoT console: Docker Pull Before Up

Obrigatório: `false`

Digite: `string`

Padrão válido: `^(true|false)$`

`StopContainersOnNewDeployment`

Indica se o conector deve interromper os contêineres docker gerenciados pelo Docker Deployer quando o GGC é interrompido (o GGC para quando um novo grupo é implantado ou quando o kernel é desligado). O valor padrão é `True`.

Nome de exibição no AWS IoT console: parada do Docker na nova implantação

> **Note**
>
> Recomendamos manter esse parâmetro definido com seu valor `True`padrão. O parâmetro to `False` faz com que seu contêiner Docker continue em execução mesmo depois de encerrar o AWS IoT Greengrass núcleo ou iniciar uma nova implantação. Se você definir esse parâmetro como `False`, você deverá garantir que seus contêineres do Docker sejam mantidos conforme necessário no caso de uma alteração ou adição do nome do serviço `docker-compose`.
> Para obter mais informações, consulte a documentação dos arquivos do `docker-compose` Compose.

Obrigatório: `false`

Digite: `string`

Padrão válido: `^(true|false)$`

`DockerOfflineMode`

Indica se é preciso usar o arquivo do Docker Compose existente quando o AWS IoT Greengrass iniciar offline. O valor padrão é `False`.

Obrigatório: `false`

Digite: `string`

Padrão válido: `^(true|false)$`

Version 6

`DockerComposeFileS3Bucket`

O nome do bucket do S3 que contém o arquivo Compose do Docker. Ao criar o bucket, certifique-se de seguir as regras para nomes de bucket descritas no Guia do usuário do Amazon Simple Storage Service.

Nome de exibição no AWS IoT console: arquivo Docker Compose no S3

> **Note**
>
> No console, a propriedade Docker Compose file in S3 (Arquivo Compose do Docker no S3) combina os parâmetros `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` e `DockerComposeFileS3Version`.

Obrigatório: `true`

Digite: `string`

Padrão válido `[a-zA-Z0-9\\-\\.]{3,63}`

`DockerComposeFileS3Key`

A chave de objeto do arquivo do Docker Compose no Amazon S3. Para obter mais informações, incluindo diretrizes de nomenclatura das chaves de objeto, consulte Metadados e chaves de objetos no Guia do usuário do Amazon Simple Storage Service.

> **Note**
>
> No console, a propriedade Docker Compose file in S3 (Arquivo Compose do Docker no S3) combina os parâmetros `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` e `DockerComposeFileS3Version`.

Obrigatório: `true`

Digite: `string`

Padrão válido `.+`

`DockerComposeFileS3Version`

A versão de objeto do arquivo do Docker Compose no Amazon S3. Para obter mais informações, incluindo diretrizes de nomenclatura das chaves de objeto, consulte Uso do versionamento no Guia do usuário do Amazon Simple Storage Service.

> **Note**
>
> No console, a propriedade Docker Compose file in S3 (Arquivo Compose do Docker no S3) combina os parâmetros `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` e `DockerComposeFileS3Version`.

Obrigatório: `false`

Digite: `string`

Padrão válido `.+`

`DockerComposeFileDestinationPath`

O caminho absoluto do diretório local usado para armazenar uma cópia do arquivo Compose do Docker. Esse deve ser um diretório existente. O usuário especificado para `DockerUserId` deve ter permissão para criar um arquivo neste diretório. Para obter mais informações, consulte the section called “Configurar o usuário do Docker no núcleo”.

> **Important**
>
> Este diretório armazena seu arquivo Compose do Docker e as credenciais para suas imagens do Docker de repositórios privados. Você é responsável por proteger este diretório. Para obter mais informações, consulte the section called “Observações de segurança”.

Nome de exibição no AWS IoT console: caminho do diretório para o arquivo Compose local

Obrigatório: `true`

Digite: `string`

Padrão válido `\/.*\/?`

Exemplo: `/home/username/myCompose`

`DockerUserId`

O UID do usuário Linux com o qual o conector é executado. Esse usuário deve pertencer ao grupo Linux `docker` no dispositivo de núcleo e ter permissões de gravação no diretório `DockerComposeFileDestinationPath`. Para obter mais informações, consulte Configurar o usuário do Docker no núcleo.

> **Note**
>
> Convém evitar a execução como raiz, a menos que isso seja absolutamente necessário. Se você especificar o usuário raiz, é necessário permitir que as funções do Lambda sejam executadas como raiz no núcleo AWS IoT Greengrass . Para obter mais informações, consulte the section called “Executar uma função do Lambda como raiz”.

Nome de exibição no AWS IoT console: ID de usuário do Docker

Obrigatório: `false`

Digite: `string`

Padrão válido: `^[0-9]{1,5}$`

`AWSSecretsArnList`

Os nomes de recursos da Amazon (ARNs) dos segredos AWS Secrets Manager que contêm as informações de login usadas para acessar suas imagens do Docker em repositórios privados. Para obter mais informações, consulte the section called “Acessar imagens do Docker de repositórios privados”.

Nome de exibição no AWS IoT console: Credenciais para repositórios privados

Obrigatório:`false`. Esse parâmetro é necessário para acessar imagens do Docker armazenadas em repositórios privados.

Tipo: `array` de `string`

Padrão válido: `[( ?,? ?"(arn:(aws(-[a-z]+)):secretsmanager:[a-z0-9-]+:[0-9]{12}:secret:([a-zA-Z0-9\]+/)[a-zA-Z0-9/_+=,.@-]+-[a-zA-Z0-9]+)")]`

`DockerContainerStatusLogFrequency`

A frequência (em segundos) na qual o conector registra informações de status sobre os contêineres do Docker em execução no núcleo. O padrão é 300 segundos (5 minutos).

Nome de exibição no AWS IoT console: Frequência de registro

Obrigatório: `false`

Digite: `string`

Padrão válido: `^[1-9]{1}[0-9]{0,3}$`

`ForceDeploy`

Indica se é preciso forçar a implantação do Docker em caso de falha devido à limpeza inadequada da última implantação. O valor padrão é `False`.

Nome de exibição no AWS IoT console: Forçar implantação

Obrigatório: `false`

Digite: `string`

Padrão válido: `^(true|false)$`

`DockerPullBeforeUp`

Indica se o implantador deve ser executado `docker-compose pull` antes de executar o `docker-compose up` para um comportamento pull-down-up. O valor padrão é `True`.

Nome de exibição no AWS IoT console: Docker Pull Before Up

Obrigatório: `false`

Digite: `string`

Padrão válido: `^(true|false)$`

`StopContainersOnNewDeployment`

Indica se o conector deve interromper os contêineres docker gerenciados pelo Docker Deployer quando o GGC é interrompido (quando uma nova implantação em grupo é feita ou quando o kernel é desligado). O valor padrão é `True`.

Nome de exibição no AWS IoT console: parada do Docker na nova implantação

> **Note**
>
> Recomendamos manter esse parâmetro definido com seu valor `True`padrão. O parâmetro to `False` faz com que seu contêiner Docker continue em execução mesmo depois de encerrar o AWS IoT Greengrass núcleo ou iniciar uma nova implantação. Se você definir esse parâmetro como `False`, você deverá garantir que seus contêineres do Docker sejam mantidos conforme necessário no caso de uma alteração ou adição do nome do serviço `docker-compose`.
> Para obter mais informações, consulte a documentação dos arquivos do `docker-compose` Compose.

Obrigatório: `false`

Digite: `string`

Padrão válido: `^(true|false)$`

Version 5

`DockerComposeFileS3Bucket`

O nome do bucket do S3 que contém o arquivo Compose do Docker. Ao criar o bucket, certifique-se de seguir as regras para nomes de bucket descritas no Guia do usuário do Amazon Simple Storage Service.

Nome de exibição no AWS IoT console: arquivo Docker Compose no S3

> **Note**
>
> No console, a propriedade Docker Compose file in S3 (Arquivo Compose do Docker no S3) combina os parâmetros `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` e `DockerComposeFileS3Version`.

Obrigatório: `true`

Digite: `string`

Padrão válido `[a-zA-Z0-9\\-\\.]{3,63}`

`DockerComposeFileS3Key`

A chave de objeto do arquivo do Docker Compose no Amazon S3. Para obter mais informações, incluindo diretrizes de nomenclatura das chaves de objeto, consulte Metadados e chaves de objetos no Guia do usuário do Amazon Simple Storage Service.

> **Note**
>
> No console, a propriedade Docker Compose file in S3 (Arquivo Compose do Docker no S3) combina os parâmetros `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` e `DockerComposeFileS3Version`.

Obrigatório: `true`

Digite: `string`

Padrão válido `.+`

`DockerComposeFileS3Version`

A versão de objeto do arquivo do Docker Compose no Amazon S3. Para obter mais informações, incluindo diretrizes de nomenclatura das chaves de objeto, consulte Uso do versionamento no Guia do usuário do Amazon Simple Storage Service.

> **Note**
>
> No console, a propriedade Docker Compose file in S3 (Arquivo Compose do Docker no S3) combina os parâmetros `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` e `DockerComposeFileS3Version`.

Obrigatório: `false`

Digite: `string`

Padrão válido `.+`

`DockerComposeFileDestinationPath`

O caminho absoluto do diretório local usado para armazenar uma cópia do arquivo Compose do Docker. Esse deve ser um diretório existente. O usuário especificado para `DockerUserId` deve ter permissão para criar um arquivo neste diretório. Para obter mais informações, consulte the section called “Configurar o usuário do Docker no núcleo”.

> **Important**
>
> Este diretório armazena seu arquivo Compose do Docker e as credenciais para suas imagens do Docker de repositórios privados. Você é responsável por proteger este diretório. Para obter mais informações, consulte the section called “Observações de segurança”.

Nome de exibição no AWS IoT console: caminho do diretório para o arquivo Compose local

Obrigatório: `true`

Digite: `string`

Padrão válido `\/.*\/?`

Exemplo: `/home/username/myCompose`

`DockerUserId`

O UID do usuário Linux com o qual o conector é executado. Esse usuário deve pertencer ao grupo Linux `docker` no dispositivo de núcleo e ter permissões de gravação no diretório

`DockerComposeFileDestinationPath`. Para obter mais informações, consulte Configurar o usuário do Docker no núcleo.

> **Note**
>
> Convém evitar a execução como raiz, a menos que isso seja absolutamente necessário. Se você especificar o usuário raiz, é necessário permitir que as funções do Lambda sejam executadas como raiz no núcleo AWS IoT Greengrass . Para obter mais informações, consulte the section called “Executar uma função do Lambda como raiz”.

Nome de exibição no AWS IoT console: ID de usuário do Docker

Obrigatório: `false`

Digite: `string`

Padrão válido: `^[0-9]{1,5}$`

`AWSSecretsArnList`

Os nomes de recursos da Amazon (ARNs) dos segredos AWS Secrets Manager que contêm as informações de login usadas para acessar suas imagens do Docker em repositórios privados. Para obter mais informações, consulte the section called “Acessar imagens do Docker de repositórios privados”.

Nome de exibição no AWS IoT console: Credenciais para repositórios privados

Obrigatório:`false`. Esse parâmetro é necessário para acessar imagens do Docker armazenadas em repositórios privados.

Tipo: `array` de `string`

Padrão válido: `[( ?,? ?"(arn:(aws(-[a-z]+)):secretsmanager:[a-z0-9-]+:[0-9]{12}:secret:([a-zA-Z0-9\]+/)[a-zA-Z0-9/_+=,.@-]+-[a-zA-Z0-9]+)")]`

`DockerContainerStatusLogFrequency`

A frequência (em segundos) na qual o conector registra informações de status sobre os contêineres do Docker em execução no núcleo. O padrão é 300 segundos (5 minutos).

Nome de exibição no AWS IoT console: Frequência de registro

Obrigatório: `false`

Digite: `string`

Padrão válido: `^[1-9]{1}[0-9]{0,3}$`

`ForceDeploy`

Indica se é preciso forçar a implantação do Docker em caso de falha devido à limpeza inadequada da última implantação. O valor padrão é `False`.

Nome de exibição no AWS IoT console: Forçar implantação

Obrigatório: `false`

Digite: `string`

Padrão válido: `^(true|false)$`

`DockerPullBeforeUp`

Indica se o implantador deve ser executado `docker-compose pull` antes de executar o `docker-compose up` para um comportamento pull-down-up. O valor padrão é `True`.

Nome de exibição no AWS IoT console: Docker Pull Before Up

Obrigatório: `false`

Digite: `string`

Padrão válido: `^(true|false)$`

Versions 2 - 4

`DockerComposeFileS3Bucket`

O nome do bucket do S3 que contém o arquivo Compose do Docker. Ao criar o bucket, certifique-se de seguir as regras para nomes de bucket descritas no Guia do usuário do Amazon Simple Storage Service.

Nome de exibição no AWS IoT console: arquivo Docker Compose no S3

> **Note**
>
> No console, a propriedade Docker Compose file in S3 (Arquivo Compose do Docker no S3) combina os parâmetros `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` e `DockerComposeFileS3Version`.

Obrigatório: `true`

Digite: `string`

Padrão válido `[a-zA-Z0-9\\-\\.]{3,63}`

`DockerComposeFileS3Key`

A chave de objeto do arquivo do Docker Compose no Amazon S3. Para obter mais informações, incluindo diretrizes de nomenclatura das chaves de objeto, consulte Metadados e chaves de objetos no Guia do usuário do Amazon Simple Storage Service.

> **Note**
>
> No console, a propriedade Docker Compose file in S3 (Arquivo Compose do Docker no S3) combina os parâmetros `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` e `DockerComposeFileS3Version`.

Obrigatório: `true`

Digite: `string`

Padrão válido `.+`

`DockerComposeFileS3Version`

A versão de objeto do arquivo do Docker Compose no Amazon S3. Para obter mais informações, incluindo diretrizes de nomenclatura das chaves de objeto, consulte Uso do versionamento no Guia do usuário do Amazon Simple Storage Service.

> **Note**
>
> No console, a propriedade Docker Compose file in S3 (Arquivo Compose do Docker no S3) combina os parâmetros `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` e `DockerComposeFileS3Version`.

Obrigatório: `false`

Digite: `string`

Padrão válido `.+`

`DockerComposeFileDestinationPath`

O caminho absoluto do diretório local usado para armazenar uma cópia do arquivo Compose do Docker. Esse deve ser um diretório existente. O usuário especificado para `DockerUserId` deve ter permissão para criar um arquivo neste diretório. Para obter mais informações, consulte the section called “Configurar o usuário do Docker no núcleo”.

> **Important**
>
> Este diretório armazena seu arquivo Compose do Docker e as credenciais para suas imagens do Docker de repositórios privados. Você é responsável por proteger este diretório. Para obter mais informações, consulte the section called “Observações de segurança”.

Nome de exibição no AWS IoT console: caminho do diretório para o arquivo Compose local

Obrigatório: `true`

Digite: `string`

Padrão válido `\/.*\/?`

Exemplo: `/home/username/myCompose`

`DockerUserId`

O UID do usuário Linux com o qual o conector é executado. Esse usuário deve pertencer ao grupo Linux `docker` no dispositivo de núcleo e ter permissões de gravação no diretório

`DockerComposeFileDestinationPath`. Para obter mais informações, consulte Configurar o usuário do Docker no núcleo.

> **Note**
>
> Convém evitar a execução como raiz, a menos que isso seja absolutamente necessário. Se você especificar o usuário raiz, é necessário permitir que as funções do Lambda sejam executadas como raiz no núcleo AWS IoT Greengrass . Para obter mais informações, consulte the section called “Executar uma função do Lambda como raiz”.

Nome de exibição no AWS IoT console: ID de usuário do Docker

Obrigatório: `false`

Digite: `string`

Padrão válido: `^[0-9]{1,5}$`

`AWSSecretsArnList`

Os nomes de recursos da Amazon (ARNs) dos segredos AWS Secrets Manager que contêm as informações de login usadas para acessar suas imagens do Docker em repositórios privados. Para obter mais informações, consulte the section called “Acessar imagens do Docker de repositórios privados”.

Nome de exibição no AWS IoT console: Credenciais para repositórios privados

Obrigatório:`false`. Esse parâmetro é necessário para acessar imagens do Docker armazenadas em repositórios privados.

Tipo: `array` de `string`

Padrão válido: `[( ?,? ?"(arn:(aws(-[a-z]+)):secretsmanager:[a-z0-9-]+:[0-9]{12}:secret:([a-zA-Z0-9\]+/)[a-zA-Z0-9/_+=,.@-]+-[a-zA-Z0-9]+)")]`

`DockerContainerStatusLogFrequency`

A frequência (em segundos) na qual o conector registra informações de status sobre os contêineres do Docker em execução no núcleo. O padrão é 300 segundos (5 minutos).

Nome de exibição no AWS IoT console: Frequência de registro

Obrigatório: `false`

Digite: `string`

Padrão válido: `^[1-9]{1}[0-9]{0,3}$`

`ForceDeploy`

Indica se é preciso forçar a implantação do Docker em caso de falha devido à limpeza inadequada da última implantação. O valor padrão é `False`.

Nome de exibição no AWS IoT console: Forçar implantação

Obrigatório: `false`

Digite: `string`

Padrão válido: `^(true|false)$`

Version 1

`DockerComposeFileS3Bucket`

O nome do bucket do S3 que contém o arquivo Compose do Docker. Ao criar o bucket, certifique-se de seguir as regras para nomes de bucket descritas no Guia do usuário do Amazon Simple Storage Service.

Nome de exibição no AWS IoT console: arquivo Docker Compose no S3

> **Note**
>
> No console, a propriedade Docker Compose file in S3 (Arquivo Compose do Docker no S3) combina os parâmetros `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` e `DockerComposeFileS3Version`.

Obrigatório: `true`

Digite: `string`

Padrão válido `[a-zA-Z0-9\\-\\.]{3,63}`

`DockerComposeFileS3Key`

A chave de objeto do arquivo do Docker Compose no Amazon S3. Para obter mais informações, incluindo diretrizes de nomenclatura das chaves de objeto, consulte Metadados e chaves de objetos no Guia do usuário do Amazon Simple Storage Service.

> **Note**
>
> No console, a propriedade Docker Compose file in S3 (Arquivo Compose do Docker no S3) combina os parâmetros `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` e `DockerComposeFileS3Version`.

Obrigatório: `true`

Digite: `string`

Padrão válido `.+`

`DockerComposeFileS3Version`

A versão de objeto do arquivo do Docker Compose no Amazon S3. Para obter mais informações, incluindo diretrizes de nomenclatura das chaves de objeto, consulte Uso do versionamento no Guia do usuário do Amazon Simple Storage Service.

> **Note**
>
> No console, a propriedade Docker Compose file in S3 (Arquivo Compose do Docker no S3) combina os parâmetros `DockerComposeFileS3Bucket`, `DockerComposeFileS3Key` e `DockerComposeFileS3Version`.

Obrigatório: `false`

Digite: `string`

Padrão válido `.+`

`DockerComposeFileDestinationPath`

O caminho absoluto do diretório local usado para armazenar uma cópia do arquivo Compose do Docker. Esse deve ser um diretório existente. O usuário especificado para `DockerUserId`

deve ter permissão para criar um arquivo neste diretório. Para obter mais informações, consulte the section called “Configurar o usuário do Docker no núcleo”.

> **Important**
>
> Este diretório armazena seu arquivo Compose do Docker e as credenciais para suas imagens do Docker de repositórios privados. Você é responsável por proteger este diretório. Para obter mais informações, consulte the section called “Observações de segurança”.

Nome de exibição no AWS IoT console: caminho do diretório para o arquivo Compose local

Obrigatório: `true`

Digite: `string`

Padrão válido `\/.*\/?`

Exemplo: `/home/username/myCompose`

`DockerUserId`

O UID do usuário Linux com o qual o conector é executado. Esse usuário deve pertencer ao grupo Linux `docker` no dispositivo de núcleo e ter permissões de gravação no diretório `DockerComposeFileDestinationPath`. Para obter mais informações, consulte Configurar o usuário do Docker no núcleo.

> **Note**
>
> Convém evitar a execução como raiz, a menos que isso seja absolutamente necessário. Se você especificar o usuário raiz, é necessário permitir que as funções do Lambda sejam executadas como raiz no núcleo AWS IoT Greengrass . Para obter mais informações, consulte the section called “Executar uma função do Lambda como raiz”.

Nome de exibição no AWS IoT console: ID de usuário do Docker

Obrigatório: `false`

Digite: `string`

Padrão válido: `^[0-9]{1,5}$`

`AWSSecretsArnList`

Os nomes de recursos da Amazon (ARNs) dos segredos AWS Secrets Manager que contêm as informações de login usadas para acessar suas imagens do Docker em repositórios privados. Para obter mais informações, consulte the section called “Acessar imagens do Docker de repositórios privados”.

Nome de exibição no AWS IoT console: Credenciais para repositórios privados

Obrigatório:`false`. Esse parâmetro é necessário para acessar imagens do Docker armazenadas em repositórios privados.

Tipo: `array` de `string`

Padrão válido: `[( ?,? ?"(arn:(aws(-[a-z]+)):secretsmanager:[a-z0-9-]+:[0-9]{12}:secret:([a-zA-Z0-9\]+/)[a-zA-Z0-9/_+=,.@-]+-[a-zA-Z0-9]+)")]`

`DockerContainerStatusLogFrequency`

A frequência (em segundos) na qual o conector registra informações de status sobre os contêineres do Docker em execução no núcleo. O padrão é 300 segundos (5 minutos).

Nome de exibição no AWS IoT console: Frequência de registro

Obrigatório: `false`

Digite: `string`

Padrão válido: `^[1-9]{1}[0-9]{0,3}$`

Exemplo de criação de conector (AWS CLI)

O seguinte comando da CLI cria um `ConnectorDefinition` com uma versão inicial que contém o conector de implantação do aplicativo do Docker do Greengrass.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
```

```
    "Connectors": [
        {
            "Id": "MyDockerAppplicationDeploymentConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/
DockerApplicationDeployment/versions/5",
            "Parameters": {
                "DockerComposeFileS3Bucket": "amzn-s3-demo-bucket",
                "DockerComposeFileS3Key": "production-docker-compose.yml",
                "DockerComposeFileS3Version": "123",
                "DockerComposeFileDestinationPath": "/home/username/myCompose",
                "DockerUserId": "1000",
                "AWSSecretsArnList": "[\"arn:aws:secretsmanager:region:account-
id:secret:greengrass-secret1-hash\",\"arn:aws:secretsmanager:region:account-
id:secret:greengrass-secret2-hash\"]",
                "DockerContainerStatusLogFrequency": "30",
                "ForceDeploy": "True",
                "DockerPullBeforeUp": "True"
            }
        }
    ]
}'
```

> **Note**
>
> A função do Lambda nesse conector tem um ciclo de vida longo.

Dados de entrada

Este conector não requer nem aceita dados de entrada.

Dados de saída

Este conector publica o status do comando `docker-compose up` como dados de saída.

Filtro de tópico na assinatura

`dockerapplicationdeploymentconnector/message/status`

Exemplo de resultado: sucesso

```
{
```

```
  "status":"success",
  "GreengrassDockerApplicationDeploymentStatus":"Successfully triggered docker-
compose up",
  "S3Bucket":"amzn-s3-demo-bucket",
  "ComposeFileName":"production-docker-compose.yml",
  "ComposeFileVersion":"123"
}
```

Exemplo de resultado: falha

```
{
  "status":"fail",
  "error_message":"description of error",
  "error":"InvalidParameter"
}
```

O tipo de erro pode ser `InvalidParameter` ou `InternalError`.

Configurando o usuário do Docker no AWS IoT Greengrass core

O conector de implantação do aplicativo do Docker do Greengrass é executado como o usuário especificado para o parâmetro `DockerUserId`. Se você não especificar um valor, o conector será executado como `ggc_user`, que é a identidade de acesso padrão do Greengrass.

Para permitir que o conector interaja com o daemon do Docker, o usuário do Docker deve pertencer ao grupo Linux do `docker` no núcleo. O usuário do Docker também deve ter permissões de gravação no diretório `DockerComposeFileDestinationPath`. Esse é o local em que o conector armazena o arquivo local `docker-compose.yml` e as credenciais do Docker.

> **Note**
>
> - Recomendamos que você crie um usuário Linux em vez de usar o padrão `ggc_user`. Caso contrário, qualquer função do Lambda no grupo do Greengrass poderá acessar o arquivo Compose e as credenciais do Docker.
> - Convém evitar a execução como raiz, a menos que isso seja absolutamente necessário. Se você especificar o usuário raiz, é necessário permitir que as funções do Lambda sejam executadas como raiz no núcleo AWS IoT Greengrass . Para obter mais informações, consulte the section called “Executar uma função do Lambda como raiz”.

1. Crie o usuário. Você pode executar o comando `useradd` e incluir a opção `-u` opcional para atribuir um UID. Por exemplo:

   ```
   sudo useradd -u 1234 user-name
   ```

2. Adicione o usuário ao grupo `docker` no núcleo. Por exemplo:

   ```
   sudo usermod -aG docker user-name
   ```

   Para obter mais informações, incluindo como criar o grupo `docker`, consulte Gerenciar Docker como usuário não raiz na documentação do Docker.

3. Conceda ao usuário permissão para gravação no diretório especificado para o parâmetro `DockerComposeFileDestinationPath`. Por exemplo:

   a. Para definir o usuário como proprietário do diretório. Este exemplo usa o UID da etapa 1.

      ```
      chown 1234 docker-compose-file-destination-path
      ```

   b. Conceda permissões de leitura e gravação ao proprietário.

      ```
      chmod 700 docker-compose-file-destination-path
      ```

      Para obter mais informações, consulte Como gerenciar permissões para arquivos e pastas no Linux na documentação do Linux Foundation.

   c. Caso você não tenha atribuído um UID ao criar o usuário, ou ainda, caso tenha usado um usuário existente, execute o comando `id` para pesquisar o UID.

      ```
      id -u user-name
      ```

      Use o UID para configurar o parâmetro `DockerUserId` para o conector.

Informações de uso

Ao usar o conector de implantação do aplicativo do Docker do Greengrass, você deve estar ciente das seguintes informações de uso específicas da implementação.

- Prefixo fixo para nomes de projeto. O conector prepõe o prefixo `greengrassdockerapplicationdeployment` para os nomes dos contêineres do Docker que

inicia. O conector usa esse prefixo como o nome do projeto nos comandos `docker-compose` que executa.

- Comportamento de log. O conector grava informações de status e de solução de problemas em um arquivo de log. Você pode configurar AWS IoT Greengrass para enviar registros para CloudWatch Logs e gravar registros localmente. Para obter mais informações, consulte the section called “Registro em log”. Este é o caminho para o log local do conector:

  ```
  /greengrass-root/ggc/var/log/user/region/aws/DockerApplicationDeployment.log
  ```

  Você deve ter permissões root para acessar logs locais.

- Atualizando as imagens do Docker. O Docker armazena imagens em cache no dispositivo de núcleo. Se você atualizar uma imagem do Docker e quiser propagar a alteração para o dispositivo de núcleo, certifique-se de alterar a tag da imagem no arquivo Compose. As alterações entram em vigor após a implantação do grupo do Greengrass.
- Tempo limite de 10 minutos para operações de limpeza. Quando o daemon do Greengrass é interrompido durante uma reinicialização, o comando `docker-compose down` é iniciado. Todos os contêineres do Docker têm duração máxima de 10 minutos após o `docker-compose down` ser iniciado para executar uma operação de limpeza. Se a limpeza não for concluída em 10 minutos, você deverá limpar os contêineres restantes manualmente. Para obter mais informações, consulte docker rm na documentação da CLI do Docker.
- Executando comandos do Docker. Para solucionar problemas, você pode executar comandos do Docker em uma janela de terminal no dispositivo de núcleo. Por exemplo, execute o seguinte comando para visualizar os contêineres do Docker que foram iniciados pelo conector:

  ```
  docker ps --filter name="greengrassdockerapplicationdeployment"
  ```

- ID de recurso reservado. O conector usa o ID `DOCKER_DEPLOYER_SECRET_RESOURCE_RESERVED_ID_index` para os recursos do Greengrass criados por ele no grupo do Greengrass. Os IDs de recurso devem ser exclusivos no grupo, portanto, não atribua um ID de recurso que possa entrar em conflito com esse ID de recurso reservado.
- Modo off-line. Quando você define o parâmetro de configuração `DockerOfflineMode` como `True`, o conector do Docker pode operar no modo offline. Isso pode acontecer quando a implantação de um grupo do Greengrass é reiniciada enquanto o dispositivo principal está off-line e o conector não consegue estabelecer uma conexão com o Amazon S3 nem com o Amazon ECR para recuperar o arquivo do Docker Compose.

Com o modo off-line ativado, o conector tenta baixar seu arquivo do Compose e executar comandos `docker login` como faria em uma reinicialização normal. Se essas tentativas falharem, o conector procurará um arquivo do Compose armazenado localmente na pasta especificada usando o parâmetro `DockerComposeFileDestinationPath`. Se existir um arquivo do Compose local, o conector seguirá a sequência normal de comandos `docker-compose` e extrairá as imagens locais. Se o arquivo do Compose ou as imagens locais não estiverem presentes, o conector falhará. O comportamento dos parâmetros `ForceDeploy` e `StopContainersOnNewDeployment` permanece o mesmo no modo off-line.

Comunicar-se com os contêineres do Docker

AWS IoT Greengrass suporta os seguintes canais de comunicação entre os componentes do Greengrass e os contêineres do Docker:

- As funções do Lambda do Greengrass podem usar APIs REST para se comunicar com processos em contêineres do Docker. Você pode configurar um servidor em um contêiner do Docker que abre uma porta. As funções do Lambda podem se comunicar com o contêiner nessa porta.
- Os processos em contêineres do Docker podem trocar mensagens MQTT por meio do atendente de mensagens local do Greengrass. Você pode configurar o contêiner Docker como um dispositivo cliente no grupo Greengrass e, em seguida, criar assinaturas para permitir que o contêiner se comunique com funções do Greengrass Lambda, dispositivos cliente e outros conectores no grupo ou com AWS IoT o serviço paralelo local. Para obter mais informações, consulte the section called “Configurar a comunicação MQTT com contêineres do Docker”.
- As funções do Lambda do Greengrass podem atualizar um arquivo compartilhado para passar informações para contêineres do Docker. Você pode usar o arquivo Compose para a vinculação-montagem do caminho do arquivo compartilhado para um contêiner do Docker.

Configurar a comunicação MQTT com contêineres do Docker

Você pode configurar um contêiner do Docker como dispositivo cliente do Greengrass e adicioná-lo a um grupo do Greengrass. Em seguida, você pode criar assinaturas que permitem a comunicação MQTT entre o contêiner do Docker e os componentes do Greengrass ou a AWS IoT. No procedimento a seguir, você cria uma assinatura que permite que o dispositivo de contêiner do Docker receba mensagens de atualização de shadow do serviço de shadow local. Você pode seguir esse padrão para criar outras assinaturas.

> **Note**
>
> Esse procedimento supõe que você já criou um grupo do Greengrass e um núcleo do Greengrass (v1.10 ou posterior). Para obter informações sobre a criação de em grupo e de núcleos do Greengrass, consulte Começando com AWS IoT Greengrass.

Para configurar um contêiner do Docker como dispositivo cliente do Greengrass e adicioná-lo a um grupo do Greengrass

1. Crie uma pasta no dispositivo de núcleo para armazenar os certificados e chaves usados para autenticar o dispositivo do Greengrass.

   O caminho do arquivo deve ser montado no contêiner do Docker que você deseja iniciar. O snippet a seguir mostra como montar um caminho de arquivo no arquivo Compose. Neste exemplo, *path-to-device-certs* representa a pasta que você criou nesta etapa.

   ```
   version: '3.3'
   services:
     myService:
       image: user-name/repo:image-tag
       volumes:
         - /path-to-device-certs/:/path-accessible-in-container
   ```

2. No painel de navegação do AWS IoT console, em Gerenciar, expanda dispositivos Greengrass e escolha Grupos (V1).
3. Selecione o grupo de destino.
4. Na página de configuração do grupo, selecione Dispositivos cliente e, em seguida, selecione Associar.
5. No modal Associar um dispositivo cliente a esse grupo, escolha Criar AWS IoT coisa nova.

   A página Criar coisas é aberta em uma nova guia.
6. Na página Criar coisas, selecione Criar uma única coisa, em seguida, selecione Avançar.
7. Na página Especificar propriedades da coisa, insira um nome para o dispositivo e, em seguida, selecione Avançar.
8. Na página Configurar certificado do dispositivo, selecione Avançar.
9. Na página Anexar políticas ao certificado, execute uma das seguintes ações:

- Selecione uma política existente que conceda as permissões exigidas pelos dispositivos cliente e, em seguida, selecione Criar coisa.

  Um modal é aberto, onde você pode baixar os certificados e as chaves que o dispositivo usa para se conectar ao núcleo Nuvem AWS e ao núcleo.

- Crie e anexe uma nova política que conceda permissões ao dispositivo cliente. Faça o seguinte:

  a. Selecione Create policy (Criar política).

     A página Create policy (Criar política) é aberta em uma nova guia.

  b. Na página Create policy (Criar política) faça o seguinte:

     i. Em Nome da política, insira um nome que descreva a política, como **`GreengrassV1ClientDevicePolicy`**.

     ii. Na guia Instruções da política, em Documento da política, selecione JSON.

     iii. Insira o seguinte documento de política. Essa política permite que o dispositivo cliente descubra os núcleos do Greengrass e se comunique sobre todos os tópicos do MQTT. Para obter informações sobre como restringir o acesso a essa política, consulte Autenticação e autorização de dispositivos para AWS IoT Greengrass.

     JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iot:Publish",
        "iot:Subscribe",
        "iot:Connect",
        "iot:Receive"
      ],
      "Resource": [
        "*"
      ]
    },
    {
```

```
      "Effect": "Allow",
      "Action": [
        "greengrass:*"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

iv. Selecione Create (Criar) para criar a política.

c. Volte para a guia do navegador com a página Anexar políticas ao certificado aberta. Faça o seguinte:

i. Na lista Políticas, selecione a política que você criou, como GreengrassV1ClientDevicePolicy.

Se a política não for exibida, selecione o botão de atualização.

ii. Selecione Criar coisa.

Um modal é aberto, onde você pode baixar os certificados e as chaves que o dispositivo usa para se conectar ao núcleo Nuvem AWS e ao núcleo.

10. No modal Baixar certificados e chaves, baixe os certificados do dispositivo.

> **Important**
>
> Antes de selecionar Concluído, faça download dos recursos de segurança.

Faça o seguinte:

a. Em Certificado do dispositivo, selecione Download para baixar o certificado do dispositivo.

b. Em Arquivo de chave pública, selecione Download para baixar a chave pública do certificado.

c. Em Arquivo de chave privada, selecione Download para baixar o arquivo de chave privada para o certificado.

d. Revise Autenticação do servidor no Guia do desenvolvedor da AWS IoT e, em seguida selecione o certificado de CA raiz apropriado. Recomendamos que você use endpoints do

Amazon Trust Services (ATS) e certificados raiz da CA do ATS. Em Certificados CA raiz, selecione Download para obter um certificado de CA raiz.

e. Selecione Done (Concluído).

Anote a ID do certificado, que é comum nos nomes dos arquivos do certificado e das chaves do dispositivo. Você precisará disso mais tarde.

11. Copie os certificados e as chaves na pasta criada na etapa 1.

Em seguida, crie uma assinatura no grupo. Neste exemplo, você cria uma assinatura que permite que o dispositivo de contêiner do Docker receba mensagens MQTT do serviço de shadow local.

> **Note**
>
> O tamanho máximo de um documento do shadow é 8 KB. Para obter mais informações, consulte Cotas do AWS IoT no Guia do desenvolvedor do AWS IoT .

Para criar uma assinatura que permita que o dispositivo de contêiner do Docker receba mensagens MQTT do serviço de shadow local

1. Na página de configuração do grupo, selecione a guia Assinaturas e, em seguida, Adicionar assinatura.
2. Na página Select your source and target, configure a origem e o destino, da seguinte forma:

   a. Para Select a source (Selecione uma origem), selecione Serviços e Serviço de sombra local.

   b. Em Selecionar um destino, selecione Devices (Dispositivos) e o seu dispositivo.

   c. Selecione Next (Próximo).

   d. Na página Filtrar seus dados com um tópico, em Filtro de tópico, selecione **`$aws/things/MyDockerDevice/shadow/update/accepted`** e, em seguida selecione Avançar. *`MyDockerDevice`*Substitua pelo nome do dispositivo que você criou anteriormente.

   e. Selecione Finish.

Inclua o snippet de código a seguir na imagem do Docker a qual você faz referência no arquivo Compose. Esse é o código de dispositivo do Greengrass. Além disso, adicione o código ao contêiner do Docker que inicia o dispositivo do Greengrass dentro do contêiner. Ele pode ser executado como processo separado na imagem, ou em um thread separado.

```
import os
import sys
import time
import uuid

from AWSIoTPythonSDK.core.greengrass.discovery.providers import DiscoveryInfoProvider
from AWSIoTPythonSDK.exception.AWSIoTExceptions import DiscoveryInvalidRequestException
from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient

# Replace thingName with the name you registered for the Docker device.
thingName = "MyDockerDevice"
clientId = thingName

# Replace host with the IoT endpoint for your &AWS-account;.
host = "myPrefix.iot.region.amazonaws.com"

# Replace topic with the topic where the Docker container subscribes.
topic = "$aws/things/MyDockerDevice/shadow/update/accepted"

# Replace these paths based on the download location of the certificates for the Docker
 container.
rootCAPath = "/path-accessible-in-container/AmazonRootCA1.pem"
certificatePath = "/path-accessible-in-container/certId-certificate.pem.crt"
privateKeyPath = "/path-accessible-in-container/certId-private.pem.key"


# Discover Greengrass cores.
discoveryInfoProvider = DiscoveryInfoProvider()
discoveryInfoProvider.configureEndpoint(host)
discoveryInfoProvider.configureCredentials(rootCAPath, certificatePath, privateKeyPath)
discoveryInfoProvider.configureTimeout(10)  # 10 seconds.

GROUP_CA_PATH = "./groupCA/"
MQTT_QOS = 1

discovered = False
groupCA = None
coreInfo = None
```

```
try:
    # Get discovery info from AWS IoT.
    discoveryInfo = discoveryInfoProvider.discover(thingName)
    caList = discoveryInfo.getAllCas()
    coreList = discoveryInfo.getAllCores()

    # Use first discovery result.
    groupId, ca = caList[0]
    coreInfo = coreList[0]

    # Save the group CA to a local file.
    groupCA = GROUP_CA_PATH + groupId + "_CA_" + str(uuid.uuid4()) + ".crt"
    if not os.path.exists(GROUP_CA_PATH):
        os.makedirs(GROUP_CA_PATH)
    groupCAFile = open(groupCA, "w")
    groupCAFile.write(ca)
    groupCAFile.close()
    discovered = True
except DiscoveryInvalidRequestException as e:
    print("Invalid discovery request detected!")
    print("Type: %s" % str(type(e)))
    print("Error message: %s" % str(e))
    print("Stopping...")
except BaseException as e:
    print("Error in discovery!")
    print("Type: %s" % str(type(e)))
    print("Error message: %s" % str(e))
    print("Stopping...")

myAWSIoTMQTTClient = AWSIoTMQTTClient(clientId)
myAWSIoTMQTTClient.configureCredentials(groupCA, privateKeyPath, certificatePath)


# Try to connect to the Greengrass core.
connected = False
for connectivityInfo in coreInfo.connectivityInfoList:
    currentHost = connectivityInfo.host
    currentPort = connectivityInfo.port
    myAWSIoTMQTTClient.configureEndpoint(currentHost, currentPort)
    try:
        myAWSIoTMQTTClient.connect()
        connected = True
    except BaseException as e:
```

```
        print("Error in connect!")
        print("Type: %s" % str(type(e)))
        print("Error message: %s" % str(e))
    if connected:
        break

if not connected:
    print("Cannot connect to core %s. Exiting..." % coreInfo.coreThingArn)
    sys.exit(-2)

# Handle the MQTT message received from GGShadowService.
def customCallback(client, userdata, message):
    print("Received an MQTT message")
    print(message)

# Subscribe to the MQTT topic.
myAWSIoTMQTTClient.subscribe(topic, MQTT_QOS, customCallback)

# Keep the process alive to listen for messages.
while True:
    time.sleep(1)
```

Observações de segurança

Ao usar o conector de implantação do aplicativo do Docker do Greengrass, esteja ciente das seguintes considerações de segurança.

Armazenamento local do arquivo Compose do Docker

O conector armazena uma cópia do arquivo Compose no diretório especificado para o parâmetro `DockerComposeFileDestinationPath`.

Você é responsável por proteger esse diretório. Você deve usar permissões do sistema de arquivos para restringir o acesso ao diretório.

Armazenamento local das credenciais do Docker

Se as imagens do Docker estiverem armazenadas em repositórios privados, o conector armazenará suas credenciais do Docker no diretório especificado para o parâmetro `DockerComposeFileDestinationPath`.

Você é responsável por proteger essas credenciais. Por exemplo, use credential-helper no dispositivo de núcleo ao instalar o Docker Engine.

Instalar o Docker Engine de uma fonte confiável

Você é responsável por instalar o Docker Engine de uma fonte confiável. Esse conector usa o daemon do Docker no dispositivo de núcleo para acessar seus ativos do Docker e gerenciar contêineres do Docker.

Escopo das permissões de função de grupo do Greengrass

As permissões adicionadas à função de grupo do Greengrass podem ser assumidas por todos os conectores e funções do Lambda no grupo do Greengrass. Esse conector requer acesso ao arquivo Compose do Docker armazenado em um bucket do S3. Ele também requer acesso ao seu token de autorização do Amazon ECR se suas imagens do Docker estiverem armazenadas em um repositório privado no Amazon ECR.

Licenças

O conector de implantação do aplicativo Greengrass Docker inclui os seguintes terceiros: software/licensing

- AWS SDK para 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 conector é liberado de acordo com o Contrato de licença de software do Greengrass Core.

Changelog

A tabela a seguir descreve as alterações em cada versão do conector.

Versão	Alterações
7	Adicionado `DockerOfflineMode` para usar um arquivo Docker Compose existente quando AWS IoT Greengrass inicia offline. Novas tentativas implementadas para o comando `docker login`. Support para UIDs de 32 bits.
6	Adicionado `StopContainersOnNewDeployment` para substituir a limpeza do contêiner quando uma nova implantação é feita ou quando o GGC é interromp ido. Mecanismos mais seguros de desligamento e inicialização. Correção de erro de validação YAML.
5	As imagens são extraídas antes da execução `docker-compose down` .
4	Foi adicionado o comportamento pull-before-up para atualizar as imagens do Docker.
3	Correção de um problema com a localização de variáveis de ambiente.
2	Adição do parâmetro `ForceDeploy` .
1	Versão inicial.

Um grupo do Greengrass só pode conter uma versão do conector por vez. Para obter informações sobre como fazer upgrade de uma versão do conector, consulte the section called “Atualizar a versões do conector”.

Consulte também

- Integrar a serviços e protocolos usando conectores
- the section called “Conceitos básicos de conectores (console)”
- the section called “Conceitos básicos de conectores (CLI)”

Conector IoT Analytics (descontinuado)

> **Warning**
>
> Esse conector passou para a fase de vida útil estendida e AWS IoT Greengrass não lançará atualizações que forneçam atributos, aprimoramentos para atributos existentes, patches

de segurança ou correções de erros. Para obter mais informações, consulte AWS IoT Greengrass Version 1 política de manutenção.

AWS IoT Analytics descontinuado

AWS IoT Analytics foi descontinuado em 15 de dezembro de 2025. O conector do IoT Analytics não funciona mais. Se você tiver esse conector configurado em seu grupo do Greengrass, remova-o para evitar erros.

O conector do IoT Analytics enviava anteriormente dados do dispositivo local aos AWS IoT Analytics canais. Como AWS IoT Analytics foi descontinuado, esse conector não pode mais fornecer dados. Todas as chamadas de AWS IoT Analytics API agora retornam um`AccessDeniedException`.

Para remover esse conector do seu grupo do Greengrass, use o console AWS IoT ou a AWS IoT Greengrass API para excluir o conector da definição de conector do seu grupo e, em seguida, implante o grupo.

Conector do adaptador de protocolo IP Ethernet IoT

O conector do adaptador de protocolo IP Ethernet IoT coleta dados de dispositivos locais usando o protocolo. Ethernet/IP Você pode usar esse conector para coletar dados de vários dispositivos e publicá-los em um fluxo de mensagens `StreamManager`.

Você também pode usar esse conector com o conector de IoT e seu SiteWise gateway de IoT. SiteWise Seu gateway deve fornecer a configuração do conector. Para obter mais informações, consulte Configurar uma fonte Ethernet/IP (EIP) no guia do usuário de SiteWise IoT.

Note

Esse conector é executado no modo Sem isolamento de contêiner, então você pode implantá-lo em um AWS IoT Greengrass grupo executado em um contêiner Docker.

Esse conector tem as seguintes versões.

Versão	ARN
2 (recomendado)	`arn:aws:greengrass:  region::/connectors/IoTEIPProtocolAdaptor/versions/2`
1	`arn:aws:greengrass:  region::/connectors/IoTEIPProtocolAdaptor/versions/1`

Para obter informações sobre alterações de versão, consulte o Changelog.

Requisitos

Esse conector tem os seguintes requisitos:

Version 1 and 2

- AWS IoT Greengrass Software principal v1.10.2 ou posterior.
- Gerenciador de transmissões ativado no AWS IoT Greengrass grupo.
- Java 8 instalado no dispositivo de núcleo e adicionado à variável de ambiente `PATH`.
- Um mínimo de 256 MB de RAM adicional. Esse requisito é adicional aos requisitos de memória AWS IoT Greengrass principal.

> **Note**
>
> Esse conector só está disponível nas seguintes Regiões:
>
> - cn-north-1
> - ap-southeast-1
> - ap-southeast-2
> - eu-central-1
> - eu-west-1
> - us-east-1
> - us-west-2

Parâmetros do conector

Esse conector oferece suporte aos seguintes parâmetros:

`LocalStoragePath`

O diretório no AWS IoT Greengrass host no qual o SiteWise conector de IoT pode gravar dados persistentes. O diretório padrão é `/var/sitewise`.

Nome de exibição no AWS IoT console: caminho de armazenamento local

Obrigatório: `false`

Digite: `string`

Padrão válido: `^\s*$|\/.`

`ProtocolAdapterConfiguration`

O conjunto de configurações do Ethernet/IP coletor das quais o conector coleta dados ou às quais se conecta. Ele pode ser uma lista vazia.

Nome de exibição no AWS IoT console: Configuração do adaptador de protocolo

Obrigatório: `true`

Tipo: uma string JSON bem-formada que define o conjunto de configurações de feedback compatíveis.

Veja a seguir um exemplo de um `ProtocolAdapterConfiguration`:

```
{
    "sources": [
        {
            "type": "EIPSource",
            "name": "TestSource",
            "endpoint": {
                "ipAddress": "52.89.2.42",
                "port": 44818
            },
            "destination": {
```

```
                "type": "StreamManager",
                "streamName": "MyOutput_Stream",
                "streamBufferSize": 10
            },
            "destinationPathPrefix": "EIPSource_Prefix",
            "propertyGroups": [
                {
                    "name": "DriveTemperatures",
                    "scanMode": {
                        "type": "POLL",
                        "rate": 10000
                    },
                    "tagPathDefinitions": [
                        {
                            "type": "EIPTagPath",
                            "path": "arrayREAL[0]",
                            "dstDataType": "double"
                        }
                    ]
                }
            ]
        }
    ]
}
```

Exemplo de criação de conector (AWS CLI)

O seguinte comando da CLI cria um `ConnectorDefinition` com uma versão inicial que contém o conector do adaptador de protocolo IP Ethernet IoT.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version
'{
    "Connectors": [
        {
            "Id": "MyIoTEIPProtocolConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/
IoTEIPProtocolAdaptor/versions/2",
            "Parameters": {
                "ProtocolAdaptorConfiguration": "{ \"sources\": [{ \"type
\": \"EIPSource\", \"name\": \"Source1\", \"endpoint\": { \"ipAddress\":
 \"54.245.77.218\", \"port\": 44818 }, \"destinationPathPrefix\": \"EIPConnector_Prefix
\", \"propertyGroups\": [{ \"name\": \"Values\", \"scanMode\": { \"type\": \"POLL\",
```

```
 \"rate\": 2000 }, \"tagPathDefinitions\": [{ \"type\": \"EIPTagPath\", \"path\":
 \"arrayREAL[0]\", \"dstDataType\": \"double\" }]}]}]}",
                "LocalStoragePath": "/var/MyIoTEIPProtocolConnectorState"
            }
        }
    ]
}'
```

> **Note**
>
> A função do Lambda nesse conector tem um ciclo de vida longo.

Dados de entrada

Esse conector não aceita mensagens MQTT como dados de entrada.

Dados de saída

Esse conector publica dados para `StreamManager`: Você deve configurar o fluxo de mensagens de destino. As mensagens de saída têm a seguinte estrutura:

```
{
    "alias": "string",
    "messages": [
        {
            "name": "string",
            "value": boolean|double|integer|string,
            "timestamp": number,
            "quality": "string"
        }
    ]
}
```

Licenças

O conector do adaptador de protocolo IP Ethernet IoT inclui os seguintes componentes de terceiros: software/licensing

- Ethernet/IP cliente
- MapDB

- Elsa

Esse conector é liberado de acordo com o Contrato de licença de software do Greengrass Core.

Changelog

A tabela a seguir descreve as alterações em cada versão do conector.

Versão	Alterações	Data
2	Esta versão contém correções de erros.	23 de dezembro de 2021
1	Versão inicial.	15 de dezembro de 2020

Um grupo do Greengrass só pode conter uma versão do conector por vez. Para obter informações sobre como fazer upgrade de uma versão do conector, consulte the section called “Atualizar a versões do conector”.

Consulte também

- Integrar a serviços e protocolos usando conectores
- the section called “Conceitos básicos de conectores (console)”
- the section called “Conceitos básicos de conectores (CLI)”

Conector IoT SiteWise

O SiteWise conector IoT envia dados locais de dispositivos e equipamentos para as propriedades dos ativos em. AWS IoT SiteWise Você pode usar esse conector para coletar dados de vários OPC-UA servidores e publicá-los na IoT SiteWise. O conector envia os dados para as propriedades do ativo na região atual Conta da AWS e na região.

> **Note**
>
> A IoT SiteWise é um serviço totalmente gerenciado que coleta, processa e visualiza dados de dispositivos e equipamentos industriais. Você pode configurar propriedades de ativos

que processam os dados brutos enviados deste conector para as propriedades de medição dos seus ativos. Por exemplo, você pode definir uma propriedade de transformação que converta os pontos de dados de temperatura Celsius de um dispositivo em Fahrenheit ou pode definir uma propriedade métrica que calcule a temperatura média por hora. Para obter mais informações, consulte O que é o AWS IoT SiteWise? no Guia do usuário do AWS IoT SiteWise .

O conector envia dados para a IoT SiteWise com os caminhos de fluxo de OPC-UA dados enviados dos OPC-UA servidores. Por exemplo, o caminho do fluxo de dados `/company/windfarm/3/turbine/7/temperature` pode representar o sensor de temperatura da turbina #7 na fazenda de energia eólica #3. Se o AWS IoT Greengrass núcleo perder a conexão com a Internet, o conector armazena os dados em cache até que ele possa se conectar com êxito ao Nuvem AWS. Você pode configurar o tamanho máximo do buffer de disco usado para armazenar dados em cache. Se o tamanho do cache exceder o tamanho máximo do buffer de disco, o conector descartará os dados mais antigos da fila.

Depois de configurar e implantar o SiteWise conector de IoT, você pode adicionar um gateway e OPC-UA fontes no console de IoT. SiteWise Ao configurar uma fonte no console, você pode filtrar ou prefixar os caminhos do fluxo de OPC-UA dados enviados pelo conector de SiteWise IoT. Para obter instruções sobre como concluir a configuração do gateway e das origens, consulte Adicionar o gateway no Guia do Usuário AWS IoT SiteWise .

A IoT SiteWise recebe dados somente de fluxos de dados que você mapeou para as propriedades de medição dos ativos de IoT. SiteWise Para mapear fluxos de dados para propriedades de ativos, você pode definir o alias de uma propriedade para ser equivalente a um caminho de fluxo de OPC-UA dados. Para saber mais sobre como definir modelos de ativos e criar ativos, consulte Modelagem de ativos industriais no Guia do usuário do AWS IoT SiteWise .

Observações

Você pode usar o gerenciador de streams para fazer upload de dados para a IoT SiteWise de fontes que OPC-UA não sejam servidores. O gerenciador de fluxo também fornece suporte personalizável para gerenciamento de persistência e largura de banda. Para obter mais informações, consulte Gerenciamento de fluxos de dados.
Esse conector é executado no modo de isolamento Sem contêiner para que você possa implantá-lo em um grupo do Greengrass que é executado em um contêiner do Docker.

Esse conector tem as seguintes versões.

Versão	ARN
12 (recomendado)	`arn:aws:greengrass:region::/connectors/IoTSiteWise/versions/12`
11	`arn:aws:greengrass:region::/connectors/IoTSiteWise/versions/11`
10	`arn:aws:greengrass:region::/connectors/IoTSiteWise/versions/10`
9	`arn:aws:greengrass:region::/connectors/IoTSiteWise/versions/9`
8	`arn:aws:greengrass:region::/connectors/IoTSiteWise/versions/8`
7	`arn:aws:greengrass:region::/connectors/IoTSiteWise/versions/7`
6	`arn:aws:greengrass:region::/connectors/IoTSiteWise/versions/6`
5	`arn:aws:greengrass:region::/connectors/IoTSiteWise/versions/5`
4	`arn:aws:greengrass:region::/connectors/IoTSiteWise/versions/4`

Versão	ARN
3	`arn:aws:greengrass:  region::/connectors/IoTSiteWise/versions/ 3`
2	`arn:aws:greengrass:  region::/connectors/IoTSiteWise/versions/ 2`
1	`arn:aws:greengrass:  region::/connectors/IoTSiteWise/versions/ 1`

Para obter informações sobre alterações de versão, consulte o Changelog.

Requisitos

Esse conector tem os seguintes requisitos:

Version 9, 10, 11, and 12

> **Important**
>
> Esta versão apresenta novos requisitos: software AWS IoT Greengrass principal v1.10.2 e gerenciador de streams.

- AWS IoT Greengrass Software principal v1.10.2.
- Gerenciador de fluxo ativado no grupo do Greengrass.
- Java 8 instalado no dispositivo de núcleo e adicionado à variável de ambiente PATH.
- Esse conector pode ser usado somente nas regiões da Amazon Web Services em que ambos AWS IoT Greengrasse IoT SiteWise são compatíveis.
- Uma política do IAM adicionada à função do grupo do Greengrass. Essa função permite que o AWS IoT Greengrass grupo acesse a `iotsitewise:BatchPutAssetPropertyValue` ação no ativo raiz de destino e seus filhos, conforme mostrado no exemplo a seguir. Você pode

remover o `Condition` da política para permitir que o conector acesse todos os seus ativos de IoT SiteWise .

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
             "Effect": "Allow",
             "Action": "iotsitewise:BatchPutAssetPropertyValue",
             "Resource": "*",
             "Condition": {
                 "StringLike": {
                     "iotsitewise:assetHierarchyPath": [
                         "/root node asset ID",
                         "/root node asset ID/*"
                     ]
                 }
             }
        }
    ]
}
```

Para obter mais informações, consulte Adicionando e removendo políticas do IAM no Guia do usuário do IAM.

Versions 6, 7, and 8

> **Important**
>
> Esta versão apresenta novos requisitos: software AWS IoT Greengrass principal v1.10.0 e gerenciador de streams.

- AWS IoT Greengrass Software principal v1.10.0.
- Gerenciador de fluxo ativado no grupo do Greengrass.
- Java 8 instalado no dispositivo de núcleo e adicionado à variável de ambiente PATH.
- Esse conector pode ser usado somente nas regiões da Amazon Web Services em que ambos AWS IoT Greengrasse IoT SiteWise são compatíveis.

- Uma política do IAM adicionada à função do grupo do Greengrass. Essa função permite que o AWS IoT Greengrass grupo acesse a `iotsitewise:BatchPutAssetPropertyValue` ação no ativo raiz de destino e seus filhos, conforme mostrado no exemplo a seguir. Você pode remover o `Condition` da política para permitir que o conector acesse todos os seus ativos de IoT SiteWise .

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
             "Effect": "Allow",
             "Action": "iotsitewise:BatchPutAssetPropertyValue",
             "Resource": "*",
             "Condition": {
                 "StringLike": {
                     "iotsitewise:assetHierarchyPath": [
                         "/root node asset ID",
                         "/root node asset ID/*"
                     ]
                 }
             }
        }
    ]
}
```

  Para obter mais informações, consulte Adicionando e removendo políticas do IAM no Guia do usuário do IAM.

Version 5

- AWS IoT Greengrass Software principal v1.9.4.
- Java 8 instalado no dispositivo de núcleo e adicionado à variável de ambiente PATH.
- Esse conector pode ser usado somente nas regiões da Amazon Web Services em que ambos AWS IoT Greengrasse IoT SiteWise são compatíveis.
- Uma política do IAM adicionada à função do grupo do Greengrass. Essa função permite que o AWS IoT Greengrass grupo acesse a `iotsitewise:BatchPutAssetPropertyValue` ação no ativo raiz de destino e seus filhos, conforme mostrado no exemplo a seguir. Você pode

remover o `Condition` da política para permitir que o conector acesse todos os seus ativos de IoT SiteWise .

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
             "Effect": "Allow",
             "Action": "iotsitewise:BatchPutAssetPropertyValue",
             "Resource": "*",
             "Condition": {
                 "StringLike": {
                     "iotsitewise:assetHierarchyPath": [
                         "/root node asset ID",
                         "/root node asset ID/*"
                     ]
                 }
             }
        }
    ]
}
```

Para obter mais informações, consulte Adicionando e removendo políticas do IAM no Guia do usuário do IAM.

Version 4

- AWS IoT Greengrass Software principal v1.10.0.
- Java 8 instalado no dispositivo de núcleo e adicionado à variável de ambiente PATH.
- Esse conector pode ser usado somente nas regiões da Amazon Web Services em que ambos AWS IoT Greengrasse IoT SiteWise são compatíveis.
- Uma política do IAM adicionada à função do grupo do Greengrass. Essa função permite que o AWS IoT Greengrass grupo acesse a `iotsitewise:BatchPutAssetPropertyValue` ação no ativo raiz de destino e seus filhos, conforme mostrado no exemplo a seguir. Você pode remover o `Condition` da política para permitir que o conector acesse todos os seus ativos de IoT SiteWise .

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
             "Effect": "Allow",
             "Action": "iotsitewise:BatchPutAssetPropertyValue",
             "Resource": "*",
             "Condition": {
                 "StringLike": {
                     "iotsitewise:assetHierarchyPath": [
                         "/root node asset ID",
                         "/root node asset ID/*"
                     ]
                 }
             }
        }
    ]
}
```

Para obter mais informações, consulte Adicionando e removendo políticas do IAM no Guia do usuário do IAM.

Version 3

- AWS IoT Greengrass Software principal v1.9.4.
- Java 8 instalado no dispositivo de núcleo e adicionado à variável de ambiente PATH.
- Esse conector pode ser usado somente nas regiões da Amazon Web Services em que ambos AWS IoT Greengrasse IoT SiteWise são compatíveis.
- Uma política do IAM adicionada à função do grupo do Greengrass. Essa função permite que o AWS IoT Greengrass grupo acesse a `iotsitewise:BatchPutAssetPropertyValue` ação no ativo raiz de destino e seus filhos, conforme mostrado no exemplo a seguir. Você pode remover o `Condition` da política para permitir que o conector acesse todos os seus ativos de IoT SiteWise .

  JSON

```
{
```

```
    "Version":"2012-10-17",
    "Statement": [
        {
             "Effect": "Allow",
             "Action": "iotsitewise:BatchPutAssetPropertyValue",
             "Resource": "*",
             "Condition": {
                 "StringLike": {
                     "iotsitewise:assetHierarchyPath": [
                         "/root node asset ID",
                         "/root node asset ID/*"
                     ]
                 }
             }
        }
    ]
}
```

Para obter mais informações, consulte Adicionando e removendo políticas do IAM no Guia do usuário do IAM.

Versions 1 and 2

- AWS IoT Greengrass Software principal v1.9.4.
- Java 8 instalado no dispositivo de núcleo e adicionado à variável de ambiente PATH.
- Esse conector pode ser usado somente nas regiões da Amazon Web Services em que ambos AWS IoT Greengrasse IoT SiteWise são compatíveis.
- Uma política do IAM adicionada à função de grupo do Greengrass que permite o acesso AWS IoT Core e a `iotsitewise:BatchPutAssetPropertyValue` ação no ativo raiz de destino e em seus filhos, conforme mostrado no exemplo a seguir. Você pode remover o `Condition` da política para permitir que o conector acesse todos os seus ativos de IoT SiteWise .

  JSON

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

```
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "iotsitewise:assetHierarchyPath": [
                        "/root node asset ID",
                        "/root node asset ID/*"
                    ]
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect",
                "iot:DescribeEndpoint",
                "iot:Publish",
                "iot:Receive",
                "iot:Subscribe"
            ],
            "Resource": "*"
        }
    ]
}
```

Para saber mais, consulte Adicionar e remover permissões de identidade do IAM no Guia do usuário do IAM.

Parâmetros

Versions 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, and 12

`SiteWiseLocalStoragePath`

O diretório no AWS IoT Greengrass host no qual o SiteWise conector de IoT pode gravar dados persistentes. O padrão é `/var/sitewise`.

Nome de exibição no AWS IoT console: caminho de armazenamento local

Obrigatório: `false`

Digite: `string`

Padrão válido: `^\s*$|\/.`

`AWSSecretsArnList`

Uma lista de segredos em AWS Secrets Manager que cada um contém um par de valores-chave de nome de OPC-UA usuário e senha. Esse segredo deve ser um segredo do tipo par de chave-valor.

Nome de exibição no AWS IoT console: Lista de ARNs para segredos OPC-UA username/password

Obrigatório: `false`

Digite: `JsonArrayOfStrings`

Padrão válido: `\[( ?,? ?\"(arn:(aws(-[a-z]+)*):secretsmanager:[a-z0-9\\-]+:[0-9]{12}:secret:([a-zA-Z0-9\\\\]+\/)*[a-zA-Z0-9\/_+=,.@\\-]+-[a-zA-Z0-9]+)*\")*\]`

`MaximumBufferSize`

O tamanho máximo em GB para uso do SiteWise disco de IoT. O padrão é 10 GB.

Nome de exibição no AWS IoT console: tamanho máximo do buffer de disco

Obrigatório: `false`

Digite: `string`

Padrão válido: `^\s*$|[0-9]+`

Version 1

`SiteWiseLocalStoragePath`

O diretório no AWS IoT Greengrass host no qual o SiteWise conector de IoT pode gravar dados persistentes. O padrão é `/var/sitewise`.

Nome de exibição no AWS IoT console: caminho de armazenamento local

Obrigatório: `false`

Digite: `string`

Padrão válido: `^\s*$|\/.`

`SiteWiseOpcuaUserIdentityTokenSecretArn`

O segredo contém AWS Secrets Manager o par de valores-chave OPC-UA do nome de usuário e da senha. Esse segredo deve ser um segredo do tipo par de chave/valor.

Nome de exibição no AWS IoT console: ARN do segredo OPC-UA username/password

Obrigatório: `false`

Digite: `string`

Padrão válido: `^$|arn:(aws(-[a-z]+)*):secretsmanager:[a-z0-9\\-]+:[0-9]{12}:secret:([a-zA-Z0-9\\\\]+/)*[a-zA-Z0-9/_+=,.@\\-]+-[a-zA-Z0-9]+`

`SiteWiseOpcuaUserIdentityTokenSecretArn-ResourceId`

O recurso secreto no AWS IoT Greengrass grupo que faz referência a um segredo de nome de OPC-UA usuário e senha.

Nome de exibição no AWS IoT console: recurso OPC-UA username/password secreto

Obrigatório: `false`

Digite: `string`

Padrão válido: `^$|.+`

`MaximumBufferSize`

O tamanho máximo em GB para uso do SiteWise disco de IoT. O padrão é 10 GB.

Nome de exibição no AWS IoT console: tamanho máximo do buffer de disco

Obrigatório: `false`

Digite: `string`

Padrão válido: `^\s*$|[0-9]+`

Exemplo de criação de conector (AWS CLI)

O AWS CLI comando a seguir cria um `ConnectorDefinition` com uma versão inicial que contém o conector de IoT SiteWise .

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MyIoTSiteWiseConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/IoTSiteWise/
versions/11"
        }
    ]
}'
```

> **Note**
>
> As funções do Lambda nesse conector têm um ciclo de vida longo.

No AWS IoT Greengrass console, você pode adicionar um conector na página Conectores do grupo. Para obter mais informações, consulte the section called “Conceitos básicos de conectores (console)”.

Dados de entrada

Esse conector não aceita mensagens MQTT como dados de entrada.

Dados de saída

Esse conector não publica mensagens MQTT como dados de saída.

Limites

Esse conector está sujeito aos seguintes limites impostos pela IoT SiteWise, incluindo os seguintes. Para obter mais informações, consulte Endpoints e cotas do AWS IoT SiteWise na Referência geral da AWS.

- Número máximo de gateways por Conta da AWS.
- Número máximo de OPC-UA fontes por gateway.
- Taxa máxima de pontos de dados de timestamp-qualidade-valor (TQV) armazenados por. Conta da AWS
- Taxa máxima de pontos de dados TQV armazenados por propriedade de ativo.

Licenças

Version 9, 10, 11, and 12

O SiteWise conector de IoT inclui os seguintes terceiros: software/licensing

- MapDB
- Elsa
- Eclipse Milo

Esse conector é liberado de acordo com o Contrato de licença de software do Greengrass Core.

Versions 6, 7, and 8

O SiteWise conector de IoT inclui os seguintes terceiros: software/licensing

- Milo / EDL 1.0

Esse conector é liberado de acordo com o Contrato de licença de software do Greengrass Core.

Versions 1, 2, 3, 4, and 5

O SiteWise conector de IoT inclui os seguintes terceiros: software/licensing

- Milo / EDL 1.0
- Chronicle-Queue/Licença Apache 2.0

Esse conector é liberado de acordo com o Contrato de licença de software do Greengrass Core.

Changelog

A tabela a seguir descreve as alterações em cada versão do conector.

Versão	Alterações	Data
12	• Esta versão contém correções de erros.	22 de dezembro de 2021

Versão	Alterações	Data
11	• Suporte para strings que contêm caracteres ocultos ou não imprimíveis. Caracteres ocultos e não imprimíveis são removidos automaticamente antes que os strings sejam enviados para o Nuvem AWS. • Correção de um problema que fazia com que o SiteWise gateway de IoT repetisse infinitamente solicitações inválidas. • Correção de um problema que causava um ponto de verificação corrompid o quando o gateway de SiteWise IoT estava conectado a uma fonte de dados de alta frequência. • Mensagens de erro aprimoradas para ajudar a solucionar problemas na configuração do gateway.	24 de março de 2021

Versão	Alterações	Data
10	Configuração do `StreamManager` para melhorar o tratamento quando a conexão de origem é perdida e restabelecida. Essa versão também aceita OPC-UA valores com um `ServerTimestamp` quando não `SourceTimestamp` está disponível.	22 de janeiro de 2021
9	Support lançado para destinos de `StreamManager` streaming personalizados do Greengrass, OPC-UA faixa morta, modo de escaneamento personalizado e taxa de escaneamento personalizada. Também inclui desempenho aprimorado durante as atualizações de configuração feitas a partir do gateway de IoT SiteWise .	15 de dezembro de 2020
8	Estabilidade aprimorada quando o conector experimenta conectividade de rede intermitente.	19 de novembro de 2020
7	Correção de um problema com as métricas do gateway.	14 de agosto de 2020

Versão	Alterações	Data
6	Foi adicionado suporte para CloudWatch métricas e descoberta automática de novas OPC-UA tags. Esta versão requer o gerenciador de streams e o software AWS IoT Greengrass Core v1.10.0 ou superior.	29 de abril de 2020
5	Corrigido um problema de compatibilidade com o software AWS IoT Greengrass Core v1.9.4.	12 de fevereiro de 2020
4	Corrigido um problema com a reconexão OPC-UA do servidor.	7 de fevereiro de 2020
3	Requisito de permissões `iot:*` removido.	17 de dezembro de 2019
2	Foi adicionado suporte para vários recursos OPC-UA secretos.	10 de dezembro de 2019
1	Versão inicial.	2 de dezembro de 2019

Um grupo do Greengrass só pode conter uma versão do conector por vez. Para obter informações sobre como fazer upgrade de uma versão do conector, consulte the section called “Atualizar a versões do conector”.

Consulte também

- Integrar a serviços e protocolos usando conectores
- the section called “Conceitos básicos de conectores (console)”

- the section called “Conceitos básicos de conectores (CLI)”
- Consulte os tópicos a seguir no Guia do usuário do AWS IoT SiteWise :
  - O que AWS IoT SiteWiseé
  - Usar um gateway
  - CloudWatch Métricas do gateway
  - Solução de problemas de um gateway de IoT SiteWise

Kinesis Firehose

O conector Kinesis Firehose publica dados por meio de um stream de entrega do Amazon Data Firehose para destinos como Amazon S3, Amazon Redshift ou Amazon Service. OpenSearch

Esse conector é um produtor de dados para um fluxo de entrega do Kinesis. Ele recebe dados de entrada em um tópico MQTT e envia os dados para um fluxo de entrega especificado. Em seguida, o fluxo de entrega envia o registro de dados ao destino configurado (por exemplo, um bucket do S3).

Esse conector tem as seguintes versões.

Versão	ARN
5	`arn:aws:greengrass:  region::/connectors/KinesisFirehose/versions/5`
4	`arn:aws:greengrass:  region::/connectors/KinesisFirehose/versions/4`
3	`arn:aws:greengrass:  region::/connectors/KinesisFirehose/versions/3`
2	`arn:aws:greengrass:  region::/connectors/KinesisFirehose/versions/2`

Versão	ARN
1	`arn:aws:greengrass:  region::/connectors/KinesisFirehose/versions/1`

Para obter informações sobre alterações de versão, consulte o Changelog.

Requisitos

Esse conector tem os seguintes requisitos:

Version 4 - 5

- AWS IoT Greengrass Software principal v1.9.3 ou posterior.
- Python, versão 3.7 ou 3.8, instalado no dispositivo de núcleo e adicionado à variável de ambiente PATH.

  > **Note**
  >
  > Para usar o Python 3.8, execute o comando a seguir para criar um symblink da pasta de instalação padrão do Python 3.7 para os binários instalados do Python 3.8.
  >
  > ```
  > sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
  > ```
  >
  > Isso configura seu dispositivo para atender ao requisito Python para AWS IoT Greengrass.

- Um fluxo de entrega configurado do Kinesis. Para ter mais informações, consulte Creating an Amazon Data Firehose delivery stream no Guia do desenvolvedor do Amazon Kinesis Firehose.
- A função de grupo do Greengrass configurada para permitir as ações `firehose:PutRecord` e `firehose:PutRecordBatch` no fluxo de entrega de destino, conforme mostrado no seguinte exemplo de política do IAM.

  JSON

  ```
  {
      "Version":"2012-10-17",
  ```

```
    "Statement":[
        {
            "Sid":"Stmt1528133056761",
            "Action":[
                "firehose:PutRecord",
                "firehose:PutRecordBatch"
            ],
            "Effect":"Allow",
            "Resource":[
            "arn:aws:firehose:us-east-1:123456789012:deliverystream/stream-
name"
            ]
        }
    ]
 }
```

Esse conector permite que você substitua dinamicamente o fluxo de entrega padrão na carga da mensagem de entrada. Se a implementação usar esse atributo, a política do IAM deverá 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 o requisito de função de grupo, você deve configurar a função para conceder as permissões necessárias e certificar-se de que a função tenha sido adicionada ao grupo. Para acessar mais informações, consulte the section called “Gerenciar a função de grupo (console)” ou the section called “Gerenciar a função de grupo (CLI)”.

Versions 2 - 3

- AWS IoT Greengrass Software principal v1.7 ou posterior.
- Python versão 2.7 instalado no dispositivo de núcleo e adicionado à variável de ambiente PATH.
- Um fluxo de entrega configurado do Kinesis. Para ter mais informações, consulte Creating an Amazon Data Firehose delivery stream no Guia do desenvolvedor do Amazon Kinesis Firehose.
- A função de grupo do Greengrass configurada para permitir as ações `firehose:PutRecord` e `firehose:PutRecordBatch` no fluxo de entrega de destino, conforme mostrado no seguinte exemplo de política do IAM.

JSON

```
{
    "Version":"2012-10-17",
    "Statement":[
        {
            "Sid":"Stmt1528133056761",
            "Action":[
                "firehose:PutRecord",
                "firehose:PutRecordBatch"
            ],
            "Effect":"Allow",
            "Resource":[
            "arn:aws:firehose:us-east-1:123456789012:deliverystream/stream-
name"
            ]
        }
    ]
 }
```

Esse conector permite que você substitua dinamicamente o fluxo de entrega padrão na carga da mensagem de entrada. Se a implementação usar esse atributo, a política do IAM deverá 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 o requisito de função de grupo, você deve configurar a função para conceder as permissões necessárias e certificar-se de que a função tenha sido adicionada ao grupo. Para acessar mais informações, consulte the section called “Gerenciar a função de grupo (console)” ou the section called “Gerenciar a função de grupo (CLI)”.

Version 1

- AWS IoT Greengrass Software principal v1.7 ou posterior.
- Python versão 2.7 instalado no dispositivo de núcleo e adicionado à variável de ambiente PATH.
- Um fluxo de entrega configurado do Kinesis. Para ter mais informações, consulte Creating an Amazon Data Firehose delivery stream no Guia do desenvolvedor do Amazon Kinesis Firehose.

- A função de grupo do Greengrass configurada para permitir a ação `firehose:PutRecord` no fluxo de entrega de destino, conforme mostrado no seguinte exemplo de política do IAM.

  JSON

  ```
  {
      "Version":"2012-10-17",
      "Statement":[
          {
              "Sid":"Stmt1528133056761",
              "Action":[
                  "firehose:PutRecord"
              ],
              "Effect":"Allow",
              "Resource":[
              "arn:aws:firehose:us-east-1:123456789012:deliverystream/stream-
  name"
              ]
          }
      ]
   }
  ```

  Esse conector permite que você substitua dinamicamente o fluxo de entrega padrão na carga da mensagem de entrada. Se a implementação usar esse atributo, a política do IAM deverá 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 o requisito de função de grupo, você deve configurar a função para conceder as permissões necessárias e certificar-se de que a função tenha sido adicionada ao grupo. Para acessar mais informações, consulte the section called “Gerenciar a função de grupo (console)” ou the section called “Gerenciar a função de grupo (CLI)”.

Parâmetros do conector

Esse conector oferece os seguintes parâmetros:

Versions 5

`DefaultDeliveryStreamArn`

O ARN do fluxo de entrega padrão do Firehose para o qual enviar dados. O fluxo de destino pode ser substituído pela propriedade `delivery_stream_arn` na carga da mensagem de entrada.

> **Note**
>
> A função do grupo deve permitir as ações apropriadas em todos os fluxos de entrega de destino. Para obter mais informações, consulte the section called “Requisitos”.

Nome de exibição no AWS IoT console: ARN do fluxo de entrega padrão

Obrigatório: `true`

Digite: `string`

Padrão válido: `arn:aws:firehose:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):deliverystream/([a-zA-Z0-9_\-.]+)$`

`DeliveryStreamQueueSize`

O número máximo de registros e serem retidos na memória antes que os novos registros para o mesmo fluxo de entrega sejam rejeitados. O valor mínimo é 2000.

Nome de exibição no AWS IoT console: número máximo de registros a serem armazenados em buffer (por stream)

Obrigatório: `true`

Digite: `string`

Padrão válido: `^([2-9]\\d{3}|[1-9]\\d{4,})$`

`MemorySize`

A quantidade de memória (em KB) a ser alocada para esse conector.

Nome de exibição no AWS IoT console: Tamanho da memória

Obrigatório: `true`

Digite: `string`

Padrão válido: `^[0-9]+$`

`PublishInterval`

O intervalo (em segundos) para a publicação de registros no Firehose. Para desabilitar o agrupamento em lotes, defina esse valor como 0.

Nome de exibição no AWS IoT console: intervalo de publicação

Obrigatório: `true`

Digite: `string`

Valores válidos: `0 - 900`

Padrão válido: `[0-9]|[1-9]\\d|[1-9]\\d\\d|900`

`IsolationMode`

O modo de conteinerização para este conector. O padrão é`GreengrassContainer`, o que significa que o conector é executado em um ambiente de execução isolado dentro do AWS IoT Greengrass contêiner.

> **Note**
>
> A configuração padrão de conteinerização para o grupo não se aplica aos conectores.

Nome de exibição no AWS IoT console: modo de isolamento de contêiner

Obrigatório: `false`

Digite: `string`

Valores válidos: `GreengrassContainer` ou `NoContainer`

Padrão válido: `^NoContainer$|^GreengrassContainer$`

Versions 2 - 4

`DefaultDeliveryStreamArn`

O ARN do fluxo de entrega padrão do Firehose para o qual enviar dados. O fluxo de destino pode ser substituído pela propriedade `delivery_stream_arn` na carga da mensagem de entrada.

> **Note**
>
> A função do grupo deve permitir as ações apropriadas em todos os fluxos de entrega de destino. Para obter mais informações, consulte the section called “Requisitos”.

Nome de exibição no AWS IoT console: ARN do fluxo de entrega padrão

Obrigatório: `true`

Digite: `string`

Padrão válido: `arn:aws:firehose:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):deliverystream/([a-zA-Z0-9_\-.]+)$`

`DeliveryStreamQueueSize`

O número máximo de registros e serem retidos na memória antes que os novos registros para o mesmo fluxo de entrega sejam rejeitados. O valor mínimo é 2000.

Nome de exibição no AWS IoT console: número máximo de registros a serem armazenados em buffer (por stream)

Obrigatório: `true`

Digite: `string`

Padrão válido: `^([2-9]\\d{3}|[1-9]\\d{4,})$`

`MemorySize`

A quantidade de memória (em KB) a ser alocada para esse conector.

Nome de exibição no AWS IoT console: Tamanho da memória

Obrigatório: `true`

Digite: `string`

Padrão válido: `^[0-9]+$`

`PublishInterval`

O intervalo (em segundos) para a publicação de registros no Firehose. Para desabilitar o agrupamento em lotes, defina esse valor como 0.

Nome de exibição no AWS IoT console: intervalo de publicação

Obrigatório: `true`

Digite: `string`

Valores válidos: `0 - 900`

Padrão válido: `[0-9]|[1-9]\\d|[1-9]\\d\\d|900`

Version 1

`DefaultDeliveryStreamArn`

O ARN do fluxo de entrega padrão do Firehose para o qual enviar dados. O fluxo de destino pode ser substituído pela propriedade `delivery_stream_arn` na carga da mensagem de entrada.

> **Note**
>
> A função do grupo deve permitir as ações apropriadas em todos os fluxos de entrega de destino. Para obter mais informações, consulte the section called “Requisitos”.

Nome de exibição no AWS IoT console: ARN do fluxo de entrega padrão

Obrigatório: `true`

Digite: `string`

Padrão válido: `arn:aws:firehose:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):deliverystream/([a-zA-Z0-9_\-.]+)$`

Example

Exemplo de criação de conector (AWS CLI)

O seguinte comando da CLI cria um `ConnectorDefinition` com uma versão inicial que contém o conector .

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MyKinesisFirehoseConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/KinesisFirehose/
versions/5",
            "Parameters": {
                "DefaultDeliveryStreamArn": "arn:aws:firehose:region:account-
id:deliverystream/stream-name",
                "DeliveryStreamQueueSize": "5000",
                "MemorySize": "65535",
                "PublishInterval": "10",
                "IsolationMode" : "GreengrassContainer"
            }
        }
    ]
}'
```

No AWS IoT Greengrass console, você pode adicionar um conector na página Conectores do grupo. Para obter mais informações, consulte the section called “Conceitos básicos de conectores (console)”.

Dados de entrada

Esse conector aceita conteúdo de fluxos em tópicos MQTT e envia o conteúdo para o fluxo de entrega de destino. Ele aceita dois tipos de dados de entrada:

- Dados JSON no tópico `kinesisfirehose/message`.
- Dados binários no tópico `kinesisfirehose/message/binary/#`.

Versions 2 - 5

Filtro de tópico: `kinesisfirehose/message`

Use esse tópico para enviar uma mensagem que contenha dados JSON.

Propriedades de mensagens

`request`

Os dados a serem enviados para o fluxo de entrega e o fluxo de entrega de destino, se diferentes do fluxo padrão.

Obrigatório: `true`

Tipo: `object` que inclui as seguintes propriedades:

`data`

Os dados a serem enviados para o fluxo de entrega.

Obrigatório: `true`

Digite: `string`

`delivery_stream_arn`

O ARN do fluxo de entrega do Kinesis de destino. Inclua essa propriedade para substituir o fluxo de entrega padrão.

Obrigatório: `false`

Digite: `string`

Padrão válido: `arn:aws:firehose:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):deliverystream/([a-zA-Z0-9_\-.]+)$`

`id`

Um ID arbitrário para a solicitação. Essa propriedade é usada para mapear a solicitação de entrada para uma resposta de saída. Quando especificada, a propriedade `id` no objeto de resposta é definida para esse valor. Se você não usa esse atributo, é possível omitir essa propriedade ou especificar uma string vazia.

Obrigatório: `false`

Digite: `string`

Padrão válido: `.*`

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"
}
```

Filtro de tópico: `kinesisfirehose/message/binary/#`

Use esse tópico para enviar uma mensagem que contenha dados binários. O conector não analisa dados binários. Os dados são transmitidos da forma que se encontram.

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/`. Lembre-se de incluir uma barra no final.

Version 1

Filtro de tópico: `kinesisfirehose/message`

Use esse tópico para enviar uma mensagem que contenha dados JSON.

Propriedades de mensagens

`request`

Os dados a serem enviados para o fluxo de entrega e o fluxo de entrega de destino, se diferentes do fluxo padrão.

Obrigatório: `true`

Tipo: `object` que inclui as seguintes propriedades:

`data`

Os dados a serem enviados para o fluxo de entrega.

Obrigatório: `true`

Digite: `string`

`delivery_stream_arn`

O ARN do fluxo de entrega do Kinesis de destino. Inclua essa propriedade para substituir o fluxo de entrega padrão.

Obrigatório: `false`

Digite: `string`

Padrão válido: `arn:aws:firehose:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):deliverystream/([a-zA-Z0-9_\-.]+)$`

`id`

Um ID arbitrário para a solicitação. Essa propriedade é usada para mapear a solicitação de entrada para uma resposta de saída. Quando especificada, a propriedade `id` no objeto de resposta é definida para esse valor. Se você não usa esse atributo, é possível omitir essa propriedade ou especificar uma string vazia.

Obrigatório: `false`

Digite: `string`

Padrão válido: `.*`

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"
}
```

Filtro de tópico: `kinesisfirehose/message/binary/#`

Use esse tópico para enviar uma mensagem que contenha dados binários. O conector não analisa dados binários. Os dados são transmitidos da forma que se encontram.

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/`. Lembre-se de incluir uma barra no final.

Dados de saída

O conector publica informações de status como dados de saída em um tópico MQTT.

Versions 2 - 5

Filtro de tópico na assinatura

`kinesisfirehose/message/status`

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 conector detectar um erro que pode ser repetido (por exemplo, erros de conexão), ele tentará publicar novamente no próximo lote. O recuo exponencial é gerenciado pelo SDK. AWS As solicitações que falham com erros repetíveis são adicionadas de volta ao final da fila para publicação.

Version 1

Filtro de tópico na assinatura

`kinesisfirehose/message/status`

Exemplo de resultado: sucesso

```
{
   "response": {
       "firehose_record_id": "1lxfuuuFomkpJYzt/34ZU/r8JYPf8Wyf7AXqlXm",
       "status": "success"
    },
    "id": "request123"
}
```

Exemplo de resultado: falha

```
{
   "response" : {
       "error": "ResourceNotFoundException",
       "error_message": "An error occurred (ResourceNotFoundException) when
 calling the PutRecord operation: Firehose test1 not found under account
 123456789012.",
       "status": "fail"
```

```
    },
    "id": "request123"
}
```

Exemplo de uso

Use as seguintes etapas de alto nível para configurar um exemplo de função do Lambda Python 3.7 que pode ser usado para testar o conector.

> **Note**
>
> - Se você usar outros tempos de execução do Python, poderá criar um link simbólico para Python3.x o Python 3.7.
> - Os tópicos Conceitos básicos de conectores (console) e Conceitos básicos de conectores (CLI) contêm etapas detalhadas que mostram como configurar e implantar um exemplo do conector de notificações do Twilio.

1. Certifique-se de cumprir os requisitos para o conector.

   Para o requisito de função de grupo, você deve configurar a função para conceder as permissões necessárias e certificar-se de que a função tenha sido adicionada ao grupo. Para acessar mais informações, consulte the section called “Gerenciar a função de grupo (console)” ou the section called “Gerenciar a função de grupo (CLI)”.
2. Crie e publique uma função do Lambda que envie dados de entrada para o conector.

   Salve o código de exemplo como arquivo PY. Baixe e descompacte o SDK do AWS IoT Greengrass Core para Python. Crie então um pacote zip que contenha o arquivo PY e a pasta `greengrasssdk` no nível raiz. Este pacote zip é o pacote de implantação que você transfere por upload para o AWS Lambda.

   Depois de criar a função do Lambda Python 3.7, publique uma versão de função e crie um alias.
3. Configure o grupo do Greengrass.

   a. Adicione a função do Lambda pelo seu alias (recomendado). Configure o ciclo de vida do Lambda como de longa duração (ou `"Pinned": true` na CLI).

   b. Adicione o conector e configure seus parâmetros.

c. Adicione assinaturas que permitam que o conector receba dados de entrada JSON e envie dados de saída em filtros de tópico compatíveis.

   - Defina a função do Lambda como origem, o conector como destino e use um filtro de tópico de entrada compatível.
   - Defina o conector como origem, o AWS IoT Core como destino, e use um filtro de tópico de saída compatível. Você usa essa assinatura para visualizar mensagens de status no AWS IoT console.

4. Implante o grupo.
5. No AWS IoT console, na página Teste, inscreva-se no tópico de dados de saída para ver as mensagens de status do conector. A função de exemplo do Lambda é de longa duração e começa a enviar mensagens imediatamente após o grupo ser implantado.

   Ao finalizar o teste, você pode definir o ciclo de vida do Lambda como sob demanda (ou `"Pinned": false` na CLI) e implantar o grupo. Isso impede o envio de mensagens pela função.

Exemplo

O exemplo a seguir da função do Lambda envia uma mensagem de entrada para o conector. Esta mensagem contém dados JSON.

```
import greengrasssdk
import time
import json

iot_client = greengrasssdk.client('iot-data')
send_topic = 'kinesisfirehose/message'

def create_request_with_all_fields():
    return  {
        "request": {
            "data": "Message from Firehose Connector Test"
        },
        "id" : "req_123"
    }

def publish_basic_message():
    messageToPublish = create_request_with_all_fields()
    print("Message To Publish: ", messageToPublish)
```

```
    iot_client.publish(topic=send_topic,
        payload=json.dumps(messageToPublish))

publish_basic_message()

def lambda_handler(event, context):
    return
```

Licenças

O conector Kinesis Firehose inclui os seguintes componentes de terceiros: software/licensing

- AWS SDK para 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 conector é liberado de acordo com o Contrato de licença de software do Greengrass Core.

Changelog

A tabela a seguir descreve as alterações em cada versão do conector.

Versão	Alterações
5	Adicionado o parâmetro `IsolationMode` para configurar o modo de conteinerização para o conector.
4	Atualização do runtime do Lambda para Python 3.7, o que altera o requisito de runtime.
3	Correção para reduzir o excesso de registro em log e outras correções de erros secundárias.

Versão	Alterações
2	Suporte adicionado para o envio de registros de dados em lote ao Firehose em um intervalo especificado. • Também requer a ação `firehose:PutRecordBatch` na função do grupo. • Novos parâmetros `MemorySize`, `DeliveryStreamQueueSize` e `PublishInterval`. • A mensagem de saída contém uma matriz de respostas de status para os registros de dados publicados.
1	Versão inicial.

Um grupo do Greengrass só pode conter uma versão do conector por vez. Para obter informações sobre como fazer upgrade de uma versão do conector, consulte the section called “Atualizar a versões do conector”.

Consulte também

- Integrar a serviços e protocolos usando conectores
- the section called “Conceitos básicos de conectores (console)”
- the section called “Conceitos básicos de conectores (CLI)”
- O que é o Amazon Kinesis Data Firehose? no Guia do desenvolvedor do Amazon Kinesis.

Conector ML Feedback

> **Warning**
>
> Esse conector passou para a fase de vida útil estendida e AWS IoT Greengrass não lançará atualizações que forneçam atributos, aprimoramentos para atributos existentes, patches

de segurança ou correções de erros. Para obter mais informações, consulte AWS IoT Greengrass Version 1 política de manutenção.

O conector de feedback do ML facilita o acesso aos dados do seu modelo de machine learning (ML) para retreinamento e análise de modelos. O conector:

- Faz upload dos dados de entrada (amostras) usados pelo seu modelo de ML no Amazon S3. A entrada do modelo pode estar em qualquer formato, como imagens, JSON ou áudio. Depois que as amostras são carregadas na nuvem, você pode usá-las para retreinar o modelo para melhorar a acurácia e a precisão de suas previsões. Por exemplo, você pode usar o SageMaker AI Ground Truth para rotular suas amostras e a SageMaker IA para treinar novamente o modelo.
- Publica os resultados da previsão do modelo como mensagens MQTT. Isso permite monitorar e analisar a qualidade da inferência do modelo em tempo real. Você também pode armazenar os resultados da previsão e usá-los para analisar tendências ao longo do tempo.
- Publica métricas sobre uploads de amostras e dados de amostra para a Amazon. CloudWatch

Para configurar esse conector, descreva suas configurações de feedback compatíveis no formato JSON. Uma configuração de feedback define propriedades como o bucket de destino do Amazon S3, o tipo de conteúdo e a estratégia de amostragem. (Uma estratégia de amostragem é usada para determinar quais amostras devem ser carregadas.)

Você pode usar o conector de feedback do ML nos seguintes cenários:

- Com funções do Lambda definidas pelo usuário. Suas funções Lambda de inferência local usam o SDK do AWS IoT Greengrass Machine Learning para invocar esse conector e transmitir a configuração de feedback de destino, a entrada do modelo e a saída do modelo (resultados da previsão). Para ver um exemplo, consulte the section called “Exemplo de uso”.
- Com o conector de classificação de imagem (v2). Para usar esse conector com o conector de classificação de imagem, configure o parâmetro `MLFeedbackConnectorConfigId` para o conector de classificação de imagem.
- Com o conector de detecção de objetos do ML. Para usar esse conector com o conector de detecção de objetos do ML, configure o parâmetro `MLFeedbackConnectorConfigId` para o conector de detecção de objetos do ML.

ARN: `arn:aws:greengrass:region::/connectors/MLFeedback/versions/1`

Requisitos

Esse conector tem os seguintes requisitos:

- AWS IoT Greengrass Core Software v1.9.3 ou posterior.
- Python, versão 3.7 ou 3.8, instalado no dispositivo de núcleo e adicionado à variável de ambiente PATH.

  > **Note**
  >
  > Para usar o Python 3.8, execute o comando a seguir para criar um symblink da pasta de instalação padrão do Python 3.7 para os binários instalados do Python 3.8.
  >
  > ```
  > sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
  > ```
  >
  > Isso configura seu dispositivo para atender ao requisito Python para AWS IoT Greengrass.

- Um ou mais buckets do Amazon S3. O número de buckets que você usa depende da sua estratégia de amostragem.
- A função de grupo do Greengrass configurada para permitir a ação `s3:PutObject` em objetos do bucket do Amazon S3 de destino, conforme mostrado no exemplo de política do IAM a seguir.

  JSON

  ```
  {
      "Version":"2012-10-17",
      "Statement": [
          {
              "Effect": "Allow",
              "Action": "s3:PutObject",
              "Resource": [
                  "arn:aws:s3:::bucket-name/*"
              ]
          }
      ]
  }
  ```

  A política deve incluir todos os buckets de destino como recursos. Você pode conceder acesso granular ou condicional aos recursos (por exemplo, usando um esquema de nomeação * curinga).

Para o requisito de função de grupo, você deve configurar a função para conceder as permissões necessárias e certificar-se de que a função tenha sido adicionada ao grupo. Para acessar mais informações, consulte the section called “Gerenciar a função de grupo (console)” ou the section called “Gerenciar a função de grupo (CLI)”.

- O conector CloudWatch Metrics foi adicionado ao grupo Greengrass e configurado. Isso só é necessário se você deseja usar o atributo de relatório de métricas.
- AWS IoT Greengrass O SDK do Machine Learning v1.1.0 é necessário para interagir com esse conector.

Parâmetros

`FeedbackConfigurationMap`

Um conjunto de uma ou mais configurações de feedback que o conector pode usar para fazer upload de amostras no Amazon S3. Uma configuração de feedback define parâmetros como o bucket de destino, o tipo de conteúdo e a estratégia de amostragem. Quando esse conector é invocado, a função do Lambda de chamada ou o conector especifica uma configuração de feedback de destino.

Nome de exibição no AWS IoT console: mapa de configuração de feedback

Obrigatório: `true`

Tipo: uma string JSON bem-formada que define o conjunto de configurações de feedback compatíveis. Para ver um exemplo, consulte the section called “FeedbackConfigurationMap exemplo”.

O ID de um objeto de configuração de feedback tem os requisitos a seguir.

O ID:

- Deve ser exclusivo entre os objetos de configuração.
- Deve começar com uma letra ou um número. Pode conter letras minúsculas e maiúsculas, números e hifens.
- Deve ter de 2 a 63 caracteres de comprimento.

Obrigatório: `true`

Digite: `string`

Padrão válido: `^[a-zA-Z0-9][a-zA-Z0-9-]{1,62}$`

Exemplos: `MyConfig0`, `config-a`, `12id`

O corpo de um objeto de configuração de comentários contém as propriedades a seguir.

`s3-bucket-name`

O nome do bucket de destino do Amazon S3.

> **Note**
>
> A função de grupo deve permitir a ação `s3:PutObject` em todos os buckets de destino. Para obter mais informações, consulte the section called “Requisitos”.

Obrigatório: `true`

Digite: `string`

Padrão válido: `^[a-z0-9\.\-]{3,63}$`

`content-type`

O tipo de conteúdo das amostras a serem carregadas. Todo o conteúdo de uma configuração de feedback individual deve ser do mesmo tipo.

Obrigatório: `true`

Digite: `string`

Exemplos: `image/jpeg`, `application/json`, `audio/ogg`

`s3-prefix`

O prefixo de chaves a ser usado para amostras carregadas. Um prefixo é semelhante a um nome de diretório. Ele permite que você armazene dados semelhantes no mesmo diretório em um bucket. Para obter mais informações, consulte Chave e metadados de objeto no Guia do usuário do Amazon Simple Storage Service.

Obrigatório: `false`

Digite: `string`

`file-ext`

A extensão de arquivo a ser usada para amostras carregadas. Deve ser uma extensão de arquivo válida para o tipo de conteúdo.

Obrigatório: `false`

Digite: `string`

Exemplos: `jpg`, `json`, `ogg`

`sampling-strategy`

A estratégia de amostragem a ser usada para filtrar quais amostras devem ser carregadas. Se omitido, o conector tentará fazer upload de todas as amostras que ele receber.

Obrigatório: `false`

Tipo: uma string JSON bem-formada que contém as propriedades a seguir.

`strategy-name`

O nome da estratégia de amostragem.

Obrigatório: `true`

Digite: `string`

Valores válidos: `RANDOM_SAMPLING`, `LEAST_CONFIDENCE`, `MARGIN` ou `ENTROPY`

`rate`

A taxa para a estratégia de amostragem Random.

Obrigatório: `true` se `strategy-name` for `RANDOM_SAMPLING`.

Tipo: `number`

Valores válidos: `0.0 - 1.0`

`threshold`

O limite para a estratégia de amostragem Least Confidence, Margin ou Entropy.

Obrigatório: `true` se `strategy-name` for `LEAST_CONFIDENCE`, `MARGIN` ou `ENTROPY`.

Tipo: `number`

Valores válidos:

- `0.0 - 1.0` para a estratégia `LEAST_CONFIDENCE` ou `MARGIN`.
- `0.0 - no limit` para a estratégia `ENTROPY`.

`RequestLimit`

O número máximo de solicitações que o conector pode processar por vez.

Você pode usar esse parâmetro para restringir o consumo de memória limitando o número de solicitações que o conector processa ao mesmo tempo. As solicitações que excederem esse limite serão ignoradas.

Nome de exibição no AWS IoT console: limite de solicitações

Obrigatório: `false`

Digite: `string`

Valores válidos: `0 - 999`

Padrão válido: `^$|^[0-9]{1,3}$`

Exemplo de criação de conector (AWS CLI)

O seguinte comando da CLI cria um `ConnectorDefinition` com uma versão inicial que contém o conector de feedback do ML.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MyMLFeedbackConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/MLFeedback/
versions/1",
            "Parameters": {
                "FeedbackConfigurationMap": "{  \"RandomSamplingConfiguration\":
 {  \"s3-bucket-name\": \"my-aws-bucket-random-sampling\",  \"content-type\":
 \"image/png\",  \"file-ext\": \"png\",  \"sampling-strategy\": {  \"strategy-name
\": \"RANDOM_SAMPLING\",  \"rate\": 0.5  } },  \"LeastConfidenceConfiguration\": {
  \"s3-bucket-name\": \"my-aws-bucket-least-confidence-sampling\",  \"content-type\":
 \"image/png\",  \"file-ext\": \"png\",  \"sampling-strategy\": {  \"strategy-name\":
 \"LEAST_CONFIDENCE\",  \"threshold\": 0.4  } } }",
```

```
                "RequestLimit": "10"
            }
        }
    ]
}'
```

FeedbackConfigurationMap exemplo

A seguir está um valor de exemplo expandido para o parâmetro `FeedbackConfigurationMap`. Este exemplo inclui várias configurações de feedback que usam diferentes estratégias de amostragem.

```
{
    "ConfigID1": {
        "s3-bucket-name": "my-aws-bucket-random-sampling",
        "content-type": "image/png",
        "file-ext": "png",
        "sampling-strategy": {
            "strategy-name": "RANDOM_SAMPLING",
            "rate": 0.5
        }
    },
    "ConfigID2": {
        "s3-bucket-name": "my-aws-bucket-margin-sampling",
        "content-type": "image/png",
        "file-ext": "png",
        "sampling-strategy": {
            "strategy-name": "MARGIN",
            "threshold": 0.4
        }
    },
    "ConfigID3": {
        "s3-bucket-name": "my-aws-bucket-least-confidence-sampling",
        "content-type": "image/png",
        "file-ext": "png",
        "sampling-strategy": {
            "strategy-name": "LEAST_CONFIDENCE",
            "threshold": 0.4
        }
    },
    "ConfigID4": {
        "s3-bucket-name": "my-aws-bucket-entropy-sampling",
        "content-type": "image/png",
```

```
        "file-ext": "png",
        "sampling-strategy": {
            "strategy-name": "ENTROPY",
            "threshold": 2
        }
    },
    "ConfigID5": {
        "s3-bucket-name": "my-aws-bucket-no-sampling",
        "s3-prefix": "DeviceA",
        "content-type": "application/json"
    }
}
```

Estratégias de amostragem

O conector oferece suporte a quatro estratégias de amostragem que determinam se deve ser feito upload de amostras que são passadas para o conector. As amostras são instâncias distintas de dados que um modelo usa para uma previsão. Você poderá usar estratégias de amostragem para filtrar as amostras com maior probabilidade de melhorar a precisão do modelo.

`RANDOM_SAMPLING`

Carrega aleatoriamente amostras com base na taxa fornecida. Ela fará upload de uma amostra se um valor gerado aleatoriamente for menor que a taxa. Quanto maior a taxa, mais amostras serão carregadas.

> **Note**
>
> Essa estratégia ignora qualquer previsão de modelo fornecida.

`LEAST_CONFIDENCE`

Faz upload de amostras cuja probabilidade máxima de confiança fica abaixo do limite fornecido.

Cenário de exemplo:

Limite: `.6`

Previsão do modelo: `[.2, .2, .4, .2]`

Probabilidade máxima de confiança: `.4`

Resultado:

Use a amostra porque a probabilidade de confiança máxima (`.4`) é menor ou igual ao limite (`.6`).

`MARGIN`

Carregará amostras se a margem entre as duas principais probabilidades de confiança estiver dentro do limite fornecido. A margem é a diferença entre as duas principais probabilidades.

Cenário de exemplo:

Limite: `.02`

Previsão do modelo: `[.3, .35, .34, .01]`

As duas principais probabilidades de confiança: `[.35, .34]`

Margem: `.01` (`.35 - .34`)

Resultado:

Use o exemplo porque a margem (`.01`) é menor ou igual ao limite (`.02`).

`ENTROPY`

Faz upload de amostras cuja entropia é maior que o limite fornecido. Usa a entropia normalizada da previsão do modelo.

Cenário de exemplo:

Limite: `0.75`

Previsão do modelo: `[.5, .25, .25]`

Entropia para previsão: `1.03972`

Resultado:

Use o exemplo porque a entropia (`1.03972`) é maior que o limite (`0.75`).

Dados de entrada

User-defined As funções do Lambda usam a `publish` função do `feedback` cliente no SDK do AWS IoT Greengrass Machine Learning para invocar o conector. Para ver um exemplo, consulte the section called “Exemplo de uso”.

> **Note**
>
> Esse conector não aceita mensagens MQTT como dados de entrada.

A função `publish` usa os seguintes argumentos:

ConfigId

O ID da configuração de feedback de destino. Isso deve corresponder ao ID de uma configuração de feedback definida no FeedbackConfigurationMap parâmetro do conector de feedback de ML.

Obrigatório: verdadeiro

Tipo: string

ModelInput

Os dados de entrada que foram passados para um modelo para inferência. Esses dados de entrada são carregados usando a configuração de destino, a menos que sejam filtrados com base na estratégia de amostragem.

Obrigatório: true

Tipo: bytes

ModelPrediction

Os resultados da previsão do modelo. O tipo de resultado pode ser um dicionário ou uma lista. Por exemplo, os resultados da previsão do conector de classificação de imagens do ML são uma lista de probabilidades (como `[0.25, 0.60, 0.15]`). Esses dados são publicados no tópico `/feedback/message/prediction`.

Obrigatório: true

Tipo: dicionário ou lista de valores `float`

Metadados

Customer-defined, metadados específicos do aplicativo anexados à amostra carregada e publicados no tópico. `/feedback/message/prediction` O conector também insere uma chave `publish-ts` com um valor de timestamp nos metadados.

Obrigatório: false

Tipo: dicionário

Exemplo: `{"some-key": "some value"}`

Dados de saída

Esse conector publica dados em três tópicos MQTT:

- As informações de status do conector no tópico `feedback/message/status`.
- Resultados da previsão no tópico `feedback/message/prediction`.
- Métricas destinadas ao CloudWatch `cloudwatch/metric/put` tópico.

Você deve configurar assinaturas para permitir que o conector se comunique em tópicos MQTT. Para obter mais informações, consulte the section called “Entradas e saídas”.

Filtro de tópico: `feedback/message/status`

Use esse tópico para monitorar o status de uploads de amostra e amostras descartadas. O conector publica nesse tópico sempre que recebe uma solicitação.

Exemplo de saída: upload de amostra bem-sucedido


```
{
  "response": {
    "status": "success",
    "s3_response": {
      "ResponseMetadata": {
        "HostId": "IOWQ4fDEXAMPLEQM+ey7N9WgVhSnQ6JEXAMPLEZb7hSQDASK
+Jd1vEXAMPLEa3Km",
        "RetryAttempts": 1,
        "HTTPStatusCode": 200,
        "RequestId": "79104EXAMPLEB723",
        "HTTPHeaders": {
          "content-length": "0",
          "x-amz-id-2":
 "lbbqaDVFOhMlyU3gRvAX1ZIdg8P0WkGkCSSFsYFvSwLZk3j7QZhG5EXAMPLEdd4/pEXAMPLEUqU=",
          "server": "AmazonS3",
          "x-amz-expiration": "expiry-date=\"Wed, 17 Jul 2019 00:00:00 GMT\",
 rule-id=\"OGZjYWY3OTgtYWI2Zi00ZDllLWE4YmQtNzMyYzEXAMPLEoUw\"",
          "x-amz-request-id": "79104EXAMPLEB723",
          "etag": "\"b9c4f172e64458a5fd674EXAMPLE5628\"",
```

```
          "date": "Thu, 11 Jul 2019 00:12:50 GMT",
          "x-amz-server-side-encryption": "AES256"
        }
      },
      "bucket": "greengrass-feedback-connector-data-us-west-2",
      "ETag": "\"b9c4f172e64458a5fd674EXAMPLE5628\"",
      "Expiration": "expiry-date=\"Wed, 17 Jul 2019 00:00:00 GMT\", rule-id=
\"OGZjYWY3OTgtYWI2Zi00ZDllLWE4YmQtNzMyYzEXAMPLEoUw\"",
      "key": "s3-key-prefix/UUID.file_ext",
      "ServerSideEncryption": "AES256"
    }
  },
  "id": "5aaa913f-97a3-48ac-5907-18cd96b89eeb"
}
```

O conector adiciona os campos `bucket` e `key` à resposta do Amazon S3. Para obter mais informações sobre as respostas do Amazon S3, consulte o PUT object (Objeto PUT) na Referência de APIs do Amazon Simple Storage Service.

Exemplo de saída: amostra descartada devido à estratégia de amostragem

```
{
  "response": {
    "status": "sample_dropped_by_strategy"
  },
  "id": "4bf5aeb0-d1e4-4362-5bb4-87c05de78ba3"
}
```

Exemplo de saída: falha no upload de amostra

Um status de falha inclui a mensagem de erro como o valor `error_message` e a classe de exceção como o valor `error`.

```
{
  "response": {
    "status": "fail",
    "error_message": "[RequestId: 4bf5aeb0-d1e4-4362-5bb4-87c05de78ba3] Failed
 to upload model input data due to exception. Model prediction will not be
 published. Exception type: NoSuchBucket, error: An error occurred (NoSuchBucket)
 when calling the PutObject operation: The specified bucket does not exist",
    "error": "NoSuchBucket"
  },
  "id": "4bf5aeb0-d1e4-4362-5bb4-87c05de78ba3"
```

```
}
```

Exemplo de saída: solicitação limitada devido ao limite de solicitações

```
{
  "response": {
    "status": "fail",
    "error_message": "Request limit has been reached (max request: 10 ). Dropping
 request.",
    "error": "Queue.Full"
  },
  "id": "4bf5aeb0-d1e4-4362-5bb4-87c05de78ba3"
}
```

Filtro de tópico: `feedback/message/prediction`

Use este tópico para ouvir previsões com base em dados de amostra carregados. Isso permite que você analise o desempenho do seu modelo em tempo real. As previsões de modelo serão publicadas nesse tópico somente se os dados forem carregados com êxito no Amazon S3. As mensagens publicadas neste tópico estão no formato JSON. Elas contêm o link para o objeto de dados carregado, a previsão do modelo e os metadados incluídos na solicitação.

Você também pode armazenar resultados de previsão e usá-los para relatar e analisar tendências ao longo do tempo. As tendências podem fornecer insights valiosos. Por exemplo, uma tendência de precisão decrescente ao longo do tempo pode ajudar você a decidir se o modelo precisa ser treinado novamente.

Exemplo de saída

```
{
  "source-ref": "s3://greengrass-feedback-connector-data-us-west-2/s3-key-prefix/
UUID.file_ext",
  "model-prediction": [
    0.5,
    0.2,
    0.2,
    0.1
  ],
  "config-id": "ConfigID2",
  "metadata": {
    "publish-ts": "2019-07-11 00:12:48.816752"
  }
```

```
}
```

> **Note**
>
> AWS IoT Analytics foi descontinuado em 15 de dezembro de 2025. O conector do IoT Analytics não funciona mais. Considere usar AWS serviços alternativos, como o Amazon Kinesis Data Streams ou o Amazon S3, para armazenar e analisar os resultados da previsão.

Filtro de tópico: `cloudwatch/metric/put`

Este é o tópico de saída usado para publicar métricas no CloudWatch. Esse recurso exige que você instale e configure o conector CloudWatch Metrics.

As métricas incluem:

- O número de amostras carregadas.
- O tamanho das amostras carregadas.
- O número de erros de uploads para o Amazon S3.
- O número de amostras descartadas com base na estratégia de amostragem.
- O número de solicitações limitadas.

Exemplo de saída: tamanho da amostra de dados (publicada antes do upload real)

```
{
  "request": {
    "namespace": "GreengrassFeedbackConnector",
    "metricData": {
      "value": 47592,
      "unit": "Bytes",
      "metricName": "SampleSize"
    }
  }
}
```

Exemplo de saída: upload de amostra bem-sucedido

```
{
  "request": {
    "namespace": "GreengrassFeedbackConnector",
```

```
    "metricData": {
      "value": 1,
      "unit": "Count",
      "metricName": "SampleUploadSuccess"
    }
  }
}
```

Exemplo de saída: upload de amostra bem-sucedido e resultado da previsão publicado

```
{
  "request": {
    "namespace": "GreengrassFeedbackConnector",
    "metricData": {
      "value": 1,
      "unit": "Count",
      "metricName": "SampleAndPredictionPublished"
    }
  }
}
```

Exemplo de saída: falha no upload de amostra

```
{
  "request": {
    "namespace": "GreengrassFeedbackConnector",
    "metricData": {
      "value": 1,
      "unit": "Count",
      "metricName": "SampleUploadFailure"
    }
  }
}
```

Exemplo de saída: amostra descartada devido à estratégia de amostragem

```
{
  "request": {
    "namespace": "GreengrassFeedbackConnector",
    "metricData": {
      "value": 1,
      "unit": "Count",
      "metricName": "SampleNotUsed"
```

```
    }
  }
}
```

Exemplo de saída: solicitação limitada devido ao limite de solicitações

```
{
  "request": {
    "namespace": "GreengrassFeedbackConnector",
    "metricData": {
      "value": 1,
      "unit": "Count",
      "metricName": "ErrorRequestThrottled"
    }
  }
}
```

Exemplo de uso

O exemplo a seguir é uma função do Lambda definida pelo usuário que usa o SDK de machine learning do AWS IoT Greengrass para enviar dados ao conector de feedback do ML.

> **Note**
>
> Você pode baixar o SDK do AWS IoT Greengrass Machine Learning na página de AWS IoT Greengrass downloads.

```
import json
import logging
import os
import sys
import greengrass_machine_learning_sdk as ml

client = ml.client('feedback')

try:
    feedback_config_id = os.environ["FEEDBACK_CONFIG_ID"]
    model_input_data_dir = os.environ["MODEL_INPUT_DIR"]
    model_prediction_str = os.environ["MODEL_PREDICTIONS"]
```

```
    model_prediction = json.loads(model_prediction_str)
except Exception as e:
    logging.info("Failed to open environment variables. Failed with exception:
{}".format(e))
    sys.exit(1)

try:
    with open(os.path.join(model_input_data_dir, os.listdir(model_input_data_dir)[0]),
 'rb') as f:
        content = f.read()
except Exception as e:
    logging.info("Failed to open model input directory. Failed with exception:
{}".format(e))
    sys.exit(1)

def invoke_feedback_connector():
    logging.info("Invoking feedback connector.")
    try:
        client.publish(
            ConfigId=feedback_config_id,
            ModelInput=content,
            ModelPrediction=model_prediction
        )
    except Exception as e:
        logging.info("Exception raised when invoking feedback connector:{}".format(e))
        sys.exit(1)

invoke_feedback_connector()

def function_handler(event, context):
    return
```

Licenças

O conector de feedback de ML inclui os seguintes terceiros software/licensing:

- AWS SDK para 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

- six/MIT

Esse conector é liberado de acordo com o Contrato de licença de software do Greengrass Core.

Consulte também

- Integrar a serviços e protocolos usando conectores
- the section called “Conceitos básicos de conectores (console)”
- the section called “Conceitos básicos de conectores (CLI)”

Conector de classificação de imagem do ML

> **Warning**
>
> Esse conector passou para a fase de vida útil estendida e AWS IoT Greengrass não lançará atualizações que forneçam atributos, aprimoramentos para atributos existentes, patches de segurança ou correções de erros. Para obter mais informações, consulte AWS IoT Greengrass Version 1 política de manutenção.

Os conectores ML Image Classification fornecem um serviço de inferência de aprendizado de máquina (ML) executado no AWS IoT Greengrass núcleo. Esse serviço de inferência local realiza a classificação de imagens usando um modelo treinado pelo algoritmo de classificação de imagens de SageMaker IA.

User-defined As funções Lambda usam o SDK do AWS IoT Greengrass Machine Learning para enviar solicitações de inferência ao serviço de inferência local. O serviço executa inferência localmente e retorna probabilidades de a imagem de entrada pertencer às categorias específicas.

AWS IoT Greengrass fornece as seguintes versões desse conector, que está disponível para várias plataformas.

Version 2

Conector	Descrição e ARN
Classificação de imagens do ML Aarch64 JTX2	Serviço de inferência de classificação de imagem para NVIDIA Jetson TX2. Compatível com a aceleração de GPU. ARN: `arn:aws:greengrass:region::/connectors/ImageClassificationAarch64JTX2/versions/2`
Classificação de imagens do ML x86_64	Serviço de inferência de classificação de imagem para plataformas x86_64. ARN: `arn:aws:greengrass:region::/connectors/ImageClassificationx86-64/versions/2`
Classificação de imagens do ML ARMv7	Serviço de inferência de classificação de imagem para plataformas ARMv7. ARN: `arn:aws:greengrass:region::/connectors/ImageClassificationARMv7/versions/2`

Version 1

Conector	Descrição e ARN
Classificação de imagens do ML Aarch64 JTX2	Serviço de inferência de classificação de imagem para NVIDIA Jetson TX2. Compatível com a aceleração de GPU.

Conector	Descrição e ARN
	ARN: `arn:aws:greengrass:region::/connectors/ImageClassificationAarch64JTX2/versions/1`
Classificação de imagens do ML x86_64	Serviço de inferência de classificação de imagem para plataformas x86_64. ARN: `arn:aws:greengrass:region::/connectors/ImageClassificationx86-64/versions/1`
Classificação de imagens do ML Armv7	Serviço de inferência de classificação de imagem para plataformas Armv7. ARN: `arn:aws:greengrass:region::/connectors/ImageClassificationARMv7/versions/1`

Para obter informações sobre alterações de versão, consulte o Changelog.

Requisitos

Esses conectores têm os seguintes requisitos:

Version 2

- AWS IoT Greengrass Core Software v1.9.3 ou posterior.
- Python, versão 3.7 ou 3.8, instalado no dispositivo de núcleo e adicionado à variável de ambiente PATH.

> **Note**
>
> Para usar o Python 3.8, execute o comando a seguir para criar um symblink da pasta de instalação padrão do Python 3.7 para os binários instalados do Python 3.8.

```
sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
```

Isso configura seu dispositivo para atender ao requisito Python para AWS IoT Greengrass.

- Dependências para a estrutura Apache MXNet instalada no dispositivo de núcleo. Para obter mais informações, consulte the section called “Como instalar as dependências do MXNet”.
- Um recurso de ML no grupo Greengrass que faz referência a uma fonte de modelo de SageMaker IA. Esse modelo deve ser treinado pelo algoritmo de classificação de imagens de SageMaker IA. Para obter mais informações, consulte Algoritmo de classificação de imagens no Amazon SageMaker AI Developer Guide.
- O conector de feedback do ML adicionado ao grupo do Greengrass e configurado. Isso será necessário somente se você quiser usar o conector para fazer upload de dados de entrada do modelo e publicar previsões em um tópico MQTT.
- A função de grupo do Greengrass configurada para permitir a ação `sagemaker:DescribeTrainingJob` no trabalho de treinamento de destino, conforme mostrado no seguinte exemplo de política do IAM.

  JSON

  ```
  {
      "Version":"2012-10-17",
      "Statement": [
          {
              "Effect": "Allow",
              "Action": [
                  "sagemaker:DescribeTrainingJob"
              ],
              "Resource": "arn:aws:sagemaker:us-east-1:123456789012:training-job/training-job-name"
          }
      ]
  }
  ```

  Para o requisito de função de grupo, você deve configurar a função para conceder as permissões necessárias e certificar-se de que a função tenha sido adicionada ao grupo. Para acessar mais informações, consulte the section called “Gerenciar a função de grupo (console)” ou the section called “Gerenciar a função de grupo (CLI)”.

Você pode conceder acesso granular ou condicional aos recursos (por exemplo, usando um esquema de nomeação * curinga). Se você alterar o trabalho de treinamento de destino no futuro, certifique-se de atualizar a função do grupo, conforme necessário.

- AWS IoT Greengrass O Machine Learning SDK v1.1.0 é necessário para interagir com esse conector.

Version 1

- AWS IoT Greengrass Core Software v1.7 ou posterior.
- Python versão 2.7 instalado no dispositivo de núcleo e adicionado à variável de ambiente PATH.
- Dependências para a estrutura Apache MXNet instalada no dispositivo de núcleo. Para obter mais informações, consulte the section called “Como instalar as dependências do MXNet”.
- Um recurso de ML no grupo Greengrass que faz referência a uma fonte de modelo de SageMaker IA. Esse modelo deve ser treinado pelo algoritmo de classificação de imagens de SageMaker IA. Para obter mais informações, consulte Algoritmo de classificação de imagens no Amazon SageMaker AI Developer Guide.
- A função de grupo do Greengrass configurada para permitir a ação `sagemaker:DescribeTrainingJob` no trabalho de treinamento de destino, conforme mostrado no seguinte exemplo de política do IAM.

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sagemaker:DescribeTrainingJob"
            ],
            "Resource": "arn:aws:sagemaker:us-east-1:123456789012:training-
job/training-job-name"
        }
    ]
}
```

Para o requisito de função de grupo, você deve configurar a função para conceder as permissões necessárias e certificar-se de que a função tenha sido adicionada ao grupo. Para acessar mais informações, consulte the section called “Gerenciar a função de grupo (console)” ou the section called “Gerenciar a função de grupo (CLI)”.

Você pode conceder acesso granular ou condicional aos recursos (por exemplo, usando um esquema de nomeação * curinga). Se você alterar o trabalho de treinamento de destino no futuro, certifique-se de atualizar a função do grupo, conforme necessário.

- AWS IoT Greengrass O Machine Learning SDK v1.0.0 ou posterior é necessário para interagir com esse conector.

Parâmetros do conector

Esses conectores fornecem os parâmetros a seguir.

Version 2

`MLModelDestinationPath`

O caminho local absoluto do recurso do ML dentro do ambiente do Lambda. Esse é o caminho de destino especificado para o recurso de ML.

> **Note**
>
> Se você criou o recurso de ML no console, este é o caminho local.

Nome de exibição no AWS IoT console: caminho de destino do modelo

Obrigatório: `true`

Digite: `string`

Padrão válido: `.+`

`MLModelResourceId`

O ID do recurso de ML que faz referência ao modelo de origem.

Nome de exibição no AWS IoT console: recurso ARN do SageMaker job

Obrigatório: `true`

Digite: `string`

Padrão válido: `[a-zA-Z0-9:_-]+`

`MLModelSageMakerJobArn`

O ARN do trabalho de treinamento de SageMaker IA que representa a fonte do modelo de SageMaker IA. O modelo deve ser treinado pelo algoritmo de classificação de imagens de SageMaker IA.

Nome de exibição no AWS IoT console: SageMaker job ARN

Obrigatório: `true`

Digite: `string`

Padrão válido: `^arn:aws:sagemaker:[a-zA-Z0-9-]+:[0-9]+:training-job/[a-zA-Z0-9][a-zA-Z0-9-]+$`

`LocalInferenceServiceName`

O nome do serviço de inferência local. User-defined As funções Lambda invocam o serviço passando o nome para a `invoke_inference_service` função do SDK do AWS IoT Greengrass Machine Learning. Para ver um exemplo, consulte the section called “Exemplo de uso”.

Nome de exibição no AWS IoT console: nome do serviço de inferência local

Obrigatório: `true`

Digite: `string`

Padrão válido: `[a-zA-Z0-9][a-zA-Z0-9-]{1,62}`

`LocalInferenceServiceTimeoutSeconds`

O tempo (em segundos) até que a solicitação de inferência seja encerrada. O valor mínimo é 1.

Nome de exibição no AWS IoT console: Tempo limite (segundo)

Obrigatório: `true`

Digite: `string`

Padrão válido: `[1-9][0-9]*`

`LocalInferenceServiceMemoryLimitKB`

A quantidade de memória (em KB) que o serviço tem acesso. O valor mínimo é 1.

Nome de exibição no AWS IoT console: limite de memória (KB)

Obrigatório: `true`

Digite: `string`

Padrão válido: `[1-9][0-9]*`

`GPUAcceleration`

O contexto de computação CPU ou GPU (acelerada). Essa propriedade aplica-se somente ao conector Aarch64 JTX2 de classificação de imagens do ML.

Nome de exibição no AWS IoT console: aceleração de GPU

Obrigatório: `true`

Digite: `string`

Valores válidos: `CPU` ou `GPU`

`MLFeedbackConnectorConfigId`

O ID da configuração de feedback a ser usada para fazer upload dos dados de entrada do modelo. Ele deve corresponder ao ID de uma configuração de feedback definida para o conector ML Feedback.

Esse parâmetro é necessário somente se você quiser usar o conector ML Feedback para fazer upload de dados de entrada do modelo e publicar previsões em um tópico MQTT.

Nome de exibição no AWS IoT console: ID de configuração do conector ML Feedback

Obrigatório: `false`

Digite: `string`

Padrão válido: `^$|^[a-zA-Z0-9][a-zA-Z0-9-]{1,62}$`

Version 1

`MLModelDestinationPath`

O caminho local absoluto do recurso do ML dentro do ambiente do Lambda. Esse é o caminho de destino especificado para o recurso de ML.

> **Note**
>
> Se você criou o recurso de ML no console, este é o caminho local.

Nome de exibição no AWS IoT console: caminho de destino do modelo

Obrigatório: `true`

Digite: `string`

Padrão válido: `.+`

`MLModelResourceId`

O ID do recurso de ML que faz referência ao modelo de origem.

Nome de exibição no AWS IoT console: recurso ARN do SageMaker job

Obrigatório: `true`

Digite: `string`

Padrão válido: `[a-zA-Z0-9:_-]+`

`MLModelSageMakerJobArn`

O ARN do trabalho de treinamento de SageMaker IA que representa a fonte do modelo de SageMaker IA. O modelo deve ser treinado pelo algoritmo de classificação de imagens de SageMaker IA.

Nome de exibição no AWS IoT console: SageMaker job ARN

Obrigatório: `true`

Digite: `string`

Padrão válido: `^arn:aws:sagemaker:[a-zA-Z0-9-]+:[0-9]+:training-job/[a-zA-Z0-9][a-zA-Z0-9-]+$`

`LocalInferenceServiceName`

O nome do serviço de inferência local. User-defined As funções Lambda invocam o serviço passando o nome para a `invoke_inference_service` função do SDK do AWS IoT Greengrass Machine Learning. Para ver um exemplo, consulte the section called “Exemplo de uso”.

Nome de exibição no AWS IoT console: nome do serviço de inferência local

Obrigatório: `true`

Digite: `string`

Padrão válido: `[a-zA-Z0-9][a-zA-Z0-9-]{1,62}`

`LocalInferenceServiceTimeoutSeconds`

O tempo (em segundos) até que a solicitação de inferência seja encerrada. O valor mínimo é 1.

Nome de exibição no AWS IoT console: Tempo limite (segundo)

Obrigatório: `true`

Digite: `string`

Padrão válido: `[1-9][0-9]*`

`LocalInferenceServiceMemoryLimitKB`

A quantidade de memória (em KB) que o serviço tem acesso. O valor mínimo é 1.

Nome de exibição no AWS IoT console: limite de memória (KB)

Obrigatório: `true`

Digite: `string`

Padrão válido: `[1-9][0-9]*`

`GPUAcceleration`

O contexto de computação CPU ou GPU (acelerada). Essa propriedade aplica-se somente ao conector Aarch64 JTX2 de classificação de imagens do ML.

Nome de exibição no AWS IoT console: aceleração de GPU

Obrigatório: `true`

Digite: `string`

Valores válidos: `CPU` ou `GPU`

Exemplo de criação de conector (AWS CLI)

Os seguintes comandos da CLI criam um `ConnectorDefinition` com uma versão inicial que contém um conector de classificação de imagem do ML.

Exemplo: instância de CPU

Este exemplo cria uma instância do conector Armv7l de classificação de imagem do ML.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MyImageClassificationConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/
ImageClassificationARMv7/versions/2",
            "Parameters": {
                "MLModelDestinationPath": "/path-to-model",
                "MLModelResourceId": "my-ml-resource",
                "MLModelSageMakerJobArn": "arn:aws:sagemaker:us-
west-2:123456789012:training-job:MyImageClassifier",
                "LocalInferenceServiceName": "imageClassification",
                "LocalInferenceServiceTimeoutSeconds": "10",
                "LocalInferenceServiceMemoryLimitKB": "500000",
                "MLFeedbackConnectorConfigId": "MyConfig0"
            }
        }
    ]
}'
```

Exemplo: instância de GPU

Este exemplo cria uma instância do conector Aarch64 JTX2 de classificação de imagens do ML, que é compatível com a aceleração da GPU em uma placa NVIDIA Jetson TX2.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MyImageClassificationConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/
ImageClassificationAarch64JTX2/versions/2",
            "Parameters": {
                "MLModelDestinationPath": "/path-to-model",
                "MLModelResourceId": "my-ml-resource",
                "MLModelSageMakerJobArn": "arn:aws:sagemaker:us-
west-2:123456789012:training-job:MyImageClassifier",
                "LocalInferenceServiceName": "imageClassification",
                "LocalInferenceServiceTimeoutSeconds": "10",
                "LocalInferenceServiceMemoryLimitKB": "500000",
                "GPUAcceleration": "GPU",
                "MLFeedbackConnectorConfigId": "MyConfig0"
            }
        }
    ]
}'
```

> **Note**
>
> A função do Lambda nesses conectores tem um ciclo de vida longo.

No AWS IoT Greengrass console, você pode adicionar um conector na página Conectores do grupo. Para obter mais informações, consulte the section called “Conceitos básicos de conectores (console)”.

Dados de entrada

Esses conectores aceitam um arquivo de imagem como entrada. Os arquivos de imagem de entrada devem estar no formato `jpeg` ou `png`. Para obter mais informações, consulte the section called “Exemplo de uso”.

Esses conectores não aceitam mensagens MQTT como dados de entrada.

Dados de saída

Esses conectores retornam uma previsão formatada para o objeto identificado na imagem de entrada:

```
[0.3,0.1,0.04,...]
```

A previsão contém uma lista de valores que correspondem às categorias usadas no conjunto de dados de treinamento durante o treinamento do modelo. Cada valor representa a probabilidade de a imagem ficar na categoria correspondente. A categoria com a probabilidade mais alta é a previsão dominante.

Esses conectores não publicam mensagens MQTT como dados de saída.

Exemplo de uso

O exemplo de função do Lambda a seguir usa o SDK AWS IoT Greengrass de Machine Learning para interagir com um conector de classificação de imagem do ML.

> **Note**
>
> Você pode baixar o SDK na página de downloads do AWS IoT Greengrass SDK de Machine Learning.

O exemplo inicializa um cliente do SDK e de forma síncrona chama a função `invoke_inference_service` do SDK para invocar o serviço de inferência local. Ela passa o tipo de algoritmo, o nome do serviço, o tipo de imagem e o conteúdo da imagem. Em seguida, o exemplo analisa a resposta do serviço para obter os resultados de probabilidade (previsões).

Python 3.7

```
import logging
from threading import Timer

import numpy as np

import greengrass_machine_learning_sdk as ml
```

```
# We assume the inference input image is provided as a local file
# to this inference client Lambda function.
with open('/test_img/test.jpg', 'rb') as f:
    content = bytearray(f.read())

client = ml.client('inference')

def infer():
    logging.info('invoking Greengrass ML Inference service')

    try:
        resp = client.invoke_inference_service(
            AlgoType='image-classification',
            ServiceName='imageClassification',
            ContentType='image/jpeg',
            Body=content
        )
    except ml.GreengrassInferenceException as e:
        logging.info('inference exception {}("{}")'.format(e.__class__.__name__, e))
        return
    except ml.GreengrassDependencyException as e:
        logging.info('dependency exception {}("{}")'.format(e.__class__.__name__,
 e))
        return

    logging.info('resp: {}'.format(resp))
    predictions = resp['Body'].read().decode("utf-8")
    logging.info('predictions: {}'.format(predictions))

    # The connector output is in the format: [0.3,0.1,0.04,...]
    # Remove the '[' and ']' at the beginning and end.
    predictions = predictions[1:-1]
    count = len(predictions.split(','))
    predictions_arr = np.fromstring(predictions, count=count, sep=',')

    # Perform business logic that relies on the predictions_arr, which is an array
    # of probabilities.

    # Schedule the infer() function to run again in one second.
    Timer(1, infer).start()
    return

infer()
```

```
def function_handler(event, context):
    return
```

Python 2.7

```
import logging
from threading import Timer

import numpy

import greengrass_machine_learning_sdk as gg_ml

# The inference input image.
with open("/test_img/test.jpg", "rb") as f:
    content = f.read()

client = gg_ml.client("inference")


def infer():
    logging.info("Invoking Greengrass ML Inference service")

    try:
        resp = client.invoke_inference_service(
            AlgoType="image-classification",
            ServiceName="imageClassification",
            ContentType="image/jpeg",
            Body=content,
        )
    except gg_ml.GreengrassInferenceException as e:
        logging.info('Inference exception %s("%s")', e.__class__.__name__, e)
        return
    except gg_ml.GreengrassDependencyException as e:
        logging.info('Dependency exception %s("%s")', e.__class__.__name__, e)
        return

    logging.info("Response: %s", resp)
    predictions = resp["Body"].read()
    logging.info("Predictions: %s", predictions)

    # The connector output is in the format: [0.3,0.1,0.04,...]
    # Remove the '[' and ']' at the beginning and end.
    predictions = predictions[1:-1]
```

```
    predictions_arr = numpy.fromstring(predictions, sep=",")
    logging.info("Split into %s predictions.", len(predictions_arr))

    # Perform business logic that relies on predictions_arr, which is an array
    # of probabilities.

    # Schedule the infer() function to run again in one second.
    Timer(1, infer).start()


infer()


# In this example, the required AWS Lambda handler is never called.
def function_handler(event, context):
    return
```

A `invoke_inference_service` função no SDK do AWS IoT Greengrass Machine Learning aceita os seguintes argumentos.

Argumento	Description
`AlgoType`	O nome do tipo de algoritmo a ser usado para inferência. No momento, só há compatibilidade com `image-classification` . Obrigatório: `true` Digite: `string` Valores válidos: `image-classification`
`ServiceName`	O nome do serviço de inferência local. Use o nome que você especificou para o parâmetro `LocalInferenceServiceName` quando você configurou o conector. Obrigatório: `true`

Argumento	Description
	Digite: `string`
`ContentType`	O tipo de mime da imagem de entrada. Obrigatório: `true` Digite: `string` Valores válidos: `image/jpeg, image/png`
`Body`	O conteúdo do arquivo de imagem de entrada. Obrigatório: `true` Digite: `binary`

Instalando dependências do MXNet no AWS IoT Greengrass core

Para usar um conector de classificação de imagem do ML, você deverá instalar as dependências para a estrutura Apache MXNet no dispositivo de núcleo. Os conectores usam a estrutura para fornecer o modelo de ML.

> **Note**
>
> Esses conectores acompanham uma biblioteca do MXNet pré-compilada, para que você não precise instalar a própria estrutura do MXNet no dispositivo de núcleo.

AWS IoT Greengrass fornece scripts para instalar as dependências das seguintes plataformas e dispositivos comuns (ou para usar como referência para instalá-los). Se você estiver usando uma plataforma ou dispositivo diferente, consulte a documentação do MXNet para sua configuração.

Antes de instalar as dependências do MXNet, certifique-se de que as bibliotecas do sistema necessárias (com as versões mínimas especificadas) estejam presentes no dispositivo.

NVIDIA Jetson TX2

1. Instale CUDA Toolkit 9.0 e cuDNN 7.0. Você pode seguir as instruções em the section called "Configurar outros dispositivos" no tutorial Conceitos básicos.
2. Habilite repositórios universe para que o conector possa instalar softwares livres mantidos pela comunidade. Para obter mais informações, consulte Repositories/Ubuntua documentação do Ubuntu.

   a. Abra o arquivo `/etc/apt/sources.list`.

   b. Certifique-se de que as seguintes linhas permaneçam sem comentário.

   ```
   deb http://ports.ubuntu.com/ubuntu-ports/ xenial universe
   deb-src http://ports.ubuntu.com/ubuntu-ports/ xenial universe
   deb http://ports.ubuntu.com/ubuntu-ports/ xenial-updates universe
   deb-src http://ports.ubuntu.com/ubuntu-ports/ xenial-updates universe
   ```

3. Salve uma cópia do script de instalação a seguir em um arquivo denominado `nvidiajtx2.sh` no dispositivo de núcleo.

   Python 3.7

   ```
   #!/bin/bash
   set -e

   echo "Installing dependencies on the system..."
   echo 'Assuming that universe repos are enabled and checking dependencies...'
   apt-get -y update
   apt-get -y dist-upgrade
   apt-get install -y liblapack3 libopenblas-dev liblapack-dev libatlas-base-
   dev
   apt-get install -y python3.7 python3.7-dev

   python3.7 -m pip install --upgrade pip
   python3.7 -m pip install numpy==1.15.0
   python3.7 -m pip install opencv-python || echo 'Error: Unable to install
    OpenCV with pip on this platform. Try building the latest OpenCV from source
    (https://github.com/opencv/opencv).'

   echo 'Dependency installation/upgrade complete.'
   ```

> **Note**
>
> Se o OpenCV não for instalado com êxito usando esse script, você poderá tentar compilar a partir do código-fonte. Para obter mais informações, consulte Instalação no Linux na documentação do OpenCV ou consulte outros recursos online da sua plataforma.

Python 2.7

```
#!/bin/bash
set -e

echo "Installing dependencies on the system..."
echo 'Assuming that universe repos are enabled and checking dependencies...'
apt-get -y update
apt-get -y dist-upgrade
apt-get install -y liblapack3 libopenblas-dev liblapack-dev libatlas-base-dev
 python-dev

echo 'Install latest pip...'
wget https://bootstrap.pypa.io/get-pip.py
python get-pip.py
rm get-pip.py

pip install numpy==1.15.0 scipy

echo 'Dependency installation/upgrade complete.'
```

4. A partir do diretório em que você salvou o arquivo, execute o seguinte comando:

```
sudo nvidiajtx2.sh
```

x86_64 (Ubuntu or Amazon Linux)

1. Salve uma cópia do script de instalação a seguir em um arquivo denominado `x86_64.sh` no dispositivo de núcleo.

Python 3.7

```
#!/bin/bash
set -e

echo "Installing dependencies on the system..."

release=$(awk -F= '/^NAME/{print $2}' /etc/os-release)

if [ "$release" == '"Ubuntu"' ]; then
  # Ubuntu. Supports EC2 and DeepLens. DeepLens has all the dependencies
 installed, so
  # this is mostly to prepare dependencies on Ubuntu EC2 instance.
  apt-get -y update
  apt-get -y dist-upgrade

  apt-get install -y libgfortran3 libsm6 libxext6 libxrender1
  apt-get install -y python3.7 python3.7-dev
elif [ "$release" == '"Amazon Linux"' ]; then
  # Amazon Linux. Expect python to be installed already
  yum -y update
  yum -y upgrade

  yum install -y compat-gcc-48-libgfortran libSM libXrender libXext
else
  echo "OS Release not supported: $release"
  exit 1
fi

python3.7 -m pip install --upgrade pip
python3.7 -m pip install numpy==1.15.0
python3.7 -m pip install opencv-python || echo 'Error: Unable to install
 OpenCV with pip on this platform. Try building the latest OpenCV from source
 (https://github.com/opencv/opencv).'

echo 'Dependency installation/upgrade complete.'
```

> **Note**
>
> Se o OpenCV não for instalado com êxito usando esse script, você poderá tentar compilar a partir do código-fonte. Para obter mais informações, consulte

Instalação no Linux na documentação do OpenCV ou consulte outros recursos online da sua plataforma.

Python 2.7

```
#!/bin/bash
set -e

echo "Installing dependencies on the system..."

release=$(awk -F= '/^NAME/{print $2}' /etc/os-release)

if [ "$release" == '"Ubuntu"' ]; then
  # Ubuntu. Supports EC2 and DeepLens. DeepLens has all the dependencies
 installed, so
  # this is mostly to prepare dependencies on Ubuntu EC2 instance.
  apt-get -y update
  apt-get -y dist-upgrade

  apt-get install -y libgfortran3 libsm6 libxext6 libxrender1 python-dev
 python-pip
elif [ "$release" == '"Amazon Linux"' ]; then
  # Amazon Linux. Expect python to be installed already
  yum -y update
  yum -y upgrade

  yum install -y compat-gcc-48-libgfortran libSM libXrender libXext python-
pip
else
  echo "OS Release not supported: $release"
  exit 1
fi

pip install numpy==1.15.0 scipy opencv-python

echo 'Dependency installation/upgrade complete.'
```

2. A partir do diretório em que você salvou o arquivo, execute o seguinte comando:

```
sudo x86_64.sh
```

Armv7 (Raspberry Pi)

1. Salve uma cópia do script de instalação a seguir em um arquivo denominado `armv7l.sh` no dispositivo de núcleo.

   Python 3.7

   ```
   #!/bin/bash
   set -e

   echo "Installing dependencies on the system..."

   apt-get update
   apt-get -y upgrade

   apt-get install -y liblapack3 libopenblas-dev liblapack-dev
   apt-get install -y python3.7 python3.7-dev

   python3.7 -m pip install --upgrade pip
   python3.7 -m pip install numpy==1.15.0
   python3.7 -m pip install opencv-python || echo 'Error: Unable to install
    OpenCV with pip on this platform. Try building the latest OpenCV from source
    (https://github.com/opencv/opencv).'

   echo 'Dependency installation/upgrade complete.'
   ```

   > **Note**
   >
   > Se o OpenCV não for instalado com êxito usando esse script, você poderá tentar compilar a partir do código-fonte. Para obter mais informações, consulte Instalação no Linux na documentação do OpenCV ou consulte outros recursos online da sua plataforma.

   Python 2.7

   ```
   #!/bin/bash
   set -e

   echo "Installing dependencies on the system..."
   ```

```
apt-get update
apt-get -y upgrade

apt-get install -y liblapack3 libopenblas-dev liblapack-dev python-dev

# python-opencv depends on python-numpy. The latest version in the APT
 repository is python-numpy-1.8.2
# This script installs python-numpy first so that python-opencv can be
 installed, and then install the latest
# numpy-1.15.x with pip
apt-get install -y python-numpy python-opencv
dpkg --remove --force-depends python-numpy

echo 'Install latest pip...'
wget https://bootstrap.pypa.io/get-pip.py
python get-pip.py
rm get-pip.py

pip install --upgrade numpy==1.15.0 picamera scipy

echo 'Dependency installation/upgrade complete.'
```

2. A partir do diretório em que você salvou o arquivo, execute o seguinte comando:

```
sudo bash armv7l.sh
```

> **Note**
>
> Em um Raspberry Pi, usar o `pip` para instalar dependências de machine learning é uma operação com uso intensivo de memória que pode fazer com que o dispositivo fique sem memória e deixe de responder. Como alternativa, você pode aumentar temporariamente o tamanho da permuta:
> Em `/etc/dphys-swapfile`, aumente o valor da variável `CONF_SWAPSIZE` e, em seguida, execute o seguinte comando para reiniciar `dphys-swapfile`.
>
> ```
> /etc/init.d/dphys-swapfile restart
> ```

Registro em log e solução de problemas

Dependendo das configurações do seu grupo, os registros de eventos e erros são gravados CloudWatch nos Registros, no sistema de arquivos local ou em ambos. Os logs desse conector usam o prefixo `LocalInferenceServiceName`. Se o conector se comportar de forma inesperada, verifique os logs do conector. Normalmente, eles contêm informações úteis de depuração, como uma dependência de biblioteca de ML ausente ou a causa de uma falha de startup do conector.

Se o AWS IoT Greengrass grupo estiver configurado para gravar registros locais, o conector grava arquivos de log em*greengrass-root*`/ggc/var/log/user/`*region*`/aws/`. Para obter mais informações sobre o registro em log do Greengrass, consulte the section called “Monitoramento com logs do AWS IoT Greengrass ”.

Use as informações a seguir para ajudar a solucionar os problemas com os conectores de classificação de imagem do ML.

Bibliotecas do sistema necessárias

As guias a seguir listam as bibliotecas do sistema necessárias para cada conector de classificação de imagem do ML.

ML Image Classification Aarch64 JTX2

Biblioteca	Versão mínima
ld-linux-aarch64.so.1	GLIBC_2.17
libc.so.6	GLIBC_2.17
libcublas.so.9.0	não aplicável
libcudart.so.9.0	não aplicável
libcudnn.so.7	não aplicável
libcufft.so.9.0	não aplicável
libcurand.so.9.0	não aplicável
libcusolver.so.9.0	não aplicável

Biblioteca	Versão mínima
libgcc_s.so.1	GCC_4.2.0
libgomp.so.1	GOMP_4.0, OMP_1.0
libm.so.6	GLIBC_2.23
libpthread.so.0	GLIBC_2.17
librt.so.1	GLIBC_2.17
libstdc++.so.6	GLIBCXX_3.4.21, CXXABI_1.3.8

ML Image Classification x86_64

Biblioteca	Versão mínima
ld-linux-x86-64.so.2	GCC_4.0.0
libc.so.6	GLIBC_2.4
libgfortran.so.3	GFORTRAN_1.0
libm.so.6	GLIBC_2.23
libpthread.so.0	GLIBC_2.2.5
librt.so.1	GLIBC_2.2.5
libstdc++.so.6	CXXABI_1.3.8, GLIBCXX_3.4.21

ML Image Classification Armv7

Biblioteca	Versão mínima
ld-linux-armhf.so.3	GLIBC_2.4
libc.so.6	GLIBC_2.7

Biblioteca	Versão mínima
libgcc_s.so.1	GCC_4.0.0
libgfortran.so.3	GFORTRAN_1.0
libm.so.6	GLIBC_2.4
libpthread.so.0	GLIBC_2.4
librt.so.1	GLIBC_2.4
libstdc++.so.6	CXXABI_1.3.8, CXXABI_ARM_1.3.3, GLIBCXX_3.4.20

Problemas

Sintomas	Solução
Em um Raspberry Pi, a mensagem de erro a seguir será registrada em log e você não está usando a câmera: `Failed to initialize libdc1394`	Execute o comando a seguir para desabilitar o driver: `sudo ln /dev/null /dev/raw1394` Essa operação é temporária e o symblink desaparecerá após a reinicialização. Consulte o manual de distribuição do seu sistema operacional para saber como criar automatic amente o link na reinicialização.

Licenças

Os conectores ML Image Classification incluem os seguintes conectores de terceiros software/licensing:

- AWS SDK para 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

- Deep Neural Network Library (DNNL)/Licença Apache 2.0
- OpenMP* Runtime Library/Consulte o licenciamento da Intel OpenMP Runtime Library.
- mxnet/Licença Apache 2.0
- six/MIT

Licenciamento da Intel OpenMP Runtime Library. O runtime Intel® OpenMP* tem licença dupla, com uma licença comercial (COM) como parte dos produtos Intel® Parallel Studio XE Suite e uma licença de código aberto (OSS) BSD.

Esse conector é liberado de acordo com o Contrato de licença de software do Greengrass Core.

Changelog

A tabela a seguir descreve as alterações em cada versão do conector.

Versão	Alterações
2	Foi adicionado o `MLFeedbackConnectorConfigId` parâmetro para suportar o uso do conector ML Feedback para carregar dados de entrada do modelo, publicar previsões em um tópico do MQTT e publicar métricas na Amazon. CloudWatch
1	Versão inicial.

Um grupo do Greengrass só pode conter uma versão do conector por vez. Para obter informações sobre como fazer upgrade de uma versão do conector, consulte the section called “Atualizar a versões do conector”.

Consulte também

- Integrar a serviços e protocolos usando conectores
- the section called “Conceitos básicos de conectores (console)”
- the section called “Conceitos básicos de conectores (CLI)”
- Executar a inferência de machine learning
- Algoritmo de classificação de imagens no Amazon SageMaker AI Developer Guide

Conector de detecção de objetos do ML

> **Warning**
>
> Esse conector passou para a fase de vida útil estendida e AWS IoT Greengrass não lançará atualizações que forneçam atributos, aprimoramentos para atributos existentes, patches de segurança ou correções de erros. Para obter mais informações, consulte AWS IoT Greengrass Version 1 política de manutenção.

Os conectores ML Object Detection fornecem um serviço de inferência de aprendizado de máquina (ML) executado no AWS IoT Greengrass núcleo. Esse serviço de inferência local realiza a detecção de objetos usando um modelo de detecção de objetos compilado pelo compilador de aprendizado profundo SageMaker AI Neo. Dois tipos de modelos de detecção de objetos são compatíveis: Single Shot Multibox Detector (SSD) e You Only Look Once (YOLO) v3. Para obter mais informações, consulte Requisitos do modelo de detecção de objetos.

User-defined As funções Lambda usam o SDK do AWS IoT Greengrass Machine Learning para enviar solicitações de inferência ao serviço de inferência local. O serviço realiza a inferência local em uma imagem de entrada e retorna uma lista de previsões para cada objeto detectado na imagem. Cada previsão contém uma categoria de objeto, uma pontuação de confiança de previsão e coordenadas de pixels que especificam uma caixa delimitadora em torno do objeto previsto.

AWS IoT Greengrass fornece conectores de detecção de objetos de ML para várias plataformas:

Conector	Descrição e ARN
Detecção de objetos do ML Aarch64 JTX2	Serviço de inferência de detecção de objetos para NVIDIA Jetson TX2. Compatível com a aceleração de GPU. ARN: `arn:aws:greengrass:` *`region`*`::/connectors/ObjectDetectionAarch64JTX2/versions/1`
Detecção de objetos do ML x86_64	Serviço de inferência de detecção de objetos para plataformas x86_64. ARN: `arn:aws:greengrass:` *`region`*`::/connectors/ObjectDetectionx86-64/versions/1`
Detecção de objetos do ML ARMv7	Serviço de inferência de detecção de objetos para plataformas ARMv7. ARN: `arn:aws:greengrass:` *`region`*`::/connectors/ObjectDetectionARMv7/versions/1`

Requisitos

Esses conectores têm os seguintes requisitos:

- AWS IoT Greengrass Core Software v1.9.3 ou posterior.
- Python, versão 3.7 ou 3.8, instalado no dispositivo de núcleo e adicionado à variável de ambiente PATH.

> **Note**
>
> Para usar o Python 3.8, execute o comando a seguir para criar um symblink da pasta de instalação padrão do Python 3.7 para os binários instalados do Python 3.8.
>
> ```
> sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
> ```

Isso configura seu dispositivo para atender ao requisito Python para AWS IoT Greengrass.

- Dependências do tempo de execução de aprendizado profundo SageMaker AI Neo instalado no dispositivo principal. Para obter mais informações, consulte the section called “Instalar dependências do runtime de aprendizado profundo do Neo”.
- Um recurso de ML no grupo do Greengrass. O recurso de ML deve fazer referência a um bucket do Amazon S3 que contém um modelo de detecção de objeto. Para obter mais informações, consulte Origem de modelo do Amazon S3.

> **Note**
>
> O modelo deve ser um tipo de modelo de detecção de objetos Single Shot Multibox Detector ou You Only Look Once v3. Ele deve ser compilado usando o compilador de aprendizado profundo SageMaker AI Neo. Para obter mais informações, consulte Requisitos do modelo de detecção de objetos.

- O conector de feedback do ML adicionado ao grupo do Greengrass e configurado. Isso será necessário somente se você quiser usar o conector para fazer upload de dados de entrada do modelo e publicar previsões em um tópico MQTT.
- AWS IoT Greengrass O Machine Learning SDK v1.1.0 é necessário para interagir com esse conector.

Requisitos do modelo de detecção de objetos

Os conectores de detecção de objetos do ML oferecem suporte aos tipos de modelo de detecção de objetos Single Shot multibox Detector (SSD) e You Only Look Once (YOLO) v3. Você pode usar os componentes de detecção de objetos fornecidos pelo GluonCV para treinar o modelo com seu próprio conjunto de dados. Ou pode usar modelos pré-treinados do GluonCV Model Zoo:

- Pre-trained Modelo SSD
- Pre-trained Modelo YOLO v3

Seu modelo de detecção de objetos deve ser treinado com imagens de entrada 512 x 512. Os modelos pré-treinados do GluonCV Model Zoo já atendem a esse requisito.

Modelos treinados de detecção de objetos devem ser compilados com o compilador de aprendizado profundo SageMaker AI Neo. Ao compilar, verifique se o hardware de destino corresponde ao hardware do seu dispositivo de núcleo do Greengrass. Para obter mais informações, consulte SageMaker AI Neo no Amazon SageMaker AI Developer Guide.

O modelo compilado deve ser adicionado como um recurso de ML (origem de modelo do Amazon S3) ao mesmo grupo do Greengrass que o conector.

Parâmetros do conector

Esses conectores fornecem os parâmetros a seguir.

`MLModelDestinationPath`

O caminho absoluto para o bucket do Amazon S3 que contém o modelo de Neo-compatible ML. Esse é o caminho de destino especificado para o recurso de modelo de ML.

Nome de exibição no AWS IoT console: caminho de destino do modelo

Obrigatório: `true`

Digite: `string`

Padrão válido: `.+`

`MLModelResourceId`

O ID do recurso de ML que faz referência ao modelo de origem.

Nome de exibição no AWS IoT console: recurso de ML do grupo Greengrass

Obrigatório: `true`

Digite: `S3MachineLearningModelResource`

Padrão válido: `^[a-zA-Z0-9:_-]+$`

`LocalInferenceServiceName`

O nome do serviço de inferência local. User-defined As funções Lambda invocam o serviço passando o nome para a `invoke_inference_service` função do SDK do AWS IoT Greengrass Machine Learning. Para ver um exemplo, consulte the section called “Exemplo de uso”.

Nome de exibição no AWS IoT console: nome do serviço de inferência local

Obrigatório: `true`

Digite: `string`

Padrão válido: `^[a-zA-Z0-9][a-zA-Z0-9-]{1,62}$`

`LocalInferenceServiceTimeoutSeconds`

O tempo (em segundos) até que a solicitação de inferência seja encerrada. O valor mínimo é 1. O valor padrão é 10.

Nome de exibição no AWS IoT console: Tempo limite (segundo)

Obrigatório: `true`

Digite: `string`

Padrão válido: `^[1-9][0-9]*$`

`LocalInferenceServiceMemoryLimitKB`

A quantidade de memória (em KB) que o serviço tem acesso. O valor mínimo é 1.

Nome de exibição no AWS IoT console: limite de memória

Obrigatório: `true`

Digite: `string`

Padrão válido: `^[1-9][0-9]*$`

`GPUAcceleration`

O contexto de computação CPU ou GPU (acelerada). Essa propriedade aplica-se somente ao conector Aarch64 JTX2 de classificação de imagens do ML.

Nome de exibição no AWS IoT console: aceleração de GPU

Obrigatório: `true`

Digite: `string`

Valores válidos: `CPU` ou `GPU`

`MLFeedbackConnectorConfigId`

O ID da configuração de feedback a ser usada para fazer upload dos dados de entrada do modelo. Ele deve corresponder ao ID de uma configuração de feedback definida para o conector ML Feedback.

Esse parâmetro é necessário somente se você quiser usar o conector ML Feedback para fazer upload de dados de entrada do modelo e publicar previsões em um tópico MQTT.

Nome de exibição no AWS IoT console: ID de configuração do conector ML Feedback

Obrigatório: `false`

Digite: `string`

Padrão válido: `^$|^[a-zA-Z0-9][a-zA-Z0-9-]{1,62}$`

Exemplo de criação de conector (AWS CLI)

O seguinte comando da CLI cria um `ConnectorDefinition` com uma versão inicial que contém um conector de detecção de objeto do ML. Este exemplo cria uma instância do conector ARMv7l detecção de objeto do ML.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MyObjectDetectionConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/
ObjectDetectionARMv7/versions/1",
            "Parameters": {
                "MLModelDestinationPath": "/path-to-model",
                "MLModelResourceId": "my-ml-resource",
                "LocalInferenceServiceName": "objectDetection",
                "LocalInferenceServiceTimeoutSeconds": "10",
                "LocalInferenceServiceMemoryLimitKB": "500000",
                "MLFeedbackConnectorConfigId" : "object-detector-random-sampling"
            }
        }
    ]
```

```
}'
```

> **Note**
>
> A função do Lambda nesses conectores tem um ciclo de vida longo.

No AWS IoT Greengrass console, você pode adicionar um conector na página Conectores do grupo. Para obter mais informações, consulte the section called “Conceitos básicos de conectores (console)”.

Dados de entrada

Esses conectores aceitam um arquivo de imagem como entrada. Os arquivos de imagem de entrada devem estar no formato `jpeg` ou `png`. Para obter mais informações, consulte the section called “Exemplo de uso”.

Esses conectores não aceitam mensagens MQTT como dados de entrada.

Dados de saída

Esses conectores retornam uma lista formatada de resultados de previsão para os objetos identificados na imagem de entrada:

```
    {
        "prediction": [
            [
                14,
                0.9384938478469849,
                0.37763649225234985,
                0.5110225081443787,
                0.6697432398796082,
                0.8544386029243469
            ],
            [
                14,
                0.8859519958496094,
                0,
                0.43536216020584106,
                0.3314110040664673,
```

```
                0.9538808465003967
            ],
            [
                12,
                0.04128098487854004,
                0.5976729989051819,
                0.5747185945510864,
                0.704264223575592,
                0.857937216758728
            ],
            ...
        ]
    }
```

Cada previsão na lista está contida entre colchetes e contém seis valores:

- O primeiro valor representa a categoria de objeto prevista para o objeto identificado. As categorias de objetos e seus valores correspondentes são determinados ao treinar seu modelo de machine learning de detecção de objetos no compilador de aprendizado profundo do Neo.
- O segundo valor é a pontuação de confiança para a previsão da categoria de objeto. Isso representa a probabilidade de a previsão estar correta.
- Os últimos quatro valores correspondem às dimensões de pixels que representam uma caixa delimitadora em torno do objeto previsto na imagem.

Esses conectores não publicam mensagens MQTT como dados de saída.

Exemplo de uso

O exemplo de função do Lambda a seguir usa o SDK AWS IoT Greengrass de machine learning para interagir com um conector de detecção de objetos do ML.

> **Note**
>
> Você pode baixar o SDK na página de downloads do SDK AWS IoT Greengrass de Machine Learning.

O exemplo inicializa um cliente do SDK e de forma síncrona chama a função `invoke_inference_service` do SDK para invocar o serviço de inferência local. Ela passa o tipo

de algoritmo, o nome do serviço, o tipo de imagem e o conteúdo da imagem. Em seguida, o exemplo analisa a resposta do serviço para obter os resultados de probabilidade (previsões).

```
import logging
from threading import Timer

import numpy as np

import greengrass_machine_learning_sdk as ml

# We assume the inference input image is provided as a local file
# to this inference client Lambda function.
with open('/test_img/test.jpg', 'rb') as f:
    content = bytearray(f.read())

client = ml.client('inference')

def infer():
    logging.info('invoking Greengrass ML Inference service')

    try:
        resp = client.invoke_inference_service(
            AlgoType='object-detection',
            ServiceName='objectDetection',
            ContentType='image/jpeg',
            Body=content
        )
    except ml.GreengrassInferenceException as e:
        logging.info('inference exception {}("{}")'.format(e.__class__.__name__, e))
        return
    except ml.GreengrassDependencyException as e:
        logging.info('dependency exception {}("{}")'.format(e.__class__.__name__, e))
        return

    logging.info('resp: {}'.format(resp))
    predictions = resp['Body'].read().decode("utf-8")
    logging.info('predictions: {}'.format(predictions))
    predictions = eval(predictions)

    # Perform business logic that relies on the predictions.

    # Schedule the infer() function to run again in ten second.
    Timer(10, infer).start()
```

```
    return

infer()

def function_handler(event, context):
    return
```

A `invoke_inference_service` função no AWS IoT Greengrass Machine Learning SDK aceita os seguintes argumentos.

Argumento	Description
`AlgoType`	O nome do tipo de algoritmo a ser usado para inferência. No momento, só há compatibilidade com `object-detection` . Obrigatório: `true` Digite: `string` Valores válidos: `object-detection`
`ServiceName`	O nome do serviço de inferência local. Use o nome que você especificou para o parâmetro `LocalInferenceServiceName` quando você configurou o conector. Obrigatório: `true` Digite: `string`
`ContentType`	O tipo de mime da imagem de entrada. Obrigatório: `true` Digite: `string` Valores válidos: `image/jpeg, image/png`
`Body`	O conteúdo do arquivo de imagem de entrada.

Argumento	Description
	Obrigatório: `true` Digite: `binary`

Instalando as dependências de tempo de execução do Neo deep learning no AWS IoT Greengrass core

Os conectores ML Object Detection são fornecidos com o SageMaker AI Neo deep learning runtime (DLR). Os conectores usam o runtime para atender ao modelo de ML. Para usar esses conectores, você deve instalar as dependências do DLR no dispositivo de núcleo.

Antes de instalar as dependências do DLR, certifique-se de que as bibliotecas do sistema necessárias (com as versões mínimas especificadas) estejam presentes no dispositivo.

NVIDIA Jetson TX2

1. Instale CUDA Toolkit 9.0 e cuDNN 7.0. Você pode seguir as instruções em the section called "Configurar outros dispositivos" no tutorial Conceitos básicos.
2. Habilite repositórios universe para que o conector possa instalar softwares livres mantidos pela comunidade. Para obter mais informações, consulte Repositories/Ubuntua documentação do Ubuntu.

   a. Abra o arquivo `/etc/apt/sources.list`.

   b. Certifique-se de que as seguintes linhas permaneçam sem comentário.

   ```
   deb http://ports.ubuntu.com/ubuntu-ports/ xenial universe
   deb-src http://ports.ubuntu.com/ubuntu-ports/ xenial universe
   deb http://ports.ubuntu.com/ubuntu-ports/ xenial-updates universe
   deb-src http://ports.ubuntu.com/ubuntu-ports/ xenial-updates universe
   ```

3. Salve uma cópia do script de instalação a seguir em um arquivo denominado `nvidiajtx2.sh` no dispositivo de núcleo.

   ```
   #!/bin/bash
   set -e

   echo "Installing dependencies on the system..."
   ```

```
echo 'Assuming that universe repos are enabled and checking dependencies...'
apt-get -y update
apt-get -y dist-upgrade
apt-get install -y liblapack3 libopenblas-dev liblapack-dev libatlas-base-dev
apt-get install -y python3.7 python3.7-dev

python3.7 -m pip install --upgrade pip
python3.7 -m pip install numpy==1.15.0
python3.7 -m pip install opencv-python || echo 'Error: Unable to install OpenCV
 with pip on this platform. Try building the latest OpenCV from source (https://
github.com/opencv/opencv).'

echo 'Dependency installation/upgrade complete.'
```

> **Note**
>
> Se o OpenCV não for instalado com êxito usando esse script, você poderá tentar compilar a partir do código-fonte. Para obter mais informações, consulte Instalação no Linux na documentação do OpenCV ou consulte outros recursos online da sua plataforma.

4. A partir do diretório em que você salvou o arquivo, execute o seguinte comando:

```
sudo nvidiajtx2.sh
```

x86_64 (Ubuntu or Amazon Linux)

1. Salve uma cópia do script de instalação a seguir em um arquivo denominado `x86_64.sh` no dispositivo de núcleo.

```
#!/bin/bash
set -e

echo "Installing dependencies on the system..."

release=$(awk -F= '/^NAME/{print $2}' /etc/os-release)

if [ "$release" == '"Ubuntu"' ]; then
  # Ubuntu. Supports EC2 and DeepLens. DeepLens has all the dependencies
 installed, so
```

```
  # this is mostly to prepare dependencies on Ubuntu EC2 instance.
  apt-get -y update
  apt-get -y dist-upgrade

  apt-get install -y libgfortran3 libsm6 libxext6 libxrender1
  apt-get install -y python3.7 python3.7-dev
elif [ "$release" == '"Amazon Linux"' ]; then
  # Amazon Linux. Expect python to be installed already
  yum -y update
  yum -y upgrade

  yum install -y compat-gcc-48-libgfortran libSM libXrender libXext
else
  echo "OS Release not supported: $release"
  exit 1
fi

python3.7 -m pip install --upgrade pip
python3.7 -m pip install numpy==1.15.0
python3.7 -m pip install opencv-python || echo 'Error: Unable to install OpenCV
 with pip on this platform. Try building the latest OpenCV from source (https://
github.com/opencv/opencv).'

echo 'Dependency installation/upgrade complete.'
```

> **Note**
>
> Se o OpenCV não for instalado com êxito usando esse script, você poderá tentar compilar a partir do código-fonte. Para obter mais informações, consulte Instalação no Linux na documentação do OpenCV ou consulte outros recursos online da sua plataforma.

2. A partir do diretório em que você salvou o arquivo, execute o seguinte comando:

```
sudo x86_64.sh
```

ARMv7 (Raspberry Pi)

1. Salve uma cópia do script de instalação a seguir em um arquivo denominado `armv7l.sh` no dispositivo de núcleo.

```
#!/bin/bash
set -e

echo "Installing dependencies on the system..."

apt-get update
apt-get -y upgrade

apt-get install -y liblapack3 libopenblas-dev liblapack-dev
apt-get install -y python3.7 python3.7-dev

python3.7 -m pip install --upgrade pip
python3.7 -m pip install numpy==1.15.0
python3.7 -m pip install opencv-python || echo 'Error: Unable to install OpenCV
 with pip on this platform. Try building the latest OpenCV from source (https://
github.com/opencv/opencv).'

echo 'Dependency installation/upgrade complete.'
```

> **Note**
>
> Se o OpenCV não for instalado com êxito usando esse script, você poderá tentar compilar a partir do código-fonte. Para obter mais informações, consulte Instalação no Linux na documentação do OpenCV ou consulte outros recursos online da sua plataforma.

2. A partir do diretório em que você salvou o arquivo, execute o seguinte comando:

```
sudo bash armv7l.sh
```

> **Note**
>
> Em um Raspberry Pi, usar o `pip` para instalar dependências de machine learning é uma operação com uso intensivo de memória que pode fazer com que o dispositivo fique sem memória e deixe de responder. Como alternativa, você pode aumentar temporariamente o tamanho da permuta. Em `/etc/dphys-swapfile`, aumente o valor da variável `CONF_SWAPSIZE` e, em seguida, execute o seguinte comando para reiniciar `dphys-swapfile`.

```
/etc/init.d/dphys-swapfile restart
```

Registro em log e solução de problemas

Dependendo das configurações do seu grupo, os registros de eventos e erros são gravados CloudWatch nos Registros, no sistema de arquivos local ou em ambos. Os logs desse conector usam o prefixo `LocalInferenceServiceName`. Se o conector se comportar de forma inesperada, verifique os logs do conector. Normalmente, eles contêm informações úteis de depuração, como uma dependência de biblioteca de ML ausente ou a causa de uma falha de startup do conector.

Se o AWS IoT Greengrass grupo estiver configurado para gravar registros locais, o conector grava arquivos de log em*greengrass-root*`/ggc/var/log/user/`*region*`/aws/`. Para obter mais informações sobre o registro em log do Greengrass, consulte the section called “Monitoramento com logs do AWS IoT Greengrass ”.

Use as informações a seguir para ajudar a solucionar os problemas com os conectores de detecção de objetos do ML.

Bibliotecas do sistema necessárias

As guias a seguir listam as bibliotecas do sistema necessárias para cada conector de detecção de objetos do ML.

ML Object Detection Aarch64 JTX2

Biblioteca	Versão mínima
ld-linux-aarch64.so.1	GLIBC_2.17
libc.so.6	GLIBC_2.17
libcublas.so.9.0	não aplicável
libcudart.so.9.0	não aplicável
libcudnn.so.7	não aplicável
libcufft.so.9.0	não aplicável

Biblioteca	Versão mínima
libcurand.so.9.0	não aplicável
libcusolver.so.9.0	não aplicável
libgcc_s.so.1	GCC_4.2.0
libgomp.so.1	GOMP_4.0, OMP_1.0
libm.so.6	GLIBC_2.23
libnvinfer.so.4	não aplicável
libnvrm_gpu.so	não aplicável
libnvrm.so	não aplicável
libnvidia-fatbinaryloader.so.28.2.1	não aplicável
libnvos.so	não aplicável
libpthread.so.0	GLIBC_2.17
librt.so.1	GLIBC_2.17
libstdc++.so.6	GLIBCXX_3.4.21, CXXABI_1.3.8

ML Object Detection x86_64

Biblioteca	Versão mínima
ld-linux-x86-64.so.2	GCC_4.0.0
libc.so.6	GLIBC_2.4
libgfortran.so.3	GFORTRAN_1.0
libm.so.6	GLIBC_2.23

Biblioteca	Versão mínima
libpthread.so.0	GLIBC_2.2.5
librt.so.1	GLIBC_2.2.5
libstdc++.so.6	CXXABI_1.3.8, GLIBCXX_3.4.21

ML Object Detection ARMv7

Biblioteca	Versão mínima
ld-linux-armhf.so.3	GLIBC_2.4
libc.so.6	GLIBC_2.7
libgcc_s.so.1	GCC_4.0.0
libgfortran.so.3	GFORTRAN_1.0
libm.so.6	GLIBC_2.4
libpthread.so.0	GLIBC_2.4
librt.so.1	GLIBC_2.4
libstdc++.so.6	CXXABI_1.3.8, CXXABI_ARM_1.3.3, GLIBCXX_3.4.20

Problemas

Sintomas	Solução
Em um Raspberry Pi, a mensagem de erro a seguir será registrada em log e você não está usando a câmera: `Failed to initialize libdc1394`	Execute o comando a seguir para desabilitar o driver: `sudo ln /dev/null /dev/raw1394`

Sintomas	Solução
	Essa operação é efêmera. O symblink desaparece após a reinicialização. Consulte o manual de distribuição do seu sistema operacional para saber como criar o link automaticamente na reinicialização.

Licenças

Os conectores ML Object Detection incluem os seguintes conectores de terceiros software/licensing:

- AWS SDK para 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

- Deep Learning Runtime/Licença Apache 2.0
- six/MIT

Esse conector é liberado de acordo com o Contrato de licença de software do Greengrass Core.

Consulte também

- Integrar a serviços e protocolos usando conectores
- the section called “Conceitos básicos de conectores (console)”
- the section called “Conceitos básicos de conectores (CLI)”
- Executar a inferência de machine learning
- Algoritmo de detecção de objetos no Amazon SageMaker AI Developer Guide

Modbus-RTU Conector de adaptador de protocolo

O conector do adaptador de Modbus-RTU protocolo pesquisa informações dos dispositivos Modbus RTU que estão no grupo. AWS IoT Greengrass

Esse conector recebe parâmetros para uma solicitação Modbus RTU de uma função do Lambda definida pelo usuário. Ele envia a solicitação correspondente e, em seguida, publica a resposta do dispositivo de destino como uma mensagem MQTT.

Esse conector tem as seguintes versões.

Versão	ARN
3	`arn:aws:greengrass:region::/connectors/ModbusRTUProtocolAdapter/versions/3`
2	`arn:aws:greengrass:region::/connectors/ModbusRTUProtocolAdapter/versions/2`
1	`arn:aws:greengrass:region::/connectors/ModbusRTUProtocolAdapter/versions/1`

Para obter informações sobre alterações de versão, consulte o Changelog.

Requisitos

Esse conector tem os seguintes requisitos:

Version 3

- AWS IoT Greengrass Software principal v1.9.3 ou posterior.
- Python, versão 3.7 ou 3.8, instalado no dispositivo de núcleo e adicionado à variável de ambiente PATH.

> **Note**
>
> Para usar o Python 3.8, execute o comando a seguir para criar um symblink da pasta de instalação padrão do Python 3.7 para os binários instalados do Python 3.8.
>
> ```
> sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
> ```
>
> Isso configura seu dispositivo para atender ao requisito Python para AWS IoT Greengrass.

- Uma conexão física entre o AWS IoT Greengrass núcleo e os dispositivos Modbus. O núcleo deve estar fisicamente conectado à rede Modbus RTU por meio de uma porta serial, por exemplo, uma porta USB.
- Um recurso do dispositivo local no grupo do Greengrass que aponta para a porta serial Modbus física.
- Uma função do Lambda definida pelo usuário que envia parâmetros de solicitação Modbus RTU para esse conector. Os parâmetros de solicitação devem estar em conformidade com os padrões esperados e incluir os IDs e os endereços dos dispositivos de destino na rede Modbus RTU. Para obter mais informações, consulte the section called “Dados de entrada”.

Versions 1 - 2

- AWS IoT Greengrass Software principal v1.7 ou posterior.
- Python versão 2.7 instalado no dispositivo de núcleo e adicionado à variável de ambiente PATH.
- Uma conexão física entre o AWS IoT Greengrass núcleo e os dispositivos Modbus. O núcleo deve estar fisicamente conectado à rede Modbus RTU por meio de uma porta serial, por exemplo, uma porta USB.
- Um recurso do dispositivo local no grupo do Greengrass que aponta para a porta serial Modbus física.
- Uma função do Lambda definida pelo usuário que envia parâmetros de solicitação Modbus RTU para esse conector. Os parâmetros de solicitação devem estar em conformidade com os padrões esperados e incluir os IDs e os endereços dos dispositivos de destino na rede Modbus RTU. Para obter mais informações, consulte the section called “Dados de entrada”.

Parâmetros do conector

Esse conector oferece suporte aos seguintes parâmetros:

`ModbusSerialPort-ResourceId`

O ID do recurso de dispositivo local que representa a porta serial Modbus física.

> **Note**
>
> Esse conector recebe acesso de leitura e gravação ao recurso.

Nome de exibição no AWS IoT console: recurso de porta serial Modbus

Obrigatório: `true`

Digite: `string`

Padrão válido: `.+`

`ModbusSerialPort`

O caminho absoluto para a porta serial Modbus física no dispositivo. Este é o caminho de origem especificado para o recurso de dispositivo local Modbus.

Nome de exibição no AWS IoT console: caminho de origem do recurso da porta serial Modbus

Obrigatório: `true`

Digite: `string`

Padrão válido: `.+`

Exemplo de criação de conector (AWS CLI)

O comando CLI a seguir cria um `ConnectorDefinition` com uma versão inicial que contém o conector do adaptador de Modbus-RTU protocolo.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
```

```
        {
            "Id": "MyModbusRTUProtocolAdapterConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/
ModbusRTUProtocolAdapter/versions/3",
            "Parameters": {
                "ModbusSerialPort-ResourceId": "MyLocalModbusSerialPort",
                "ModbusSerialPort": "/path-to-port"
            }
        }
    ]
}'
```

> **Note**
>
> A função do Lambda nesse conector tem um ciclo de vida longo.

No AWS IoT Greengrass console, você pode adicionar um conector na página Conectores do grupo. Para obter mais informações, consulte the section called “Conceitos básicos de conectores (console)”.

> **Note**
>
> Depois de implantar o conector do adaptador de Modbus-RTU protocolo, você pode usá-lo AWS IoT Things Graph para orquestrar interações entre dispositivos em seu grupo. Para obter mais informações, consulte Modbus no Guia do usuário do AWS IoT Things Graph .

Dados de entrada

Esse conector aceita parâmetros de solicitação Modbus RTU de uma função do Lambda definida pelo usuário em um tópico MQTT. As mensagens de entrada devem estar no formato JSON.

Filtro de tópico na assinatura

`modbus/adapter/request`

Propriedades de mensagens

A mensagem de solicitação varia de acordo com o tipo de solicitação Modbus RTU que ela representa. As propriedades a seguir são necessárias para todas as solicitações:

- No objeto `request`:
  - `operation`. O nome da operação a ser executada. Por exemplo, especifique `"operation": "ReadCoilsRequest"` para bobinas de leitura. Esse valor deve ser uma string Unicode. Para ver as operações suportadas, consulte the section called “Respostas e solicitações Modbus RTU”.
  - `device`. O dispositivo de destino da solicitação. O valor deve estar entre `0 - 247`.
- A propriedade `id`. Um ID para a solicitação. Esse valor é usado para a eliminação de duplicação de dados e é retornado da forma que se encontra na propriedade `id` de todas as respostas, incluindo respostas de erro. Esse valor deve ser uma string Unicode.

> **Note**
>
> Se sua solicitação incluir um campo de endereço, você deve especificar o valor como um inteiro. Por exemplo, ."`address": 1`

Os outros parâmetros a ser incluídos na solicitação dependem da operação. Todos os parâmetros de solicitação são necessários, exceto o CRC, que é tratado separadamente. Para obter exemplos, consulte the section called “Exemplos de solicitações e respostas”.

Exemplo de entrada: solicitação de bobinas de leitura

```
{
    "request": {
        "operation": "ReadCoilsRequest",
     "device": 1,
     "address": 1,
     "count": 1
    },
    "id": "TestRequest"
}
```

Dados de saída

Esse conector publica respostas em solicitações Modbus RTU de entrada.

Filtro de tópico na assinatura

```
modbus/adapter/response
```

Propriedades de mensagens

O formato da mensagem de resposta varia de acordo com a solicitação correspondente e o status da resposta. Para obter exemplos, consulte the section called “Exemplos de solicitações e respostas”.

> **Note**
>
> Uma resposta para uma operação de gravação é simplesmente um eco da solicitação. Embora nenhuma informação significativa seja retornada para respostas de gravação, é uma boa prática verificar o status da resposta.

Cada resposta inclui as seguintes propriedades:

- No objeto `response`:
  - `status`. O status da solicitação. O status pode ser um dos valores a seguir:
    - `Success`. A solicitação era válida, enviada para a rede Modbus RTU, e uma resposta foi retornada.
    - `Exception`. A solicitação era válida, enviada para a rede Modbus RTU, e uma resposta de exceção foi retornada. Para obter mais informações, consulte the section called “Status da resposta: Exceção”.
    - `No Response`. A solicitação era inválida, e o conector capturou o erro antes que a solicitação fosse enviada por meio da rede Modbus RTU. Para obter mais informações, consulte the section called “Status de resposta: Sem resposta”.
  - `device`. O dispositivo para o qual a solicitação foi enviada.
  - `operation`. O tipo de solicitação que foi enviada.
  - `payload`. O conteúdo da resposta que foi retornada. Se `status` for `No Response`, esse objeto conterá apenas uma propriedade `error` com a descrição do erro (por exemplo, `"error": "[Input/Output] No Response received from the remote unit"`).
- A propriedade `id`. O ID da solicitação, usado para eliminação de duplicação de dados.

Exemplo de resultado: sucesso

```
{
    "response" : {
        "status" : "success",
        "device": 1,
```

```
     "operation": "ReadCoilsRequest",
     "payload": {
         "function_code": 1,
         "bits": [1]
     }
     },
     "id" : "TestRequest"
}
```

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" : "TestRequest"
}
```

Para obter mais exemplos, consulte the section called “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

Nome da operação na solicitação	Código da função em resposta
ReadHoldingRegistersRequest	03
ReadInputRegistersRequest	04
WriteSingleCoilRequest	05
WriteSingleRegisterRequest	06
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"
}
```

Ler 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"
}
```

Registros de leitura em espera

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"
}
```

Registros de entrada de leitura

Exemplo de solicitação:

```
{
    "request": {
        "operation": "ReadInputRegistersRequest",
     "device": 1,
     "address": 1,
     "value": 1
    },
    "id": "TestRequest"
}
```

Bobina de gravação única

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"
```

Registro de gravação único

Exemplo de solicitação:

```
{
    "request": {
        "operation": "WriteSingleRegisterRequest",
     "device": 1,
     "address": 1,
     "value": 1
    },
    "id": "TestRequest"
}
```

Várias bobinas de gravação

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"
}
```

Vários registros de gravação

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 Mask

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"
}
```

Vários registros de leitura/gravação

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**
>
> Os registros retornados nessa resposta são os registos lidos.

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 <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"
}
```

Exemplo de uso

Use as seguintes etapas de alto nível para configurar um exemplo de função do Lambda Python 3.7 que pode ser usado para testar o conector.

> **Note**
>
> - Se você usar outros tempos de execução do Python, poderá criar um link simbólico para Python3.x o Python 3.7.
> - Os tópicos Conceitos básicos de conectores (console) e Conceitos básicos de conectores (CLI) contêm etapas detalhadas que mostram como configurar e implantar um exemplo do conector de notificações do Twilio.

1. Certifique-se de cumprir os requisitos para o conector.

2. Crie e publique uma função do Lambda que envie dados de entrada para o conector.

   Salve o código de exemplo como arquivo PY. Baixe e descompacte o SDK do AWS IoT Greengrass Core para Python. Crie então um pacote zip que contenha o arquivo PY e a pasta `greengrasssdk` no nível raiz. Este pacote zip é o pacote de implantação que você transfere por upload para o AWS Lambda.

   Depois de criar a função do Lambda Python 3.7, publique uma versão de função e crie um alias.

3. Configure o grupo do Greengrass.

   a. Adicione a função do Lambda pelo seu alias (recomendado). Configure o ciclo de vida do Lambda como de longa duração (ou `"Pinned": true` na CLI).

   b. Adicione o recurso de dispositivo local necessário e conceda read/write acesso à função Lambda.

   c. Adicione o conector e configure seus parâmetros.

   d. Adicione assinaturas que permitam que o conector receba dados de entrada e envie dados de saída em filtros de tópico compatíveis.

      - Defina a função do Lambda como origem, o conector como destino e use um filtro de tópico de entrada compatível.
      - Defina o conector como origem, o AWS IoT Core como destino, e use um filtro de tópico de saída compatível. Você usa essa assinatura para visualizar mensagens de status no AWS IoT console.

4. Implante o grupo.

5. No AWS IoT console, na página Teste, inscreva-se no tópico de dados de saída para ver as mensagens de status do conector. A função de exemplo do Lambda é de longa duração e começa a enviar mensagens imediatamente após o grupo ser implantado.

   Ao finalizar o teste, você pode definir o ciclo de vida do Lambda como sob demanda (ou `"Pinned": false` na CLI) e implantar o grupo. Isso impede o envio de mensagens pela função.

Exemplo

O exemplo a seguir da função do Lambda envia uma mensagem de entrada para o conector.

```
import greengrasssdk
```

```
import json

TOPIC_REQUEST = 'modbus/adapter/request'

# Creating a greengrass core sdk client
iot_client = greengrasssdk.client('iot-data')

def create_read_coils_request():
 request = {
  "request": {
   "operation": "ReadCoilsRequest",
   "device": 1,
   "address": 1,
   "count": 1
  },
  "id": "TestRequest"
 }
 return request

def publish_basic_request():
 iot_client.publish(payload=json.dumps(create_read_coils_request()),
 topic=TOPIC_REQUEST)

publish_basic_request()

def lambda_handler(event, context):
 return
```

Licenças

O conector do adaptador de Modbus-RTU protocolo inclui os seguintes terceiros software/licensing:

- pymodbus/BSD
- pyserial/BSD

Esse conector é liberado de acordo com o Contrato de licença de software do Greengrass Core.

Changelog

A tabela a seguir descreve as alterações em cada versão do conector.

Versão	Alterações
3	Atualização do runtime do Lambda para Python 3.7, o que altera o requisito de runtime.
2	ARN do conector atualizado para Região da AWS suporte. Melhoria do registro em log de erros.
1	Versão inicial.

Um grupo do Greengrass só pode conter uma versão do conector por vez. Para obter informações sobre como fazer upgrade de uma versão do conector, consulte the section called “Atualizar a versões do conector”.

Consulte também

- Integrar a serviços e protocolos usando conectores
- the section called “Conceitos básicos de conectores (console)”
- the section called “Conceitos básicos de conectores (CLI)”

Modbus-TCP Conector de adaptador de protocolo

O conector do adaptador de Modbus-TCP protocolo coleta dados de dispositivos locais por meio do protocolo ModbusTCP e os publica nos fluxos selecionados. `StreamManager`

Você também pode usar esse conector com o conector de IoT e seu SiteWise gateway de IoT. SiteWise Seu gateway deve fornecer a configuração do conector. Para obter mais informações, consulte Configurar uma fonte Modbus TCP no guia do usuário de SiteWise IoT.

> **Note**
>
> Esse conector é executado no modo Sem isolamento de contêiner, então você pode implantá-lo em um AWS IoT Greengrass grupo executado em um contêiner Docker.

Esse conector tem as seguintes versões.

Versão	ARN
3	`arn:aws:greengrass:region::/connectors/ModbusTCPConnector/versions/3`
2	`arn:aws:greengrass:region::/connectors/ModbusTCPConnector/versions/2`
1	`arn:aws:greengrass:region::/connectors/ModbusTCPConnector/versions/1`

Para obter informações sobre alterações de versão, consulte o Changelog.

Requisitos

Esse conector tem os seguintes requisitos:

Version 1 - 3

- AWS IoT Greengrass Software principal v1.10.2 ou posterior.
- Gerenciador de transmissões ativado no AWS IoT Greengrass grupo.
- Java 8 instalado no dispositivo de núcleo e adicionado à variável de ambiente `PATH`.

> **Note**
>
> Esse conector só está disponível nas seguintes regiões:
>
> - ap-southeast-1
> - ap-southeast-2
> - eu-central-1
> - eu-west-1
> - us-east-1
> - us-west-2

- cn-north-1

Parâmetros do conector

Esse conector oferece suporte aos seguintes parâmetros:

`LocalStoragePath`

O diretório no AWS IoT Greengrass host no qual o SiteWise conector de IoT pode gravar dados persistentes. O diretório padrão é `/var/sitewise`.

Nome de exibição no AWS IoT console: caminho de armazenamento local

Obrigatório: `false`

Digite: `string`

Padrão válido: `^\s*$|\/.`

`MaximumBufferSize`

O tamanho máximo em GB para uso do SiteWise disco de IoT. O tamanho padrão é 10 GB.

Nome de exibição no AWS IoT console: tamanho máximo do buffer de disco

Obrigatório: `false`

Digite: `string`

Padrão válido: `^\s*$|[0-9]+`

`CapabilityConfiguration`

O conjunto de configurações do coletor Modbus TCP do qual o conector coleta dados e ao qual ele se conecta.

Nome de exibição no AWS IoT console: CapabilityConfiguration

Obrigatório: `false`

Tipo: uma string JSON bem-formada que define o conjunto de configurações de feedback compatíveis.

Veja a seguir um exemplo de um `CapabilityConfiguration`:

```
{
    "sources": [
        {
            "type": "ModBusTCPSource",
            "name": "SourceName1",
            "measurementDataStreamPrefix": "SourceName1_Prefix",
            "destination": {
                "type": "StreamManager",
                "streamName": "SiteWise_Stream_1",
                "streamBufferSize": 8
            },
            "endpoint": {
                "ipAddress": "127.0.0.1",
                "port": 8081,
                "unitId": 1
            },
            "propertyGroups": [
                {
                    "name": "GroupName",
                    "tagPathDefinitions": [
                        {
                            "type": "ModBusTCPAddress",
                            "tag": "TT-001",
                            "address": "30001",
                            "size": 2,
                            "srcDataType": "float",
                            "transformation": "byteWordSwap",
                            "dstDataType": "double"
                        }
                    ],
                    "scanMode": {
                        "type": "POLL",
                        "rate": 100
                    }
                }
            ]
        }
    ]
}
```

Exemplo de criação de conector (AWS CLI)

O comando CLI a seguir cria um `ConnectorDefinition` com uma versão inicial que contém o conector do adaptador de Modbus-TCP protocolo.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '
{
    "Connectors": [
        {
            "Id": "MyModbusTCPConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/ModbusTCP/
versions/3",
            "Parameters": {
                "capability_configuration": "{\"version\":1,\"namespace\":
\"iotsitewise:modbuscollector:1\",\"configuration\":\"{\"sources\":[{\"type
\":\"ModBusTCPSource\",\"name\":\"SourceName1\",\"measurementDataStreamPrefix
\":\"\",\"endpoint\":{\"ipAddress\":\"127.0.0.1\",\"port\":8081,\"unitId\":1},
\"propertyGroups\":[{\"name\":\"PropertyGroupName\",\"tagPathDefinitions\":[{\"type
\":\"ModBusTCPAddress\",\"tag\":\"TT-001\",\"address\":\"30001\",\"size\":2,
\"srcDataType\":\"hexdump\",\"transformation\":\"noSwap\",\"dstDataType\":\"string
\"}],\"scanMode\":{\"rate\":200,\"type\":\"POLL\"}}],\"destination\":{\"type\":
\"StreamManager\",\"streamName\":\"SiteWise_Stream\",\"streamBufferSize\":10},
\"minimumInterRequestDuration\":200}]}\"}"
            }
        }
    ]
}'
```

> **Note**
>
> A função do Lambda nesse conector tem um ciclo de vida longo.

Dados de entrada

Esse conector não aceita mensagens MQTT como dados de entrada.

Dados de saída

Esse conector publica dados para `StreamManager`: Você deve configurar o fluxo de mensagens de destino. As mensagens de saída têm a seguinte estrutura:

```
{
    "alias": "string",
    "messages": [
        {
            "name": "string",
            "value": boolean|double|integer|string,
            "timestamp": number,
            "quality": "string"
        }
    ]
}
```

Licenças

O conector do adaptador de Modbus-TCP protocolo inclui os seguintes terceiros software/licensing:

- Digital Petri Modbus

Esse conector é liberado de acordo com o Contrato de licença de software do Greengrass Core.

Changelog

A tabela a seguir descreve as alterações em cada versão do conector.

Versão	Alterações	Data
3 (recomendado)	Esta versão contém correções de erros.	22 de dezembro de 2021
2	Foi adicionado suporte para strings de caracteres de origem codificados em ASCII, UTF8 e ISO8859.	24 de maio de 2021
1	Versão inicial.	15 de dezembro de 2020

Um grupo do Greengrass só pode conter uma versão do conector por vez. Para obter informações sobre como fazer upgrade de uma versão do conector, consulte the section called “Atualizar a versões do conector”.

Consulte também

- Integrar a serviços e protocolos usando conectores
- the section called “Conceitos básicos de conectores (console)”
- the section called “Conceitos básicos de conectores (CLI)”

Conector Raspberry Pi GPIO

> **Warning**
>
> Esse conector passou para a fase de vida útil estendida e AWS IoT Greengrass não lançará atualizações que forneçam atributos, aprimoramentos para atributos existentes, patches de segurança ou correções de erros. Para obter mais informações, consulte AWS IoT Greengrass Version 1 política de manutenção.

O conector Raspberry Pi GPIO controla os pinos de uso geral input/output (GPIO) em um dispositivo principal Raspberry Pi.

Esse conector consulta pins de entrada em um intervalo especificado e publica alterações de estado em tópicos MQTT. Ele também aceita solicitações de leitura e gravação como mensagens MQTT de funções do Lambda definidas pelo usuário. As solicitações de gravação são usadas para definir o pin de alta ou baixa tensão.

O conector fornece parâmetros que você usa para designar pins de entrada e saída. Esse comportamento é configurado antes da implantação do grupo. Ele não pode ser alterado em tempo de execução.

- Pins de entrada podem ser usados para receber dados de dispositivos periféricos.
- Pins de entrada podem ser usados para controlar periféricos ou enviar dados para periféricos.

Você pode usar esse conector em muitos cenários, como:

- Controlar luzes LED verde, amarela e vermelha para um semáforo.
- Controlar um ventilador (anexado a uma retransmissão elétrica) com base nos dados de um sensor de umidade.

- Avisar funcionários de uma loja quando os clientes pressionarem um botão.
- Usar um interruptor inteligente para controlar outros dispositivos da IoT.

> **Note**
>
> Esse conector não é adequado para aplicativos que têm requisitos em tempo real. Eventos de curta duração podem ser perdidos.

Esse conector tem as seguintes versões.

Versão	ARN
3	`arn:aws:greengrass:region::/connectors/RaspberryPiGPIO/versions/3`
2	`arn:aws:greengrass:region::/connectors/RaspberryPiGPIO/versions/2`
1	`arn:aws:greengrass:region::/connectors/RaspberryPiGPIO/versions/1`

Para obter informações sobre alterações de versão, consulte o Changelog.

Requisitos

Esse conector tem os seguintes requisitos:

Version 3

- AWS IoT Greengrass Software principal v1.9.3 ou posterior.
- Python versão 3.7 instalado no dispositivo de núcleo e adicionado à variável de ambiente PATH.

- Raspberry Pi 4 Modelo B ou Raspberry Pi 3 Modelo B/B +. Você deve saber a sequência de pin do seu Raspberry Pi. Para obter mais informações, consulte the section called “Sequência de pin GPIO”.
- Um recurso do dispositivo local no grupo do Greengrass que aponta para `/dev/gpiomem` no Raspberry Pi. Se você criar o recurso no console, deverá selecionar a opção Adicionar automaticamente permissões de grupo de SO do grupo Linux que possui o recurso. Na API, defina a propriedade `GroupOwnerSetting.AutoAddGroupOwner` como `true`.
- O RPi.GPIOmódulo instalado no Raspberry Pi. No Raspbian, esse módulo é instalado por padrão. Você pode usar o comando a seguir para instalá-lo novamente:

  ```
  sudo pip install RPi.GPIO
  ```

Versions 1 - 2

- AWS IoT Greengrass Software principal v1.7 ou posterior.
- Python versão 2.7 instalado no dispositivo de núcleo e adicionado à variável de ambiente PATH.
- Raspberry Pi 4 Modelo B ou Raspberry Pi 3 Modelo B/B +. Você deve saber a sequência de pin do seu Raspberry Pi. Para obter mais informações, consulte the section called “Sequência de pin GPIO”.
- Um recurso do dispositivo local no grupo do Greengrass que aponta para `/dev/gpiomem` no Raspberry Pi. Se você criar o recurso no console, deverá selecionar a opção Adicionar automaticamente permissões de grupo de SO do grupo Linux que possui o recurso. Na API, defina a propriedade `GroupOwnerSetting.AutoAddGroupOwner` como `true`.
- O RPi.GPIOmódulo instalado no Raspberry Pi. No Raspbian, esse módulo é instalado por padrão. Você pode usar o comando a seguir para instalá-lo novamente:

  ```
  sudo pip install RPi.GPIO
  ```

Sequência de pin GPIO

O conector GPIO do Raspberry Pi faz referência aos pins GPIO pelo esquema de numeração do System on Chip (SoC) subjacente, não pelo layout físico dos pins GPIO. A ordem física dos pins pode variar nas versões do Raspberry Pi. Para obter mais informações, consulte GPIO na documentação do Raspberry Pi.

O conector não pode validar que os pins de entrada e saída que você configura mapeiem corretamente ao hardware subjacente do seu Raspberry Pi. Se a configuração do pin for inválida, o conector retornará um erro de tempo de execução ao tentar iniciar no dispositivo. Para resolver esse problema, reconfigure o conector e reimplante.

> **Note**
>
> Certifique-se de que periféricos para pins GPIO estejam corretamente cabeados para evitar danos de componente.

Parâmetros do conector

Esse conector oferece os seguintes parâmetros:

`InputGpios`

Uma lista separada por vírgulas de números de pins GPIO para configurar como entradas. Opcionalmente, acrescente `U` para definir um resistor de pin ou `D` para definir a resistência suspensa. Exemplo: `"5,6U,7D"`.

Nome de exibição no AWS IoT console: pinos GPIO de entrada

Obrigatório:`false`. Você deve especificar pins de entrada, de saída ou ambos.

Digite: `string`

Padrão válido: `^$|^[0-9]+[UD]?(,[0-9]+[UD]?)*$`

`InputPollPeriod`

O intervalo (em milissegundos) entre cada operação de sondagem, que verifica pins GPIO de entrada para alterações de estado. O valor mínimo é 1.

Esse valor depende do cenário e do tipo de dispositivos consultados. Por exemplo, o valor `50` deve ser suficientemente rápido para detectar o ato de pressionar um botão.

Nome de exibição no AWS IoT console: Insira o período de pesquisa do GPIO

Obrigatório: `false`

Digite: `string`

Padrão válido: `^$|^[1-9][0-9]*$`

`OutputGpios`

Uma lista separada por vírgulas de números de pins GPIO para configurar como saídas. Opcionalmente, anexe `H` para definir um estado superior (1), ou `L` para definir um estado inferior (0). Exemplo: `"8H,9,27L"`.

Nome de exibição no AWS IoT console: pinos GPIO de saída

Obrigatório:`false`. Você deve especificar pins de entrada, de saída ou ambos.

Digite: `string`

Padrão válido: `^$|^[0-9]+[HL]?(,[0-9]+[HL]?)*$`

`GpioMem-ResourceId`

O ID do recurso do dispositivo local que representa `/dev/gpiomem`.

> **Note**
>
> Esse conector recebe acesso de leitura e gravação ao recurso.

Nome de exibição no AWS IoT console: Recurso para/dev/gpiomem dispositivo

Obrigatório: `true`

Digite: `string`

Padrão válido: `.+`

Exemplo de criação de conector (AWS CLI)

O seguinte comando da CLI cria um `ConnectorDefinition` com uma versão inicial que contém o conector GPIO do Raspberry Pi.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
```

```
            "Id": "MyRaspberryPiGPIOConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/RaspberryPiGPIO/
versions/3",
            "Parameters": {
                "GpioMem-ResourceId": "my-gpio-resource",
                "InputGpios": "5,6U,7D",
                "InputPollPeriod": 50,
                "OutputGpios": "8H,9,27L"
            }
        }
    ]
}'
```

> **Note**
>
> A função do Lambda nesse conector tem um ciclo de vida longo.

No AWS IoT Greengrass console, você pode adicionar um conector na página Conectores do grupo. Para obter mais informações, consulte the section called “Conceitos básicos de conectores (console)”.

Dados de entrada

Esse conector aceita solicitações de leitura ou gravação para pins GPIO em dois tópicos MQTT.

- Solicitações de leitura no tópico `gpio/+/+/read`.
- Solicitações de gravação no tópico `gpio/+/+/write`.

Para publicar esses tópicos, substitua os curingas + pelo nome da coisa principal e pelo número do pin de destino, respectivamente. Por exemplo:

```
gpio/core-thing-name/gpio-number/read
```

> **Note**
>
> Atualmente, quando você cria uma assinatura que usa o conector GPIO do Raspberry Pi, deve especificar um valor para pelo menos um dos curingas + no tópico.

Filtro de tópico: `gpio/+/+/read`

Use esse tópico para orientar o conector para ler o estado do pin GPIO que é especificado no tópico.

O conector publica a resposta no tópico de saída correspondente (por exemplo, `gpio/`*`core-thing-name`*`/`*`gpio-number`*`/state`).

Propriedades de mensagens

Nenhum. As mensagens enviadas a esse tópico são ignoradas.

Filtro de tópico: `gpio/+/+/write`

Use esse tópico para enviar solicitações de gravação para um pin GPIO. Isso orienta o conector a definir o pin GPIO que é especificado no tópico como de baixa ou alta tensão.

- `0` define o pin como baixa tensão.
- `1` define o pin como de alta tensão.

O conector publica a resposta no tópico `/state` de saída correspondente (por exemplo, `gpio/`*`core-thing-name`*`/`*`gpio-number`*`/state`).

Propriedades de mensagens

O valor `0` ou `1`, como número inteiro ou string.

Exemplo de entrada

```
0
```

Dados de saída

Esse conector publica dados em dois tópicos:

- O estado superior ou inferior é alterado no tópico `gpio/+/+/state`.
- Erros no tópico `gpio/+/error`.

Filtro de tópico: `gpio/+/+/state`

Use esse tópico para ouvir alterações de estado nos pins de entrada e nas respostas às solicitações de leitura. O conector retornará a string `"0"` se o pin estiver em estado inferior ou `"1"` se estiver em estado superior.

Ao publicar nesse tópico, o conector substitui os curingas + pelo nome da coisa principal e pelo pin de destino, respectivamente. Por exemplo:

```
gpio/core-thing-name/gpio-number/state
```

> **Note**
>
> Atualmente, quando você cria uma assinatura que usa o conector GPIO do Raspberry Pi, deve especificar um valor para pelo menos um dos curingas + no tópico.

Exemplo de saída

```
0
```

Filtro de tópico: `gpio/+/error`

Use esse tópico para ouvir erros. O conector publica nesse tópico como resultado de uma solicitação inválida (por exemplo, quando uma alteração de estado é solicitada em um pin de entrada).

Ao publicar nesse tópico, o conector substitui o curinga + pelo nome da coisa principal.

Exemplo de saída

```
{
   "topic": "gpio/my-core-thing/22/write",
   "error": "Invalid GPIO operation",
   "long_description": "GPIO 22 is configured as an INPUT GPIO. Write operations
 are not permitted."
 }
```

Exemplo de uso

Use as seguintes etapas de alto nível para configurar um exemplo de função do Lambda Python 3.7 que pode ser usado para testar o conector.

> **Note**
>
> - Se você usar outros tempos de execução do Python, poderá criar um link simbólico para Python3.x o Python 3.7.
> - Os tópicos Conceitos básicos de conectores (console) e Conceitos básicos de conectores (CLI) contêm etapas detalhadas que mostram como configurar e implantar um exemplo do conector de notificações do Twilio.

1. Certifique-se de cumprir os requisitos para o conector.
2. Crie e publique uma função do Lambda que envie dados de entrada para o conector.

   Salve o código de exemplo como arquivo PY. Baixe e descompacte o SDK do AWS IoT Greengrass Core para Python. Crie então um pacote zip que contenha o arquivo PY e a pasta `greengrasssdk` no nível raiz. Este pacote zip é o pacote de implantação que você transfere por upload para o AWS Lambda.

   Depois de criar a função do Lambda Python 3.7, publique uma versão de função e crie um alias.
3. Configure o grupo do Greengrass.

   a. Adicione a função do Lambda pelo seu alias (recomendado). Configure o ciclo de vida do Lambda como de longa duração (ou `"Pinned": true` na CLI).

   b. Adicione o recurso de dispositivo local necessário e conceda read/write acesso à função Lambda.

   c. Adicione o conector e configure seus parâmetros.

   d. Adicione assinaturas que permitam que o conector receba dados de entrada e envie dados de saída em filtros de tópico compatíveis.

      - Defina a função do Lambda como origem, o conector como destino e use um filtro de tópico de entrada compatível.
      - Defina o conector como origem, o AWS IoT Core como destino, e use um filtro de tópico de saída compatível. Você usa essa assinatura para visualizar mensagens de status no AWS IoT console.
4. Implante o grupo.

5. No AWS IoT console, na página Teste, inscreva-se no tópico de dados de saída para ver as mensagens de status do conector. A função de exemplo do Lambda é de longa duração e começa a enviar mensagens imediatamente após o grupo ser implantado.

   Ao finalizar o teste, você pode definir o ciclo de vida do Lambda como sob demanda (ou `"Pinned": false` na CLI) e implantar o grupo. Isso impede o envio de mensagens pela função.

Exemplo

O exemplo a seguir da função do Lambda envia uma mensagem de entrada para o conector. Este exemplo envia solicitações de leitura para um conjunto de pins GPIO de entrada. Ele mostra como construir tópicos usando o nome e o número do pin da coisa principal.

```
import greengrasssdk
import json
import os

iot_client = greengrasssdk.client('iot-data')
INPUT_GPIOS = [6, 17, 22]

thingName = os.environ['AWS_IOT_THING_NAME']

def get_read_topic(gpio_num):
    return '/'.join(['gpio', thingName, str(gpio_num), 'read'])

def get_write_topic(gpio_num):
    return '/'.join(['gpio', thingName, str(gpio_num), 'write'])

def send_message_to_connector(topic, message=''):
    iot_client.publish(topic=topic, payload=str(message))

def set_gpio_state(gpio, state):
    send_message_to_connector(get_write_topic(gpio), str(state))

def read_gpio_state(gpio):
    send_message_to_connector(get_read_topic(gpio))

def publish_basic_message():
    for i in INPUT_GPIOS:
     read_gpio_state(i)
```

```
publish_basic_message()

def lambda_handler(event, context):
    return
```

Licenças

O conector Raspberry Pi GPIO; inclui os seguintes componentes de terceiros: software/licensing

- RPi.GPIO/MIT

Esse conector é liberado de acordo com o Contrato de licença de software do Greengrass Core.

Changelog

A tabela a seguir descreve as alterações em cada versão do conector.

Versão	Alterações
3	Atualização do runtime do Lambda para Python 3.7, o que altera o requisito de runtime.
2	ARN do conector atualizado para Região da AWS suporte.
1	Versão inicial.

Um grupo do Greengrass só pode conter uma versão do conector por vez. Para obter informações sobre como fazer upgrade de uma versão do conector, consulte the section called “Atualizar a versões do conector”.

Consulte também

- Integrar a serviços e protocolos usando conectores
- the section called “Conceitos básicos de conectores (console)”
- the section called “Conceitos básicos de conectores (CLI)”
- GPIO na documentação do Raspberry Pi

Conector de fluxo serial

> **Warning**
>
> Esse conector passou para a fase de vida útil estendida e AWS IoT Greengrass não lançará atualizações que forneçam atributos, aprimoramentos para atributos existentes, patches de segurança ou correções de erros. Para obter mais informações, consulte AWS IoT Greengrass Version 1 política de manutenção.

O conector Serial Stream lê e grava em uma porta serial em um dispositivo AWS IoT Greengrass principal.

Esse conector oferece suporte a dois modos de operação:

- Read-On-Demand. Recebe solicitações de leitura e gravação sobre tópicos do MQTT e publica a resposta da operação de leitura ou o status da operação de gravação.
- Polling-Read. Lê a partir da porta serial em intervalos regulares. Esse modo também suporta Read-On-Demand solicitações.

> **Note**
>
> As solicitações de leitura são limitadas a um tamanho máximo de leitura de 63994 bytes. As solicitações de gravação são limitadas um tamanho máximo de dados de 128000 bytes.

Esse conector tem as seguintes versões.

Versão	ARN
3	`arn:aws:greengrass:  region::/connectors/SerialStream/versions/3`
2	`arn:aws:greengrass:  region::/connectors/SerialStream/versions/2`

Versão	ARN
1	`arn:aws:greengrass:  region::/connectors/SerialStream/versions/1`

Para obter informações sobre alterações de versão, consulte o Changelog.

Requisitos

Esse conector tem os seguintes requisitos:

Version 3

- AWS IoT Greengrass Software principal v1.9.3 ou posterior.
- Python, versão 3.7 ou 3.8, instalado no dispositivo de núcleo e adicionado à variável de ambiente PATH.

  > **Note**
  >
  > Para usar o Python 3.8, execute o comando a seguir para criar um symblink da pasta de instalação padrão do Python 3.7 para os binários instalados do Python 3.8.
  >
  > ```
  > sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
  > ```
  >
  > Isso configura seu dispositivo para atender ao requisito Python para AWS IoT Greengrass.

- Um recurso do dispositivo local no grupo do Greengrass que aponta para a porta serial de destino.

  > **Note**
  >
  > Antes de implantar esse conector, recomendamos que você configure a porta serial e verifique se é possível fazer leituras e gravações nela.

Versions 1 - 2

- AWS IoT Greengrass Software principal v1.7 ou posterior.
- Python versão 2.7 instalado no dispositivo de núcleo e adicionado à variável de ambiente PATH.
- Um recurso do dispositivo local no grupo do Greengrass que aponta para a porta serial de destino.

> **Note**
>
> Antes de implantar esse conector, recomendamos que você configure a porta serial e verifique se é possível fazer leituras e gravações nela.

Parâmetros do conector

Esse conector oferece os seguintes parâmetros:

`BaudRate`

A taxa de baud da conexão serial.

Nome de exibição no AWS IoT console: Taxa de transmissão

Obrigatório: `true`

Digite: `string`

Valores válidos: `110, 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 56000, 57600, 115200, 230400`

Padrão válido: `^110$|^300$|^600$|^1200$|^2400$|^4800$|^9600$|^14400$|^19200$|^28800$|^38400$|^56000$|^57600$|^115200$|^230400$`

`Timeout`

O tempo limite (em segundos) para a operação de leitura.

Nome de exibição no AWS IoT console: Timeout

Obrigatório: `true`

Digite: `string`

Valores válidos: `1 - 59`

Padrão válido: `^([1-9]|[1-5][0-9])$`

`SerialPort`

O caminho absoluto para a porta serial física no dispositivo. Esse é o caminho de origem especificado para o recurso de dispositivo local.

Nome de exibição no AWS IoT console: Porta serial

Obrigatório: `true`

Digite: `string`

Padrão válido: `[/a-zA-Z0-9_-]+`

`SerialPort-ResourceId`

O ID do recurso de dispositivo local que representa a porta serial física.

> **Note**
>
> Esse conector recebe acesso de leitura e gravação ao recurso.

Nome de exibição no AWS IoT console: recurso de porta serial

Obrigatório: `true`

Digite: `string`

Padrão válido: `[a-zA-Z0-9_-]+`

`PollingRead`

Define o modo de leitura: Polling-Read ou Read-On-Demand.

- Para o Polling-Read modo, especifique`true`. Nesse modo, as propriedades `PollingInterval`, `PollingReadType` e `PollingReadLength` são necessárias.
- Para o Read-On-Demand modo, especifique`false`. Nesse modo, os valores de tamanho e tipo são especificados na solicitação de leitura.

Nome de exibição no AWS IoT console: Modo de leitura

Obrigatório: `true`

Digite: `string`

Valores válidos: `true, false`

Padrão válido: `^([Tt][Rr][Uu][Ee]|[Ff][Aa][Ll][Ss][Ee])$`

`PollingReadLength`

O tamanho de dados (em bytes) a serem lidos em cada operação de leitura de sondagem. Isso se aplica somente ao usar o Polling-Read modo.

Nome de exibição no AWS IoT console: comprimento de leitura da pesquisa

Obrigatório:`false`. Essa propriedade é necessária quando `PollingRead` é `true`.

Digite: `string`

Padrão válido: `^(|[1-9][0-9]{0,3}|[1-5][0-9]{4}|6[0-2][0-9]{3}|63[0-8][0-9]{2}|639[0-8][0-9]|6399[0-4])$`

`PollingReadInterval`

O intervalo (em segundos) em que a leitura de sondagem ocorre. Isso se aplica somente ao usar o Polling-Read modo.

Nome de exibição no AWS IoT console: intervalo de leitura da pesquisa

Obrigatório:`false`. Essa propriedade é necessária quando `PollingRead` é `true`.

Tipo: `string`

Valores válidos: 1 a 999

Padrão válido: `^(|[1-9]|[1-9][0-9]|[1-9][0-9][0-9])$`

`PollingReadType`

O tipo de dados que o thread de sondagem lê. Isso se aplica somente ao usar o Polling-Read modo.

Nome de exibição no AWS IoT console: Tipo de leitura da pesquisa

Obrigatório:`false`. Essa propriedade é necessária quando `PollingRead` é `true`.

Tipo: `string`

Valores válidos: `ascii, hex`

Padrão válido: `^(|[Aa][Ss][Cc][Ii][Ii]|[Hh][Ee][Xx])$`

`RtsCts`

Indica se o controle de RTS/CTS fluxo deve ser ativado. O valor padrão é `false`. Para obter mais informações, consulte RTS, CTS e RTR.

Nome de exibição no AWS IoT console: controle RTS/CTS de fluxo

Obrigatório: `false`

Digite: `string`

Valores válidos: `true, false`

Padrão válido: `^(|[Tt][Rr][Uu][Ee]|[Ff][Aa][Ll][Ss][Ee])$`

`XonXoff`

Indica se é necessário habilitar o controle de fluxo de software. O valor padrão é `false`. Para obter mais informações, consulte Controle de fluxo de software.

Nome de exibição no AWS IoT console: Controle de fluxo de software

Obrigatório: `false`

Digite: `string`

Valores válidos: `true, false`

Padrão válido: `^(|[Tt][Rr][Uu][Ee]|[Ff][Aa][Ll][Ss][Ee])$`

`Parity`

A paridade da porta serial. O valor padrão é `N`. Para obter mais informações, consulte Paridade.

Nome de exibição no AWS IoT console: paridade de porta serial

Obrigatório: `false`

Digite: `string`

Valores válidos: `N, E, O, S, M`

Padrão válido: `^(|[NEOSMneosm])$`

Exemplo de criação de conector (AWS CLI)

O seguinte comando da CLI cria um `ConnectorDefinition` com uma versão inicial que contém o conector de fluxo serial. Ele configura o conector para o Polling-Read modo.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MySerialStreamConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/SerialStream/
versions/3",
            "Parameters": {
                "BaudRate" : "9600",
                "Timeout" : "25",
                "SerialPort" : "/dev/serial1",
                "SerialPort-ResourceId" : "my-serial-port-resource",
                "PollingRead" : "true",
                "PollingReadLength" : "30",
                "PollingReadInterval" : "30",
                "PollingReadType" : "hex"
            }
        }
    ]
}'
```

No AWS IoT Greengrass console, você pode adicionar um conector na página Conectores do grupo. Para obter mais informações, consulte the section called “Conceitos básicos de conectores (console)”.

Dados de entrada

Esse conector aceita solicitações de leitura ou gravação para portas seriais em dois tópicos MQTT. As mensagens de entrada devem estar no formato JSON.

- Solicitações de leitura no tópico `serial/+/read/#`.
- Solicitações de gravação no tópico `serial/+/write/#`.

Para publicar nesses tópicos, substitua o caractere curinga + pelo nome da coisa principal e o caractere curinga # pelo caminho para a porta serial. Por exemplo:

```
serial/core-thing-name/read/dev/serial-port
```

Filtro de tópico: `serial/+/read/#`

Use esse tópico para enviar solicitações de leitura sob demanda para um pin serial. As solicitações de leitura são limitadas a um tamanho máximo de leitura de 63994 bytes.

Propriedades de mensagens

`readLength`

O tamanho de dados a serem lidos na porta serial.

Obrigatório: `true`

Digite: `string`

Padrão válido: `^[1-9][0-9]*$`

`type`

O tipo de dados a serem lidos.

Obrigatório: `true`

Digite: `string`

Valores válidos: `ascii, hex`

Padrão válido: `(?i)^(ascii|hex)$`

`id`

Um ID arbitrário para a solicitação. Essa propriedade é usada para mapear a solicitação de entrada para uma resposta de saída.

Obrigatório: `false`

Digite: `string`

Padrão válido: `.+`

Exemplo de entrada

```
{
    "readLength": "30",
    "type": "ascii",
    "id": "abc123"
}
```

Filtro de tópico: `serial/+/write/#`

Use esse tópico para enviar solicitações de gravação para um pin serial. As solicitações de gravação são limitadas um tamanho máximo de dados de 128000 bytes.

Propriedades de mensagens

`data`

A string a ser gravada na porta serial.

Obrigatório: `true`

Digite: `string`

Padrão válido: `^[1-9][0-9]*$`

`type`

O tipo de dados a serem lidos.

Obrigatório: `true`

Digite: `string`

Valores válidos: `ascii, hex`

Padrão válido: `^(ascii|hex|ASCII|HEX)$`

`id`

Um ID arbitrário para a solicitação. Essa propriedade é usada para mapear a solicitação de entrada para uma resposta de saída.

Obrigatório: `false`

Digite: `string`

Padrão válido: `.+`

Exemplo de entrada: solicitação ASCII

```
{
    "data": "random serial data",
    "type": "ascii",
    "id": "abc123"
}
```

Exemplo de entrada: solicitação hexadecimal

```
{
    "data": "base64 encoded data",
    "type": "hex",
    "id": "abc123"
}
```

Dados de saída

O conector publica dados de saída em dois tópicos:

- As informações de status do conector no tópico `serial/+/status/#`.
- Respostas de solicitações de leitura no tópico `serial/+/read_response/#`.

Ao publicar nesse tópico, o conector substitui o caractere curinga + pelo nome da coisa principal e o caractere curinga # pelo o caminho para a porta serial. Por exemplo:

```
serial/core-thing-name/status/dev/serial-port
```

Filtro de tópico: `serial/+/status/#`

Use esse tópico para monitorar o status de solicitações de leitura e gravação. Se uma propriedade `id` for incluída na solicitação, ela será retornada na resposta.

Exemplo de resultado: sucesso

```
{
    "response": {
        "status": "success"
    },
    "id": "abc123"
```

```
}
```

Exemplo de resultado: falha

Uma resposta com falha inclui uma propriedade `error_message` que descreve o erro ou tempo limite encontrado ao executar a operação de leitura e gravação.

```
{
    "response": {
        "status": "fail",
        "error_message": "Could not write to port"
    },
    "id": "abc123"
}
```

Filtro de tópico: `serial/+/read_response/#`

Use esse tópico para receber dados de resposta de uma operação de leitura. Os dados de resposta são codificados em Base64 se o tipo é `hex`.

Exemplo de saída

```
{
    "data": "output of serial read operation"
    "id": "abc123"
}
```

Exemplo de uso

Use as seguintes etapas de alto nível para configurar um exemplo de função do Lambda Python 3.7 que pode ser usado para testar o conector.

> **Note**
>
> - Se você usar outros tempos de execução do Python, poderá criar um link simbólico para Python3.x o Python 3.7.
> - Os tópicos Conceitos básicos de conectores (console) e Conceitos básicos de conectores (CLI) contêm etapas detalhadas que mostram como configurar e implantar um exemplo do conector de notificações do Twilio.

1. Certifique-se de cumprir os requisitos para o conector.
2. Crie e publique uma função do Lambda que envie dados de entrada para o conector.

   Salve o código de exemplo como arquivo PY. Baixe e descompacte o SDK do AWS IoT Greengrass Core para Python. Crie então um pacote zip que contenha o arquivo PY e a pasta `greengrasssdk` no nível raiz. Este pacote zip é o pacote de implantação que você transfere por upload para o AWS Lambda.

   Depois de criar a função do Lambda Python 3.7, publique uma versão de função e crie um alias.
3. Configure o grupo do Greengrass.

   a. Adicione a função do Lambda pelo seu alias (recomendado). Configure o ciclo de vida do Lambda como de longa duração (ou `"Pinned": true` na CLI).

   b. Adicione o recurso de dispositivo local necessário e conceda read/write acesso à função Lambda.

   c. Adicione o conector ao grupo e configure seus parâmetros.

   d. Adicione assinaturas ao grupo que permitam que o conector receba dados de entrada e envie dados de saída em filtros de tópico compatíveis.

      - Defina a função do Lambda como origem, o conector como destino e use um filtro de tópico de entrada compatível.
      - Defina o conector como origem, o AWS IoT Core como destino, e use um filtro de tópico de saída compatível. Você usa essa assinatura para visualizar mensagens de status no AWS IoT console.
4. Implante o grupo.
5. No AWS IoT console, na página Teste, inscreva-se no tópico de dados de saída para ver as mensagens de status do conector. A função de exemplo do Lambda é de longa duração e começa a enviar mensagens imediatamente após o grupo ser implantado.

   Ao finalizar o teste, você pode definir o ciclo de vida do Lambda como sob demanda (ou `"Pinned": false` na CLI) e implantar o grupo. Isso impede o envio de mensagens pela função.

Exemplo

O exemplo a seguir da função do Lambda envia uma mensagem de entrada para o conector.

```
import greengrasssdk
import json

TOPIC_REQUEST = 'serial/CORE_THING_NAME/write/dev/serial1'

# Creating a greengrass core sdk client
iot_client = greengrasssdk.client('iot-data')

def create_serial_stream_request():
 request = {
  "data": "TEST",
  "type": "ascii",
  "id": "abc123"
 }
 return request

def publish_basic_request():
 iot_client.publish(payload=json.dumps(create_serial_stream_request()),
 topic=TOPIC_REQUEST)

publish_basic_request()

def lambda_handler(event, context):
 return
```

Licenças

O conector Serial Stream inclui os seguintes terceiros software/licensing:

- pyserial/BSD

Esse conector é liberado de acordo com o Contrato de licença de software do Greengrass Core.

Changelog

A tabela a seguir descreve as alterações em cada versão do conector.

Versão	Alterações
3	Atualização do runtime do Lambda para Python 3.7, o que altera o requisito de runtime.

Versão	Alterações
2	ARN do conector atualizado para Região da AWS suporte.
1	Versão inicial.

Um grupo do Greengrass só pode conter uma versão do conector por vez. Para obter informações sobre como fazer upgrade de uma versão do conector, consulte the section called “Atualizar a versões do conector”.

Consulte também

- Integrar a serviços e protocolos usando conectores
- the section called “Conceitos básicos de conectores (console)”
- the section called “Conceitos básicos de conectores (CLI)”

ServiceNow MetricBase Conector de integração

> **Warning**
>
> Esse conector passou para a fase de vida útil estendida e AWS IoT Greengrass não lançará atualizações que forneçam atributos, aprimoramentos para atributos existentes, patches de segurança ou correções de erros. Para obter mais informações, consulte AWS IoT Greengrass Version 1 política de manutenção.

O conector de ServiceNow MetricBase integração publica métricas de séries temporais dos dispositivos Greengrass para. ServiceNow MetricBase Isso permite que você armazene, analise e visualize dados de séries temporais do ambiente do núcleo do Greengrass e atue em eventos locais.

Esse conector recebe dados de séries temporais sobre um tópico do MQTT e publica os dados na ServiceNow API em intervalos regulares.

Você pode usar esse conector para oferecer suporte a cenários, como:

- Crie alertas baseados em limites e alarmes baseados em dados de séries temporais coletados em Dispositivos Greengrass.
- Use dados de serviços de horário dos dispositivos Greengrass com aplicativos personalizados criados na ServiceNow plataforma.

Esse conector tem as seguintes versões.

Versão	ARN
4	`arn:aws:greengrass:  region::/connectors/ServiceNowMetricBaseIntegration/versions/4`
3	`arn:aws:greengrass:  region::/connectors/ServiceNowMetricBaseIntegration/versions/3`
2	`arn:aws:greengrass:  region::/connectors/ServiceNowMetricBaseIntegration/versions/2`
1	`arn:aws:greengrass:  region::/connectors/ServiceNowMetricBaseIntegration/versions/1`

Para obter informações sobre alterações de versão, consulte o Changelog.

Requisitos

Esse conector tem os seguintes requisitos:

Version 3 - 4

- AWS IoT Greengrass Software principal v1.9.3 ou posterior. AWS IoT Greengrass deve ser configurado para oferecer suporte a segredos locais, conforme descrito em Requisitos de segredos.

> **Note**
>
> Este requisito inclui permitir o acesso aos seus segredos do Secrets Manager. Se você estiver usando a função de serviço padrão do Greengrass, o Greengrass terá permissão para obter os valores de segredos com nomes que começam com greengrass-.

- Python, versão 3.7 ou 3.8, instalado no dispositivo de núcleo e adicionado à variável de ambiente PATH.

> **Note**
>
> Para usar o Python 3.8, execute o comando a seguir para criar um symblink da pasta de instalação padrão do Python 3.7 para os binários instalados do Python 3.8.
>
> ```
> sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
> ```
>
> Isso configura seu dispositivo para atender ao requisito Python para AWS IoT Greengrass.

- Uma ServiceNow conta com uma assinatura ativada do MetricBase. Além disso, uma métrica e tabela de métrica devem ser criadas na conta. Para obter mais informações, consulte MetricBasea ServiceNow documentação.
- Um tipo de texto secreto AWS Secrets Manager que armazena o nome de usuário e a senha para fazer login na sua ServiceNow instância com autenticação básica. O segredo deve conter chaves "user" e "password" com valores correspondentes. Para obter mais informações, consulte Criar um segredo básico no Guia do usuário do AWS Secrets Manager .
- Um recurso de segredo no grupo do Greengrass que faz referência ao segredo do Secrets Manager. Para obter mais informações, consulte Implantar segredos no núcleo.

Versions 1 - 2

- AWS IoT Greengrass Software principal v1.7 ou posterior. AWS IoT Greengrass deve ser configurado para oferecer suporte a segredos locais, conforme descrito em Requisitos de segredos.

> **Note**
>
> Este requisito inclui permitir o acesso aos seus segredos do Secrets Manager. Se você estiver usando a função de serviço padrão do Greengrass, o Greengrass terá permissão para obter os valores de segredos com nomes que começam com greengrass-.

- Python versão 2.7 instalado no dispositivo de núcleo e adicionado à variável de ambiente PATH.
- Uma ServiceNow conta com uma assinatura ativada do MetricBase. Além disso, uma métrica e tabela de métrica devem ser criadas na conta. Para obter mais informações, consulte MetricBasea ServiceNow documentação.
- Um tipo de texto secreto AWS Secrets Manager que armazena o nome de usuário e a senha para fazer login na sua ServiceNow instância com autenticação básica. O segredo deve conter chaves "user" e "password" com valores correspondentes. Para obter mais informações, consulte Criar um segredo básico no Guia do usuário do AWS Secrets Manager .
- Um recurso de segredo no grupo do Greengrass que faz referência ao segredo do Secrets Manager. Para obter mais informações, consulte Implantar segredos no núcleo.

Parâmetros do conector

Esse conector oferece os seguintes parâmetros:

Version 4

`PublishInterval`

O número máximo de segundos de espera entre os eventos de publicação ServiceNow. O valor máximo é 900.

O conector publica até ServiceNow quando `PublishBatchSize` é atingido ou `PublishInterval` expira.

Nome de exibição no AWS IoT console: intervalo de publicação em segundos

Obrigatório: `true`

Digite: `string`

Valores válidos: `1 - 900`

Padrão válido: `[1-9]|[1-9]\d|[1-9]\d\d|900`

`PublishBatchSize`

O número máximo de valores métricos que podem ser agrupados antes de serem ServiceNow publicados no.

O conector publica até ServiceNow quando `PublishBatchSize` é atingido ou `PublishInterval` expira.

Nome de exibição no AWS IoT console: tamanho do lote de publicação

Obrigatório: `true`

Digite: `string`

Padrão válido: `^[0-9]+$`

`InstanceName`

O nome da instância usada para se conectar ServiceNow.

Nome de exibição no AWS IoT console: Nome da ServiceNow instância

Obrigatório: `true`

Digite: `string`

Padrão válido: `.+`

`DefaultTableName`

O nome da tabela que contém o `GlideRecord` associado ao MetricBase banco de dados de séries temporais. A propriedade `table` na carga da mensagem de entrada pode ser usada para substituir esse valor.

Nome de exibição no AWS IoT console: nome da tabela para conter a métrica

Obrigatório: `true`

Digite: `string`

Padrão válido: `.+`

`MaxMetricsToRetain`

O número máximo de métricas a serem salvas na memória antes de serem substituídas por novas métricas.

Esse limite aplica-se quando não há conexão com a Internet, e o conector começa a armazenar as métricas em buffer para publicar posteriormente. Quando o buffer está cheio, as métricas mais antigas são substituídas por novas métricas.

> **Note**
>
> As métricas não são salvas se o processo de host do conector é interrompido. Por exemplo, isso pode ocorrer durante a implantação do grupo ou quando o dispositivo é reiniciado.

Esse valor deve ser maior do que o tamanho do lote e grande o suficiente para armazenar mensagens com base na taxa de entrada de mensagens MQTT.

Nome de exibição no AWS IoT console: Métricas máximas a serem retidas na memória

Obrigatório: `true`

Digite: `string`

Padrão válido: `^[0-9]+$`

`AuthSecretArn`

O segredo AWS Secrets Manager é que armazena o nome de ServiceNow usuário e a senha. Ele deve ser um segredo de tipo de texto. O segredo deve conter chaves "user" e "password" com valores correspondentes.

Nome de exibição no AWS IoT console: ARN do segredo de autenticação

Obrigatório: `true`

Digite: `string`

Padrão válido: `arn:aws:secretsmanager:[a-z0-9\-]+:[0-9]{12}:secret:([a-zA-Z0-9\\]+/)*[a-zA-Z0-9/_+=,.@\-]+-[a-zA-Z0-9]+`

`AuthSecretArn-ResourceId`

O recurso secreto no grupo que faz referência ao segredo do Secrets Manager para as ServiceNow credenciais.

Nome de exibição no AWS IoT console: recurso de token de autenticação

Obrigatório: `true`

Digite: `string`

Padrão válido: `.+`

`IsolationMode`

O modo de conteinerização para este conector. O padrão é`GreengrassContainer`, o que significa que o conector é executado em um ambiente de execução isolado dentro do AWS IoT Greengrass contêiner.

> **Note**
>
> A configuração padrão de conteinerização para o grupo não se aplica aos conectores.

Nome de exibição no AWS IoT console: modo de isolamento de contêiner

Obrigatório: `false`

Digite: `string`

Valores válidos: `GreengrassContainer` ou `NoContainer`

Padrão válido: `^NoContainer$|^GreengrassContainer$`

Version 1 - 3

`PublishInterval`

O número máximo de segundos de espera entre os eventos de publicação ServiceNow. O valor máximo é 900.

O conector publica até ServiceNow quando `PublishBatchSize` é atingido ou `PublishInterval` expira.

Nome de exibição no AWS IoT console: intervalo de publicação em segundos

Obrigatório: `true`

Digite: `string`

Valores válidos: `1 - 900`

Padrão válido: `[1-9]|[1-9]\d|[1-9]\d\d|900`

`PublishBatchSize`

O número máximo de valores métricos que podem ser agrupados antes de serem ServiceNow publicados no.

O conector publica até ServiceNow quando `PublishBatchSize` é atingido ou `PublishInterval` expira.

Nome de exibição no AWS IoT console: tamanho do lote de publicação

Obrigatório: `true`

Digite: `string`

Padrão válido: `^[0-9]+$`

`InstanceName`

O nome da instância usada para se conectar ServiceNow.

Nome de exibição no AWS IoT console: Nome da ServiceNow instância

Obrigatório: `true`

Digite: `string`

Padrão válido: `.+`

`DefaultTableName`

O nome da tabela que contém o `GlideRecord` associado ao MetricBase banco de dados de séries temporais. A propriedade `table` na carga da mensagem de entrada pode ser usada para substituir esse valor.

Nome de exibição no AWS IoT console: nome da tabela para conter a métrica

Obrigatório: `true`

Digite: `string`

Padrão válido: `.+`

`MaxMetricsToRetain`

O número máximo de métricas a serem salvas na memória antes de serem substituídas por novas métricas.

Esse limite aplica-se quando não há conexão com a Internet, e o conector começa a armazenar as métricas em buffer para publicar posteriormente. Quando o buffer está cheio, as métricas mais antigas são substituídas por novas métricas.

> **Note**
>
> As métricas não são salvas se o processo de host do conector é interrompido. Por exemplo, isso pode ocorrer durante a implantação do grupo ou quando o dispositivo é reiniciado.

Esse valor deve ser maior do que o tamanho do lote e grande o suficiente para armazenar mensagens com base na taxa de entrada de mensagens MQTT.

Nome de exibição no AWS IoT console: Métricas máximas a serem retidas na memória

Obrigatório: `true`

Digite: `string`

Padrão válido: `^[0-9]+$`

`AuthSecretArn`

O segredo AWS Secrets Manager é que armazena o nome de ServiceNow usuário e a senha. Ele deve ser um segredo de tipo de texto. O segredo deve conter chaves "user" e "password" com valores correspondentes.

Nome de exibição no AWS IoT console: ARN do segredo de autenticação

Obrigatório: `true`

Digite: `string`

Padrão válido: `arn:aws:secretsmanager:[a-z0-9\-]+:[0-9]{12}:secret:([a-zA-Z0-9\\]+/)*[a-zA-Z0-9/_+=,.@\-]+-[a-zA-Z0-9]+`

`AuthSecretArn-ResourceId`

O recurso secreto no grupo que faz referência ao segredo do Secrets Manager para as ServiceNow credenciais.

Nome de exibição no AWS IoT console: recurso de token de autenticação

Obrigatório: `true`

Digite: `string`

Padrão válido: `.+`

Exemplo de criação de conector (AWS CLI)

O comando CLI a seguir cria um `ConnectorDefinition` com uma versão inicial que contém o conector de ServiceNow MetricBase integração.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MyServiceNowMetricBaseIntegrationConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/
ServiceNowMetricBaseIntegration/versions/4",
            "Parameters": {
                "PublishInterval" : "10",
                "PublishBatchSize" : "50",
                "InstanceName" : "myinstance",
                "DefaultTableName" : "u_greengrass_app",
                "MaxMetricsToRetain" : "20000",
                "AuthSecretArn" : "arn:aws:secretsmanager:region:account-
id:secret:greengrass-secret-hash",
                "AuthSecretArn-ResourceId" : "MySecretResource",
                "IsolationMode" : "GreengrassContainer"
            }
        }
    ]
```

```
}'
```

> **Note**
>
> A função do Lambda nesse conector tem um ciclo de vida longo.

No AWS IoT Greengrass console, você pode adicionar um conector na página Conectores do grupo. Para obter mais informações, consulte the section called “Conceitos básicos de conectores (console)”.

Dados de entrada

Esse conector aceita métricas de séries temporais em um tópico do MQTT e publica as métricas em. ServiceNow As mensagens de entrada devem estar no formato JSON.

Filtro de tópico na assinatura

`servicenow/metricbase/metric`

Propriedades de mensagens

`request`

As informações sobre a tabela, o registro e a métrica. Essa solicitação representa o objeto `seriesRef` em uma solicitação POST de séries temporais. Para obter mais informações, consulte API - POST de séries temporais Clotho.

Obrigatório: `true`

Tipo: `object` que inclui as seguintes propriedades:

`subject`

O `sys_id` do registro específico na tabela.

Obrigatório: `true`

Digite: `string`

`metric_name`

O nome do campo da métrica.

Obrigatório: `true`

Digite: `string`

`table`

O nome da tabela na qual armazenar o registro. Especifique esse valor para substituir o parâmetro `DefaultTableName`.

Obrigatório: `false`

Digite: `string`

`value`

O valor do ponto de dados individual.

Obrigatório: `true`

Digite: `float`

`timestamp`

O timestamp do ponto de dados individual. O valor padrão é o horário atual.

Obrigatório: `false`

Digite: `string`

Exemplo de entrada

```
{
    "request": {
        "subject":"ef43c6d40a0a0b5700c77f9bf387afe3",
        "metric_name":"u_count",
        "table": "u_greengrass_app"
        "value": 1.0,
        "timestamp": "2018-10-14T10:30:00"
    }
}
```

Dados de saída

O conector publica informações de status como dados de saída em um tópico MQTT.

Filtro de tópico na assinatura

```
servicenow/metricbase/metric/status
```

Exemplo de resultado: sucesso

```
{
    "response": {
        "metric_name": "Errors",
        "table_name": "GliderProd",
        "processed_on": "2018-10-14T10:35:00",
        "response_id": "khjKSkj132qwr23fcba",
        "status": "success",
        "values": [
            {
                "timestamp": "2016-10-14T10:30:00",
                "value": 1.0
            },
            {
                "timestamp": "2016-10-14T10:31:00",
                "value": 1.1
            }
        ]
    }
}
```

Exemplo de resultado: falha

```
{
    "response": {
        "error": "InvalidInputException",
        "error_message": "metric value is invalid",
        "status": "fail"
    }
}
```

> **Note**
>
> Se o conector detectar um erro que pode ser repetido (por exemplo, erros de conexão), ele tentará publicar novamente no próximo lote.

Exemplo de uso

Use as seguintes etapas de alto nível para configurar um exemplo de função do Lambda Python 3.7 que pode ser usado para testar o conector.

> **Note**
>
> - Se você usar outros tempos de execução do Python, poderá criar um link simbólico para Python3.x o Python 3.7.
> - Os tópicos Conceitos básicos de conectores (console) e Conceitos básicos de conectores (CLI) contêm etapas detalhadas que mostram como configurar e implantar um exemplo do conector de notificações do Twilio.

1. Certifique-se de cumprir os requisitos para o conector.
2. Crie e publique uma função do Lambda que envie dados de entrada para o conector.

   Salve o código de exemplo como arquivo PY. Baixe e descompacte o SDK do AWS IoT Greengrass Core para Python. Crie então um pacote zip que contenha o arquivo PY e a pasta `greengrasssdk` no nível raiz. Este pacote zip é o pacote de implantação que você transfere por upload para o AWS Lambda.

   Depois de criar a função do Lambda Python 3.7, publique uma versão de função e crie um alias.
3. Configure o grupo do Greengrass.

   a. Adicione a função do Lambda pelo seu alias (recomendado). Configure o ciclo de vida do Lambda como de longa duração (ou `"Pinned": true` na CLI).

   b. Adicione o recurso de segredode segredo necessário e conceda acesso de leitura à função do Lambda.

   c. Adicione o conector e configure seus parâmetros.

   d. Adicione assinaturas que permitam que o conector receba dados de entrada e envie dados de saída em filtros de tópico compatíveis.

      - Defina a função do Lambda como origem, o conector como destino e use um filtro de tópico de entrada compatível.

- Defina o conector como origem, o AWS IoT Core como destino, e use um filtro de tópico de saída compatível. Você usa essa assinatura para visualizar mensagens de status no AWS IoT console.

4. Implante o grupo.
5. No AWS IoT console, na página Teste, inscreva-se no tópico de dados de saída para ver as mensagens de status do conector. A função de exemplo do Lambda é de longa duração e começa a enviar mensagens imediatamente após o grupo ser implantado.

   Ao finalizar o teste, você pode definir o ciclo de vida do Lambda como sob demanda (ou `"Pinned": false` na CLI) e implantar o grupo. Isso impede o envio de mensagens pela função.

Exemplo

O exemplo a seguir da função do Lambda envia uma mensagem de entrada para o conector.

```
import greengrasssdk
import json

iot_client = greengrasssdk.client('iot-data')
SEND_TOPIC = 'servicenow/metricbase/metric'

def create_request_with_all_fields():
    return {
        "request": {
             "subject": '2efdf6badbd523803acfae441b961961',
             "metric_name": 'u_count',
             "value": 1234,
             "timestamp": '2018-10-20T20:22:20',
             "table": 'u_greengrass_metricbase_test'
        }
    }

def publish_basic_message():
    messageToPublish = create_request_with_all_fields()
    print("Message To Publish: ", messageToPublish)
    iot_client.publish(topic=SEND_TOPIC,
        payload=json.dumps(messageToPublish))

publish_basic_message()
```

```
def lambda_handler(event, context):
    return
```

Licenças

O conector de ServiceNow MetricBase integração inclui os seguintes terceiros software/licensing:

- pysnow/MIT

Esse conector é liberado de acordo com o Contrato de licença de software do Greengrass Core.

Changelog

A tabela a seguir descreve as alterações em cada versão do conector.

Versão	Alterações
4	Adicionado o parâmetro `IsolationMode` para configurar o modo de conteinerização para o conector.
3	Atualização do runtime do Lambda para Python 3.7, o que altera o requisito de runtime.
2	Corrija para reduzir o registro excessivo.
1	Versão inicial.

Um grupo do Greengrass só pode conter uma versão do conector por vez. Para obter informações sobre como fazer upgrade de uma versão do conector, consulte the section called “Atualizar a versões do conector”.

Consulte também

- Integrar a serviços e protocolos usando conectores
- the section called “Conceitos básicos de conectores (console)”
- the section called “Conceitos básicos de conectores (CLI)”

Conector do SNS

O conector do SNS publica mensagens em um tópico do Amazon SNS. Isso permite que os servidores web, endereços de e-mail e outros assinantes de mensagens respondam a eventos no grupo do Greengrass.

Esse conector recebe informações de mensagens do SNS em um tópico MQTT e envia a mensagem para um determinado tópico do SNS. Você também pode usar funções do Lambda personalizadas para implementar a lógica de formatação ou filtragem em mensagens antes que elas sejam publicadas nesse conector.

Esse conector tem as seguintes versões.

Versão	ARN
4	`arn:aws:greengrass:region::/connectors/SNS/versions/4`
3	`arn:aws:greengrass:region::/connectors/SNS/versions/3`
2	`arn:aws:greengrass:region::/connectors/SNS/versions/2`
1	`arn:aws:greengrass:region::/connectors/SNS/versions/1`

Para obter informações sobre alterações de versão, consulte o Changelog.

Requisitos

Esse conector tem os seguintes requisitos:

Version 3 - 4

- AWS IoT Greengrass Software principal v1.9.3 ou posterior.
- Python, versão 3.7 ou 3.8, instalado no dispositivo de núcleo e adicionado à variável de ambiente PATH.

> **Note**
>
> Para usar o Python 3.8, execute o comando a seguir para criar um symblink da pasta de instalação padrão do Python 3.7 para os binários instalados do Python 3.8.
>
> ```
> sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
> ```
>
> Isso configura seu dispositivo para atender ao requisito Python para AWS IoT Greengrass.

- Um tópico do SNS configurado. Para obter mais informações, consulte Criação de um tópico do Amazon SNS no Guia do desenvolvedor do Amazon Simple Notification Service.
- A função de grupo do Greengrass configurada para permitir a ação `sns:Publish` no tópico de destino do Amazon SNS, conforme mostrado no exemplo de política do IAM a seguir.

  JSON

  ```
  {
      "Version":"2012-10-17",
      "Statement": [
          {
              "Sid": "Stmt1528133056761",
              "Action": [
                  "sns:Publish"
              ],
              "Effect": "Allow",
              "Resource": [
              "arn:aws:sns:us-east-1:123456789012:topic-name"
              ]
          }
      ]
  }
  ```

  Esse conector permite que você substitua dinamicamente o tópico padrão na carga da mensagem de entrada. Se a sua implementação usa esse atributo, a política do IAM deve conceder a permissão `sns:Publish` em todos os tópicos de destino. Você pode conceder acesso granular ou condicional aos recursos (por exemplo, usando um esquema de nomeação * curinga).

Para o requisito de função de grupo, você deve configurar a função para conceder as permissões necessárias e certificar-se de que a função tenha sido adicionada ao grupo. Para acessar mais informações, consulte the section called “Gerenciar a função de grupo (console)” ou the section called “Gerenciar a função de grupo (CLI)”.

Versions 1 - 2

- AWS IoT Greengrass Software principal v1.7 ou posterior.
- Python versão 2.7 instalado no dispositivo de núcleo e adicionado à variável de ambiente PATH.
- Um tópico do SNS configurado. Para obter mais informações, consulte Criação de um tópico do Amazon SNS no Guia do desenvolvedor do Amazon Simple Notification Service.
- A função de grupo do Greengrass configurada para permitir a ação `sns:Publish` no tópico de destino do Amazon SNS, conforme mostrado no exemplo de política do IAM a seguir.

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1528133056761",
            "Action": [
                "sns:Publish"
            ],
            "Effect": "Allow",
            "Resource": [
            "arn:aws:sns:us-east-1:123456789012:topic-name"
            ]
        }
    ]
}
```

  Esse conector permite que você substitua dinamicamente o tópico padrão na carga da mensagem de entrada. Se a sua implementação usa esse atributo, a política do IAM deve conceder a permissão `sns:Publish` em todos os tópicos de destino. Você pode conceder acesso granular ou condicional aos recursos (por exemplo, usando um esquema de nomeação * curinga).

Para o requisito de função de grupo, você deve configurar a função para conceder as permissões necessárias e certificar-se de que a função tenha sido adicionada ao grupo. Para acessar mais informações, consulte the section called “Gerenciar a função de grupo (console)” ou the section called “Gerenciar a função de grupo (CLI)”.

Parâmetros do conector

Esse conector oferece os seguintes parâmetros:

Version 4

`DefaultSNSArn`

O ARN do tópico padrão do SNS no qual publicar mensagens. O tópico de destino pode ser substituído pela propriedade `sns_topic_arn` na carga da mensagem de entrada.

> **Note**
>
> A função do grupo deve conceder a permissão `sns:Publish` em todos os tópicos de destino. Para obter mais informações, consulte the section called “Requisitos”.

Nome de exibição no AWS IoT console: ARN do tópico SNS padrão

Obrigatório: `true`

Digite: `string`

Padrão válido: `arn:aws:sns:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):([a-zA-Z0-9-_]+)$`

`IsolationMode`

O modo de conteinerização para este conector. O padrão é`GreengrassContainer`, o que significa que o conector é executado em um ambiente de execução isolado dentro do AWS IoT Greengrass contêiner.

> **Note**
>
> A configuração padrão de conteinerização para o grupo não se aplica aos conectores.

Nome de exibição no AWS IoT console: modo de isolamento de contêiner

Obrigatório: `false`

Digite: `string`

Valores válidos: `GreengrassContainer` ou `NoContainer`

Padrão válido: `^NoContainer$|^GreengrassContainer$`

Versions 1 - 3

`DefaultSNSArn`

O ARN do tópico padrão do SNS no qual publicar mensagens. O tópico de destino pode ser substituído pela propriedade `sns_topic_arn` na carga da mensagem de entrada.

> **Note**
>
> A função do grupo deve conceder a permissão `sns:Publish` em todos os tópicos de destino. Para obter mais informações, consulte the section called “Requisitos”.

Nome de exibição no AWS IoT console: ARN do tópico SNS padrão

Obrigatório: `true`

Digite: `string`

Padrão válido: `arn:aws:sns:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):([a-zA-Z0-9-_]+)$`

Exemplo de criação de conector (AWS CLI)

O seguinte comando da CLI cria um `ConnectorDefinition` com uma versão inicial que contém o conector do SNS.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
```

```
        {
            "Id": "MySNSConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/SNS/versions/4",
            "Parameters": {
                "DefaultSNSArn": "arn:aws:sns:region:account-id:topic-name",
                "IsolationMode" : "GreengrassContainer"
            }
        }
    ]
}'
```

No AWS IoT Greengrass console, você pode adicionar um conector na página Conectores do grupo. Para obter mais informações, consulte the section called “Conceitos básicos de conectores (console)”.

Dados de entrada

Esse conector aceita informações de mensagens do SNS em um tópico MQTT e publica a mensagem da maneira que se encontra como um tópico do SNS de destino. As mensagens de entrada devem estar no formato JSON.

Filtro de tópico na assinatura

`sns/message`

Propriedades de mensagens

`request`

Informações sobre a mensagem a ser enviada para o tópico do SNS.

Obrigatório: `true`

Tipo: `object` que inclui as seguintes propriedades:

`message`

O conteúdo da mensagem como uma string ou no formato JSON. Para obter exemplos, consulte Exemplos de entrada.

Para enviar JSON, a propriedade `message_structure` deve ser definida como `json` e a mensagem deve ser uma string codificada por objeto JSON com uma chave `default`.

Obrigatório: `true`

Digite: `string`

Padrão válido: `.*`

`subject`

O assunto da mensagem.

Obrigatório: `false`

Tipo: O texto ASCII, com até 100 caracteres. Deve começar com uma letra, um número ou um sinal de pontuação. Não deve incluir quebras de linha ou caracteres de controle.

Padrão válido: `.*`

`sns_topic_arn`

O ARN do tópico do SNS no qual publicar mensagens. Se especificado, o conector publicará nesse tópico em vez do tópico padrão.

> **Note**
>
> A função do grupo deve conceder a permissão `sns:Publish` a qualquer tópico de destino. Para obter mais informações, consulte the section called “Requisitos”.

Obrigatório: `false`

Digite: `string`

Padrão válido: `arn:aws:sns:([a-z]{2}-[a-z]+-\d{1}):(\d{12}):([a-zA-Z0-9-_]+)$`

`message_structure`

A estrutura da mensagem.

Obrigatório:`false`. Isso deve ser especificado para enviar uma mensagem JSON.

Tipo: `string`

Valores válidos: `json`

`id`

Um ID arbitrário para a solicitação. Essa propriedade é usada para mapear a solicitação de entrada para uma resposta de saída. Quando especificada, a propriedade `id` no objeto de resposta é definida para esse valor. Se você não usa esse atributo, é possível omitir essa propriedade ou especificar uma string vazia.

Obrigatório: `false`

Digite: `string`

Padrão válido: `.*`

Limites

O tamanho da mensagem é limitado por um tamanho máximo de mensagem do SNS de 256 KB.

Exemplo de entrada: mensagem de string

Este exemplo envia uma mensagem de string. Ele especifica a propriedade opcional `sns_topic_arn`, que substitui o tópico de destino padrão.

```
{
    "request": {
        "subject": "Message subject",
        "message": "Message data",
        "sns_topic_arn": "arn:aws:sns:region:account-id:topic2-name"
    },
    "id": "request123"
}
```

Exemplo de entrada: mensagem JSON

Este exemplo envia uma mensagem como objeto JSON codificado por string que inclui a chave `default`.

```
{
    "request": {
        "subject": "Message subject",
        "message": "{ \"default\": \"Message data\" }",
        "message_structure": "json"
    },
    "id": "request123"
```

```
}
```

Dados de saída

O conector publica informações de status como dados de saída em um tópico MQTT.

Filtro de tópico na assinatura

```
sns/message/status
```

Exemplo de resultado: sucesso

```
{
    "response": {
        "sns_message_id": "f80a81bc-f44c-56f2-a0f0-d5af6a727c8a",
        "status": "success"
    },
    "id": "request123"
}
```

Exemplo de resultado: falha

```
{
   "response" : {
        "error": "InvalidInputException",
        "error_message": "SNS Topic Arn is invalid",
        "status": "fail"
   },
   "id": "request123"
}
```

Exemplo de uso

Use as seguintes etapas de alto nível para configurar um exemplo de função do Lambda Python 3.7 que pode ser usado para testar o conector.

> **Note**
>
> - Se você usar outros tempos de execução do Python, poderá criar um link simbólico para Python3.x o Python 3.7.

- Os tópicos Conceitos básicos de conectores (console) e Conceitos básicos de conectores (CLI) contêm etapas detalhadas que mostram como configurar e implantar um exemplo do conector de notificações do Twilio.

1. Certifique-se de cumprir os requisitos para o conector.

   Para o requisito de função de grupo, você deve configurar a função para conceder as permissões necessárias e certificar-se de que a função tenha sido adicionada ao grupo. Para acessar mais informações, consulte the section called “Gerenciar a função de grupo (console)” ou the section called “Gerenciar a função de grupo (CLI)”.
2. Crie e publique uma função do Lambda que envie dados de entrada para o conector.

   Salve o código de exemplo como arquivo PY. Baixe e descompacte o SDK do AWS IoT Greengrass Core para Python. Crie então um pacote zip que contenha o arquivo PY e a pasta `greengrasssdk` no nível raiz. Este pacote zip é o pacote de implantação que você transfere por upload para o AWS Lambda.

   Depois de criar a função do Lambda Python 3.7, publique uma versão de função e crie um alias.
3. Configure o grupo do Greengrass.

   a. Adicione a função do Lambda pelo seu alias (recomendado). Configure o ciclo de vida do Lambda como de longa duração (ou `"Pinned": true` na CLI).

   b. Adicione o conector e configure seus parâmetros.

   c. Adicione assinaturas que permitam que o conector receba dados de entrada e envie dados de saída em filtros de tópico compatíveis.

      - Defina a função do Lambda como origem, o conector como destino e use um filtro de tópico de entrada compatível.
      - Defina o conector como origem, o AWS IoT Core como destino, e use um filtro de tópico de saída compatível. Você usa essa assinatura para visualizar mensagens de status no AWS IoT console.
4. Implante o grupo.
5. No AWS IoT console, na página Teste, inscreva-se no tópico de dados de saída para ver as mensagens de status do conector. A função de exemplo do Lambda é de longa duração e começa a enviar mensagens imediatamente após o grupo ser implantado.

Ao finalizar o teste, você pode definir o ciclo de vida do Lambda como sob demanda (ou `"Pinned": false` na CLI) e implantar o grupo. Isso impede o envio de mensagens pela função.

Exemplo

O exemplo a seguir da função do Lambda envia uma mensagem de entrada para o conector.

```
import greengrasssdk
import time
import json

iot_client = greengrasssdk.client('iot-data')
send_topic = 'sns/message'

def create_request_with_all_fields():
    return  {
        "request": {
            "message": "Message from SNS Connector Test"
        },
        "id" : "req_123"
    }

def publish_basic_message():
    messageToPublish = create_request_with_all_fields()
    print("Message To Publish: ", messageToPublish)
    iot_client.publish(topic=send_topic,
        payload=json.dumps(messageToPublish))

publish_basic_message()

def lambda_handler(event, context):
    return
```

Licenças

O conector SNS inclui os seguintes terceiros software/licensing:

- AWS SDK para 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 conector é liberado de acordo com o Contrato de licença de software do Greengrass Core.

Changelog

A tabela a seguir descreve as alterações em cada versão do conector.

Versão	Alterações
4	Adicionado o parâmetro `IsolationMode` para configurar o modo de conteinerização para o conector.
3	Atualização do runtime do Lambda para Python 3.7, o que altera o requisito de runtime.
2	Corrija para reduzir o registro excessivo.
1	Versão inicial.

Um grupo do Greengrass só pode conter uma versão do conector por vez. Para obter informações sobre como fazer upgrade de uma versão do conector, consulte the section called “Atualizar a versões do conector”.

Consulte também

- Integrar a serviços e protocolos usando conectores
- the section called “Conceitos básicos de conectores (console)”
- the section called “Conceitos básicos de conectores (CLI)”
- Ação de publicação na documentação do Boto 3

- O que é o Amazon Simple Notification Service? no Guia do desenvolvedor do Amazon Simple Notification Service

Conector de integração Splunk

> **Warning**
>
> Esse conector passou para a fase de vida útil estendida e AWS IoT Greengrass não lançará atualizações que forneçam atributos, aprimoramentos para atributos existentes, patches de segurança ou correções de erros. Para obter mais informações, consulte AWS IoT Greengrass Version 1 política de manutenção.

O conector de integração Splunk publica dados de Dispositivos Greengrass no Splunk. Isso permite que você use o Splunk para monitorar e analisar o ambiente do núcleo do Greengrass e atue em eventos locais. O conector integra-se ao Coletor de eventos HTTP (HEC). Para obter mais informações, consulte Introdução ao Coletor de eventos HTTP do Splunk na documentação do Splunk.

Esse conector recebe o registro em log e os dados de eventos em um tópico MQTT e publica os dados da maneira que se encontram na API do Splunk.

Você pode usar esse conector para oferecer suporte a cenários industriais, como:

- Os operadores podem usar dados periódicos de sensores e acionadores (por exemplo, temperatura, pressão e leituras de água) para iniciar alarmes quando os valores excederem determinados limites.
- Os desenvolvedores usam dados coletados de máquinas industriais para criar modelos de ML que podem monitorar o equipamento em relação a possíveis problemas.

Esse conector tem as seguintes versões.

Versão	ARN
4	`arn:aws:greengrass:  region::/connectors/SplunkIntegration/versions/4`

Versão	ARN
3	`arn:aws:greengrass:region::/connectors/SplunkIntegration/versions/3`
2	`arn:aws:greengrass:region::/connectors/SplunkIntegration/versions/2`
1	`arn:aws:greengrass:region::/connectors/SplunkIntegration/versions/1`

Para obter informações sobre alterações de versão, consulte o Changelog.

Requisitos

Esse conector tem os seguintes requisitos:

Version 3 - 4

- AWS IoT Greengrass Software principal v1.9.3 ou posterior. AWS IoT Greengrass deve ser configurado para oferecer suporte a segredos locais, conforme descrito em Requisitos de segredos.

  > **Note**
  >
  > Este requisito inclui permitir o acesso aos seus segredos do Secrets Manager. Se você estiver usando a função de serviço padrão do Greengrass, o Greengrass terá permissão para obter os valores de segredos com nomes que começam com greengrass-.

- Python, versão 3.7 ou 3.8, instalado no dispositivo de núcleo e adicionado à variável de ambiente PATH.

> **Note**
>
> Para usar o Python 3.8, execute o comando a seguir para criar um symblink da pasta de instalação padrão do Python 3.7 para os binários instalados do Python 3.8.
>
> ```
> sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
> ```
>
> Isso configura seu dispositivo para atender ao requisito Python para AWS IoT Greengrass.

- A funcionalidade do Coletor de eventos HTTP deve ser habilitada no Splunk. Para obter mais informações, consulte Configurar e usar o eEvent Collector HTTP no Splunk Web na documentação do Splunk.
- Um segredo de tipo de texto AWS Secrets Manager que armazena seu token do Splunk HTTP Event Collector. Para obter mais informações, consulte Sobre os tokens do coletor de eventos na documentação do Splunk e Criando um segredo básico no Guia do usuário do AWS Secrets Manager .

> **Note**
>
> Para criar o segredo no console do Secrets Manager, insira seu token na guia Texto simples. Não inclua aspas ou outra formatação. Na API, especifique o token como o valor da propriedade `SecretString`.

- Um recurso de segredo no grupo do Greengrass que faz referência ao segredo do Secrets Manager. Para obter mais informações, consulte Implantar segredos no núcleo.

Versions 1 - 2

- AWS IoT Greengrass Software principal v1.7 ou posterior. AWS IoT Greengrass deve ser configurado para oferecer suporte a segredos locais, conforme descrito em Requisitos de segredos.

> **Note**
>
> Este requisito inclui permitir o acesso aos seus segredos do Secrets Manager. Se você estiver usando a função de serviço padrão do Greengrass, o Greengrass

terá permissão para obter os valores de segredos com nomes que começam com greengrass-.

- Python versão 2.7 instalado no dispositivo de núcleo e adicionado à variável de ambiente PATH.
- A funcionalidade do Coletor de eventos HTTP deve ser habilitada no Splunk. Para obter mais informações, consulte Configurar e usar o eEvent Collector HTTP no Splunk Web na documentação do Splunk.
- Um segredo de tipo de texto AWS Secrets Manager que armazena seu token do Splunk HTTP Event Collector. Para obter mais informações, consulte Sobre os tokens do coletor de eventos na documentação do Splunk e Criando um segredo básico no Guia do usuário do AWS Secrets Manager .

> **Note**
>
> Para criar o segredo no console do Secrets Manager, insira seu token na guia Texto simples. Não inclua aspas ou outra formatação. Na API, especifique o token como o valor da propriedade `SecretString`.

- Um recurso de segredo no grupo do Greengrass que faz referência ao segredo do Secrets Manager. Para obter mais informações, consulte Implantar segredos no núcleo.

Parâmetros do conector

Esse conector oferece os seguintes parâmetros:

Version 4

`SplunkEndpoint`

Um endpoint da sua instância do Splunk. Esse valor deve conter o protocolo, o nome do host e a porta.

Nome de exibição no AWS IoT console: endpoint Splunk

Obrigatório: `true`

Digite: `string`

Padrão válido: `^(http:\/\/|https:\/\/)?[a-z0-9]+([-.]{1}[a-z0-9]+)*.[a-z]{2,5}(:[0-9]{1,5})?(\/.*)?$`

`MemorySize`

A quantidade de memória (em KB) a ser alocada para o conector.

Nome de exibição no AWS IoT console: Tamanho da memória

Obrigatório: `true`

Digite: `string`

Padrão válido: `^[0-9]+$`

`SplunkQueueSize`

O número máximo de itens a serem salvos na memória antes de serem enviados ou descartados. Quando esse limite for atingido, os itens mais antigos na fila serão substituídos pelos itens mais recentes. Esse limite normalmente se aplica quando não há conexão com a Internet.

Nome de exibição no AWS IoT console: máximo de itens a serem retidos

Obrigatório: `true`

Digite: `string`

Padrão válido: `^[0-9]+$`

`SplunkFlushIntervalSeconds`

O intervalo (em segundos) para a publicação de dados recebidos no HEC do Splunk. O valor máximo é 900. Para configurar o conector a fim de publicar itens conforme são recebidos (sem agrupamento em lotes), especifique 0.

Nome de exibição no AWS IoT console: intervalo de publicação do Splunk

Obrigatório: `true`

Digite: `string`

Padrão válido: `[0-9]|[1-9]\d|[1-9]\d\d|900`

`SplunkTokenSecretArn`

O segredo é AWS Secrets Manager que armazena o token Splunk. Ele deve ser um segredo de tipo de texto.

Nome de exibição no AWS IoT console: ARN do segredo do token de autenticação do Splunk

Obrigatório: `true`

Digite: `string`

Padrão válido: `arn:aws:secretsmanager:[a-z]{2}-[a-z]+-\d{1}:\d{12}?:secret:[a-zA-Z0-9-_]+-[a-zA-Z0-9-_]+`

`SplunkTokenSecretArn-ResourceId`

O recurso de segredo no grupo do Greengrass que faz referência ao segredo do Splunk.

Nome de exibição no AWS IoT console: recurso de token de autenticação Splunk

Obrigatório: `true`

Digite: `string`

Padrão válido: `.+`

`SplunkCustomCALocation`

O caminho do arquivo da autoridade de certificação (CA) personalizada para o Splunk (por exemplo, `/etc/ssl/certs/splunk.crt`).

Nome de exibição no AWS IoT console: localização da autoridade de certificação personalizada do Splunk

Obrigatório: `false`

Digite: `string`

Padrão válido: `^$|/.*`

`IsolationMode`

O modo de conteinerização para este conector. O padrão é`GreengrassContainer`, o que significa que o conector é executado em um ambiente de execução isolado dentro do AWS IoT Greengrass contêiner.

> **Note**
>
> A configuração padrão de conteinerização para o grupo não se aplica aos conectores.

Nome de exibição no AWS IoT console: modo de isolamento de contêiner

Obrigatório: `false`

Digite: `string`

Valores válidos: `GreengrassContainer` ou `NoContainer`

Padrão válido: `^NoContainer$|^GreengrassContainer$`

Version 1 - 3

`SplunkEndpoint`

Um endpoint da sua instância do Splunk. Esse valor deve conter o protocolo, o nome do host e a porta.

Nome de exibição no AWS IoT console: endpoint Splunk

Obrigatório: `true`

Digite: `string`

Padrão válido: `^(http:\/\/|https:\/\/)?[a-z0-9]+([-.]{1}[a-z0-9]+)*.[a-z]{2,5}(:[0-9]{1,5})?(\/.*)?$`

`MemorySize`

A quantidade de memória (em KB) a ser alocada para o conector.

Nome de exibição no AWS IoT console: Tamanho da memória

Obrigatório: `true`

Digite: `string`

Padrão válido: `^[0-9]+$`

`SplunkQueueSize`

O número máximo de itens a serem salvos na memória antes de serem enviados ou descartados. Quando esse limite for atingido, os itens mais antigos na fila serão substituídos pelos itens mais recentes. Esse limite normalmente se aplica quando não há conexão com a Internet.

Nome de exibição no AWS IoT console: máximo de itens a serem retidos

Obrigatório: `true`

Digite: `string`

Padrão válido: `^[0-9]+$`

`SplunkFlushIntervalSeconds`

O intervalo (em segundos) para a publicação de dados recebidos no HEC do Splunk. O valor máximo é 900. Para configurar o conector a fim de publicar itens conforme são recebidos (sem agrupamento em lotes), especifique 0.

Nome de exibição no AWS IoT console: intervalo de publicação do Splunk

Obrigatório: `true`

Digite: `string`

Padrão válido: `[0-9]|[1-9]\d|[1-9]\d\d|900`

`SplunkTokenSecretArn`

O segredo é AWS Secrets Manager que armazena o token Splunk. Ele deve ser um segredo de tipo de texto.

Nome de exibição no AWS IoT console: ARN do segredo do token de autenticação do Splunk

Obrigatório: `true`

Digite: `string`

Padrão válido: `arn:aws:secretsmanager:[a-z]{2}-[a-z]+-\d{1}:\d{12}?:secret:[a-zA-Z0-9-_]+-[a-zA-Z0-9-_]+`

`SplunkTokenSecretArn-ResourceId`

O recurso de segredo no grupo do Greengrass que faz referência ao segredo do Splunk.

Nome de exibição no AWS IoT console: recurso de token de autenticação Splunk

Obrigatório: `true`

Digite: `string`

Padrão válido: `.+`

`SplunkCustomCALocation`

O caminho do arquivo da autoridade de certificação (CA) personalizada para o Splunk (por exemplo, `/etc/ssl/certs/splunk.crt`).

Nome de exibição no AWS IoT console: localização da autoridade de certificação personalizada do Splunk

Obrigatório: `false`

Digite: `string`

Padrão válido: `^$|/.*`

Exemplo de criação de conector (AWS CLI)

O seguinte comando da CLI cria um `ConnectorDefinition` com uma versão inicial que contém o conector de integração Splunk.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
    "Connectors": [
        {
            "Id": "MySplunkIntegrationConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/SplunkIntegration/
versions/4",
            "Parameters": {
                "SplunkEndpoint": "https://myinstance.cloud.splunk.com:8088",
                "MemorySize": 200000,
                "SplunkQueueSize": 10000,
                "SplunkFlushIntervalSeconds": 5,
                "SplunkTokenSecretArn":"arn:aws:secretsmanager:region:account-
id:secret:greengrass-secret-hash",
                "SplunkTokenSecretArn-ResourceId": "MySplunkResource",
                "IsolationMode" : "GreengrassContainer"
```

```
            }
        }
    ]
}'
```

> **Note**
>
> A função do Lambda nesse conector tem um ciclo de vida longo.

No AWS IoT Greengrass console, você pode adicionar um conector na página Conectores do grupo. Para obter mais informações, consulte the section called “Conceitos básicos de conectores (console)”.

Dados de entrada

Esse conector aceita o registro em log e os dados de eventos em um tópico MQTT e publica os dados recebidos da maneira que se encontram na API do Splunk. As mensagens de entrada devem estar no formato JSON.

Filtro de tópico na assinatura

`splunk/logs/put`

Propriedades de mensagens

`request`

Os dados de eventos a serem enviados para a API do Splunk. Os eventos devem atender às especificações da services/collectorAPI.

Obrigatório: `true`

Tipo:`object`. Somente a propriedade `event` é obrigatória.

`id`

Um ID arbitrário para a solicitação. Essa propriedade é usada para mapear a solicitação de entrada para um status de saída.

Obrigatório: `false`

Digite: `string`

Limites

Todos os limites impostos pela API do Splunk se aplicam ao usar esse conector. Para obter mais informações, consulte services/collector.

Exemplo de entrada

```
{
    "request": {
        "event": "some event",
        "fields": {
            "severity": "INFO",
            "category": [
                "value1",
                "value2"
            ]
        }
    },
    "id": "request123"
}
```

Dados de saída

Esse conector publica dados de saída em dois tópicos:

- Informações de status no tópico `splunk/logs/put/status`.
- Erros no tópico `splunk/logs/put/error`.

Filtro de tópico: `splunk/logs/put/status`

Use esse tópico para monitorar o status das solicitações. Sempre que o conector envia um lote de dados recebidos para a API do Splunk, ele publica uma lista dos IDs das solicitações que foram bem-sucedidas e que falharam.

Exemplo de saída

```
{
    "response": {
        "succeeded": [
            "request123",
            ...
        ],
```

```
        "failed": [
            "request789",
            ...
        ]
    }
}
```

Filtro de tópico: `splunk/logs/put/error`

Use esse tópico para monitorar erros do conector. A propriedade `error_message` que descreve o erro ou tempo limite encontrado ao processar a solicitação.

Exemplo de saída

```
{
    "response": {
        "error": "UnauthorizedException",
        "error_message": "invalid splunk token",
        "status": "fail"
    }
}
```

> **Note**
>
> Se o conector detectar um erro que pode ser repetido (por exemplo, erros de conexão), ele tentará publicar novamente no próximo lote.

Exemplo de uso

Use as seguintes etapas de alto nível para configurar um exemplo de função do Lambda Python 3.7 que pode ser usado para testar o conector.

> **Note**
>
> - Se você usar outros tempos de execução do Python, poderá criar um link simbólico para Python3.x o Python 3.7.
> - Os tópicos Conceitos básicos de conectores (console) e Conceitos básicos de conectores (CLI) contêm etapas detalhadas que mostram como configurar e implantar um exemplo do conector de notificações do Twilio.

1. Certifique-se de cumprir os requisitos para o conector.
2. Crie e publique uma função do Lambda que envie dados de entrada para o conector.

   Salve o código de exemplo como arquivo PY. Baixe e descompacte o SDK do AWS IoT Greengrass Core para Python. Crie então um pacote zip que contenha o arquivo PY e a pasta `greengrasssdk` no nível raiz. Este pacote zip é o pacote de implantação que você transfere por upload para o AWS Lambda.

   Depois de criar a função do Lambda Python 3.7, publique uma versão de função e crie um alias.
3. Configure o grupo do Greengrass.

   a. Adicione a função do Lambda pelo seu alias (recomendado). Configure o ciclo de vida do Lambda como de longa duração (ou `"Pinned": true` na CLI).

   b. Adicione o recurso de segredode segredo necessário e conceda acesso de leitura à função do Lambda.

   c. Adicione o conector e configure seus parâmetros.

   d. Adicione assinaturas que permitam que o conector receba dados de entrada e envie dados de saída em filtros de tópico compatíveis.

      - Defina a função do Lambda como origem, o conector como destino e use um filtro de tópico de entrada compatível.
      - Defina o conector como origem, o AWS IoT Core como destino, e use um filtro de tópico de saída compatível. Você usa essa assinatura para visualizar mensagens de status no AWS IoT console.
4. Implante o grupo.
5. No AWS IoT console, na página Teste, inscreva-se no tópico de dados de saída para ver as mensagens de status do conector. A função de exemplo do Lambda é de longa duração e começa a enviar mensagens imediatamente após o grupo ser implantado.

   Ao finalizar o teste, você pode definir o ciclo de vida do Lambda como sob demanda (ou `"Pinned": false` na CLI) e implantar o grupo. Isso impede o envio de mensagens pela função.

Exemplo

O exemplo a seguir da função do Lambda envia uma mensagem de entrada para o conector.

```
import greengrasssdk
import time
import json

iot_client = greengrasssdk.client('iot-data')
send_topic = 'splunk/logs/put'

def create_request_with_all_fields():
    return {
        "request": {
            "event": "Access log test message."
        },
        "id" : "req_123"
    }

def publish_basic_message():
    messageToPublish = create_request_with_all_fields()
    print("Message To Publish: ", messageToPublish)
    iot_client.publish(topic=send_topic,
        payload=json.dumps(messageToPublish))

publish_basic_message()

def lambda_handler(event, context):
    return
```

Licenças

Esse conector é liberado de acordo com o Contrato de licença de software do Greengrass Core.

Changelog

A tabela a seguir descreve as alterações em cada versão do conector.

Versão	Alterações
4	Adicionado o parâmetro `IsolationMode` para configurar o modo de conteinerização para o conector.

Versão	Alterações
3	Atualização do runtime do Lambda para Python 3.7, o que altera o requisito de runtime.
2	Corrija para reduzir o registro excessivo.
1	Versão inicial.

Um grupo do Greengrass só pode conter uma versão do conector por vez. Para obter informações sobre como fazer upgrade de uma versão do conector, consulte the section called “Atualizar a versões do conector”.

Consulte também

- Integrar a serviços e protocolos usando conectores
- the section called “Conceitos básicos de conectores (console)”
- the section called “Conceitos básicos de conectores (CLI)”

Conector de notificações Twilio

> **Warning**
>
> Esse conector passou para a fase de vida útil estendida e AWS IoT Greengrass não lançará atualizações que forneçam atributos, aprimoramentos para atributos existentes, patches de segurança ou correções de erros. Para obter mais informações, consulte AWS IoT Greengrass Version 1 política de manutenção.

O conector de notificações Twilio faz chamadas telefônicas automatizadas ou envia mensagens SMS por meio do Twilio. Você pode usar esse conector para enviar notificações em resposta a eventos no grupo do Greengrass. Para chamadas telefônicas, o conector pode encaminhar uma mensagem de voz para o destinatário.

Esse conector recebe informações de mensagens do Twilio em um tópico MQTT e aciona uma notificação do Twilio.

> **Note**
>
> Para obter um tutorial que mostra como usar o conector de notificações Twilio, consulte the section called “Conceitos básicos de conectores (console)” ou the section called “Conceitos básicos de conectores (CLI)”.

Esse conector tem as seguintes versões.

Versão	ARN
5	`arn:aws:greengrass:  region::/connectors/TwilioNotifications/versions/5`
4	`arn:aws:greengrass:  region::/connectors/TwilioNotifications/versions/4`
3	`arn:aws:greengrass:  region::/connectors/TwilioNotifications/versions/3`
2	`arn:aws:greengrass:  region::/connectors/TwilioNotifications/versions/2`
1	`arn:aws:greengrass:  region::/connectors/TwilioNotifications/versions/1`

Para obter informações sobre alterações de versão, consulte o Changelog.

Requisitos

Esse conector tem os seguintes requisitos:

Version 4 - 5

- AWS IoT Greengrass Software principal v1.9.3 ou posterior. AWS IoT Greengrass deve ser configurado para oferecer suporte a segredos locais, conforme descrito em Requisitos de segredos.

  > **Note**
  >
  > Este requisito inclui permitir o acesso aos seus segredos do Secrets Manager. Se você estiver usando a função de serviço padrão do Greengrass, o Greengrass terá permissão para obter os valores de segredos com nomes que começam com greengrass-.

- Python, versão 3.7 ou 3.8, instalado no dispositivo de núcleo e adicionado à variável de ambiente PATH.

  > **Note**
  >
  > Para usar o Python 3.8, execute o comando a seguir para criar um symblink da pasta de instalação padrão do Python 3.7 para os binários instalados do Python 3.8.
  >
  > ```
  > sudo ln -s path-to-python-3.8/python3.8 /usr/bin/python3.7
  > ```
  >
  > Isso configura seu dispositivo para atender ao requisito Python para AWS IoT Greengrass.

- Um SID, token de autenticação e número de telefone da conta Twilio. Twilio-enabled Depois de criar um projeto do Twilio, esses valores estarão disponíveis no painel do projeto.

  > **Note**
  >
  > Você pode usar uma conta de teste do Twilio. Se você estiver usando uma conta de teste, você deve adicionar números de telefone que não são de destinatários do Twilio a uma lista de números de telefone verificados. Para obter mais informações, consulte Como trabalhar com a sua conta de teste gratuita do Twilio.

- Um segredo de tipo de texto AWS Secrets Manager que armazena o token de autenticação do Twilio. Para obter mais informações, consulte Criar um segredo básico no Guia do usuário do AWS Secrets Manager .

  > **Note**
  >
  > Para criar o segredo no console do Secrets Manager, insira seu token na guia Texto simples. Não inclua aspas ou outra formatação. Na API, especifique o token como o valor da propriedade `SecretString`.

- Um recurso de segredo no grupo do Greengrass que faz referência ao segredo do Secrets Manager. Para obter mais informações, consulte Implantar segredos no núcleo.

Versions 1 - 3

- AWS IoT Greengrass Software principal v1.7 ou posterior. AWS IoT Greengrass deve ser configurado para oferecer suporte a segredos locais, conforme descrito em Requisitos de segredos.

  > **Note**
  >
  > Este requisito inclui permitir o acesso aos seus segredos do Secrets Manager. Se você estiver usando a função de serviço padrão do Greengrass, o Greengrass terá permissão para obter os valores de segredos com nomes que começam com greengrass-.

- Python versão 2.7 instalado no dispositivo de núcleo e adicionado à variável de ambiente PATH.
- Um SID, token de autenticação e número de telefone da conta Twilio. Twilio-enabled Depois de criar um projeto do Twilio, esses valores estarão disponíveis no painel do projeto.

  > **Note**
  >
  > Você pode usar uma conta de teste do Twilio. Se você estiver usando uma conta de teste, você deve adicionar números de telefone que não são de destinatários do Twilio a uma lista de números de telefone verificados. Para obter mais informações, consulte Como trabalhar com a sua conta de teste gratuita do Twilio.

- Um segredo de tipo de texto AWS Secrets Manager que armazena o token de autenticação do Twilio. Para obter mais informações, consulte Criar um segredo básico no Guia do usuário do AWS Secrets Manager .

  > **Note**
  >
  > Para criar o segredo no console do Secrets Manager, insira seu token na guia Texto simples. Não inclua aspas ou outra formatação. Na API, especifique o token como o valor da propriedade `SecretString`.

- Um recurso de segredo no grupo do Greengrass que faz referência ao segredo do Secrets Manager. Para obter mais informações, consulte Implantar segredos no núcleo.

Parâmetros do conector

Esse conector oferece os seguintes parâmetros.

Version 5

`TWILIO_ACCOUNT_SID`

O SID da conta do Twilio que é usado para invocar a API do Twilio.

Nome de exibição no AWS IoT console: SID da conta Twilio

Obrigatório: `true`

Digite: `string`

Padrão válido: `.+`

`TwilioAuthTokenSecretArn`

O ARN do segredo do Secrets Manager que armazena o token de autenticação do Twilio.

> **Note**
>
> É usado para acessar o valor do segredo local no núcleo.

Nome de exibição no AWS IoT console: ARN do segredo do token de autenticação do Twilio

Obrigatório: `true`

Digite: `string`

Padrão válido: `arn:aws:secretsmanager:[a-z0-9\-]+:[0-9]{12}:secret:([a-zA-Z0-9\\]+/)*[a-zA-Z0-9/_+=,.@\-]+-[a-zA-Z0-9]+`

`TwilioAuthTokenSecretArn-ResourceId`

O ID do recurso de segredo no grupo do Greengrass que faz referência ao segredo para o token de autenticação do Twilio.

Nome de exibição no AWS IoT console: recurso de token de autenticação Twilio

Obrigatório: `true`

Digite: `string`

Padrão válido: `.+`

`DefaultFromPhoneNumber`

O número de Twilio-enabled telefone padrão que o Twilio usa para enviar mensagens. O Twilio usa esse número para iniciar a mensagem ou chamada.

- Se você não configurar um número de telefone padrão, deverá especificar um número de telefone na propriedade `from_number` no corpo da mensagem de entrada.
- Se você configurar um número de telefone padrão, poderá substituir o padrão especificando a propriedade `from_number` no corpo da mensagem de entrada.

Nome de exibição no AWS IoT console: padrão a partir do número de telefone

Obrigatório: `false`

Digite: `string`

Padrão válido: `^$|\+[0-9]+`

`IsolationMode`

O modo de conteinerização para este conector. O padrão é`GreengrassContainer`, o que significa que o conector é executado em um ambiente de execução isolado dentro do AWS IoT Greengrass contêiner.

> **Note**
>
> A configuração padrão de conteinerização para o grupo não se aplica aos conectores.

Nome de exibição no AWS IoT console: modo de isolamento de contêiner

Obrigatório: `false`

Digite: `string`

Valores válidos: `GreengrassContainer` ou `NoContainer`

Padrão válido: `^NoContainer$|^GreengrassContainer$`

Version 1 - 4

`TWILIO_ACCOUNT_SID`

O SID da conta do Twilio que é usado para invocar a API do Twilio.

Nome de exibição no AWS IoT console: SID da conta Twilio

Obrigatório: `true`

Digite: `string`

Padrão válido: `.+`

`TwilioAuthTokenSecretArn`

O ARN do segredo do Secrets Manager que armazena o token de autenticação do Twilio.

> **Note**
>
> É usado para acessar o valor do segredo local no núcleo.

Nome de exibição no AWS IoT console: ARN do segredo do token de autenticação do Twilio

Obrigatório: `true`

Digite: `string`

Padrão válido: `arn:aws:secretsmanager:[a-z0-9\-]+:[0-9]{12}:secret:([a-zA-Z0-9\\]+/)*[a-zA-Z0-9/_+=,.@\-]+-[a-zA-Z0-9]+`

`TwilioAuthTokenSecretArn-ResourceId`

O ID do recurso de segredo no grupo do Greengrass que faz referência ao segredo para o token de autenticação do Twilio.

Nome de exibição no AWS IoT console: recurso de token de autenticação Twilio

Obrigatório: `true`

Digite: `string`

Padrão válido: `.+`

`DefaultFromPhoneNumber`

O número de Twilio-enabled telefone padrão que o Twilio usa para enviar mensagens. O Twilio usa esse número para iniciar a mensagem ou chamada.

- Se você não configurar um número de telefone padrão, deverá especificar um número de telefone na propriedade `from_number` no corpo da mensagem de entrada.
- Se você configurar um número de telefone padrão, poderá substituir o padrão especificando a propriedade `from_number` no corpo da mensagem de entrada.

Nome de exibição no AWS IoT console: padrão a partir do número de telefone

Obrigatório: `false`

Digite: `string`

Padrão válido: `^$|\+[0-9]+`

Exemplo de criação de conector (AWS CLI)

O seguinte comando da CLI de exemplo cria um `ConnectorDefinition` com uma versão inicial que contém o conector de notificações Twilio.

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --initial-
version '{
```

```
    "Connectors": [
        {
            "Id": "MyTwilioNotificationsConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/
TwilioNotifications/versions/5",
            "Parameters": {
                "TWILIO_ACCOUNT_SID": "abcd12345xyz",
                "TwilioAuthTokenSecretArn": "arn:aws:secretsmanager:region:account-
id:secret:greengrass-secret-hash",
                "TwilioAuthTokenSecretArn-ResourceId": "MyTwilioSecret",
                "DefaultFromPhoneNumber": "+19999999999",
                "IsolationMode" : "GreengrassContainer"
            }
        }
    ]
}'
```

Para obter tutoriais que mostram como adicionar o conector de notificações Twilio a um grupo, consulte the section called “Conceitos básicos de conectores (CLI)” e the section called “Conceitos básicos de conectores (console)”.

Dados de entrada

Esse conector aceita informações de mensagens do Twilio em dois tópicos MQTT. As mensagens de entrada devem estar no formato JSON.

- Informações de mensagens SMS no tópico `twilio/txt`.
- Informações de mensagens de voz no tópico `twilio/call`.

> **Note**
>
> A carga da mensagem de entrada pode incluir uma mensagem SMS (`message`) ou mensagem de voz (`voice_message_location`), mas não as duas.

Filtro de tópico: **`twilio/txt`**

Propriedades de mensagens

`request`

Informações sobre a notificação do Twilio.

Obrigatório: `true`

Tipo: `object` que inclui as seguintes propriedades:

`recipient`

O destinatário da mensagem. Somente um destinatário tem suporte.

Obrigatório: `true`

Tipo: `object` que inclui as seguintes propriedades:

`name`

O nome do destinatário.

Obrigatório: `true`

Digite: `string`

Padrão válido: `.*`

`phone_number`

O número de telefone do destinatário.

Obrigatório: `true`

Digite: `string`

Padrão válido: `\+[1-9]+`

`message`

O conteúdo de texto da mensagem SMS. Somente mensagens SMS têm suporte neste tópico. Para mensagens de voz, use `twilio/call`.

Obrigatório: `true`

Digite: `string`

Padrão válido: `.+`

`from_number`

O número de telefone do remetente. O Twilio usa esse número de telefone para iniciar a mensagem. Essa propriedade será necessária se

o parâmetro `DefaultFromPhoneNumber` não estiver configurado. Se `DefaultFromPhoneNumber` estiver configurado, você poderá usar essa propriedade para substituir o padrão.

Obrigatório: `false`

Digite: `string`

Padrão válido: `\+[1-9]+`

`retries`

O número de novas tentativas. O padrão é 0.

Obrigatório: `false`

Digite: `integer`

`id`

Um ID arbitrário para a solicitação. Essa propriedade é usada para mapear a solicitação de entrada para uma resposta de saída.

Obrigatório: `true`

Digite: `string`

Padrão válido: `.+`

Exemplo de entrada

```
{
    "request": {
        "recipient": {
            "name": "Darla",
            "phone_number": "+12345000000",
            "message": "Hello from the edge"
        },
        "from_number": "+19999999999",
        "retries": 3
    },
    "id": "request123"
}
```

Filtro de tópico: **`twilio/call`**

Propriedades de mensagens

`request`

Informações sobre a notificação do Twilio.

Obrigatório: `true`

Tipo: `object` que inclui as seguintes propriedades:

`recipient`

O destinatário da mensagem. Somente um destinatário tem suporte.

Obrigatório: `true`

Tipo: `object` que inclui as seguintes propriedades:

`name`

O nome do destinatário.

Obrigatório: `true`

Digite: `string`

Padrão válido: `.+`

`phone_number`

O número de telefone do destinatário.

Obrigatório: `true`

Digite: `string`

Padrão válido: `\+[1-9]+`

`voice_message_location`

O URL do conteúdo de áudio para a mensagem de voz. Deve estar no formato TwiML. Somente mensagens de voz têm suporte neste tópico. Para mensagens SMS, use `twilio/txt`.

Obrigatório: `true`

Digite: `string`

Padrão válido: `.+`

`from_number`

O número de telefone do remetente. O Twilio usa esse número de telefone para iniciar a mensagem. Essa propriedade será necessária se o parâmetro `DefaultFromPhoneNumber` não estiver configurado. Se `DefaultFromPhoneNumber` estiver configurado, você poderá usar essa propriedade para substituir o padrão.

Obrigatório: `false`

Digite: `string`

Padrão válido: `\+[1-9]+`

`retries`

O número de novas tentativas. O padrão é 0.

Obrigatório: `false`

Digite: `integer`

`id`

Um ID arbitrário para a solicitação. Essa propriedade é usada para mapear a solicitação de entrada para uma resposta de saída.

Obrigatório: `true`

Digite: `string`

Padrão válido: `.+`

Exemplo de entrada

```
{
    "request": {
        "recipient": {
```

```
            "name": "Darla",
            "phone_number": "+12345000000",
            "voice_message_location": "https://some-public-TwiML"
        },
        "from_number": "+19999999999",
        "retries": 3
    },
    "id": "request123"
}
```

Dados de saída

O conector publica informações de status como dados de saída em um tópico MQTT.

Filtro de tópico na assinatura

```
twilio/message/status
```

Exemplo de resultado: sucesso

```
{
    "response": {
        "status": "success",
        "payload": {
            "from_number": "+19999999999",
            "messages": {
                "message_status": "queued",
                "to_number": "+12345000000",
                "name": "Darla"
            }
        }
    },
    "id": "request123"
}
```

Exemplo de resultado: falha

```
{
    "response": {
        "status": "fail",
        "error_message": "Recipient name cannot be None",
        "error": "InvalidParameter",
```

```
        "payload": None
        }
    },
    "id": "request123"
}
```

A propriedade `payload` na saída é a resposta da API do Twilio quando a mensagem é enviada. Se o conector detecta que os dados de entrada são inválidos (por exemplo, se ele não especifica um campo de entrada obrigatório), o conector retorna um erro e define o valor como `None`. Veja as cargas de exemplo a seguir:

```
{
    'from_number':'+19999999999',
    'messages': {
        'name':'Darla',
        'to_number':'+12345000000',
        'message_status':'undelivered'
    }
}
```

```
{
    'from_number':'+19999999999',
    'messages': {
        'name':'Darla',
        'to_number':'+12345000000',
        'message_status':'queued'
    }
}
```

Exemplo de uso

Use as seguintes etapas de alto nível para configurar um exemplo de função do Lambda Python 3.7 que pode ser usado para testar o conector.

> **Note**
>
> Os tópicos the section called “Conceitos básicos de conectores (console)” e the section called “Conceitos básicos de conectores (CLI)” contêm etapas de ponta a ponta que mostram como configurar, implantar e testar o conector de notificações do Twilio.

1. Certifique-se de cumprir os requisitos para o conector.
2. Crie e publique uma função do Lambda que envie dados de entrada para o conector.

   Salve o código de exemplo como arquivo PY. Baixe e descompacte o SDK do AWS IoT Greengrass Core para Python. Crie então um pacote zip que contenha o arquivo PY e a pasta `greengrasssdk` no nível raiz. Este pacote zip é o pacote de implantação que você transfere por upload para o AWS Lambda.

   Depois de criar a função do Lambda Python 3.7, publique uma versão de função e crie um alias.
3. Configure o grupo do Greengrass.

   a. Adicione a função do Lambda pelo seu alias (recomendado). Configure o ciclo de vida do Lambda como de longa duração (ou `"Pinned": true` na CLI).

   b. Adicione o recurso de segredode segredo necessário e conceda acesso de leitura à função do Lambda.

   c. Adicione o conector e configure seus parâmetros.

   d. Adicione assinaturas que permitam que o conector receba dados de entrada e envie dados de saída em filtros de tópico compatíveis.

      - Defina a função do Lambda como origem, o conector como destino e use um filtro de tópico de entrada compatível.
      - Defina o conector como origem, o AWS IoT Core como destino, e use um filtro de tópico de saída compatível. Você usa essa assinatura para visualizar mensagens de status no AWS IoT console.
4. Implante o grupo.
5. No AWS IoT console, na página Teste, inscreva-se no tópico de dados de saída para ver as mensagens de status do conector. A função de exemplo do Lambda é de longa duração e começa a enviar mensagens imediatamente após o grupo ser implantado.

   Ao finalizar o teste, você pode definir o ciclo de vida do Lambda como sob demanda (ou `"Pinned": false` na CLI) e implantar o grupo. Isso impede o envio de mensagens pela função.

Exemplo

O exemplo a seguir da função do Lambda envia uma mensagem de entrada para o conector. Este exemplo aciona uma mensagem de texto.

```
import greengrasssdk
import json

iot_client = greengrasssdk.client('iot-data')
TXT_INPUT_TOPIC = 'twilio/txt'
CALL_INPUT_TOPIC = 'twilio/call'

def publish_basic_message():

    txt = {
        "request": {
            "recipient" : {
                "name": "Darla",
                "phone_number": "+12345000000",
                "message": 'Hello from the edge'
            },
            "from_number" : "+19999999999"
        },
        "id" : "request123"
    }

    print("Message To Publish: ", txt)

    client.publish(topic=TXT_INPUT_TOPIC,
                   payload=json.dumps(txt))

publish_basic_message()

def lambda_handler(event, context):
    return
```

Licenças

O conector Twilio Notifications inclui os seguintes terceiros: software/licensing

- twilio-python/MIT

Esse conector é liberado de acordo com o Contrato de licença de software do Greengrass Core.

Changelog

A tabela a seguir descreve as alterações em cada versão do conector.

Versão	Alterações
5	Adicionado o parâmetro `IsolationMode` para configurar o modo de conteinerização para o conector.
4	Atualização do runtime do Lambda para Python 3.7, o que altera o requisito de runtime.
3	Corrija para reduzir o registro excessivo.
2	Correções de bugs secundários e melhorias.
1	Versão inicial.

Um grupo do Greengrass só pode conter uma versão do conector por vez. Para obter informações sobre como fazer upgrade de uma versão do conector, consulte the section called “Atualizar a versões do conector”.

Consulte também

- Integrar a serviços e protocolos usando conectores
- the section called “Conceitos básicos de conectores (console)”
- the section called “Conceitos básicos de conectores (CLI)”
- Referência da API do Twilio

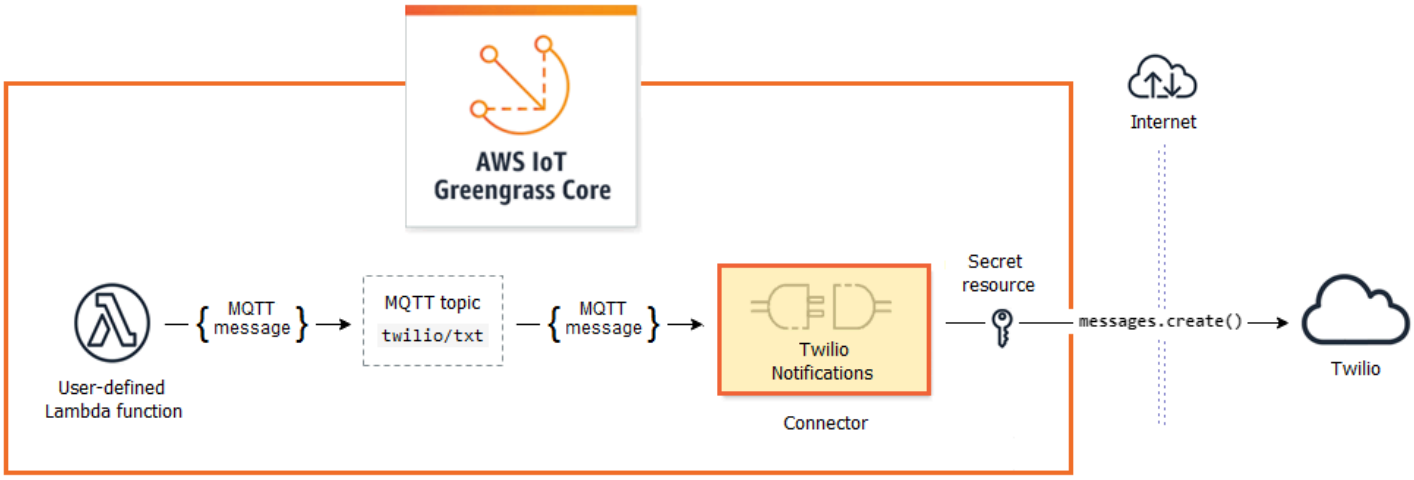
Conceitos básicos de conectores do Greengrass (console)

Esse recurso está disponível para AWS IoT Greengrass Core v1.7 e versões posteriores.

Este tutorial mostra como usar o Console de gerenciamento da AWS para trabalhar com conectores.

Use conectores para acelerar o ciclo de vida de desenvolvimento. Os conectores são módulos pré-criados e reutilizáveis que podem facilitar a interação com serviços, protocolos e recursos. Eles podem ajudar você a implantar a lógica de negócios para Dispositivos Greengrass mais rapidamente. Para obter mais informações, consulte Integrar a serviços e protocolos usando conectores.

Neste tutorial, você configura e implanta o conector de notificações do Twilio. O conector recebe informações de mensagem do Twilio como dados de entrada e aciona uma mensagem de texto do Twilio. O fluxo de dados é exibido no diagrama a seguir.



Depois de configurar o conector, crie uma função do Lambda e uma assinatura.

- A função avalia os dados simulados do sensor de temperatura. Ele publica, de forma condicional, as informações de mensagem do Twilio para um tópico MQTT. Esse é o tópico que o conector assina.
- A assinatura permite que a função publique no tópico e que o conector receba os dados do tópico.

O conector de notificações do Twilio exige um token de autenticação do Twilio para interagir com a API do Twilio. O token é um segredo de tipo de texto criado AWS Secrets Manager e referenciado a partir de um recurso de grupo. Isso permite AWS IoT Greengrass criar uma cópia local do segredo no núcleo do Greengrass, onde ele é criptografado e disponibilizado para o conector. Para obter mais informações, consulte Implantar segredos no núcleo.

O tutorial contém as seguintes etapas de nível elevado:

1. Crie um segredo do Secrets Manager
2. Adicionar um recurso de segredo a um grupo
3. Adicionar um conector ao grupo
4. Crie um pacote de implantação para a função do Lambda
5. Criar uma função do Lambda

6. Adicionar uma função ao grupo
7. Adicionar assinaturas ao grupo
8. Implantar o grupo
9. the section called “Testar a solução”

O tutorial levará aproximadamente 20 minutos para ser concluído.

Pré-requisitos

Para concluir este tutorial, é necessário:

- Um grupo do Greengrass e um núcleo do Greengrass (versão 1.9.3 ou posterior). Para saber como criar um grupo e núcleo Greengrass, consulte Começando com AWS IoT Greengrass. O tutorial Conceitos básicos também inclui etapas para instalar o software do núcleo do AWS IoT Greengrass .
- Python 3.7 instalado no dispositivo AWS IoT Greengrass principal.
- AWS IoT Greengrass devem ser configurados para oferecer suporte a segredos locais, conforme descrito em Requisitos de segredos.

> **Note**
>
> Este requisito inclui permitir o acesso aos seus segredos do Secrets Manager. Se você estiver usando a função de serviço padrão do Greengrass, o Greengrass terá permissão para obter os valores de segredos com nomes que começam com greengrass-.

- Um SID de conta Twilio, token de autenticação e número de telefone. Twilio-enabled Depois de criar um projeto do Twilio, esses valores estarão disponíveis no painel do projeto.

> **Note**
>
> Você pode usar uma conta de teste do Twilio. Se você estiver usando uma conta de teste, você deve adicionar números de telefone que não são de destinatários do Twilio a uma lista de números de telefone verificados. Para obter mais informações, consulte Como trabalhar com a sua conta de teste gratuita do Twilio.

Etapa 1: Criar um segredo do Secrets Manager

Nesta etapa, você usa o AWS Secrets Manager console para criar um segredo de tipo de texto para seu token de autenticação do Twilio.

1. Faça login no console do AWS Secrets Manager.

   > **Note**
   >
   > Para obter mais informações sobre esse processo, consulte Etapa 1: Criar e armazenar seu segredo no AWS Secrets Manager no Guia do usuário do AWS Secrets Manager .

2. Selecione Armazenar um novo segredo.
3. Em Escolher tipo de segredo, selecione Outro tipo de segredo.
4. Em Especificar os key/value pares a serem armazenados para esse segredo, na guia Texto sem formatação, insira seu token de autenticação do Twilio. Remova toda a formatação JSON e insira apenas o valor do token.
5. Mantenha aws/secretsmanager selecionado para a chave de criptografia e, em seguida, escolha Avançar.

   > **Note**
   >
   > Você não será cobrado AWS KMS se usar a chave AWS gerenciada padrão que o Secrets Manager cria em sua conta.

6. Em Secret name, digite **`greengrass-TwilioAuthToken`** e selecione Next.

   > **Note**
   >
   > Por padrão, a função de serviço do Greengrass permite AWS IoT Greengrass obter o valor dos segredos com nomes que começam com greengrass-. Para obter mais informações, consulte os requisitos de segredos.

7. A alternância não é necessária neste tutorial, portanto, selecione Desabilitar a alternância automática e, em seguida, Avançar.
8. Na página Review (Revisar), revise as configurações e selecione Store (Armazenar).

Em seguida, crie um recurso de segredo no seu grupo do Greengrass que faz referência ao segredo.

Etapa 2: Adicionar um recurso de segredo a um grupo do Greengrass

Nesta etapa, você adiciona um recurso de segredo ao grupo do Greengrass. Esse recurso é uma referência ao segredo que você criou na etapa anterior.

1. No painel de navegação do AWS IoT console, em Gerenciar, expanda dispositivos Greengrass e escolha Grupos (V1).
2. Selecione o grupo ao qual você deseja adicionar o recurso de segredo.
3. Na página de configuração do grupo, selecione a guia Recursos e, em seguida, role para baixo até a seção Segredos. A seção Segredos exibe os recursos de segredos que pertencem ao grupo. Você pode adicionar, editar e remover recursos de segredos nessa seção.

   > **Note**
   >
   > Como alternativa, o console permite que você crie um segredo e um recurso de segredo ao configurar um conector ou uma função do Lambda. Você pode fazer isso na página Configurar parâmetros do conector ou na página Recursos da função do Lambda.

4. Selecione Adicionar na seção Segredos.
5. Na página Adicionar um recurso de segredo, insira **`MyTwilioAuthToken`** em Nome do recurso.
6. Para o Segredo, escolha greengrass-. TwilioAuthToken
7. Na seção Selecionar rótulos (Opcional), o rótulo de preparação AWSCURRENT representa a versão mais recente do segredo. Esse rótulo é sempre incluído em um recurso de segredo.

   > **Note**
   >
   > Somente o rótulo AWSCURRENT é necessário neste tutorial. Você também pode incluir rótulos que a função do Lambda ou o connector exige.

8. Selecione Adicionar recurso.

Etapa 3: Adicionar um conector ao grupo do Greengrass

Nesta etapa, você configura parâmetros para o conector de notificações Twilio e o adiciona ao grupo.

1. Na página de configuração do grupo, selecione Connectors e, em seguida, Add a connector.
2. Na página Adicionar conector, selecione Notificações do Twilio.
3. Selecione a versão .
4. Na seção Configuração:
   - Para Recurso de token de autenticação do Twilio, insira o recurso que você criou na etapa anterior.

     > **Note**
     >
     > Quando você insere o recurso, a propriedade ARN of Twilio auth token secret é preenchida para você.

   - Em Padrão do número de telefone, insira seu número de Twilio-enabled telefone.
   - Para Twilio account SID, insira seu SID da conta do Twilio.
5. Selecione Adicionar recurso.

Etapa 4: crie um pacote de implantação para a função do Lambda

Para criar uma função do Lambda, você deve, primeiro, criar um pacote de implantação da função do Lambda que contenha o código da função e as dependências. As funções do Lambda do Greengrass exigem o SDK do AWS IoT Greengrass Core para tarefas como comunicação com mensagens MQTT no ambiente de núcleo e acesso a segredos locais. Este tutorial cria uma função Python, então você usa a versão Python do SDK no pacote de implantação.

1. Na página de downloads do AWS IoT Greengrass Core SDK, baixe o AWS IoT Greengrass Core SDK para Python em seu computador.
2. Descompacte o pacote obtido por download para obter o SDK. O SDK é a pasta do `greengrasssdk`.
3. Salve a seguinte função do código Python em um arquivo local denominado `temp_monitor.py`.

```
import greengrasssdk
```

```
import json
import random

client = greengrasssdk.client('iot-data')

# publish to the Twilio Notifications connector through the twilio/txt topic
def function_handler(event, context):
    temp = event['temperature']

    # check the temperature
    # if greater than 30C, send a notification
    if temp > 30:
        data = build_request(event)
        client.publish(topic='twilio/txt', payload=json.dumps(data))
        print('published:' + str(data))

    print('temperature:' + str(temp))
    return

# build the Twilio request from the input data
def build_request(event):
    to_name = event['to_name']
    to_number = event['to_number']
    temp_report = 'temperature:' + str(event['temperature'])

    return {
        "request": {
            "recipient": {
                "name": to_name,
                "phone_number": to_number,
                "message": temp_report
            }
        },
        "id": "request_" + str(random.randint(1,101))
    }
```

4. Comprima os seguintes itens em um arquivo denominado `temp_monitor_python.zip`. Ao criar o arquivo ZIP, inclua apenas o código e suas dependências, e não a pasta que contém os arquivos.

   - temp_monitor.py. Lógica do aplicativo.
   - greengrasssdk. Biblioteca necessária para funções Python do Lambda do Greengrass que publicam mensagens MQTT.

Esse é o pacote de implantação de sua função do Lambda.

Agora, crie uma função do Lambda que usa o pacote de implantação.

Etapa 5: Crie uma função do Lambda no AWS Lambda console

Nesta etapa, você usa o AWS Lambda console para criar uma função do Lambda e configurá-la para usar seu pacote de implantação. Depois, publique uma versão da função e crie um alias.

1. Primeiro, crie a função do Lambda.

   a. No Console de gerenciamento da AWS, escolha Serviços e abra o AWS Lambda console.

   b. Selecione Criar função e, em seguida, selecione Criar do zero.

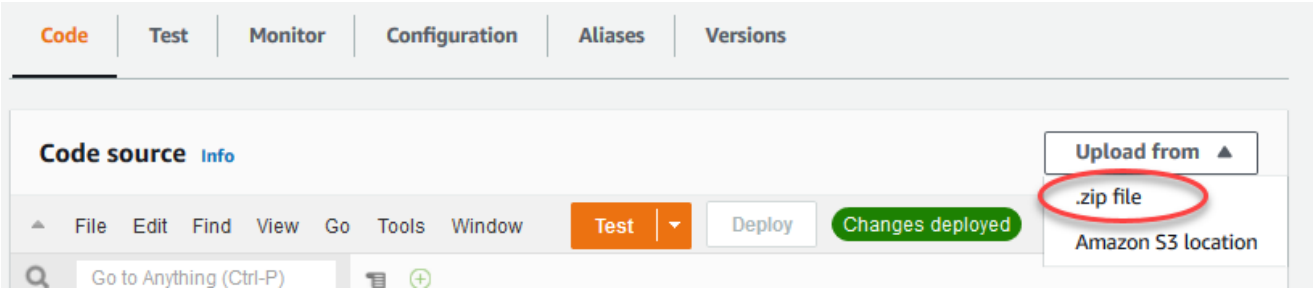
   c. Na seção Basic information (Informações básicas), use os seguintes valores:

      - Em Function name (Nome da função), insira **`TempMonitor`**.
      - Em Runtime (Tempo de execução), selecione Python 3.7.
      - Em Permissões, mantenha a configuração padrão. Isso cria uma função de execução que concede permissões básicas do Lambda. Essa função não é usada por AWS IoT Greengrass.

   d. Na parte inferior da página, selecione Create function.

2. Em seguida, registre o manipulador e faça upload do seu pacote de implantação da função do Lambda.

   a. Na guia Código, em Fonte do código, selecione Fazer upload a partir de. No menu suspenso, selecione o arquivo .zip.



   b. Selecione Upload e, em seguida, selecione seu pacote de implantação `temp_monitor_python.zip`. Selecione Salvar.

c. Na guia Código da função, em Configurações de runtime, selecione Editar e insira os valores a seguir.

   - Em Runtime (Tempo de execução), selecione Python 3.7.
   - Em Handler (Manipulador), insira **`temp_monitor.function_handler`**.

d. Escolha Salvar.

> **Note**
>
> O botão Testar no AWS Lambda console não funciona com essa função. O SDK AWS IoT Greengrass Core não contém módulos necessários para executar suas funções do Greengrass Lambda de forma independente no console. AWS Lambda Esses módulos (por exemplo, `greengrass_common`) são fornecidos às funções depois de serem implantados no núcleo do Greengrass.

3. Agora, publique a primeira versão da sua função do Lambda e crie um alias para a versão.

> **Note**
>
> Os grupos do Greengrass podem fazer referência a uma função do Lambda por alias (recomendado) ou por versão. Usar um alias facilita o gerenciamento de atualizações de código porque você não precisa alterar a tabela de assinaturas nem a definição do grupo ao atualizar a função do código. Em vez disso, você pode simplesmente apontar o alias para a nova versão da função.

a. No menu Actions, selecione Publish new version.

b. Em Version description (Descrição da versão), insira **`First version`** e, em seguida, selecione Publish (Publicar).

c. Na página de configuração TempMonitor: 1, no menu Ações, escolha Criar alias.

d. Na página Create a new alias, use os seguintes valores:

   - Em Nome, digite **`GG_TempMonitor`**.
   - Em Version, selecione 1.

> **Note**
>
> AWS IoT Greengrass não oferece suporte a aliases do Lambda para as versões $LATEST.

e. Selecione Create (Criar).

Agora você está pronto para adicionar a função do Lambda ao seu grupo do Greengrass.

Etapa 6: Adicionar uma função do Lambda ao grupo do Greengrass

Nesta etapa, você adiciona a função do Lambda ao grupo e configura o ciclo de vida e as variáveis de ambiente. Para obter mais informações, consulte the section called “Controlando a execução da função do Lambda do Greengrass”.

1. Na página de configuração do grupo, selecione a guia Funções do Lambda.
2. Em Minhas funções do Lambda, selecione Adicionar.
3. Na página Adicionar função do Lambda, escolha TempMonitor sua função do Lambda.
4. Para a versão da função Lambda, escolha Alias: GG_. TempMonitor
5. Selecione Adicionar função do Lambda.

Etapa 7: Adicionar assinaturas ao grupo do Greengrass

Nesta etapa, adicione uma assinatura que permita que a função do Lambda envie dados de entrada para o conector. O conector define os tópicos MQTT que assina, para que essa assinatura use um dos tópicos. Esse é o mesmo tópico no qual a função de exemplo publica.

Neste tutorial, você também cria assinaturas que permitem que a função receba leituras simuladas de temperatura AWS IoT e permita AWS IoT receber informações de status do conector.

1. Na página de configuração do grupo, selecione a guia Assinaturas e, em seguida, Adicionar assinatura.
2. Na página Criar uma assinatura, configure a origem e o destino, da seguinte forma:

   a. Em Tipo de origem, escolha Função Lambda e, em seguida, escolha TempMonitor.
   b. Em Tipo de destino, selecione Conector e, em seguida, selecione Notificações Twilio.

3. Para o Filtro de tópico, insira **`twilio/txt`**.
4. Selecione Criar assinatura.
5. Repita as etapas 1 a 4 para criar uma assinatura que permita AWS IoT publicar mensagens na função.

   a. Para Tipo de origem, selecione Serviço e, em seguida, selecione IoT Cloud.

   b. Em Selecionar um destino, escolha Função Lambda e, em seguida, escolha TempMonitor.

   c. Para Topic filter, insira **`temperature/input`**.
6. Repita as etapas 1 a 4 para criar uma assinatura que permita que o conector publique mensagens no AWS IoT.

   a. Em Tipo de origem, selecione Connector (Conector) e, em seguida, selecione Twilio Notifications (Notificações Twilio).

   b. Para Tipo de destino, selecione Serviço e, em seguida, IoT Cloud.

   c. Para Topic filter (Filtro de tópico), **`twilio/message/status`** é inserido para você. Este é o tópico predefinido no qual o conector publica.

Etapa 8: Implantar o grupo do Greengrass

Implante o grupo no dispositivo do núcleo.

1. Certifique-se de que o AWS IoT Greengrass núcleo esteja funcionando. Execute os seguintes comandos no seu terminal do Raspberry Pi, conforme necessário.

   a. Para verificar se o daemon está em execução:

   ```
   ps aux | grep -E 'greengrass.*daemon'
   ```

   Se a saída contém uma entrada `root` para `/greengrass/ggc/packages/`*`ggc-version`*`/bin/daemon`, o daemon está em execução.

   > **Note**
   >
   > A versão no caminho depende da versão do software AWS IoT Greengrass Core instalada em seu dispositivo principal.

   b. Para iniciar o daemon:

```
cd /greengrass/ggc/core/
sudo ./greengrassd start
```

2. Na página de configuração do grupo, selecione Implantar.
3. a. Na guia Funções do Lambda, na seção Funções do Lambda do sistema, selecione Detector de IP e selecione Editar.
   b. Na caixa de diálogo Editar configurações do detector IP, selecione Detectar e substituir automaticamente os endpoints do corretor MQTT.
   c. Selecione Salvar.

   Isso permite que os dispositivos adquiram automaticamente as informações de conectividade para o núcleo, como endereço IP, DNS e o número da porta. A detecção automática é recomendada, mas AWS IoT Greengrass também oferece suporte a endpoints especificados manualmente. Você só é solicitado pelo método de descoberta na primeira vez em que o grupo for implantado.

   > **Note**
   >
   > Se solicitado, conceda permissão para criar a função de serviço do Greengrass e associá-la à sua Conta da AWS na atual. Região da AWS Essa função permite AWS IoT Greengrass acessar seus recursos em AWS serviços.

   A página Deployments mostra a data e hora, ID de versão e status da implantação. Quando concluída, o status exibido para a implantação deve ser Concluída.

   Para obter ajuda sobre a solução de problemas, consulte Solução de problemas.

> **Note**
>
> Um grupo do Greengrass só pode conter uma versão do conector por vez. Para obter informações sobre como fazer upgrade de uma versão do conector, consulte the section called “Atualizar a versões do conector”.

Testar a solução

1. Na página inicial do AWS IoT console, escolha Testar.
2. Em Inscrever-se no tópico, use os seguintes valores e, em seguida, selecione Inscrever-se. O conector de notificações do Twilio publica informações de status nesse tópico.

Propriedade	Valor
Tópico de assinatura	twilio/message/status
Exibição da carga útil do MQTT	Exibir cargas úteis como strings

3. Para Publicar no tópico, use os valores a seguir e, em seguida selecione Publicar para invocar a função.

Propriedade	Valor
Tópico	temperature/input
Mensagem	*recipient-name* Substitua por um nome e *recipient-phone-number* pelo número de telefone do destinatário da mensagem de texto. Exemplo: +12345000000 `{ "to_name": " recipient-name ", "to_number": " recipient-phone-number ", "temperature": 31 }` Se você estiver usando uma conta de teste, você deve adicionar números de telefone que não são de destinatários do Twilio a uma lista de números de telefone verificados. Para obter mais informações, consulte Verificar seu número de telefone pessoal.

Se for bem-sucedido, o destinatário receberá a mensagem de texto e o console exibirá o `success` status dos dados de saída.

Agora, altere a `temperature` na mensagem de entrada para **29** e publique. Como isso é menor que 30, a TempMonitor função não aciona uma mensagem do Twilio.

Consulte também

- Integrar a serviços e protocolos usando conectores
- the section called “Conectores do Greengrass fornecidos pela AWS”

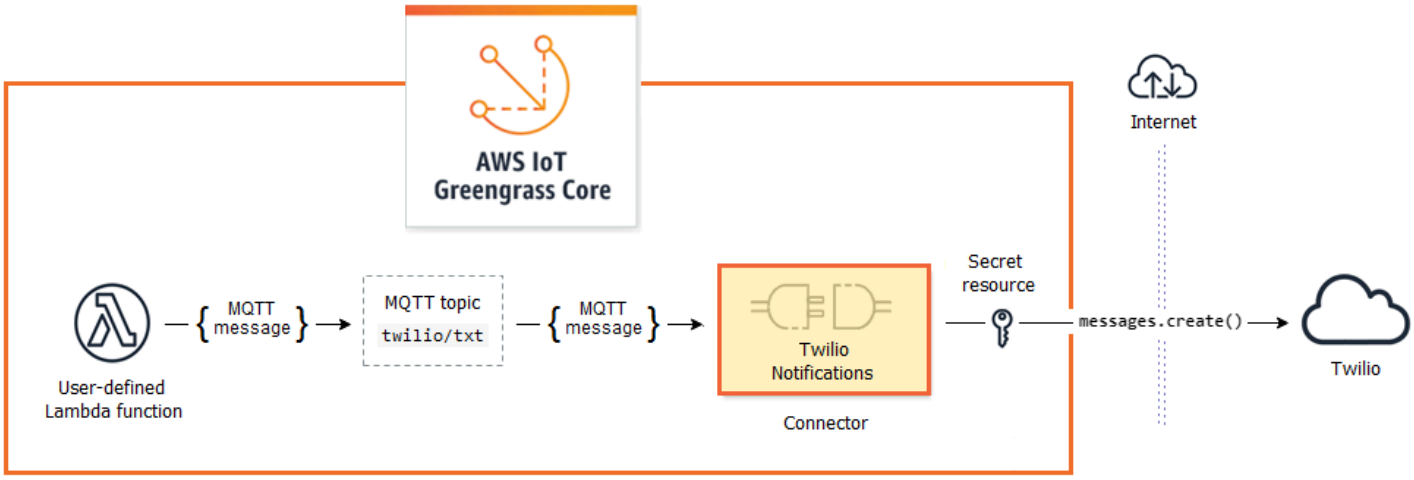
Conceitos básicos de conectores do Greengrass (CLI)

Esse recurso está disponível para AWS IoT Greengrass Core v1.7 e versões posteriores.

Este tutorial mostra como usar o AWS CLI para trabalhar com conectores.

Use conectores para acelerar o ciclo de vida de desenvolvimento. Os conectores são módulos pré-criados e reutilizáveis que podem facilitar a interação com serviços, protocolos e recursos. Eles podem ajudar você a implantar a lógica de negócios para Dispositivos Greengrass mais rapidamente. Para obter mais informações, consulte Integrar a serviços e protocolos usando conectores.

Neste tutorial, você configura e implanta o conector de notificações do Twilio. O conector recebe informações de mensagem do Twilio como dados de entrada e aciona uma mensagem de texto do Twilio. O fluxo de dados é exibido no diagrama a seguir.



Depois de configurar o conector, crie uma função do Lambda e uma assinatura.

- A função avalia os dados simulados do sensor de temperatura. Ele publica, de forma condicional, as informações de mensagem do Twilio para um tópico MQTT. Esse é o tópico que o conector assina.
- A assinatura permite que a função publique no tópico e que o conector receba os dados do tópico.

O conector de notificações do Twilio exige um token de autenticação do Twilio para interagir com a API do Twilio. O token é um segredo de tipo de texto criado AWS Secrets Manager e referenciado a partir de um recurso de grupo. Isso permite AWS IoT Greengrass criar uma cópia local do segredo no núcleo do Greengrass, onde ele é criptografado e disponibilizado para o conector. Para obter mais informações, consulte Implantar segredos no núcleo.

O tutorial contém as seguintes etapas de nível elevado:

1. Crie um segredo do Secrets Manager
2. Criar uma definição e uma versão de recurso
3. Criar uma definição e uma versão de conector
4. Crie um pacote de implantação para a função do Lambda
5. Criar uma função do Lambda
6. Criar uma definição e uma versão de função
7. Criar uma definição e uma versão de assinatura
8. Criar uma versão de grupo

9. Crie um implantação do .
10. the section called “Testar a solução”

O tutorial levará aproximadamente 30 minutos para ser concluído.

Usando a AWS IoT Greengrass API

É útil compreender os padrões a seguir ao trabalhar com componentes de grupo e grupos do Greengrass (por exemplo, os conectores, as funções e os recursos no grupo).

- No topo da hierarquia, um componente tem um objeto de definição que é um contêiner para objetos da versão. Por sua vez, uma versão é um contêiner para os conectores, as funções ou outros tipos de componentes.
- Ao implantar o núcleo do Greengrass, você implanta uma versão do grupo específica. A versão do grupo pode conter uma versão de cada tipo de componente. Um núcleo é necessário, mas os demais são incluídos conforme necessário.
- As versões são imutáveis, portanto, você deve criar novas versões quando desejar fazer alterações.

> **Tip**
>
> Se você receber um erro ao executar um AWS CLI comando, adicione o `--debug` parâmetro e execute novamente o comando para obter mais informações sobre o erro.

A AWS IoT Greengrass API permite criar várias definições para um tipo de componente. Por exemplo, você pode criar um objeto `FunctionDefinition` toda vez que criar uma `FunctionDefinitionVersion`, ou você pode adicionar novas versões a uma definição existente. Essa flexibilidade permite que você personalize seu sistema de gerenciamento de versão.

Pré-requisitos

Para concluir este tutorial, é necessário:

- Um grupo do Greengrass e um núcleo do Greengrass (versão 1.9.3 ou posterior). Para saber como criar um grupo e núcleo Greengrass, consulte Começando com AWS IoT Greengrass. O tutorial Conceitos básicos também inclui etapas para instalar o software do núcleo do AWS IoT Greengrass .

- Python 3.7 instalado no dispositivo AWS IoT Greengrass principal.
- AWS IoT Greengrass devem ser configurados para oferecer suporte a segredos locais, conforme descrito em Requisitos de segredos.

> **Note**
>
> Este requisito inclui permitir o acesso aos seus segredos do Secrets Manager. Se você estiver usando a função de serviço padrão do Greengrass, o Greengrass terá permissão para obter os valores de segredos com nomes que começam com greengrass-.

- Um SID de conta Twilio, token de autenticação e número de telefone. Twilio-enabled Depois de criar um projeto do Twilio, esses valores estarão disponíveis no painel do projeto.

> **Note**
>
> Você pode usar uma conta de teste do Twilio. Se você estiver usando uma conta de teste, você deve adicionar números de telefone que não são de destinatários do Twilio a uma lista de números de telefone verificados. Para obter mais informações, consulte Como trabalhar com a sua conta de teste gratuita do Twilio.

- AWS CLI instalado e configurado em seu computador. Para obter mais informações, consulte o Instalando o AWS Command Line Interface e Configurando o AWS CLI no Guia do usuário da AWS Command Line Interface .

Os exemplos neste tutorial foram escritos para Linux e outros Unix-based sistemas. Se você estiver usando o Windows, consulte Especificando valores de parâmetro para a AWS Command Line Interface para saber mais sobre as diferenças de sintaxe.

Se o comando contém uma string JSON, o tutorial fornece um exemplo que tem o JSON em uma única linha. Em alguns sistemas, pode ser mais fácil editar e executar comandos usando esse formato.

Etapa 1: Criar um segredo do Secrets Manager

Nesta etapa, você usa a AWS Secrets Manager API para criar um segredo para seu token de autenticação do Twilio.

1. Primeiro, crie o segredo.

   - *twilio-auth-token*Substitua pelo seu token de autenticação do Twilio.

   ```
   aws secretsmanager create-secret --name greengrass-TwilioAuthToken --secret-
   string twilio-auth-token
   ```

   > **Note**
   >
   > Por padrão, a função de serviço do Greengrass permite AWS IoT Greengrass obter o valor dos segredos com nomes que começam com greengrass-. Para obter mais informações, consulte os requisitos de segredos.

2. Copie da saída o ARN do segredo. Use-o para criar o recurso de segredo e configurar o conector de notificações do Twilio.

Etapa 2: Criar uma definição e uma versão de recurso

Nesta etapa, você usa a AWS IoT Greengrass API para criar um recurso secreto para seu segredo do Secrets Manager.

1. Crie uma definição de recurso que inclua uma versão inicial.

   - *secret-arn*Substitua pelo ARN do segredo que você copiou na etapa anterior.

   JSON Expanded

   ```
   aws greengrass create-resource-definition --name MyGreengrassResources --
   initial-version '{
       "Resources": [
           {
   ```

```
            "Id": "TwilioAuthToken",
            "Name": "MyTwilioAuthToken",
            "ResourceDataContainer": {
                "SecretsManagerSecretResourceData": {
                    "ARN": "secret-arn"
                }
            }
        }
    ]
}'
```

JSON Single-line

```
aws greengrass create-resource-definition \
--name MyGreengrassResources \
--initial-version '{"Resources": [{"Id": "TwilioAuthToken",
 "Name": "MyTwilioAuthToken", "ResourceDataContainer":
 {"SecretsManagerSecretResourceData": {"ARN": "secret-arn"}}}]}'
```

2. Copie o `LatestVersionArn` da definição de recurso da saída. Você usa esse valor para adicionar a versão de definição do recurso à versão de grupo que você implanta no núcleo.

Etapa 3: Criar uma definição e uma versão de conector

Nesta etapa, você configura parâmetros para o conector de notificações do Twilio.

1. Crie uma definição de conector com uma versão inicial.

   - *account-sid*Substitua pelo SID da sua conta Twilio.
   - *secret-arn*Substitua pelo segredo ARN do seu Secrets Manager. O conector o usa para obter o valor do segredo local.
   - *phone-number*Substitua Twilio-enabled pelo seu número de telefone. O Twilio usa esse texto para iniciar a mensagem. Isso pode ser substituído na carga da mensagem de entrada. Use o formato a seguir: +19999999999.

JSON Expanded

```
aws greengrass create-connector-definition --name MyGreengrassConnectors --
initial-version '{
    "Connectors": [
        {
            "Id": "MyTwilioNotificationsConnector",
            "ConnectorArn": "arn:aws:greengrass:region::/connectors/
TwilioNotifications/versions/4",
            "Parameters": {
                "TWILIO_ACCOUNT_SID": "account-sid",
                "TwilioAuthTokenSecretArn": "secret-arn",
                "TwilioAuthTokenSecretArn-ResourceId": "TwilioAuthToken",
                "DefaultFromPhoneNumber": "phone-number"
            }
        }
    ]
}'
```

JSON Single-line

```
aws greengrass create-connector-definition \
--name MyGreengrassConnectors \
--initial-version '{"Connectors": [{"Id": "MyTwilioNotificationsConnector",
 "ConnectorArn": "arn:aws:greengrass:region::/connectors/TwilioNotifications/
versions/4", "Parameters": {"TWILIO_ACCOUNT_SID": "account-sid",
 "TwilioAuthTokenSecretArn": "secret-arn", "TwilioAuthTokenSecretArn-
ResourceId": "TwilioAuthToken", "DefaultFromPhoneNumber": "phone-number"}}]}'
```

> **Note**
>
> `TwilioAuthToken` é o ID que você usou na etapa anterior para criar o recurso de segredo.

2. Copie o `LatestVersionArn` da definição do conector da saída. Use esse valor para adicionar a versão da definição do conector à versão de grupo que você implanta no núcleo.

Etapa 4: crie um pacote de implantação para a função do Lambda

Para criar uma função do Lambda, você deve, primeiro, criar um pacote de implantação da função do Lambda que contenha o código da função e as dependências. As funções do Lambda do Greengrass exigem o SDK do AWS IoT Greengrass Core para tarefas como comunicação com mensagens MQTT no ambiente de núcleo e acesso a segredos locais. Este tutorial cria uma função Python, então você usa a versão Python do SDK no pacote de implantação.

1. Na página de downloads do AWS IoT Greengrass Core SDK, baixe o AWS IoT Greengrass Core SDK para Python em seu computador.
2. Descompacte o pacote obtido por download para obter o SDK. O SDK é a pasta do `greengrasssdk`.
3. Salve a seguinte função do código Python em um arquivo local denominado `temp_monitor.py`.

```
import greengrasssdk
import json
import random

client = greengrasssdk.client('iot-data')

# publish to the Twilio Notifications connector through the twilio/txt topic
def function_handler(event, context):
    temp = event['temperature']

    # check the temperature
    # if greater than 30C, send a notification
    if temp > 30:
        data = build_request(event)
        client.publish(topic='twilio/txt', payload=json.dumps(data))
        print('published:' + str(data))

    print('temperature:' + str(temp))
    return

# build the Twilio request from the input data
def build_request(event):
    to_name = event['to_name']
    to_number = event['to_number']
    temp_report = 'temperature:' + str(event['temperature'])
```

```
    return {
        "request": {
            "recipient": {
                "name": to_name,
                "phone_number": to_number,
                "message": temp_report
            }
        },
        "id": "request_" + str(random.randint(1,101))
    }
```

4. Comprima os seguintes itens em um arquivo denominado `temp_monitor_python.zip`. Ao criar o arquivo ZIP, inclua apenas o código e suas dependências, e não a pasta que contém os arquivos.

   - temp_monitor.py. Lógica do aplicativo.
   - greengrasssdk. Biblioteca necessária para funções Python do Lambda do Greengrass que publicam mensagens MQTT.

   Esse é o pacote de implantação de sua função do Lambda.

Etapa 5: Criar uma função do Lambda

Agora, crie uma função do Lambda que usa o pacote de implantação.

1. Crie um perfil do IAM para você transmitir no ARN da função ao criar a função.

   JSON Expanded

   ```
   aws iam create-role --role-name Lambda_empty --assume-role-policy '{
       "Version": "2012-10-17",
       "Statement": [
           {
               "Effect": "Allow",
               "Principal": {
                   "Service": "lambda.amazonaws.com"
               },
              "Action": "sts:AssumeRole"
           }
       ]
   ```

```
}'
```

JSON Single-line

```
aws iam create-role --role-name Lambda_empty --assume-role-policy '{"Version":
 "2012-10-17",        "Statement": [{"Effect": "Allow", "Principal": {"Service":
 "lambda.amazonaws.com"},"Action": "sts:AssumeRole"}]}'
```

> **Note**
>
> AWS IoT Greengrass não usa essa função porque as permissões para suas funções do Greengrass Lambda são especificadas na função de grupo do Greengrass. Neste tutorial, você cria uma função vazia.

2. Copie a `Arn` da saída.
3. Use a AWS Lambda API para criar a TempMonitor função. O comando a seguir pressupõe que o arquivo zip esteja no diretório atual.

   - Substitua *`role-arn`* pelo `Arn` que você copiou.

```
aws lambda create-function \
--function-name TempMonitor \
--zip-file fileb://temp_monitor_python.zip \
--role role-arn \
--handler temp_monitor.function_handler \
--runtime python3.7
```

4. Publique uma versão da função.

```
aws lambda publish-version --function-name TempMonitor --description 'First
 version'
```

5. Crie um alias para a versão publicada.

   Os grupos do Greengrass podem fazer referência a uma função do Lambda por alias (recomendado) ou por versão. Usar um alias facilita o gerenciamento de atualizações de código porque você não precisa alterar a tabela de assinaturas nem a definição do grupo ao atualizar a

função do código. Em vez disso, você pode simplesmente apontar o alias para a nova versão da função.

> **Note**
>
> AWS IoT Greengrass não oferece suporte a aliases do Lambda para as versões $LATEST.

```
aws lambda create-alias --function-name TempMonitor --name GG_TempMonitor --
function-version 1
```

6. Copie a `AliasArn` da saída. Você usa esse valor ao configurar a função para AWS IoT Greengrass e ao criar uma assinatura.

Agora você está pronto para configurar a função para AWS IoT Greengrass.

Etapa 6: Criar uma definição e uma versão de função

Para usar uma função Lambda em um AWS IoT Greengrass núcleo, você cria uma versão de definição de função que faz referência à função Lambda por alias e define a configuração em nível de grupo. Para obter mais informações, consulte the section called “Controlando a execução da função do Lambda do Greengrass”.

1. Crie uma definição de função que inclua uma versão inicial.

   - *`alias-arn`*Substitua pelo `AliasArn` que você copiou ao criar o alias.

   JSON Expanded

```
aws greengrass create-function-definition --name MyGreengrassFunctions --
initial-version '{
    "Functions": [
        {
            "Id": "TempMonitorFunction",
            "FunctionArn": "alias-arn",
            "FunctionConfiguration": {
```

```
                "Executable": "temp_monitor.function_handler",
                "MemorySize": 16000,
                "Timeout": 5
            }
        }
    ]
}'
```

JSON Single-line

```
aws greengrass create-function-definition \
--name MyGreengrassFunctions \
--initial-version '{"Functions": [{"Id": "TempMonitorFunction",
 "FunctionArn": "alias-arn", "FunctionConfiguration": {"Executable":
 "temp_monitor.function_handler", "MemorySize": 16000,"Timeout": 5}}]}'
```

2. Copie a `LatestVersionArn` da saída. Você usa esse valor para adicionar a versão de definição da função à versão de grupo que você implanta no núcleo.
3. Copie a `Id` da saída. Você pode usar esse valor no futuro, ao atualizar a função.

Etapa 7: criar uma definição e uma versão de assinatura

Nesta etapa, adicione uma assinatura que permita que a função do Lambda envie dados de entrada para o conector. O conector define os tópicos MQTT que assina, para que essa assinatura use um dos tópicos. Esse é o mesmo tópico no qual a função de exemplo publica.

Neste tutorial, você também cria assinaturas que permitem que a função receba leituras simuladas de temperatura AWS IoT e permita AWS IoT receber informações de status do conector.

1. Crie uma definição de assinatura que contém uma versão inicial que inclui as assinaturas.

   - *alias-arn*Substitua pelo `AliasArn` que você copiou ao criar o alias para a função. Use esse ARN para ambas as assinaturas que o usam.

JSON Expanded

```
aws greengrass create-subscription-definition --initial-version '{
    "Subscriptions": [
```

```
        {
            "Id": "TriggerNotification",
            "Source": "alias-arn",
            "Subject": "twilio/txt",
            "Target": "arn:aws:greengrass:region::/connectors/
TwilioNotifications/versions/4"
        },
        {
            "Id": "TemperatureInput",
            "Source": "cloud",
            "Subject": "temperature/input",
            "Target": "alias-arn"
        },
        {
            "Id": "OutputStatus",
            "Source": "arn:aws:greengrass:region::/connectors/
TwilioNotifications/versions/4",
            "Subject": "twilio/message/status",
            "Target": "cloud"
        }
    ]
}'
```

JSON Single-line

```
aws greengrass create-subscription-definition \
--initial-version '{"Subscriptions": [{"Id": "TriggerNotification", "Source":
 "alias-arn", "Subject": "twilio/txt", "Target": "arn:aws:greengrass:region::/
connectors/TwilioNotifications/versions/4"},{"Id": "TemperatureInput",
 "Source": "cloud", "Subject": "temperature/input", "Target": "alias-arn"},
{"Id": "OutputStatus", "Source": "arn:aws:greengrass:region::/connectors/
TwilioNotifications/versions/4", "Subject": "twilio/message/status", "Target":
 "cloud"}]}'
```

2. Copie a `LatestVersionArn` da saída. Você usa esse valor para adicionar a versão de definição da assinatura à versão de grupo que você implanta no núcleo.

Etapa 8: Criar uma versão de grupo

Agora, você está pronto para criar uma versão de grupo que contém todos os itens que deseja implantar. Faça isso criando uma versão de grupo que faz referência à versão de destino de cada componente.

Primeiro, obtenha o ID do grupo e o ARN da versão de definição do núcleo. Esses valores são necessárias para criar a versão do grupo.

1. Obtenha o ID do grupo e a versão mais recente do grupo:

   a. Obtenha os IDs do grupo do Greengrass de destino e a versão do grupo. Esse procedimento pressupõe que esse seja o grupo e a versão mais recente do grupo. A consulta a seguir retorna o grupo criado mais recentemente.

   ```
   aws greengrass list-groups --query "reverse(sort_by(Groups,
    &CreationTimestamp))[0]"
   ```

   Ou é possível consultar por nome. Os nomes de grupo não precisam ser exclusivos, portanto, vários grupos podem ser retornados.

   ```
   aws greengrass list-groups --query "Groups[?Name=='MyGroup']"
   ```

   > **Note**
   >
   > Você também pode encontrar esses valores no AWS IoT console. O ID do grupo é exibido na página Settings (Configurações) do grupo. Os IDs de versão do grupo são exibidos na guia Implantações do grupo.

   b. Copie da saída o `Id` do grupo de destino. Você usa isso para obter a versão de definição de núcleo e ao implantar o grupo.

   c. Copie a `LatestVersion` da saída, que é o ID da última versão adicionada ao grupo. Você usa isso para obter a versão de definição do núcleo.

2. Obtenha o ARN da versão de definição de núcleo:

   a. Obtenha a versão do grupo. Para esta etapa, vamos considerar que a versão mais recente do grupo inclui uma versão de definição do núcleo.

   - *group-id*Substitua pelo `Id` que você copiou para o grupo.
   - *group-version-id*Substitua pelo `LatestVersion` que você copiou para o grupo.

   ```
   aws greengrass get-group-version \
   --group-id group-id \
   ```

```
--group-version-id group-version-id
```

b. Copie a `CoreDefinitionVersionArn` da saída.

3. Criar uma versão de grupo.

   - *group-id*Substitua pelo `Id` que você copiou para o grupo.
   - *core-definition-version-arn*Substitua pela `CoreDefinitionVersionArn` que você copiou para a versão de definição principal.
   - *resource-definition-version-arn*Substitua pelo `LatestVersionArn` que você copiou para a definição do recurso.
   - *connector-definition-version-arn*Substitua pelo `LatestVersionArn` que você copiou para a definição do conector.
   - *function-definition-version-arn*Substitua pelo `LatestVersionArn` que você copiou para a definição da função.
   - *subscription-definition-version-arn*Substitua pelo `LatestVersionArn` que você copiou para a definição da assinatura.

```
aws greengrass create-group-version \
--group-id group-id \
--core-definition-version-arn core-definition-version-arn \
--resource-definition-version-arn resource-definition-version-arn \
--connector-definition-version-arn connector-definition-version-arn \
--function-definition-version-arn function-definition-version-arn \
--subscription-definition-version-arn subscription-definition-version-arn
```

4. Copie o valor de `Version` da saída. Este é o ID da versão do grupo. Você usa esse valor para implantar a versão do grupo.

Etapa 9: Criar uma implantação

Implante o grupo no dispositivo do núcleo.

1. Em um terminal de dispositivo principal, verifique se o AWS IoT Greengrass daemon está em execução.

   a. Para verificar se o daemon está em execução:

```
ps aux | grep -E 'greengrass.*daemon'
```

Se a saída contém uma entrada `root` para `/greengrass/ggc/packages/1.11.6/bin/daemon`, o daemon está em execução.

b. Para iniciar o daemon:

```
cd /greengrass/ggc/core/
sudo ./greengrassd start
```

2. Crie um implantação do .

- *group-id*Substitua pelo `Id` que você copiou para o grupo.
- *group-version-id*Substitua pelo `Version` que você copiou para a nova versão do grupo.

```
aws greengrass create-deployment \
--deployment-type NewDeployment \
--group-id group-id \
--group-version-id group-version-id
```

3. Copie a `DeploymentId` da saída.
4. Obtenha o status de implantação.

- *group-id*Substitua pelo `Id` que você copiou para o grupo.
- *deployment-id*Substitua pelo `DeploymentId` que você copiou para a implantação.

```
aws greengrass get-deployment-status \
--group-id group-id \
--deployment-id deployment-id
```

Se o status for `Success`, a implantação foi bem-sucedida. Para obter ajuda sobre a solução de problemas, consulte Solução de problemas.

Testar a solução

1. Na página inicial do AWS IoT console, escolha Testar.

2. Em Inscrever-se no tópico, use os seguintes valores e, em seguida, selecione Inscrever-se. O conector de notificações do Twilio publica informações de status nesse tópico.

Propriedade	Valor
Tópico de assinatura	twilio/message/status
Exibição da carga útil do MQTT	Exibir cargas úteis como strings

3. Para Publicar no tópico, use os valores a seguir e, em seguida selecione Publicar para invocar a função.

Propriedade	Valor
Tópico	temperature/input
Mensagem	*recipient-name* Substitua por um nome e *recipient-phone-number* pelo número de telefone do destinatário da mensagem de texto. Exemplo: +12345000000 `{ "to_name": " recipient-name ", "to_number": " recipient-phone-number ", "temperature": 31 }` Se você estiver usando uma conta de teste, você deve adicionar números de telefone que não são de destinatários do Twilio a uma lista de números de telefone verificados. Para obter mais informações, consulte Verificar seu número de telefone pessoal.

Se for bem-sucedido, o destinatário receberá a mensagem de texto e o console exibirá o `success` status dos dados de saída.

Agora, altere a `temperature` na mensagem de entrada para **29** e publique. Como isso é menor que 30, a TempMonitor função não aciona uma mensagem do Twilio.

Consulte também

- Integrar a serviços e protocolos usando conectores
- the section called “Conectores do Greengrass fornecidos pela AWS”
- the section called “Conceitos básicos de conectores (console)”
- AWS Secrets Manager comandos na Referência de AWS CLI Comandos
- AWS Identity and Access Management Comandos (IAM) na Referência de AWS CLI Comandos
- AWS Lambda comandos na Referência de AWS CLI Comandos
- AWS IoT Greengrass comandos na Referência de AWS CLI Comandos

API RESTful de descoberta do Greengrass

Todos os dispositivos clientes que se comunicam com um AWS IoT Greengrass núcleo devem ser membros de um grupo do Greengrass. Cada grupo deve ter um núcleo do Greengrass. A API de descoberta permite que os dispositivos recuperem as informações necessárias para se conectar a um núcleo do Greengrass que esteja no mesmo grupo do Greengrass que o dispositivo cliente. Quando um dispositivo cliente ficar online pela primeira vez, ele poderá se conectar ao serviço do AWS IoT Greengrass e usar a API de descoberta para localizar:

- O grupo ao qual pertence. Um dispositivo cliente pode ser membro de até 10 grupos.
- O endereço IP e a porta do núcleo do Greengrass no grupo.
- O certificado CA do grupo, que pode ser usado para autenticar o dispositivo de núcleo do Greengrass.

> **Note**
>
> Os dispositivos cliente também podem usar os AWS IoT Device SDKs para descobrir informações de conectividade para um núcleo do Greengrass. Para obter mais informações, consulte AWS IoT SDK do dispositivo.

Para usar essa API, envie solicitações HTTP ao endpoint da API de descoberta. Por exemplo:

```
https://greengrass-ats.iot.region.amazonaws.com:port/greengrass/discover/thing/thing-name
```

Para obter uma lista de regiões e endpoints da Amazon Web Services compatíveis com a API AWS IoT Greengrass Discovery, consulte AWS IoT Greengrass endpoints e cotas no. Referência geral da AWS Esta é uma API somente de plano de dados. Os endpoints para gerenciamento e AWS IoT Core operações de grupos são diferentes dos endpoints da API Discovery.

Solicitação

A solicitação contém os cabeçalhos HTTP padrão e é enviada ao endpoint de descoberta do Greengrass, conforme mostrado nos exemplos a seguir.

O número da porta depende se o núcleo é configurado para enviar o tráfego HTTPS pela porta 8443 ou pela 443. Para obter mais informações, consulte the section called “Conectar-se à porta 443 ou por meio de um proxy de rede”.

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
```

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 .

> **Note**
>
> Estes exemplos usam o endpoint do Amazon Trust Services (ATS), que é usado com certificados da CA raiz do ATS (recomendado). Os endpoints devem corresponder ao tipo de certificado da CA raiz. Para obter mais informações, consulte the section called “Os endpoints do serviço devem corresponder ao tipo de certificado”.

Resposta

Após o sucesso, a resposta incluirá os cabeçalhos HTTP padrão e os seguintes código e corpo:

```
HTTP 200
BODY: response document
```

Para obter mais informações, consulte Exemplos de documentos de resposta de descoberta.

Autorização de descoberta

Para recuperar as informações de conectividade, é necessária uma política que permita ao chamador executar a ação `greengrass:Discover`. A autenticação TLS mútuo com certificado de cliente é a única forma aceita de autenticação. Veja a seguir um exemplo de política que permite a um chamador executar essa ação:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": "greengrass:Discover",
        "Resource": ["arn:aws:iot:us-west-2:123456789012:thing/MyThingName"]
     }]
}
```

Exemplos de documentos de resposta de descoberta

O documento a seguir mostra a resposta de um dispositivo cliente que é membro de um grupo com um núcleo do Greengrass, um endpoint e um certificado CA de grupo:

```
{
  "GGGroups": [
    {
      "GGGroupId": "gg-group-01-id",
      "Cores": [
        {
          "thingArn": "core-01-thing-arn",
          "Connectivity": [
            {
              "id": "core-01-connection-id",
              "hostAddress": "core-01-address",
              "portNumber": core-01-port,
              "metadata": "core-01-description"
            }
          ]
        }
      ],
```

```
      "CAs": [
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----"
      ]
    }
  ]
}
```

O documento a seguir mostra a resposta de um dispositivo cliente que é membro de dois grupos com um núcleo do Greengrass, vários endpoints e vários certificados CA de grupo:

```
{
  "GGGroups": [
    {
      "GGGroupId": "gg-group-01-id",
      "Cores": [
        {
          "thingArn": "core-01-thing-arn",
          "Connectivity": [
            {
              "id": "core-01-connection-id",
              "hostAddress": "core-01-address",
              "portNumber": core-01-port,
              "metadata": "core-01-connection-1-description"
            },
            {
              "id": "core-01-connection-id-2",
              "hostAddress": "core-01-address-2",
              "portNumber": core-01-port-2,
              "metadata": "core-01-connection-2-description"
            }
          ]
        }
      ],
      "CAs": [
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
        "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----"
      ]
    },
    {
      "GGGroupId": "gg-group-02-id",
      "Cores": [
         {
```

```
            "thingArn":"core-02-thing-arn",
            "Connectivity" : [
            {
              "id": "core-02-connection-id",
              "hostAddress": "core-02-address",
              "portNumber": core-02-port,
              "metadata": "core-02-connection-1-description"
            }
            ],
            "CAs": [
                "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
                "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----",
                "-----BEGIN CERTIFICATE-----cert-contents-----END CERTIFICATE-----"
            ]
        }
    ]
    }
}
```

> **Note**
>
> Um grupo do Greengrass deve definir exatamente um núcleo do Greengrass. Qualquer resposta do AWS IoT Greengrass serviço que contenha uma lista de núcleos do Greengrass contém somente um núcleo do Greengrass.

Se tiver o cURL instalado, você poderá testar a solicitação de descoberta. Por exemplo:

```
$ curl --cert 1a23bc4d56.cert.pem --key 1a23bc4d56.private.key https://greengrass-
ats.iot.us-west-2.amazonaws.com:8443/greengrass/discover/thing/MyDevice
{"GGGroups":[{"GGGroupId":"1234a5b6-78cd-901e-2fgh-3i45j6k1789","Cores":
[{"thingArn":"arn:aws:iot:us-west-2:1234567
89012:thing/MyFirstGroup_Core","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"]}]}
```

Segurança em AWS IoT Greengrass

A segurança na nuvem AWS é a maior prioridade. Como AWS cliente, você se beneficia de uma arquitetura de data center e rede criada para atender aos requisitos das organizações mais sensíveis à segurança.

A segurança é uma responsabilidade compartilhada entre você AWS e você. O modelo de responsabilidade compartilhada descreve isso como segurança da nuvem e segurança na nuvem:

- Segurança da nuvem — AWS é responsável por proteger a infraestrutura que executa AWS os serviços no Nuvem AWS. AWS também fornece serviços que você pode usar com segurança. Third-party auditores testam e verificam regularmente a eficácia de nossa segurança como parte dos Programas de Conformidade Programas de AWS de . Para saber mais sobre os programas de conformidade aplicáveis AWS IoT Greengrass, consulte AWS Serviços no escopo do programa de conformidade AWS .
- 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.

Ao usar AWS IoT Greengrass, você também é responsável por proteger seus dispositivos, conexão de rede local e chaves privadas.

Esta documentação ajuda você a entender como aplicar o modelo de responsabilidade compartilhada ao usar AWS IoT Greengrass. Os tópicos a seguir mostram como configurar para atender AWS IoT Greengrass aos seus objetivos de segurança e conformidade. Você também aprenderá a usar outros AWS serviços que ajudam a monitorar e proteger seus AWS IoT Greengrass recursos.

Tópicos

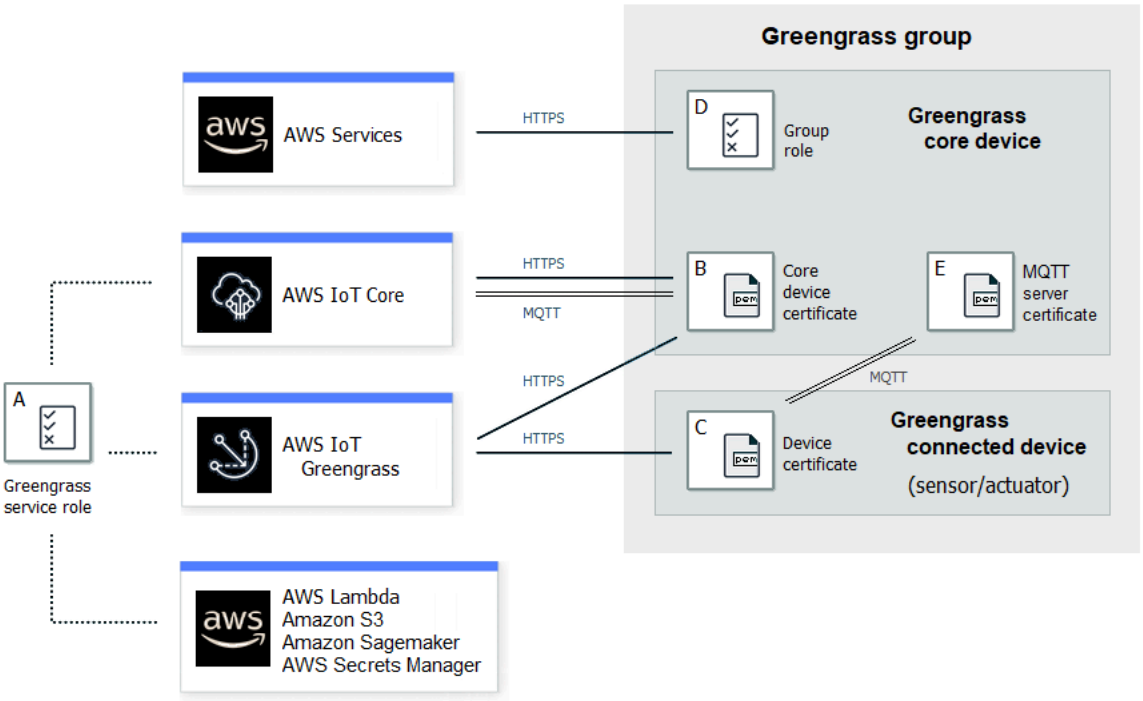
- Visão geral do AWS IoT Greengrass security
- Proteção de dados em AWS IoT Greengrass
- Autenticação e autorização de dispositivos para AWS IoT Greengrass
- Gerenciamento de identidade e acesso para AWS IoT Greengrass
- Validação de conformidade AWS IoT Greengrass
- Resiliência em AWS IoT Greengrass

- Segurança da infraestrutura em AWS IoT Greengrass
- Análise de configuração e vulnerabilidade em AWS IoT Greengrass
- AWS IoT Greengrass e endpoints VPC de interface (AWS PrivateLink)
- Práticas recomendadas de segurança para AWS IoT Greengrass

Visão geral do AWS IoT Greengrass security

AWS IoT Greengrass usa X.509 certificados, AWS IoT políticas e políticas e funções do IAM para proteger os aplicativos executados em dispositivos em seu ambiente Greengrass local.

O diagrama a seguir mostra os componentes do modelo AWS IoT Greengrass de segurança:



A — Função de serviço do Greengrass

Uma função do IAM criada pelo cliente e assumida por AWS IoT Greengrass quando você acessa seus AWS recursos a partir de AWS IoT Core AWS Lambda, e outros AWS serviços. Para obter mais informações, consulte the section called “Função de serviço do Greengrass”.

B — Certificado de dispositivo de núcleo

Um X.509 certificado usado para autenticar um núcleo do Greengrass com e. AWS IoT Core AWS IoT Greengrass Para obter mais informações, consulte the section called “Autorização e autenticação do dispositivo”.

C — Certificado de dispositivo

Um X.509 certificado usado para autenticar um dispositivo cliente, também conhecido como dispositivo conectado, com AWS IoT Core e. AWS IoT Greengrass Para obter mais informações, consulte the section called “Autorização e autenticação do dispositivo”.

D — Função de grupo

Uma função do IAM criada pelo cliente assumida por AWS IoT Greengrass quando chama AWS serviços de um núcleo do Greengrass.

Você usa essa função para especificar as permissões de acesso que suas funções e conectores do Lambda definidos pelo usuário precisam para acessar AWS serviços, como o DynamoDB. Você também o usa para permitir AWS IoT Greengrass a exportação de fluxos do gerenciador de fluxos para AWS serviços e a gravação em CloudWatch registros. Para obter mais informações, consulte the section called “Função do grupo do Greengrass.”.

> **Note**
>
> AWS IoT Greengrass não usa a função de execução do Lambda especificada na versão em AWS Lambda nuvem de uma função do Lambda.

Certificado do servidor E - MQTT

O certificado usado para autenticação mútua Transport Layer Security (TLS) entre um dispositivo de núcleo do Greengrass e dispositivos cliente no grupo do Greengrass. O certificado é assinado pelo certificado CA do grupo, que é armazenado na Nuvem AWS.

Fluxo de trabalho de conexão de dispositivo

Esta seção descreve como os dispositivos cliente se conectam ao AWS IoT Greengrass serviço e aos dispositivos principais do Greengrass. Os dispositivos cliente são AWS IoT Core dispositivos registrados que estão no mesmo grupo do Greengrass que o dispositivo principal.

- Um dispositivo principal do Greengrass usa seu certificado de dispositivo, sua chave privada e o certificado CA AWS IoT Core raiz para se conectar ao AWS IoT Greengrass serviço. No dispositivo de núcleo, o objeto `crypto` no arquivo de configuração especifica o caminho do arquivo para esses itens.
- O dispositivo de núcleo do Greengrass faz download das informações de associação em grupo do serviço do AWS IoT Greengrass .
- Quando uma implantação é feita no dispositivo de núcleo do Greengrass, o Device Certificate Manager (DCM) lida com o gerenciamento de certificados do servidor local para o dispositivo de núcleo do Greengrass.
- Um dispositivo cliente se conecta ao AWS IoT Greengrass serviço usando seu certificado de dispositivo, chave privada e o certificado CA AWS IoT Core raiz. Após estabelecer a conexão, o dispositivo cliente usa o Greengrass Discovery Service para encontrar o endereço IP do dispositivo de núcleo do Greengrass. O dispositivo cliente também faz download do certificado CA do grupo, que é usado para autenticação TLS mútua com o dispositivo de núcleo do Greengrass.
- Um dispositivo cliente tenta se conectar ao dispositivo de núcleo do Greengrass, passando seu certificado de dispositivo e ID de cliente. Se o ID do cliente corresponder ao nome de coisa do dispositivo cliente e o certificado for válido (parte do grupo do Greengrass), a conexão será estabelecida. Caso contrário, a conexão será encerrada.

A AWS IoT política para dispositivos cliente deve conceder a `greengrass:Discover` permissão para permitir que os dispositivos cliente descubram as informações de conectividade do núcleo. Para obter mais informações sobre a declaração de política, consulte the section called “Autorização de descoberta”.

Configurar AWS IoT Greengrass security

Para configurar a segurança do aplicativo Greengrass

1. Crie qualquer AWS IoT Core coisa para seu dispositivo principal do Greengrass.
2. Gere um par de chaves e um certificado de dispositivo para o dispositivo de núcleo do Greengrass.
3. Crie e anexe uma política do AWS IoT para o certificado de dispositivo. O certificado e a política permitem que o dispositivo principal do Greengrass acesse AWS IoT Core e ofereça AWS IoT Greengrass serviços. Para obter mais informações, consulte AWS IoT Política mínima para o dispositivo principal.

> **Note**
>
> O uso de variáveis de política (`iot:Connection.Thing.*`) na AWS IoT política de um dispositivo principal não é suportado. O núcleo usa o mesmo certificado de dispositivo para fazer várias conexões AWS IoT Core , mas o ID do cliente em uma conexão pode não corresponder exatamente ao nome do item principal.

4. Crie uma função de serviço do Greengrass. Essa função do IAM AWS IoT Greengrass autoriza o acesso a recursos de outros AWS serviços em seu nome. Isso permite AWS IoT Greengrass realizar tarefas essenciais, como recuperar AWS Lambda funções e gerenciar sombras de dispositivos.

   Você pode usar a mesma função de serviço em Região da AWS s, mas ela deve estar associada à sua Conta da AWS em todos os Região da AWS lugares que você usa AWS IoT Greengrass.

5. (Opcional) Crie uma função de grupo do Greengrass. Essa função do IAM concede permissão às funções e conectores do Lambda em execução em um núcleo do Greengrass para chamar AWS serviços. Por exemplo, o conector do Kinesis Firehose requer permissão para gravar registros em um fluxo de entrega do Amazon Data Firehose.

   Você pode anexar apenas uma função a um grupo do Greengrass.

6. Crie AWS IoT Core algo para cada dispositivo que se conecte ao seu núcleo do Greengrass.

   > **Note**
   >
   > Você também pode usar AWS IoT Core itens e certificados existentes.

7. Crie certificados de dispositivos, pares de chaves e AWS IoT políticas para cada dispositivo que se conecta ao seu núcleo do Greengrass.

AWS IoT Greengrass principais princípios de segurança

O núcleo do Greengrass usa os seguintes princípios de segurança: AWS IoT cliente, servidor MQTT local e gerenciador de segredos local. A configuração para essas entidades principais são armazenadas no objeto `crypto` no arquivo de configuração `config.json`. Para obter mais informações, consulte the section called “AWS IoT Greengrass arquivo de configuração principal”.

Essa configuração inclui o caminho para a chave privada usada pelo componente principal para autenticação e criptografia. O AWS IoT Greengrass oferece suporte a dois modos de chave privada: armazenamento baseado em hardware ou baseado no sistema de arquivos (padrão). Para obter mais informações sobre como armazenar chaves nos módulos de segurança de hardware, consulte the section called “Integração de segurança de hardware”.

AWS IoT Cliente

O AWS IoT cliente (cliente de IoT) gerencia a comunicação pela Internet entre o núcleo do Greengrass e. AWS IoT Core AWS IoT Greengrass usa X.509 certificados com chaves públicas e privadas para autenticação mútua ao estabelecer conexões TLS para essa comunicação. Para obter mais informações, consulte X.509 certificados e AWS IoT Core no Guia do AWS IoT Core desenvolvedor.

O cliente da IoT oferece suporte a certificados e chaves RSA e EC. Os caminhos do certificado e da chave privada são especificados para a entidade principal `IoTCertificate` em `config.json`.

Servidor MQTT

O servidor MQTT local gerencia a comunicação pela rede local entre o núcleo do Greengrass e os dispositivos clientes do grupo. AWS IoT Greengrass usa X.509 certificados com chaves públicas e privadas para autenticação mútua ao estabelecer conexões TLS para essa comunicação.

Por padrão, AWS IoT Greengrass gera uma chave privada RSA para você. Para configurar o núcleo para usar uma chave privada diferente, você deve fornecer o caminho da chave para a entidade principal `MQTTServerCertificate` em `config.json`. Você é responsável por girar uma chave fornecida pelo cliente.

Suporte à chave privada

	Chave RSA	Chave EC
Tipo de chave	Compatível	Compatível
Principais parâmetros	comprimento mínimo de 2.048 bits	Curva NIST P-256 ou P-384 NIST
Formato de disco	PKCS#1, PKCS#8	SECG1, PKCS#8

	Chave RSA	Chave EC
Versão mínima do GGC	• Usar chaves RSA padrão: 1.0 • Especificar uma chave RSA: 1.7	• Especificar uma chave EC: 1.9

A configuração da chave privada determina os processos associados. Para ver a lista de pacotes de criptografia aos quais o núcleo do Greengrass oferece suporte como um servidor, consulte the section called “Suporte a pacotes de criptografia TLS”.

Se nenhuma chave privada for especificada (padrão)

- AWS IoT Greengrass gira a chave com base nas suas configurações de rotação.
- O núcleo gera uma chave RSA, que é usada para gerar o certificado.
- O certificado do servidor MQTT tem uma chave pública RSA e uma assinatura SHA-256 RSA.

Se uma chave privada RSA for especificada (requer o GGC v1.7 ou posterior)

- Você é responsável pela rotação das chaves.
- O núcleo usa a chave especificada para gerar o certificado.
- A chave RSA deve ter um tamanho mínimo de 2.048 bits.
- O certificado do servidor MQTT tem uma chave pública RSA e uma assinatura SHA-256 RSA.

Se uma chave privada EC for especificada (requer o GGC v1.9 ou posterior)

- Você é responsável pela rotação das chaves.
- O núcleo usa a chave especificada para gerar o certificado.
- A chave privada EC deve usar uma curva NIST P-256 ou NIST P-384 .
- O certificado do servidor MQTT tem uma chave pública EC e uma assinatura SHA-256 RSA.

  O certificado do servidor MQTT apresentado pelo núcleo tem uma assinatura SHA-256 RSA, independentemente do tipo de chave. Por esse motivo, os clientes devem oferecer suporte à validação do certificado SHA-256 RSA para estabelecer uma conexão segura com o núcleo.

Secrets Manager

O gerenciador de segredos local gerencia com segurança as cópias locais dos segredos que você cria em AWS Secrets Manager. Ele usa uma chave privada para proteger a chave de dados que é usada para criptografar os segredos. Para obter mais informações, consulte Implantar segredos no núcleo.

Por padrão, a chave privada do cliente da IoT é usada, mas você pode especificar outra chave privada para a entidade principal `SecretsManager` em `config.json`. Somente o tipo de chave RSA tem suporte. Para obter mais informações, consulte the section called “Especificar a chave privada para criptografia de segredos”.

> **Note**
>
> Atualmente, AWS IoT Greengrass oferece suporte somente ao mecanismo de preenchimento PKCS #1 v1.5 para criptografia e decodificação de segredos locais ao usar chaves privadas baseadas em hardware. Se você estiver seguindo as instruções fornecidas pelo fornecedor para gerar manualmente chaves privadas baseadas em hardware, certifique-se de escolher PKCS #1 v1.5. AWS IoT Greengrass não oferece suporte ao Optimal Asymmetric Encryption Padding (OAEP).

Suporte à chave privada

	Chave RSA	Chave EC
Tipo de chave	Compatível	Não compatível
Principais parâmetros	comprimento mínimo de 2.048 bits	Não aplicável
Formato de disco	PKCS#1, PKCS#8	Não aplicável
Versão mínima do GGC	1,7	Não aplicável

Assinaturas gerenciadas no fluxo de trabalho de mensagens MQTT

AWS IoT Greengrass usa uma tabela de assinaturas para definir como as mensagens MQTT podem ser trocadas entre dispositivos, funções e conectores do cliente em um grupo do Greengrass e com

o serviço AWS IoT Core paralelo local. Cada assinatura especifica uma fonte, destino e tópico (ou assunto) do MQTT sobre os quais as mensagens são enviadas ou recebidas. AWS IoT Greengrass permite que as mensagens sejam enviadas de uma origem para um destino somente se uma assinatura correspondente for definida.

Uma assinatura define o fluxo de mensagens em uma única direção, da origem para o destino. Para oferecer suporte à troca de mensagens bidirecional, você deve criar duas assinaturas, uma para cada direção.

Suporte a pacotes de criptografia TLS

AWS IoT Greengrass usa o modelo AWS IoT Core de segurança de transporte para criptografar a comunicação com a nuvem usando pacotes de criptografia TLS. https://en.wikipedia.org/wiki/Cipher_suite Além disso, AWS IoT Greengrass os dados são criptografados quando estão em repouso (na nuvem). Para obter mais informações sobre segurança de AWS IoT Core transporte e pacotes de criptografia compatíveis, consulte Segurança de transporte no Guia do AWS IoT Core desenvolvedor.

Pacotes de criptografia com suporte para a comunicação na rede local

Ao contrário de AWS IoT Core, o AWS IoT Greengrass núcleo suporta os seguintes conjuntos de cifras TLS de rede local para algoritmos de assinatura de certificados. Todos esses pacotes de criptografia têm suporte quando as chaves privadas são armazenadas no sistema de arquivos. Um subconjunto têm suporte quando o núcleo está configurado para usar os módulos de segurança de hardware (HSM). Para obter mais informações, consulte the section called “Entidades principais de segurança” e the section called “Integração de segurança de hardware”. A tabela também inclui a versão mínima do software AWS IoT Greengrass Core necessária para suporte.

	Cifra	Suporte do HSM	Versão mínima do GGC
TLSv1.2	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	Com suporte	1,0
	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	Com suporte	1,0

	Cifra	Suporte do HSM	Versão mínima do GGC
	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	Com suporte	1,0
	TLS_RSA_WITH_AES_128_CBC_SHA	Não compatível	1,0
	TLS_RSA_WITH_AES_128_GCM_SHA256	Não compatível	1,0
	TLS_RSA_WITH_AES_256_CBC_SHA	Não compatível	1,0
	TLS_RSA_WITH_AES_256_GCM_SHA384	Não compatível	1,0
	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	Compatível	1.9
	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	Compatível	1.9
TLSv1.1	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	Com suporte	1,0

	Cifra	Suporte do HSM	Versão mínima do GGC
	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	Com suporte	1,0
	TLS_RSA_WITH_AES_128_CBC_SHA	Não compatível	1,0
	TLS_RSA_WITH_AES_256_CBC_SHA	Não compatível	1,0
TLSv1.0	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	Com suporte	1,0
	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	Com suporte	1,0
	TLS_RSA_WITH_AES_128_CBC_SHA	Não compatível	1,0
	TLS_RSA_WITH_AES_256_CBC_SHA	Não compatível	1,0

Proteção de dados em AWS IoT Greengrass

O modelo de responsabilidade AWS compartilhada O modelo se aplica à proteção de dados em AWS IoT Greengrass. Conforme descrito neste modelo, AWS é responsável por proteger a infraestrutura global que executa todos os Nuvem AWS. Você é responsável por manter o controle sobre o 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 privacidade de dados, consulte Perguntas frequentes sobre privacidade de dados . Para obter informações sobre proteção de dados na Europa, consulte o Centro de Regulamentação Geral de Proteção de Dados (GDPR).

Para fins de proteção de dados, recomendamos que você proteja Conta da AWS as credenciais e configure usuários individuais com Centro de Identidade do AWS IAM ou 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) com cada conta.
- Use SSL/TLS para se comunicar com AWS os recursos. Exigimos TLS 1.2 e recomendamos TLS 1.3.
- Configure a API e o registro de atividades do usuário com AWS CloudTrail. Para obter informações sobre o uso de CloudTrail trilhas para capturar AWS atividades, consulte Como trabalhar com CloudTrail trilhas no Guia AWS CloudTrail do usuário.
- Use soluções de AWS criptografia, juntamente com todos os controles de segurança padrão Serviços da AWS.
- Use serviços gerenciados de segurança avançada, como o Amazon Macie, que ajuda a localizar e proteger dados sensíveis armazenados no Amazon S3.
- Se você precisar de módulos criptográficos validados pelo FIPS 140-3 ao acessar AWS por meio de uma interface de linha de comando ou de uma API, use um endpoint FIPS. Para saber mais sobre os endpoints FIPS disponíveis, consulte Federal Information Processing Standard (FIPS) 140-3.

É altamente recomendável que nunca sejam colocadas informações confidenciais ou sensíveis, como endereços de e-mail de clientes, em tags ou campos de formato livre, como um campo Nome. Isso inclui quando você trabalha com AWS IoT Greengrass ou Serviços da AWS usa o console, a API ou AWS os SDKs. AWS CLI Quaisquer dados inseridos em tags ou em 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, é fortemente recomendável que não sejam incluídas informações de credenciais no URL para validar a solicitação nesse servidor.

Para obter mais informações sobre como proteger informações confidenciais em AWS IoT Greengrass, consultethe section called “Não registrar em log informações confidenciais”.

Consulte mais informações sobre proteção de dados na publicação do blog AWS Shared Responsibility Model and GDPR no Blog de segurança da AWS .

Tópicos

- Criptografia de dados
- Integração de segurança de hardware

Criptografia de dados

AWS IoT Greengrass usa criptografia para proteger os dados em trânsito (pela Internet ou rede local) e em repouso (armazenados na Nuvem AWS).

Os dispositivos em um AWS IoT Greengrass ambiente geralmente coletam dados que são enviados aos AWS serviços para processamento adicional. Para obter mais informações sobre criptografia de dados em outros AWS serviços, 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 Greengrass tem três modos de comunicação em que os dados estão em trânsito:

- the section called “Dados em trânsito pela Internet”. A comunicação entre um núcleo do Greengrass e AWS IoT Greengrass pela Internet é criptografada.
- the section called “Dados em trânsito por meio da rede local”. A comunicação entre um núcleo do Greengrass e dispositivos cliente em uma rede local é criptografada.
- 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

AWS IoT Greengrass usa Transport Layer Security (TLS) para criptografar toda a comunicação pela Internet. Todos os dados enviados para o Nuvem AWS são enviados por uma conexão TLS

usando os protocolos MQTT ou HTTPS, portanto, são seguros por padrão. AWS IoT Greengrass usa o modelo AWS IoT de segurança de transporte. Para obter mais informações, consulte Segurança de transporte no Guia do desenvolvedor do AWS IoT Core .

Dados em trânsito por meio da rede local

AWS IoT Greengrass usa TLS para criptografar toda a comunicação na rede local entre os dispositivos principais e clientes do Greengrass. Para obter mais informações, consulte Conjuntos de cifras com suporte para comunicação de rede local.

É sua responsabilidade proteger a rede local e as chaves privadas.

Para os dispositivos de núcleo do Greengrass, é sua responsabilidade:

- Manter o kernel atualizado com os patches de segurança mais recentes.
- Manter as bibliotecas do sistema atualizadas com os patches de segurança mais recentes.
- Proteger as chaves privadas. Para obter mais informações, consulte the section called “Gerenciamento de chaves”.

Para dispositivos cliente, é sua responsabilidade:

- Manter a pilha de TLS atualizada.
- Proteger as chaves privadas.

Dados no dispositivo de núcleo

AWS IoT Greengrass não criptografa dados trocados localmente no dispositivo principal do Greengrass porque os dados não saem do dispositivo. Isso inclui a comunicação entre funções Lambda definidas pelo usuário, conectores, AWS IoT Greengrass o SDK principal e componentes do sistema, como o gerenciador de streams.

Criptografia em repouso

AWS IoT Greengrass armazena seus dados:

- the section called “Dados em repouso no Nuvem 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 no Nuvem AWS

AWS IoT Greengrass criptografa os dados do cliente armazenados no Nuvem AWS. Esses dados são protegidos usando AWS KMS chaves gerenciadas por AWS IoT Greengrass.

Dados em repouso sobre o núcleo do Greengrass

AWS IoT Greengrass depende das permissões de arquivo Unix e da criptografia de disco inteiro (se ativada) para proteger os dados em repouso no núcleo. É sua responsabilidade proteger o sistema de arquivos e o dispositivo.

No AWS IoT Greengrass entanto, criptografa cópias locais de seus segredos recuperados de. AWS Secrets Manager Para obter mais informações, consulte the section called “Criptografia de segredos”.

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. O AWS IoT Greengrass usa chaves públicas e privadas para os seguintes cenários:

- 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, consulte the 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.

- A chave do servidor MQTT é usada para que o certificado do servidor MQTT autentique conexões TLS entre dispositivos de núcleo e cliente. Para obter mais informações, consulte the section called “Autorização e autenticação do dispositivo”.
- O secrets manager local também usa a chave de cliente da IoT para proteger a chave de dados usada para criptografar segredos locais, mas você pode fornecer sua própria chave privada. Para obter mais informações, consulte the section called “Criptografia de segredos”.

Um núcleo do Greengrass oferece suporte para o armazenamento de chaves privadas usando permissões do sistema de arquivos, módulos de segurança de hardware, ou ambos. Se você usar

chaves privadas baseadas no sistema de arquivos, você será responsável por seu armazenamento seguro no dispositivo de núcleo.

Em um núcleo do Greengrass, a localização de suas chaves privadas são especificadas na seção `crypto` do arquivo `config.json`. Se você configurar o núcleo para usar uma chave fornecida pelo cliente para o certificado de servidor MQTT, é sua responsabilidade girar a chave. Para obter mais informações, consulte the section called “Entidades principais de segurança”.

Para dispositivos cliente, é sua responsabilidade manter a pilha TLS atualizada e proteger chaves privadas. As chaves privadas são usadas com certificados de dispositivo para autenticar conexões TLS com o AWS IoT Greengrass serviço.

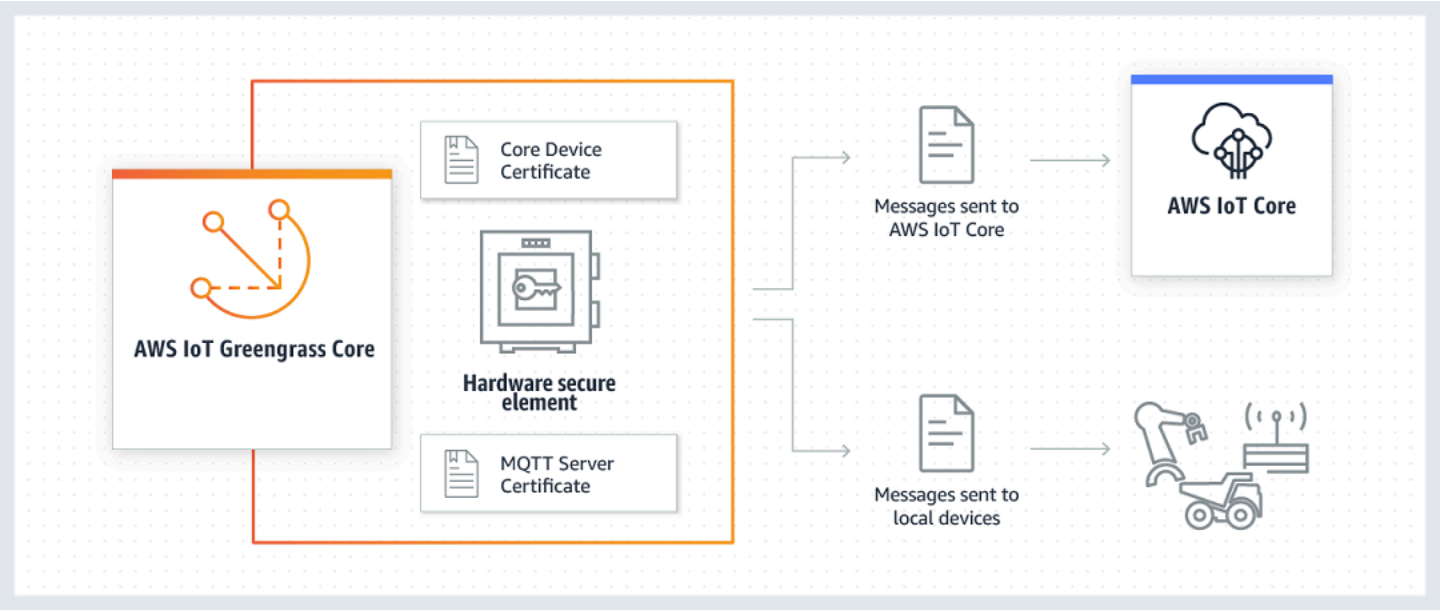
Integração de segurança de hardware

Esse recurso está disponível para AWS IoT Greengrass Core v1.7 e versões posteriores.

AWS IoT Greengrass suporta o uso de módulos de segurança de hardware (HSM) por meio da interface PKCS #11 para armazenamento seguro e descarregamento de chaves privadas. Isso impede que as chaves sejam expostas ou duplicados no software. As chaves privadas podem ser armazenados de forma segura em módulos de hardware, como HSMs, módulos de plataforma confiáveis (TPM, Trusted Platform Module), ou outros elementos de criptografia.

Pesquise dispositivos qualificados para esse atributo no Catálogo de dispositivos AWS Partner.

O diagrama a seguir mostra a arquitetura de segurança de hardware de um AWS IoT Greengrass núcleo.



Em uma instalação padrão, AWS IoT Greengrass usa duas chaves privadas. Uma chave é usada pelo componente AWS IoT cliente (cliente IoT) durante o handshake TLS (Transport Layer Security) quando um núcleo do Greengrass se conecta a. AWS IoT Core(Essa chave também é conhecida como chave privada do núcleo.) A outra chave é usada pelo servidor MQTT local, o que permite que os Dispositivos Greengrass se comuniquem com o núcleo do Greengrass. Se você quiser usar a segurança de hardware para ambos os componentes, você pode usar uma chave privada compartilhado ou chaves privadas distintas. Para obter mais informações, consulte the section called “ Práticas de provisionamento”.

> **Note**
>
> Em uma instalação padrão, o gerenciador de segredos local também usa a chave cliente da IoT para o processo de criptografia, mas você pode usar sua própria chave privada. Ela deve ser uma chave RSA com tamanho mínimo de 2048 bits. Para obter mais informações, consulte the section called “Especificar a chave privada para criptografia de segredos”.

Requisitos

Antes de configurar a segurança de hardware para um núcleo do Greengrass, é necessário ter o seguinte:

- Um módulo de segurança de hardware (HSM) que ofereça suporte à configuração de chave privada alvo para o cliente da IoT, servidor MQTT local e componentes do secrets manager. A configuração pode incluir um, dois ou três chaves privadas com base em hardware, dependendo da configuração dos componentes para compartilhar chaves. Para obter mais informações sobre a chave privada compatível, consulte the section called “Entidades principais de segurança”.
  - Para chaves RSA: um tamanho de RSA-2048 chave (ou maior) e um esquema de assinatura PKCS #1 v1.5.
  - Para chaves EC: uma curva NIST P-256 ou NIST P-384 .

  > **Note**
  >
  > Pesquise dispositivos qualificados para esse atributo no Catálogo de dispositivos AWS Partner.

- Uma biblioteca do provedor PKCS#11 carregável em tempo de execução (usando libdl) e que fornece funções PKCS#11.

- O módulo de hardware deve ser solucionado pelo rótulo do slot, conforme definido na especificação PKCS#11.
- A chave privada deve ser gerada e carregada no HSM usando as ferramentas de provisionamento fornecidas pelo fornecedor.
- A chave privada deve ser solucionada pelo rótulo do objeto.
- O certificado de dispositivo de núcleo. Este é um certificado de cliente da IoT que corresponde à chave privada.
- Se você estiver usando o atendente de atualização OTA do Greengrass, a biblioteca wrapper PKCS#11 do OpenSSL libp11 deve ser instalada. Para obter mais informações, consulte the section called “Configurar as atualizações OTA”.

Além disso, certifique-se de que as seguintes condições sejam atendidas:

- Os certificados do cliente de IoT associados à chave privada são registrados AWS IoT e ativados. Você pode verificar isso no AWS IoT console em Gerenciar, expandir Todos os dispositivos, escolher Coisas e escolher a guia Certificados para o item principal.
- O software AWS IoT Greengrass Core v1.7 ou posterior é instalado no dispositivo principal, conforme descrito no Módulo 2 do tutorial de introdução. É necessário ter a versão 1.9 ou posteriior para usar uma chave EC para o servidor MQTT.
- Os certificados são anexados ao núcleo do Greengrass. Você pode verificar isso na página Gerenciar para saber o que é essencial no AWS IoT console.

> **Note**
>
> Atualmente, AWS IoT Greengrass não dá suporte ao carregamento do certificado CA ou do certificado do cliente de IoT diretamente do HSM. Os certificados devem ser carregados como arquivos de texto simples no sistema de arquivos em um local que possa ser lido pelo Greengrass.

Configuração de segurança de hardware para um AWS IoT Greengrass core

A segurança do hardware é configurada no arquivo de configuração do Greengrass. Este é o arquivo `config.json` localizado no diretório `/greengrass-root/config`.

> **Note**
>
> Para analisar o processo de definição da configuração de um HSM usando uma implementação de software puro, consulte the section called “Módulo 7: Simular a integração de segurança de hardware”.
>
> > **Important**
> >
> > A configuração simulada no exemplo não fornece nenhum benefício de segurança. O objetivo é permitir que você saiba mais sobre a especificação PKCS#11 e faça testes iniciais do seu software se, futuramente, você planejar usar um HSM com base em hardware.

Para configurar a segurança de hardware em AWS IoT Greengrass, você edita o `crypto` objeto em`config.json`.

Ao usar a segurança de hardware, o objeto `crypto` é usado para especificar caminhos a certificados, chaves privadas e ativos para a biblioteca do provedor PKCS # 11 no núcleo, conforme exibido no exemplo a seguir.

```
"crypto": {
  "PKCS11" : {
    "OpenSSLEngine" : "/path-to-p11-openssl-engine",
    "P11Provider" : "/path-to-pkcs11-provider-so",
    "slotLabel" : "crypto-token-name",
    "slotUserPin" : "crypto-token-user-pin"
  },
  "principals" : {
    "IoTCertificate" : {
      "privateKeyPath" : "pkcs11:object=core-private-key-label;type=private",
      "certificatePath" : "file:///path-to-core-device-certificate"
    },
    "MQTTServerCertificate" : {
      "privateKeyPath" : "pkcs11:object=server-private-key-label;type=private"
    },
    "SecretsManager" : {
      "privateKeyPath": "pkcs11:object=core-private-key-label;type=private"
    }
  },
```

```
  "caPath" : "file:///path-to-root-ca"
```

O objeto `crypto` contém as seguintes propriedades:

Campo	Description	Observações
`caPath`	O caminho absoluto para a CA AWS IoT raiz.	Deve ser um URI de arquivo no formato: `file:///absolute/path/to/file` . **Note** Certifique-se de que os endpoints correspondem ao seu tipo de certificado.
`PKCS11`		
`OpenSSLEngine`	Opcional. O caminho absoluto para o arquivo `.so` do mecanismo OpenSSL para habilitar o suporte PKCS#11 no OpenSSL.	Deve ser um caminho para o arquivo no sistema de arquivos. Esta propriedade será necessária se você estiver usando o atendente de atualizações OTA do Greengrass com segurança de hardware. Para obter mais informações, consulte the section called “Configurar as atualizações OTA”.
`P11Provider`	O caminho absoluto para a biblioteca carregável libdl da implementação PKCS#11.	Deve ser um caminho para o arquivo no sistema de arquivos.

Campo	Description	Observações
`slotLabel`	O rótulo de slot usado para identificar o módulo de hardware.	Deve estar em conformidade com as especificações de rótulo PKCS#11.
`slotUserPin`	O PIN do usuário usado para autenticar o núcleo do Greengrass no módulo.	É necessário ter permissões suficientes para executar C_Sign com as chaves privadas configuradas.
`principals`		
`IoTCertificate`	O certificado e a chave privada que o núcleo usa para fazer solicitações à AWS IoT.	
`IoTCertificate.privateKeyPath`	O caminho para a chave privada do núcleo.	Para o armazenamento do sistema de arquivos, deve ser um URI de arquivo no formato: `file:///`*`absolute/path/to/file`* . Para o armazenamento HSM, deve ser um caminho RFC 7512 PKCS#11 que especifica o rótulo do objeto.
`IoTCertificate.certificatePath`	O caminho absoluto para o certificado do dispositivo do núcleo.	Deve ser um URI de arquivo no formato: `file:///`*`absolute/path/to/file`* .
`MQTTServerCertificate`	Opcional. A chave privada que o núcleo usa em combinação com o certificado para atuar como um gateway ou servidor MQTT.	

Campo	Description	Observações
`MQTTServerCertificate.privateKeyPath`	O caminho para a chave privada do servidor MQTT local.	Use esse valor para especificar sua própria chave privada para o servidor MQTT local. Para o armazenamento do sistema de arquivos, deve ser um URI de arquivo no formato: `file:///`*`absolute/path/to/file`* . Para o armazenamento HSM, deve ser um caminho RFC 7512 PKCS#11 que especifica o rótulo do objeto. Se essa propriedade for omitida, AWS IoT Greengrass gira a chave com base nas configurações de rotação. Se especificado, o cliente será responsável por girar a chave.
`SecretsManager`	A chave privada que protege a chave de dados usada para criptografia. Para obter mais informações, consulte Implantar segredos no núcleo.	

Campo	Description	Observações
`SecretsManager .privateKeyPath`	O caminho para a chave privada do secrets manager local.	Somente uma chave RSA é compatível. Para o armazenamento do sistema de arquivos, deve ser um URI de arquivo no formato: `file:///absolute/path/to/file` . Para o armazenamento HSM, deve ser um caminho RFC 7512 PKCS#11 que especific a o rótulo do objeto. A chave privada deve ser gerada usando o mecanismo de preenchimento PKCS#1 v1.5.

Campo	Description	Observações
`caPath`	O caminho absoluto para a CA AWS IoT raiz.	Deve ser um URI de arquivo no formato: `file:///absolute/path/to/file` . **Note** Certifique-se de que os endpoints correspondem ao seu tipo de certificado.
`PKCS11`		

Campo	Description	Observações
`OpenSSLEngine`	Opcional. O caminho absoluto para o arquivo `.so` do mecanismo OpenSSL para habilitar o suporte PKCS#11 no OpenSSL.	Deve ser um caminho para o arquivo no sistema de arquivos. Esta propriedade será necessária se você estiver usando o atendente de atualizações OTA do Greengrass com segurança de hardware. Para obter mais informações, consulte the section called “Configurar as atualizações OTA”.
`P11Provider`	O caminho absoluto para a biblioteca carregável libdl da implementação PKCS#11.	Deve ser um caminho para o arquivo no sistema de arquivos.
`slotLabel`	O rótulo de slot usado para identificar o módulo de hardware.	Deve estar em conformidade com as especificações de rótulo PKCS#11.
`slotUserPin`	O PIN do usuário usado para autenticar o núcleo do Greengrass no módulo.	É necessário ter permissões suficientes para executar C_Sign com as chaves privadas configuradas.
`principals`		
`IoTCertificate`	O certificado e a chave privada que o núcleo usa para fazer solicitações à AWS IoT.	

Campo	Description	Observações
`IoTCertificate.privateKeyPath`	O caminho para a chave privada do núcleo.	Para o armazenamento do sistema de arquivos, deve ser um URI de arquivo no formato: `file:///absolute/path/to/file` . Para o armazenamento HSM, deve ser um caminho RFC 7512 PKCS#11 que especifica o rótulo do objeto.
`IoTCertificate.certificatePath`	O caminho absoluto para o certificado do dispositivo do núcleo.	Deve ser um URI de arquivo no formato: `file:///absolute/path/to/file` .
`MQTTServerCertificate`	Opcional. A chave privada que o núcleo usa em combinação com o certificado para atuar como um gateway ou servidor MQTT.	

Campo	Description	Observações
`MQTTServerCertificate.privateKeyPath`	O caminho para a chave privada do servidor MQTT local.	Use esse valor para especificar sua própria chave privada para o servidor MQTT local. Para o armazenamento do sistema de arquivos, deve ser um URI de arquivo no formato: `file:///`*`absolute/path/to/file`* . Para o armazenamento HSM, deve ser um caminho RFC 7512 PKCS#11 que especifica o rótulo do objeto. Se essa propriedade for omitida, AWS IoT Greengrass gira a chave com base nas configurações de rotação. Se especificado, o cliente será responsável por girar a chave.
`SecretsManager`	A chave privada que protege a chave de dados usada para criptografia. Para obter mais informações, consulte Implantar segredos no núcleo.	

Campo	Description	Observações
`SecretsManager .privateKeyPath`	O caminho para a chave privada do secrets manager local.	Somente uma chave RSA é compatível. Para o armazenamento do sistema de arquivos, deve ser um URI de arquivo no formato: `file:///`*`absolute/path/to/file`* . Para o armazenamento HSM, deve ser um caminho RFC 7512 PKCS#11 que especific a o rótulo do objeto. A chave privada deve ser gerada usando o mecanismo de preenchimento PKCS#1 v1.5.

Campo	Description	Observações
`caPath`	O caminho absoluto para a CA AWS IoT raiz.	Deve ser um URI de arquivo no formato: `file:///`*`absolute/path/to/file`* . **Note** Certifique-se de que os endpoints correspondem ao seu tipo de certificado.
`PKCS11`		

Campo	Description	Observações
`OpenSSLEngine`	Opcional. O caminho absoluto para o arquivo `.so` do mecanismo OpenSSL para habilitar o suporte PKCS#11 no OpenSSL.	Deve ser um caminho para o arquivo no sistema de arquivos. Esta propriedade será necessária se você estiver usando o atendente de atualizações OTA do Greengrass com segurança de hardware. Para obter mais informações, consulte the section called “Configurar as atualizações OTA”.
`P11Provider`	O caminho absoluto para a biblioteca carregável libdl da implementação PKCS#11.	Deve ser um caminho para o arquivo no sistema de arquivos.
`slotLabel`	O rótulo de slot usado para identificar o módulo de hardware.	Deve estar em conformidade com as especificações de rótulo PKCS#11.
`slotUserPin`	O PIN do usuário usado para autenticar o núcleo do Greengrass no módulo.	É necessário ter permissões suficientes para executar C_Sign com as chaves privadas configuradas.
`principals`		
`IoTCertificate`	O certificado e a chave privada que o núcleo usa para fazer solicitações à AWS IoT.	

Campo	Description	Observações
`IoTCertificate .privateKeyPath`	O caminho para a chave privada do núcleo.	Para o armazenamento do sistema de arquivos, deve ser um URI de arquivo no formato: `file:///absolute/path/to/file` . Para o armazenamento HSM, deve ser um caminho RFC 7512 PKCS#11 que especifica o rótulo do objeto.
`IoTCertificate .certificatePath`	O caminho absoluto para o certificado do dispositivo do núcleo.	Deve ser um URI de arquivo no formato: `file:///absolute/path/to/file` .
`MQTTServerCertificate`	Opcional. A chave privada que o núcleo usa em combinação com o certificado para atuar como um gateway ou servidor MQTT.	

Campo	Description	Observações
`MQTTServerCertificate.privateKeyPath`	O caminho para a chave privada do servidor MQTT local.	Use esse valor para especificar sua própria chave privada para o servidor MQTT local. Para o armazenamento do sistema de arquivos, deve ser um URI de arquivo no formato: `file:///`*`absolute/path/to/file`* . Para o armazenamento HSM, deve ser um caminho RFC 7512 PKCS#11 que especifica o rótulo do objeto. Se essa propriedade for omitida, AWS IoT Greengrass gira a chave com base nas configurações de rotação. Se especificado, o cliente será responsável por girar a chave.
`SecretsManager`	A chave privada que protege a chave de dados usada para criptografia. Para obter mais informações, consulte Implantar segredos no núcleo.	

Campo	Description	Observações
`SecretsManager .privateKeyPath`	O caminho para a chave privada do secrets manager local.	Somente uma chave RSA é compatível. Para o armazenamento do sistema de arquivos, deve ser um URI de arquivo no formato: `file:///absolute/path/to/file` . Para o armazenamento HSM, deve ser um caminho RFC 7512 PKCS#11 que especific a o rótulo do objeto. A chave privada deve ser gerada usando o mecanismo de preenchimento PKCS#1 v1.5.

Práticas de provisionamento para AWS IoT Greengrass segurança de hardware

Estas são práticas de provisionamento relacionadas ao desempenho e à segurança.

Segurança

- Gerar chaves privadas diretamente no HSM usando o hardware interno gerador de número aleatório.

  > **Note**
  >
  > Se você configurar chaves privadas para usar com esse recurso (seguindo as instruções fornecidas pelo fornecedor do hardware), saiba que AWS IoT Greengrass atualmente suporta somente o mecanismo de preenchimento PKCS1 v1.5 para criptografia e descriptografia de segredos locais. ??? AWS IoT Greengrass não oferece suporte ao Optimal Asymmetric Encryption Padding (OAEP).

- Configurar chaves privadas para proibir a exportação.

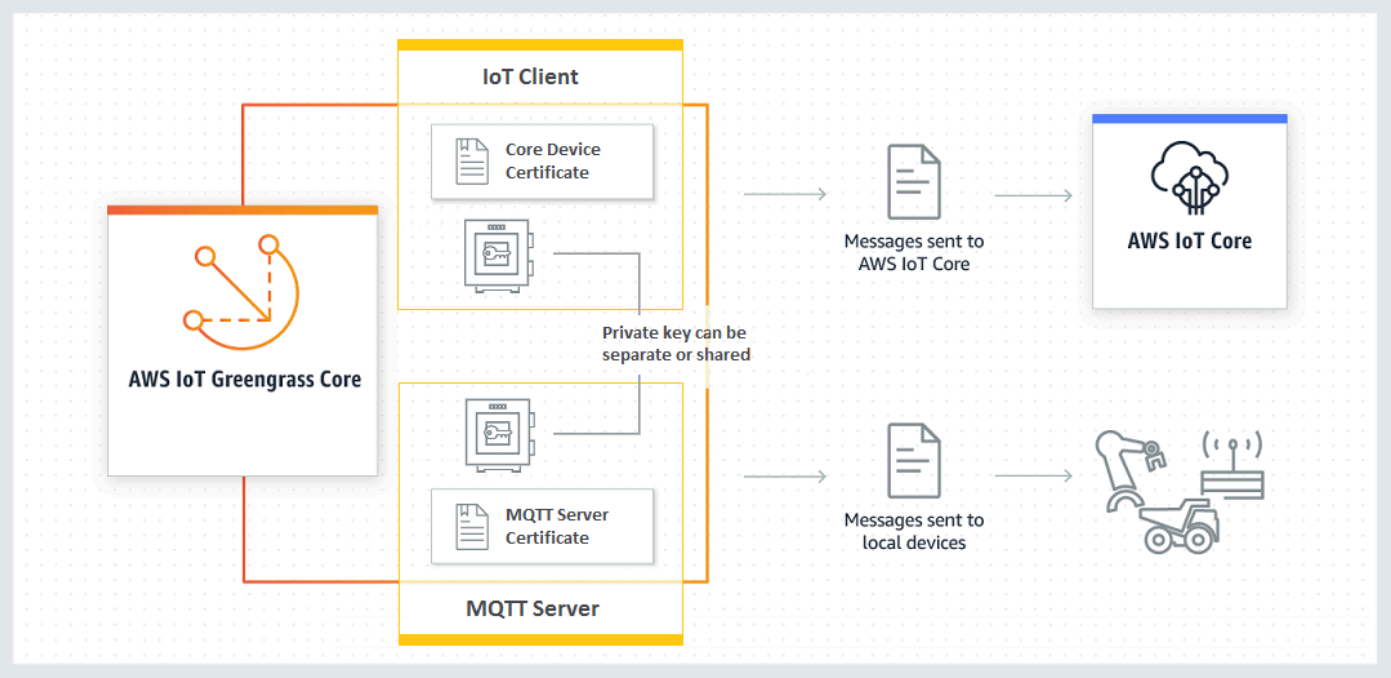
- Use a ferramenta de provisionamento fornecida pelo fornecedor do hardware para gerar uma solicitação de assinatura de certificado (CSR) usando a chave privada protegida por hardware e, em seguida, use o AWS IoT console para gerar um certificado de cliente.

> **Note**
>
> A prática de alternar chaves não se aplica quando as chaves privadas são geradas em HSM.

Desempenho

O diagrama a seguir mostra o componente cliente de IoT e o servidor MQTT local no AWS IoT Greengrass núcleo. Se você quiser usar uma configuração HSM para ambos os componentes, você pode usar a mesma chave privada ou chaves privadas distintas. Se você usar chaves separadas, elas precisam ser armazenados no mesmo slot.

> **Note**
>
> AWS IoT Greengrass não impõe nenhum limite ao número de chaves que você armazena no HSM, então você pode armazenar chaves privadas para os componentes do cliente IoT, do servidor MQTT e do gerenciador de segredos. No entanto, alguns fornecedores do HSM podem impor limites para o número de chaves que você pode armazenar em um slot.



Em geral, a chave do cliente de IoT não é usada com muita frequência porque o software AWS IoT Greengrass Core mantém conexões duradouras com a nuvem. No entanto, a chave do servidor MQTT é usada toda vez que um dispositivo do Greengrass se conecta ao núcleo. Essas interações afetam diretamente o desempenho.

Quando a chave do servidor MQTT é armazenada no HSM, a taxa na qual os dispositivos podem se conectar depende do número de operações de assinatura RSA por segundo que o HSM pode realizar. Por exemplo, se o HSM levar 300 milissegundos para realizar uma RSASSA-PKCS1-v1.5 assinatura em uma chave RSA-2048 privada, somente três dispositivos poderão se conectar ao núcleo do Greengrass por segundo. Depois que as conexões são feitas, o HSM não é mais usado e o padrão de cotas do AWS IoT Greengrass é aplicado.

Para atenuar gargalos de desempenho, você pode armazenar a chave privada para o servidor MQTT no sistema de arquivos, e não no HSM. Com essa configuração, o servidor MQTT se comporta como se a segurança de hardware não estivesse habilitada.

AWS IoT Greengrass suporta várias configurações de armazenamento de chaves para os componentes do cliente IoT e do servidor MQTT, para que você possa otimizar seus requisitos de segurança e desempenho. A tabela a seguir inclui exemplos de configurações.

Configuração	Chave da IoT	Chave MQTT	desempenho
Chave compartilhada do HSM	HSM: chave A	HSM: chave A	Limitada pelo HSM ou pela CPU
Chaves distintas do HSM	HSM: chave A	HSM: chave B	Limitada pelo HSM ou pela CPU
HSM apenas para IoT	HSM: chave A	Sistema de arquivos: chave B	Limitada pela CPU
Legada	Sistema de arquivos: chave A	Sistema de arquivos: chave B	Limitada pela CPU

Para configurar o núcleo do Greengrass para usar chaves baseadas no sistema de arquivos para o servidor MQTT, omita a `principals.MQTTServerCertificate` seção de `config.json` (ou especifique um caminho baseado em arquivo para a chave se você não estiver usando a chave padrão gerada por). AWS IoT Greengrass O objeto resultante `crypto` é semelhante a:

```
"crypto": {
  "PKCS11": {
    "OpenSSLEngine": "...",
    "P11Provider": "...",
    "slotLabel": "...",
    "slotUserPin": "..."
  },
  "principals": {
    "IoTCertificate": {
      "privateKeyPath": "...",
      "certificatePath": "..."
    },
    "SecretsManager": {
      "privateKeyPath": "..."
    }
  },
  "caPath" : "..."
}
```

Pacotes de criptografia compatíveis com a integração de segurança de hardware

AWS IoT Greengrass oferece suporte a um conjunto de pacotes de cifras quando o núcleo está configurado para segurança de hardware. Este é um subconjunto de pacotes de criptografia compatíveis quando o núcleo é configurado para usar segurança baseada em arquivos. Para obter mais informações, consulte the section called “Suporte a pacotes de criptografia TLS”.

> **Note**
>
> Quando estiver conectando ao núcleo do Greengrass em Dispositivos Greengrass em uma rede local, certifique-se de usar um dos pacotes de criptografia compatíveis para a conexão TLS.

Configurar suporte para atualizações OTA (over-the-air)

Para habilitar atualizações remotas (OTA) do software AWS IoT Greengrass Core ao usar a segurança de hardware, você deve instalar a biblioteca de wrapper OpenSC libp11 PKCS #11 e editar o arquivo de configuração do Greengrass. Para obter mais informações sobre atualizações OTA, consulte Atualizações OTA do software AWS IoT Greengrass Core.

1. Pare o daemon do Greengrass.

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

   > **Note**
   >
   > *greengrass-root*representa o caminho em que o software AWS IoT Greengrass Core está instalado em seu dispositivo. Normalmente, esse é o diretório `/greengrass`.

2. Instale o mecanismo OpenSSL. OpenSSL 1.0 ou 1.1 é compatível.

   ```
   sudo apt-get install libengine-pkcs11-openssl
   ```

3. Encontre o caminho para o mecanismo OpenSSL (`libpkcs11.so`) no seu sistema:

   a. Obtenha a lista de pacotes instalados para a biblioteca.

```
sudo dpkg -L libengine-pkcs11-openssl
```

O arquivo `libpkcs11.so` está localizado no diretório `engines`.

b. Copiar o caminho completo para o arquivo (por exemplo, `/usr/lib/ssl/engines/libpkcs11.so`).

4. Abra o arquivo de configuração do Greengrass. Este é o arquivo `config.json` no diretório */greengrass-root*`/config`.
5. Para a propriedade `OpenSSLEngine`, insira o caminho para o arquivo `libpkcs11.so`.

```
{
 "crypto": {
   "caPath" : "file:///path-to-root-ca",
   "PKCS11" : {
     "OpenSSLEngine" : "/path-to-p11-openssl-engine",
     "P11Provider" : "/path-to-pkcs11-provider-so",
     "slotLabel" : "crypto-token-name",
     "slotUserPin" : "crypto-token-user-pin"
    },
    ...
  }
  ...
}
```

> **Note**
>
> Se a propriedade `OpenSSLEngine` não existir no objeto `PKCS11`, adicione-a.

6. Inicie o daemon do Greengrass.

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

Compatibilidade retroativa com versões anteriores do AWS IoT Greengrass software principal

O software AWS IoT Greengrass Core com suporte à segurança de hardware é totalmente compatível com versões anteriores dos `config.json` arquivos gerados para a versão 1.6 e versões

anteriores. Se o `crypto` objeto não estiver presente no arquivo de `config.json` configuração, AWS IoT Greengrass usará as propriedades `coreThing.certPathcoreThing.keyPath`, e `coreThing.caPath` baseadas em arquivo. Essa compatibilidade com versões anteriores se aplica às atualizações OTA do Greengrass que não substituem uma configuração baseada em arquivo especificada em `config.json`.

Hardware sem suporte a 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), talvez seja possível usar o software de código aberto existente. No entanto, se seu hardware não tiver uma implementação de biblioteca PKCS #11 correspondente ou se você quiser criar um provedor PKCS #11 personalizado, entre em contato com seu representante de suporte AWS corporativo com perguntas relacionadas à integração.

Consulte também

- Guia de uso da interface de token criptográfico PKCS #11 Versão 2.40. Editada por John Leiseboer e Robert Griffin. 16 de novembro de 2014. Nota 02 do Comitê OASIS. http://docs.oasis-open.org/pkcs11/pkcs11-ug/v2.40/cn02/pkcs11-ug-v2.40-cn02.html. Versão mais recente: http://docs.oasis-open.org/pkcs11/pkcs11-ug/v2.40/pkcs11-ug-v2.40.html.
- RFC 7512
- PKCS #1: RSA Encryption versão 1.5

Autenticação e autorização de dispositivos para AWS IoT Greengrass

Os dispositivos em AWS IoT Greengrass ambientes usam X.509 certificados para autenticação e AWS IoT políticas para autorização. Certificados e políticas permitem que os dispositivos se conectem com segurança entre si AWS IoT Core, e. AWS IoT Greengrass

X.509 certificados são certificados digitais que usam o padrão de infraestrutura de chave X.509 pública para associar uma chave pública à identidade contida em um certificado. X.509 os certificados são emitidos por uma entidade confiável chamada autoridade de certificação (CA). A CA mantém um ou mais certificados especiais chamados certificados CA que ela usa para emitir X.509 certificados. Somente a autoridade de certificação tem acesso aos certificados CA.

AWS IoT as políticas definem o conjunto de operações permitidas para AWS IoT dispositivos. Especificamente, eles permitem e negam o acesso AWS IoT Core e as operações do plano de AWS IoT Greengrass dados, como publicar mensagens MQTT e recuperar sombras do dispositivo.

Todos os dispositivos exigem uma entrada no AWS IoT Core registro e um X.509 certificado ativado com uma AWS IoT política anexada. Os dispositivos se enquadram em duas categorias:

- Núcleos do Greengrass. Os principais dispositivos do Greengrass usam certificados e AWS IoT políticas aos quais se conectar. AWS IoT Core Os certificados e políticas também permitem AWS IoT Greengrass implantar informações de configuração, funções Lambda, conectores e assinaturas gerenciadas nos dispositivos principais.
- Dispositivos cliente. Os dispositivos cliente (também chamados dispositivos conectados, dispositivos Greengrass ou dispositivos) se conectam a um núcleo do Greengrass por meio do MQTT. Eles usam certificados e políticas para se conectar ao AWS IoT Core AWS IoT Greengrass serviço. Isso permite que dispositivos clientes usem o AWS IoT Greengrass Discovery Service para encontrar e se conectar a um dispositivo principal. Um dispositivo cliente usa o mesmo certificado para se conectar ao gateway do AWS IoT Core dispositivo e ao dispositivo principal. Os dispositivos cliente também usam informações de descoberta para autenticação mútua com o dispositivo de núcleo. Para obter mais informações, consulte the section called “Fluxo de trabalho de conexão de dispositivo” e the section called “Gerenciar a autenticação de dispositivos com o núcleo do Greengrass”.

X.509 certificados

A comunicação entre dispositivos principais e clientes e entre dispositivos AWS IoT Core e/ou AWS IoT Greengrass deve ser autenticada. Essa autenticação mútua é baseada em certificados de X.509 dispositivos registrados e chaves criptográficas.

Em um AWS IoT Greengrass ambiente, os dispositivos usam certificados com chaves públicas e privadas para as seguintes conexões TLS (Transport Layer Security):

- O componente AWS IoT cliente no núcleo do Greengrass que se conecta à AWS IoT Core e AWS IoT Greengrass pela Internet.
- Dispositivos clientes que se conectam AWS IoT Greengrass para obter as principais informações de descoberta pela Internet.
- O componente de servidor MQTT no núcleo do Greengrass que se conecta a dispositivos cliente no grupo pela rede local.

O dispositivo AWS IoT Greengrass principal armazena certificados em dois locais:

- Certificado de dispositivo de núcleo em */greengrass-root*/certs. Normalmente, o certificado de dispositivo de núcleo é nomeado *hash*.cert.pem (por exemplo, 86c84488a5.cert.pem). Esse certificado é usado pelo AWS IoT cliente para autenticação mútua quando o núcleo se conecta aos AWS IoT Greengrass serviços AWS IoT Core e.
- Certificado do servidor MQTT em */greengrass-root*/ggc/var/state/server. O certificado de servidor MQTT é nomeado server.crt. Este certificado é usado para autenticação mútua entre o servidor MQTT local (no núcleo do Greengrass) e Dispositivos Greengrass.

> **Note**
>
> *greengrass-root*representa o caminho em que o software AWS IoT Greengrass Core está instalado em seu dispositivo. Normalmente, esse é o diretório /greengrass.

Para obter mais informações, consulte the section called “Entidades principais de segurança”.

Certificados da autoridade de certificação (CA)

Os dispositivos principais e os dispositivos clientes baixam um certificado CA raiz usado para autenticação AWS IoT Core e AWS IoT Greengrass serviços. 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 .

> **Note**
>
> O tipo de certificado raiz da CA deve corresponder ao endpoint. Use um certificado de CA raiz ATS com um endpoint ATS (preferencial) ou um certificado de CA VeriSign raiz com um endpoint legado. Os endpoints legados são compatíveis apenas com algumas Regiões do Amazon Web Services. Para obter mais informações, consulte the section called “Os endpoints do serviço devem corresponder ao tipo de certificado”.

Os dispositivos cliente também fazem download do certificado CA do grupo do Greengrass. Isso é usado para validar o certificado de servidor MQTT no núcleo do Greengrass durante a autenticação

mútua. Para obter mais informações, consulte the section called “Fluxo de trabalho de conexão de dispositivo”. A expiração padrão do certificado do servidor MQTT é de sete dias.

Troca de certificados no servidor MQTT local

Os dispositivos cliente usam o certificado de servidor MQTT local para autenticação mútua com o dispositivo de núcleo do Greengrass. Por padrão, esse certificado expira em 7 dias. Esse período limitado é baseado nas melhores práticas de segurança. O certificado do servidor MQTT é assinado pelo certificado CA do grupo, que é armazenado na nuvem.

Para que a rotação do certificado ocorra, seu dispositivo principal do Greengrass deve estar on-line e ser capaz de acessar o AWS IoT Greengrass serviço diretamente regularmente. Quando o certificado expira, o dispositivo principal tenta se conectar ao AWS IoT Greengrass serviço para obter um novo certificado. Se a conexão for bem-sucedida, o dispositivo de núcleo fará download de um novo certificado do servidor MQTT e reiniciará o serviço MQTT local. Nesse ponto do processo, todos os dispositivos cliente conectados ao núcleo serão desconectados. Se o dispositivo de núcleo estiver offline no momento da expiração, ele não receberá o certificado de substituição. Todas as novas tentativas para se conectar ao dispositivo de núcleo serão rejeitadas. As conexões existentes não são afetadas. Os dispositivos cliente não podem se conectar ao dispositivo principal até que a conexão com o AWS IoT Greengrass serviço seja restaurada e um novo certificado do servidor MQTT possa ser baixado.

Você pode definir a expiração para qualquer valor entre 7 e 30 dias, dependendo das suas necessidades. Rotação mais frequente requer conexão em nuvem mais frequente. Rotação menos frequente pode apresentar problemas de segurança. Para definir a expiração do certificado para um valor acima de 30 dias, entre em contato com o Suporte.

No AWS IoT console, você pode gerenciar o certificado na página de configurações do grupo. Na AWS IoT Greengrass API, você pode usar a UpdateGroupCertificateConfigurationação.

Quando o certificado do servidor MQTT expirar, qualquer tentativa de validação deste apresentará falha. Os dispositivos cliente devem ser capazes de detectar a falha e encerrar a conexão.

AWS IoT políticas para operações de plano de dados

Use AWS IoT políticas para autorizar o acesso ao plano de AWS IoT Greengrass dados AWS IoT Core e. O plano de AWS IoT Core dados consiste em operações para dispositivos, usuários e aplicativos, como conexão AWS IoT Core e assinatura de tópicos. O plano de dados do AWS IoT Greengrass consiste em operações para Dispositivos Greengrass, como recuperação de implantações e atualização de informações de conectividade.

Uma AWS IoT política 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.

AWS IoT as políticas são suportadas `*` como um caractere curinga e tratam os caracteres curinga (+e#) do MQTT como cadeias de caracteres 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 .

> **Note**
>
> AWS IoT Core permite que você anexe AWS IoT políticas a grupos de coisas para definir permissões para grupos de dispositivos. As políticas do Thing Group não permitem acesso às operações do plano de AWS IoT Greengrass dados. Para permitir que uma coisa acesse uma operação de plano de AWS IoT Greengrass dados, adicione a permissão a uma AWS IoT política que você anexa ao certificado da coisa.

AWS IoT Greengrass ações políticas

Ações do núcleo do Greengrass

AWS IoT Greengrass define as seguintes ações políticas que os dispositivos principais do Greengrass podem usar nas AWS IoT políticas:

`greengrass:AssumeRoleForGroup`

Permissão para um dispositivo de núcleo do Greengrass recuperar credenciais usando a função do Lambda do sistema Token Exchange Service (TES). As permissões vinculadas às credenciais recuperadas baseiam-se na política anexada à função de grupo configurada.

Essa permissão é verificada quando um dispositivo de núcleo do Greengrass tenta recuperar credenciais (assumindo que as credenciais não sejam armazenadas em cache localmente).

`greengrass:CreateCertificate`

Permissão para um dispositivo de núcleo do Greengrass criar seu próprio certificado de servidor.

Esta permissão é verificada quando um dispositivo de núcleo do Greengrass cria um certificado. Os dispositivos de núcleo do Greengrass tentam criar um certificado de servidor na primeira execução, quando as informações de conectividade do núcleo mudam e em períodos de rotação designados.

`greengrass:GetConnectivityInfo`

Permissão para um dispositivo de núcleo do Greengrass recuperar suas próprias informações de conectividade.

Essa permissão é verificada quando um dispositivo principal do Greengrass tenta recuperar suas informações de conectividade do. AWS IoT Core

`greengrass:GetDeployment`

Permissão para um dispositivo de núcleo do Greengrass recuperar implantações.

Essa permissão é verificada quando um dispositivo de núcleo do Greengrass tenta recuperar implantações e status de implantação da nuvem.

`greengrass:GetDeploymentArtifacts`

Permissão para um dispositivo de núcleo do Greengrass recuperar artefatos de implantação, como informações de grupo ou funções do Lambda.

Essa permissão é verificada quando um dispositivo de núcleo do Greengrass recebe uma implantação e tenta recuperar artefatos de implantação.

`greengrass:UpdateConnectivityInfo`

Permissão para um dispositivo de núcleo do Greengrass atualizar suas próprias informações de conectividade com informações de IP ou nome de host.

Essa permissão é verificada quando um dispositivo de núcleo do Greengrass tenta atualizar suas informações de conectividade na nuvem.

`greengrass:UpdateCoreDeploymentStatus`

Permissão para um dispositivo de núcleo do Greengrass atualizar o status de uma implantação.

Essa permissão é verificada quando um dispositivo de núcleo do Greengrass recebe uma implantação e tenta atualizar o status da implantação.

Ações do dispositivo do Greengrass

AWS IoT Greengrass define a seguinte ação de política que os dispositivos cliente podem usar nas AWS IoT políticas:

`greengrass:Discover`

Permissão para um dispositivo cliente usar a API de descoberta para recuperar as informações de conectividade do núcleo do grupo e a autoridade de certificação do grupo.

Essa permissão é verificada quando um dispositivo cliente chama a API de descoberta com autenticação TLS mútua.

Mínimo AWS IoT política para o AWS IoT Greengrass dispositivo principal

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 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 funções AWS IoT Core, conectores e dispositivos cliente do Lambda no grupo Greengrass, especifique os tópicos e os filtros de tópicos que você deseja permitir. Para obter mais informações, consulte exemplos Publish/Subscribe de políticas no Guia do AWS IoT Core desenvolvedor.
- A política inclui uma seção que permite AWS IoT Core obter, atualizar e excluir a sombra do dispositivo principal. Para permitir a sincronização da sombra para dispositivos cliente no grupo do Greengrass, especifique os Nomes de recurso da Amazon (ARNs) de destino na lista `Resource` (por exemplo, `arn:aws:iot:`*`region`*`:`*`account-id`*`:thing/`*`device-name`*).
- O uso de variáveis de política de coisas (`iot:Connection.Thing.*`) na AWS IoT política de um dispositivo principal não é suportado. O núcleo usa o mesmo certificado de dispositivo para fazer várias conexões AWS IoT Core , mas o ID do cliente em uma conexão pode não corresponder exatamente ao nome do item principal.

- Para `greengrass:UpdateCoreDeploymentStatus` obter a permissão, o segmento final no `Resource` ARN é o URL-encoded ARN do dispositivo principal.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot:Connect"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:client/core-name-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Publish",
                "iot:Receive"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topic/$aws/things/core-name-
*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:Subscribe"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:topicfilter/$aws/things/core-
name-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "iot:GetThingShadow",
```

```
                "iot:UpdateThingShadow",
                "iot:DeleteThingShadow"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:thing/core-name-*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:AssumeRoleForGroup",
                "greengrass:CreateCertificate"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:GetDeployment"
            ],
            "Resource": [
                "arn:aws:greengrass:us-east-1:123456789012:/greengrass/
groups/group-id/deployments/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:GetDeploymentArtifacts"
            ],
            "Resource": [
                "arn:aws:greengrass:us-east-1:123456789012:/greengrass/
groups/group-id/deployments/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:UpdateCoreDeploymentStatus"
            ],
            "Resource": [
```

```
                "arn:aws:greengrass:us-east-1:123456789012:/
greengrass/groups/group-id/deployments/*/cores/arn%3Aaws%3Aiot
%3Aregion%3A123456789012%3Athing%2Fcore-name"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "greengrass:GetConnectivityInfo",
                "greengrass:UpdateConnectivityInfo"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:thing/core-name-*"
            ]
        }
    ]
}
```

Note

AWS IoT as políticas para dispositivos cliente geralmente exigem permissões semelhantes para `iot:Connect`iot:Publish,`iot:Receive`, e `iot:Subscribe` ações.
Para permitir que um dispositivo cliente detecte automaticamente as informações de conectividade dos núcleos nos grupos do Greengrass aos quais o dispositivo pertence, a AWS IoT política de um dispositivo cliente deve incluir a `greengrass:Discover` ação. Na seção `Resource`, especifique o ARN do dispositivo cliente, não o ARN do dispositivo de núcleo do Greengrass. Por exemplo:

```
{
    "Effect": "Allow",
    "Action": [
        "greengrass:Discover"
    ],
    "Resource": [
        "arn:aws:iot:region:account-id:thing/device-name"
    ]
}
```

A AWS IoT política para dispositivos cliente normalmente não exige permissões ou `iot:DeleteThingShadow` ações `iot:GetThingShadow`iot:UpdateThingShadow,

porque o núcleo do Greengrass gerencia as operações de sincronização de sombra para dispositivos cliente. Nesse caso, certifique-se de que a `Resource` seção de ações paralelas na AWS IoT política do núcleo inclua os ARNs dos dispositivos cliente.

No AWS IoT console, você pode visualizar e editar a política anexada ao certificado do seu núcleo.

1. No painel de navegação, em Gerenciar, expanda Todos os dispositivos e, em seguida selecione Coisas.
2. Selecione seu núcleo
3. Na página de configuração do seu núcleo, selecione a guia Certificados.
4. Na guia Certificados, selecione seu certificado.
5. Na página de configuração do certificado, selecione Políticas e, em seguida, selecione a política.

   Se quiser editar a política, selecione Editar versão ativa.
6. Revise a política e adicione, remova ou edite as permissões, conforme necessário.
7. 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.
8. Selecione Salvar como nova versão.

Gerenciamento de identidade e acesso para AWS IoT Greengrass

AWS Identity and Access Management (IAM) é uma ferramenta AWS service (Serviço da AWS) que ajuda o administrador a controlar com segurança o acesso aos AWS recursos. Os administradores do IAM controlam quem pode ser autenticado (conectado) e autorizado (tem permissões) a usar AWS IoT Greengrass os recursos. O IAM é um AWS service (Serviço da AWS) que você pode usar sem custo adicional.

> **Note**
>
> Este tópico descreve conceitos e atributos do IAM. Para obter informações sobre os recursos do IAM compatíveis com AWS IoT Greengrass, consultethe section called “Como AWS IoT Greengrass funciona com o IAM”.

Público

A forma como você usa AWS Identity and Access Management (IAM) difere com base na sua função:

- Usuário do serviço: solicite permissões ao seu administrador se você não conseguir acessar os atributos (consulte Solução de problemas de identidade e acesso para AWS IoT Greengrass).
- Administrador do serviço: determine o acesso do usuário e envie solicitações de permissão (consulte Como AWS IoT Greengrass funciona com o IAM)
- Administrador do IAM: escreva políticas para gerenciar o acesso (consulte Identity-based exemplos de políticas para AWS IoT Greengrass)

Autenticação com identidades

A autenticação é como você faz login AWS usando suas credenciais de identidade. Você deve estar autenticado como usuário do IAM ou assumindo uma função do IAM. Usuário raiz da conta da AWS

Você pode fazer login como uma identidade federada usando credenciais de uma fonte de identidade como Centro de Identidade do AWS IAM (IAM Identity Center), autenticação de login único ou credenciais. Google/Facebook Para ter mais informações sobre como fazer login, consulte Como fazer login em sua Conta da AWS no Guia do usuário do Início de Sessão da AWS .

Para acesso programático, AWS fornece um SDK e uma CLI para assinar solicitações criptograficamente. Para ter mais informações, consulte AWS Signature Version 4 para solicitações de API no Guia do usuário do IAM.

Conta da AWS usuário-raiz

Ao criar um Conta da AWS, você começa com uma identidade de login chamada usuário Conta da AWS raiz que tem acesso completo a todos Serviços da AWS os recursos. É altamente recomendável não usar o usuário-raiz em tarefas diárias. Consulte as tarefas que exigem credenciais de usuário-raiz em 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 com permissões específicas para uma única pessoa ou aplicação. É recomendável usar credenciais temporárias, em vez de usuários do IAM com credenciais de longo prazo. Para obter mais informações, consulte Exigir que usuários humanos

usem a federação com um provedor de identidade para acessar AWS usando credenciais temporárias no Guia do usuário do IAM.

Um grupo do IAM especifica um conjunto de usuários do IAM e facilita o gerenciamento de permissões para grandes conjuntos de usuários. Para ter mais informações, consulte Casos de uso de usuários do IAM no Guia do usuário do IAM.

Perfis do IAM

Uma perfil do IAM é uma identidade com permissões específicas que oferece credenciais temporárias. Você pode assumir uma função mudando de um usuário para uma função do IAM (console) ou chamando uma operação de AWS API AWS CLI ou. Para saber mais, consulte Métodos para assumir um perfil no Manual do usuário do IAM.

Os perfis do IAM são úteis para acesso de usuário federado, permissões de usuário do IAM temporárias, acesso entre contas, acesso entre serviços e aplicações em execução no Amazon EC2. Consulte mais informações em Acesso a recursos entre contas no IAM no Guia do usuário do IAM.

Gerenciar o acesso usando políticas

Você controla o acesso AWS criando políticas e anexando-as a AWS identidades ou recursos. Uma política define permissões quando associada a uma identidade ou recurso. AWS avalia essas políticas quando um diretor faz uma solicitação. A maioria das políticas é armazenada AWS como documentos JSON. Para ter mais informações sobre documentos de política JSON, consulte Visão geral das políticas JSON no Guia do usuário do IAM.

Por meio de políticas, os administradores especificam quem tem acesso a que, definindo qual entidade principal pode realizar ações em quais recursos e sob quais condições.

Por padrão, usuários e perfis não têm permissões. Um administrador do IAM cria políticas do IAM e as adiciona aos perfis, os quais os usuários podem então assumir. As políticas do IAM definem permissões, independentemente do método usado para realizar a operação.

Identity-based políticas

Identity-based políticas são documentos de políticas de permissões JSON que você anexa a uma identidade (usuário, grupo ou função). Essas políticas controlam quais ações as identidades podem realizar, em quais recursos e sob quais condições. Para saber como criar uma política baseada em identidade, consulte Definir permissões personalizadas do IAM com as políticas gerenciadas pelo cliente no Guia do Usuário do IAM.

Identity-based as políticas podem ser políticas em linha (incorporadas diretamente em uma única identidade) ou políticas gerenciadas (políticas autônomas anexadas a várias identidades). Para saber como escolher entre uma política gerenciada e políticas em linha, consulte Escolher entre políticas gerenciadas e políticas em linha no Guia do usuário do IAM.

Resource-based políticas

Resource-based políticas são documentos de política JSON que você anexa a um recurso. Entre os exemplos estão políticas de confiança de perfil do IAM e políticas de bucket do Amazon S3. Em serviços compatíveis com políticas baseadas em recursos, os administradores de serviço podem usá-las para controlar o acesso a um recurso específico. É necessário especificar uma entidade principal em uma política baseada em recursos.

Resource-based políticas são políticas em linha localizadas nesse serviço. Você não pode usar políticas AWS gerenciadas do IAM em uma política baseada em recursos.

Listas de controle de acesso (ACLs)

As listas de controle de acesso (ACLs) controlam quais entidades principais (membros, usuários ou perfis da conta) têm permissões para acessar um recurso. As ACLs são semelhantes às políticas baseadas em recursos, embora não usem o formato de documento de política JSON.

O Amazon S3 e o Amazon VPC são exemplos de serviços que oferecem suporte a ACLs. AWS WAF 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

AWS oferece suporte a tipos de políticas adicionais que podem definir o máximo de permissões concedidas por tipos de políticas mais comuns:

- Limites de permissões: definem o número máximo de permissões que uma política baseada em identidade pode conceder a uma entidade do IAM. Para saber mais sobre limites de permissões, consulte Limites de permissões para identidades do IAM no Guia do usuário do IAM.
- Políticas de Controle de Serviços (SCPs): as SCPs especificam o número máximo de permissões para uma organização ou uma unidade organizacional no AWS Organizations. Para saber mais, consulte Políticas de controle de serviço no Guia do usuário do AWS Organizations .
- Políticas de controle de recursos (RCPs): definem o número máximo de permissões disponíveis para recursos em suas contas. Consulte mais informações em Resource control policies (RCPs) no Guia do usuário do AWS Organizations .

- Políticas de sessão: políticas avançadas transmitidas como um parâmetro durante a criação de uma sessão temporária para um perfil ou um usuário federado. Para saber mais, 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 AWS determinar se uma solicitação deve ser permitida quando vários tipos de políticas estão envolvidos, consulte Lógica de avaliação de políticas no Guia do usuário do IAM.

Consulte também

- the section called “Como AWS IoT Greengrass funciona com o IAM”
- the section called “Identity-based exemplos de políticas”
- the section called “Solução de problemas de identidade e acesso”

Como AWS IoT Greengrass funciona com o IAM

Antes de usar o IAM para gerenciar o acesso AWS IoT Greengrass, você deve entender os recursos do IAM com os quais você pode usar AWS IoT Greengrass.

Recurso do IAM	Compatível com o Greengrass?
Identity-based políticas com permissões em nível de recurso	Sim
Resource-based políticas	Não
Listas de controle de acesso (ACLs)	Não
Tags-based autorização	Não
Credenciais temporárias	Sim
Service-linked funções	Não
Perfis de serviço	Sim

Para uma visão de alto nível de como outros AWS serviços funcionam com o IAM, consulte AWS os serviços que funcionam com o IAM no Guia do usuário do IAM.

Identity-based políticas para AWS IoT Greengrass

Com as políticas baseadas em identidade do IAM, você pode especificar ações e recursos permitidos ou negados e as condições sob as quais as ações são permitidas ou negadas. 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 políticas AWS JSON para especificar quem tem acesso ao quê. Ou seja, qual entidade principal pode executar ações em quais recursos e em que condições.

O elemento `Action` de uma política JSON descreve as ações que podem ser usadas para permitir ou negar acesso em uma política. Incluem ações em uma política para conceder permissões para executar a operação associada.

Ações de política para AWS IoT Greengrass usar o `greengrass:` prefixo antes da ação. Por exemplo, para permitir que alguém use a operação da `ListGroups` API para listar os grupos Conta da AWS, você inclui a `greengrass:ListGroups` ação na política dessa pessoa. As declarações de política devem incluir um elemento `Action` ou AWS IoT Greengrass . O `NotAction` 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, coloque-as entre parênteses (`[ ]`) e separe-as com vírgulas, da seguinte forma:

```
"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 ver a lista completa de AWS IoT Greengrass ações, consulte Ações definidas por AWS IoT Greengrass no Guia do usuário do IAM.

Recursos

Os administradores podem usar políticas AWS JSON para especificar quem tem acesso ao quê. Ou seja, qual entidade principal pode executar ações em quais recursos e em que condições.

O elemento de política JSON `Resource` especifica o objeto ou os objetos aos quais a ação se aplica. Como prática recomendada, especifique um recurso usando seu nome do recurso da Amazon (ARN). Para ações que não oferecem compatibilidade com permissões em nível de recurso, use um curinga (*) para indicar que a instrução se aplica a todos os recursos.

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

A tabela a seguir contém os ARNs de AWS IoT Greengrass recursos que podem ser usados no `Resource` elemento de uma declaração de política. Para um mapeamento das permissões compatíveis em nível de recurso para AWS IoT Greengrass ações, consulte Ações definidas por AWS IoT Greengrass no Guia do usuário do IAM.

Recurso	ARN
Group	`arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/groups/${GroupId}`
GroupVersion	`arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/groups/${GroupId}/versions/${VersionId}`

Recurso	ARN
CertificateAuthority	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/groups/${GroupId}/certificateauthorities/${CertificateAuthorityId}
Deployment	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/groups/${GroupId}/deployments/${DeploymentId}
BulkDeployment	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/bulk/deployments/${BulkDeploymentId}
ConnectorDefinition	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/connectors/${ConnectorDefinitionId}
ConnectorDefinitionVersion	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/connectors/${ConnectorDefinitionId}/versions/${VersionId}
CoreDefinition	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/cores/${CoreDefinitionId}
CoreDefinitionVersion	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/cores/${CoreDefinitionId}/versions/${VersionId}
DeviceDefinition	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/devices/${DeviceDefinitionId}
DeviceDefinitionVersion	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/devices/${DeviceDefinitionId}/versions/${VersionId}
FunctionDefinition	arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/functions/${FunctionDefinitionId}

Recurso	ARN
FunctionDefinitionVersion	`arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/functions/${FunctionDefinitionId}/versions/${VersionId}`
LoggerDefinition	`arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/loggers/${LoggerDefinitionId}`
LoggerDefinitionVersion	`arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/loggers/${LoggerDefinitionId}/versions/${VersionId}`
ResourceDefinition	`arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/resources/${ResourceDefinitionId}`
ResourceDefinitionVersion	`arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/resources/${ResourceDefinitionId}/versions/${VersionId}`
SubscriptionDefinition	`arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/subscriptions/${SubscriptionDefinitionId}`
SubscriptionDefinitionVersion	`arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/definition/subscriptions/${SubscriptionDefinitionId}/versions/${VersionId}`
ConnectivityInfo	`arn:${Partition}:greengrass:${Region}:${Account}:/greengrass/things/${ThingName}/connectivityInfo`

O `Resource` elemento de exemplo a seguir especifica o ARN de um grupo na região Oeste dos EUA (Oregon) no: Conta da AWS `123456789012`

```
"Resource": "arn:aws:greengrass:us-west-2:123456789012:/greengrass/groups/
a1b2c3d4-5678-90ab-cdef-EXAMPLE11111
```

Ou, para especificar todos os grupos que pertencem a um Conta da AWS em um específico Região da AWS, use o caractere curinga no lugar da ID do grupo:

```
"Resource": "arn:aws:greengrass:us-west-2:123456789012:/greengrass/groups/*"
```

Algumas AWS IoT Greengrass ações (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 recurso em uma declaração, liste-os entre parênteses (`[ ]`) e separe-os com vírgulas, da seguinte forma:

```
"Resource": [
    "resource-arn1",
    "resource-arn2",
    "resource-arn3"
]
```

Para obter mais informações sobre formatos de ARN, consulte Amazon Resource Names (ARNs) e namespaces AWS de serviços no. Referência geral da Amazon Web Services

Chaves de condição

Os administradores podem usar políticas AWS JSON para especificar quem tem acesso ao quê. Ou seja, qual entidade principal pode executar ações em quais recursos e em que condições.

O elemento `Condition` especifica quando as instruções são executadas com base em critérios definidos. É possível criar expressões condicionais que usem agentes de condição, como “igual a” ou “menor que”, para fazer a condição da política corresponder aos valores na solicitação. Para ver todas as chaves de condição AWS globais, consulte as chaves de contexto de condição AWS global no Guia do usuário do IAM.

AWS IoT Greengrass suporta as seguintes chaves de condição globais.

Chave	Description
`aws:CurrentTime`	Filtra o acesso verificando date/time as condições da data e hora atuais.

Chave	Description
`aws:EpochTime`	Filtra o acesso verificando date/time as condições da data e hora atuais na época ou na hora do Unix.
`aws:MultiFactorAuthAge`	Filtra o acesso verificando há quanto tempo (em segundos) as credenciais de segurança validadas pela autenticação multifator (MFA) na solicitação foram emitidas usando a MFA.
`aws:MultiFactorAuthPresent`	Filtra o acesso verificando se a autenticação multifator (MFA) foi usada para validar as credenciais de segurança temporárias que fizeram a solicitação atual.
`aws:RequestTag/${TagKey}`	Filtra as solicitações de criação com base no conjunto de valores permitidos para cada uma das tags obrigatórias.
`aws:ResourceTag/${TagKey}`	Filtra as ações com base no valor da tag associada ao recurso.
`aws:SecureTransport`	Filtra o acesso verificando se a solicitação foi enviada usando SSL.
`aws:TagKeys`	Filtra solicitações de criação com base na presença de etiquetas obrigatórias na solicitação.
`aws:UserAgent`	Filtra o acesso pelo aplicativo cliente do solicitante.

Para obter mais informações, consulte Chaves de contexto de condição global da AWS no Guia do usuário do IAM.

Exemplos

Para ver exemplos de políticas AWS IoT Greengrass baseadas em identidade, consulte. the section called “Identity-based exemplos de políticas”

Resource-based políticas para AWS IoT Greengrass

AWS IoT Greengrass não oferece suporte a políticas baseadas em recursos.

Listas de controle de acesso (ACLs)

AWS IoT Greengrass não oferece suporte a ACLs.

Autorização baseada em AWS IoT Greengrass tags

AWS IoT Greengrass não oferece suporte à autorização baseada em tags.

Funções do IAM para AWS IoT Greengrass

Uma função do IAM é uma entidade dentro da sua Conta da AWS que tem permissões específicas.

Usando credenciais temporárias com AWS IoT Greengrass

Credenciais temporárias são usadas para fazer login com federação, assumir um perfil do IAM ou assumir uma função entre contas. Você obtém credenciais de segurança temporárias chamando operações de AWS STS API, como AssumeRoleou GetFederationToken.

No núcleo do Greengrass, credenciais temporárias para a função de grupo são disponibilizadas para funções e conectores do Lambda definidos pelo usuário. Se suas funções do Lambda usam o AWS SDK, você não precisa adicionar lógica para obter as credenciais, pois o AWS SDK faz isso por você.

Service-linked funções

AWS IoT Greengrass não oferece suporte a funções vinculadas a serviços.

Perfis de serviço

Esse atributo permite que um serviço assuma um perfil de serviço em seu nome. O perfil permite que o serviço acesse recursos em outros serviços para concluir uma ação em seu nome. Os perfis 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 esse perfil. Porém, fazer isso pode alterar a funcionalidade do serviço.

AWS IoT Greengrass usa uma função de serviço para acessar alguns de seus AWS recursos em seu nome. Para obter mais informações, consulte the section called “Função de serviço do Greengrass”.

Escolhendo uma função do IAM no AWS IoT Greengrass console

No AWS IoT Greengrass console, talvez você precise escolher uma função de serviço do Greengrass ou uma função de grupo do Greengrass em uma lista de funções do IAM em sua conta.

- A função de serviço do Greengrass permite AWS IoT Greengrass acessar seus AWS recursos em outros serviços em seu nome. Normalmente, você não precisa escolher o perfil de serviço porque o console pode criá-la e configurá-la para você. Para obter mais informações, consulte the section called “Função de serviço do Greengrass”.
- A função de grupo do Greengrass é usada para permitir que as funções e conectores do Greengrass Lambda no grupo acessem seus recursos. AWS Ele também pode dar AWS IoT Greengrass permissões para exportar fluxos para AWS serviços e gravar CloudWatch registros. Para obter mais informações, consulte the section called “Função do grupo do Greengrass.”.

Função de serviço do Greengrass

A função de serviço do Greengrass é uma função de serviço AWS Identity and Access Management (IAM) que AWS IoT Greengrass autoriza o acesso a recursos de AWS serviços em seu nome. Isso possibilita AWS IoT Greengrass a execução de tarefas essenciais, como recuperar suas AWS Lambda funções e gerenciar AWS IoT sombras.

Para permitir AWS IoT Greengrass o acesso aos seus recursos, a função de serviço do Greengrass deve estar associada à sua Conta da AWS e ser especificada 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. Essa política é mantida AWS e define o conjunto de permissões que você AWS IoT Greengrass usa para acessar seus AWS recursos.

Você pode reutilizar a mesma função de serviço do Greengrass Região da AWS entre s, mas deve associá-la à sua conta em Região da AWS todos os lugares em que usa. AWS IoT Greengrass A implantação do grupo falhará se a função de serviço não existir na região atual Conta da AWS e na região.

As seções a seguir descrevem como criar e gerenciar a função de serviço do Greengrass no Console de gerenciamento da AWS ou. AWS CLI

- Gerenciar a função de serviço (console)
- Gerenciar a função de serviço (CLI)

> **Note**
>
> Além da função de serviço que autoriza o acesso em nível de serviço, você pode atribuir uma função de grupo a um grupo. AWS IoT Greengrass A função de grupo é uma função separada do IAM que controla como as funções e conectores do Greengrass Lambda no grupo podem acessar os serviços. AWS

Gerenciar a função de serviço do Greengrass (console)

O AWS IoT console facilita o gerenciamento de sua função de serviço no Greengrass. Por exemplo, quando você cria ou implanta um grupo do Greengrass, o console verifica se você Conta da AWS está vinculado a uma função de serviço do Greengrass na Região da AWS que está atualmente selecionada no console. Caso contrário, o console pode criar e configurar uma função de serviço para você. Para obter mais informações, consulte the section called “Criar a função de serviço do Greengrass”.

Você pode usar o AWS IoT console para as seguintes tarefas de gerenciamento de funções:

- Encontrar a função de serviço do Greengrass
- Criar a função de serviço do Greengrass
- Alterar a função de serviço do Greengrass
- Desanexar a função de serviço do Greengrass

> **Note**
>
> O usuário que está conectado no console deve ter permissões para visualizar, criar ou alterar o perfil de serviço.

Encontrar a função de serviço do Greengrass (console)

Use as etapas a seguir para encontrar a função de serviço que AWS IoT Greengrass está sendo usada no momento Região da AWS.

1. No painel de navegação do console do AWS IoT, selecione Configurações.

2. Role até a seção Função de serviço do Greengrass para ver a função de serviço e as políticas dela.

   Se não for exibida uma função de serviço, deixe que o console crie ou configure uma para você. Para obter mais informações, consulte Criar a função de serviço do Greengrass.

Criar a função de serviço do Greengrass (console)

O console pode criar e configurar uma função 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 a configuração do dispositivo do Greengrass criar o perfil de serviço, o nome da função será `GreengrassServiceRole_`*`random-string`*.

Quando você cria ou implanta um grupo do Greengrass a partir do AWS IoT console, o console verifica se uma função de serviço do Greengrass está associada à sua Conta da AWS no Região da AWS que está atualmente selecionado no console. Caso contrário, o console solicitará que você permita AWS IoT Greengrass a leitura e gravação nos AWS serviços 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 a função de serviço à sua Conta da AWS na atual Região da AWS.

- Se a função não existir, o console cria uma função de serviço padrão do Greengrass e a anexa à sua Conta da AWS na atual. Região da AWS

> **Note**
>
> Se quiser criar uma função de serviço com políticas de função personalizadas, use o console do IAM para criar ou modificar a função. Para obter mais informações, consulte Criação de uma função para delegar permissões a um AWS serviço ou Modificar uma função no Guia 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 Cross-service prevenção delegada confusa.
> Se você criar uma função de serviço, retorne ao AWS IoT console e anexe a função ao grupo. É possível fazer isso na função de serviço do Greengrass, na página Configurações do grupo.

Alterar a função de serviço do Greengrass (console)

Use o procedimento a seguir para escolher uma função de serviço diferente do Greengrass para anexar à sua Conta da AWS na Região da AWS atualmente selecionada no console.

1. No painel de navegação do console do AWS IoT, selecione Configurações.
2. Em Função de serviço do Greengrass, selecione Change role (Mudar perfil).

   A caixa de diálogo Atualizar função de serviço do Greengrass é aberta e mostra as funções do IAM em sua Conta da AWS que você define AWS IoT Greengrass como uma entidade confiável.
3. Selecione a função de serviço do Greengrass a ser anexado.
4. Selecione Anexar função.

> **Note**
>
> Para permitir que o console crie uma função de serviço padrão do Greengrass para você, selecione Create role for me (Criar função para mim) em vez de escolher uma função na lista. O link Criar função para mim não aparecerá se uma função chamada `Greengrass_ServiceRole` estiver na Conta da AWS.

Desanexar a função de serviço do Greengrass (console)

Use o procedimento a seguir para separar a função de serviço do Greengrass da Conta da AWS sua na Região da AWS atualmente selecionada no console. Isso revoga as permissões AWS IoT Greengrass para acessar AWS serviços no atual Região da AWS.

> **Important**
>
> Desanexar o perfil de serviço pode interromper operações ativas.

1. No painel de navegação do console do AWS IoT, selecione Configurações.
2. Em Função de serviço do Greengrass, selecione Detach role (Desanexar função).
3. 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 AWS IoT Greengrass o acesso aos seus 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 do Greengrass (CLI)

Nos procedimentos a seguir, presumimos que o AWS CLI esteja instalado e configurado para usar sua Conta da AWS ID. Para obter mais informações, consulte Instalação da interface de linha de AWS comando e Configuração da AWS CLI no Guia do AWS Command Line Interface Usuário.

Você pode usar o AWS CLI para as seguintes tarefas de gerenciamento de funções:

- Obter a função de serviço do Greengrass
- Criar a função de serviço do Greengrass
- Remover a função de serviço do Greengrass

Obter a função de serviço do Greengrass (CLI)

Use o procedimento a seguir para descobrir se uma função de serviço do Greengrass está associada à sua Conta da AWS em um. Região da AWS

- Obtenha a função de serviço. *region*Substitua pelo seu Região da AWS (por exemplo,`us-west-2`).

  ```
  aws Greengrass get-service-role-for-account --region region
  ```

  Se uma função de serviço do Greengrass já estiver associado à sua conta, os metadados de função serão retornados.

  ```
  {
    "AssociatedAt": "timestamp",
    "RoleArn": "arn:aws:iam::account-id:role/path/role-name"
  }
  ```

  Se nenhum metadado de função for retornado, você deverá criar a função de serviço (se ele não existir) e associá-la á sua conta na Região da AWS.

Criar a função de serviço do Greengrass (CLI)

Use as etapas a seguir para criar uma função e associá-la à sua Conta da AWS.

Como criar a função de serviço usando o IAM

1. Crie a função com uma política de confiança que AWS IoT Greengrass permita assumir 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 Cross-service prevenção delegada confusa.

   Linux, macOS, 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": {
           "StringEquals": {
             "aws:SourceAccount": "account-id"
           },
           "ArnLike": {
             "aws:SourceArn": "arn:aws:greengrass:region:account-id:*"
           }
         }
       }
     ]
   }'
   ```

Windows command prompt

```
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 a função de serviço à sua Conta da AWS

- Associe a função à sua conta. *`role-arn`*Substitua pelo ARN da função de serviço e *`region`* pelo seu Região da AWS (por exemplo,`us-west-2`).

```
aws greengrass associate-service-role-to-account --role-arn role-arn --
region region
```

Se houver êxito, a resposta a seguir será retornada.

```
{
  "AssociatedAt": "timestamp"
}
```

Remover a função de serviço do Greengrass (CLI)

Use as etapas a seguir para desassociar a função de serviço do Greengrass de sua Conta da AWS.

- Desassocie a função de serviço da conta. *`region`*Substitua pelo seu Região da AWS (por exemplo,`us-west-2`).

```
aws greengrass disassociate-service-role-from-account --region region
```

Se houver êxito, a resposta a seguir será retornada.

```
{
  "DisassociatedAt": "timestamp"
}
```

> **Note**
>
> Você deve excluir a função de serviço se não a estiver usando em nenhuma Região da AWS. Primeiro, use delete-role-policy para desvincular a política gerenciada `AWSGreengrassResourceAccessRolePolicy` do perfil, depois use delete-role para excluir o perfil. Para obter mais informações sobre como excluir uma função, consulte Excluir funções ou perfis de instância no Manual do usuário do IAM.

Consulte também

- Criação de uma função para delegar permissões a um AWS serviço 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.
- AWS IoT Greengrass comandos na Referência de AWS CLI Comandos
  - associate-service-role-to-account
  - disassociate-service-role-from-account
  - get-service-role-for-account
- Comandos do IAM disponíveis na Referência de comandos do AWS CLI
  - attach-role-policy
  - create-role
  - delete-role
  - delete-role-policy

Função do grupo do Greengrass.

A função de grupo do Greengrass é um perfil do IAM que autoriza o código em execução em um núcleo do Greengrass a acessar seus recursos da AWS . Você cria a função e gerencia as permissões no AWS Identity and Access Management (IAM) e anexa a função ao seu grupo do Greengrass. Um grupo do Greengrass tem uma função de grupo única. Para adicionar ou alterar permissões, você pode anexar uma função diferente ou alterar as políticas do IAM anexadas à função.

A função deve ser definida AWS IoT Greengrass como uma entidade confiável. Dependendo do seu caso de negócio, a função de grupo pode conter políticas do IAM que definem:

- Permissões para funções Lambda definidas pelo usuário acessarem AWS serviços.
- Permissões para conectores acessarem AWS serviços.
- Permissões para que o gerenciador de streams exporte streams para o Kinesis Data Streams AWS IoT Analytics e o Kinesis Data Streams.
- Permissões para permitir o registro em log do CloudWatch .

As seções a seguir descrevem como anexar ou desanexar uma função de grupo do Greengrass no Console de gerenciamento da AWS ou. AWS CLI

- Gerenciar a função de grupo (console)
- Gerenciar a função de grupo (CLI)

> **Note**
>
> Além da função de grupo que autoriza o acesso a partir do núcleo do Greengrass, você pode atribuir uma função de serviço do Greengrass que AWS IoT Greengrass permite AWS acessar recursos em seu nome.

Gerenciar a função de grupo do Greengrass (console)

Você pode usar o AWS IoT console para as seguintes tarefas de gerenciamento de funções:

- Encontrar a função de grupo do Greengrass
- Adicionar ou alterar a função de grupo do Greengrass

- Remover a função de grupo do Greengrass

> **Note**
>
> O usuário que está conectado ao console deve ter permissões para gerenciar a função.

Encontrar a função de grupo do Greengrass (console)

Siga estas etapas para encontrar a função anexada a um grupo do Greengrass.

1. No painel de navegação do AWS IoT console, em Gerenciar, expanda dispositivos Greengrass e escolha Grupos (V1).
2. Selecione o grupo de destino.
3. Na página de configuração do grupo, selecione Visualizar configurações.

Se uma função estiver anexada ao grupo, ela será exibida em Função de grupo.

Adicionar ou alterar a função de grupo do Greengrass (console)

Siga estas etapas para escolher uma função do IAM Conta da AWS para adicionar a um grupo do Greengrass.

Uma função de grupo tem os seguintes requisitos:

- AWS IoT Greengrass definido como uma entidade confiável.
- As políticas de permissão anexadas à função devem conceder aos seus AWS recursos as permissões exigidas pelas funções e conectores do Lambda no grupo e pelos componentes do sistema Greengrass.

> **Note**
>
> 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 Cross-service prevenção delegada confusa.

Use o console do IAM para criar e configurar o perfil e suas permissões. Para obter etapas que criam o exemplo de um perfil que permite acesso a uma tabela do Amazon DynamoDB, consulte the section called “Configurar a função do grupo”. Para ver as etapas gerais, consulte Criação de uma função para um AWS serviço (console) no Guia do usuário do IAM.

Depois que a função for configurada, use o AWS IoT console para adicionar a função ao grupo.

> **Note**
>
> Este procedimento é necessário apenas para escolher uma função para o grupo. Não é necessário depois de alterar as permissões da função de grupo selecionada no momento.

1. No painel de navegação do AWS IoT console, em Gerenciar, expanda dispositivos Greengrass e escolha Grupos (V1).
2. Selecione o grupo de destino.
3. Na página de configuração do grupo, selecione Visualizar configurações.
4. Em Função do grupo, selecione adicionar ou alterar a função:
   - Para adicionar a função, selecione Associar função e, em seguida, selecione sua função na lista de funções. Essas são as suas funções Conta da AWS que se definem AWS IoT Greengrass como uma entidade confiável.
   - Para escolher uma função diferente, selecione Editar função e, em seguida, selecione sua função na lista de funções.
5. Selecione Salvar.

Remover a função de grupo do Greengrass (console)

Siga estas etapas para separar a função de um grupo do Greengrass.

1. No painel de navegação do AWS IoT console, em Gerenciar, expanda dispositivos Greengrass e escolha Grupos (V1).
2. Selecione o grupo de destino.
3. Na página de configuração do grupo, selecione Visualizar configurações.
4. Em Função do grupo, selecione Desassociar função.
5. Na caixa de diálogo de confirmação, selecione Desassociar função. Esta etapa remove a função do grupo, mas não exclui a função. Se você quiser excluir a função, use o console do IAM.

Gerenciar a função de grupo do Greengrass (CLI)

Você pode usar o AWS CLI para as seguintes tarefas de gerenciamento de funções:

- Obter a função de grupo do Greengrass
- Criar a função de grupo do Greengrass
- Remover a função de grupo do Greengrass

Obter a função de grupo do Greengrass (CLI)

Siga estas etapas para descobrir se um grupo do Greengrass tem uma função associada.

1. Obtenha o ID do grupo-alvo na lista de seus grupos.

   ```
   aws greengrass list-groups
   ```

   Esta é uma resposta de exemplo do `list-groups`. Cada grupo na resposta inclui uma propriedade `Id` que contém o ID do grupo.

   ```
   {
       "Groups": [
           {
               "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/
   groups/00dedaaa-ac16-484d-ad77-c3eedEXAMPLE/versions/4cbc3f07-fc5e-48c4-
   a50e-7d356EXAMPLE",
               "Name": "MyFirstGroup",
               "LastUpdatedTimestamp": "2019-11-11T05:47:31.435Z",
               "LatestVersion": "4cbc3f07-fc5e-48c4-a50e-7d356EXAMPLE",
   ```

```
            "CreationTimestamp": "2019-11-11T05:47:31.435Z",
            "Id": "00dedaaa-ac16-484d-ad77-c3eedEXAMPLE",
            "Arn": "arn:aws:us-west-2:123456789012:/greengrass/groups/00dedaaa-
ac16-484d-ad77-c3eedEXAMPLE"
        },
        {
            "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE/versions/8fe9e8ec-64d1-4647-
b0b0-01dc8EXAMPLE",
            "Name": "GreenhouseSensors",
            "LastUpdatedTimestamp": "2020-01-07T19:58:36.774Z",
            "LatestVersion": "8fe9e8ec-64d1-4647-b0b0-01dc8EXAMPLE",
            "CreationTimestamp": "2020-01-07T19:58:36.774Z",
            "Id": "036ceaf9-9319-4716-ba2a-237f9EXAMPLE",
            "Arn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE"
        },
        ...
    ]
}
```

Para obter mais informações, incluindo exemplos que usam a opção `query` para filtrar resultados, consulte the section called “Obter o ID do grupo”.

2. Copie da saída o `Id` do grupo de destino.
3. Obtenha a função de grupo. *`group-id`*Substitua pelo ID do grupo-alvo.

```
aws greengrass get-associated-role --group-id group-id
```

Se uma função estiver associada ao seu grupo do Greengrass, os seguintes metadados de função serão retornados.

```
{
  "AssociatedAt": "timestamp",
  "RoleArn": "arn:aws:iam::account-id:role/path/role-name"
}
```

Se o seu grupo não tiver uma função associada, o seguinte erro será retornado.

```
An error occurred (404) when calling the GetAssociatedRole operation: You need to
 attach an IAM role to this deployment group.
```

Criar a função de grupo do Greengrass (CLI)

Siga estas etapas para criar uma função e associá-la a um grupo do Greengrass.

Para criar a função de grupo usando o IAM

1. Crie a função com uma política de confiança que AWS IoT Greengrass permita assumir a função. Este exemplo cria uma função chamada `MyGreengrassGroupRole`, 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 Cross-service prevenção delegada confusa.

   Linux, macOS, or Unix

```
aws iam create-role --role-name MyGreengrassGroupRole --assume-role-policy-
document '{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "greengrass.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "account-id"
        },
        "ArnLike": {
          "aws:SourceArn": "arn:aws:greengrass:region:account-id:/greengrass/
groups/group-id"
        }
      }
    }
  ]
}'
```

Windows command prompt

```
aws iam create-role --role-name MyGreengrassGroupRole --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:/greengrass/groups/group-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 ao seu grupo.
3. Anexe políticas gerenciadas ou em linha à função para dar suporte ao seu caso de negócio. Por exemplo, se uma função do Lambda definida pelo usuário ler do Amazon S3, você pode anexar a política gerenciada `AmazonS3ReadOnlyAccess` à função.

```
aws iam attach-role-policy --role-name MyGreengrassGroupRole --policy-arn
 arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess
```

Se for bem-sucedido, nenhuma resposta será retornada.

Como associar a função ao seu grupo do Greengrass

1. Obtenha o ID do grupo-alvo na lista de seus grupos.

```
aws greengrass list-groups
```

Esta é uma resposta de exemplo do `list-groups`. Cada grupo na resposta inclui uma propriedade `Id` que contém o ID do grupo.

```
{
    "Groups": [
        {
            "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/00dedaaa-ac16-484d-ad77-c3eedEXAMPLE/versions/4cbc3f07-fc5e-48c4-
a50e-7d356EXAMPLE",
            "Name": "MyFirstGroup",
            "LastUpdatedTimestamp": "2019-11-11T05:47:31.435Z",
```

```
            "LatestVersion": "4cbc3f07-fc5e-48c4-a50e-7d356EXAMPLE",
            "CreationTimestamp": "2019-11-11T05:47:31.435Z",
            "Id": "00dedaaa-ac16-484d-ad77-c3eedEXAMPLE",
            "Arn": "arn:aws:us-west-2:123456789012:/greengrass/groups/00dedaaa-
ac16-484d-ad77-c3eedEXAMPLE"
        },
        {
            "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE/versions/8fe9e8ec-64d1-4647-
b0b0-01dc8EXAMPLE",
            "Name": "GreenhouseSensors",
            "LastUpdatedTimestamp": "2020-01-07T19:58:36.774Z",
            "LatestVersion": "8fe9e8ec-64d1-4647-b0b0-01dc8EXAMPLE",
            "CreationTimestamp": "2020-01-07T19:58:36.774Z",
            "Id": "036ceaf9-9319-4716-ba2a-237f9EXAMPLE",
            "Arn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE"
        },
        ...
    ]
}
```

Para obter mais informações, incluindo exemplos que usam a opção `query` para filtrar resultados, consulte the section called “Obter o ID do grupo”.

2. Copie da saída o `Id` do grupo de destino.
3. Associe a função ao grupo. *`group-id`*Substitua pelo ID do grupo-alvo e *`role-arn`* pelo ARN da função do grupo.

```
aws greengrass associate-role-to-group --group-id group-id --role-arn role-arn
```

Se houver êxito, a resposta a seguir será retornada.

```
{
  "AssociatedAt": "timestamp"
}
```

Remover a função de grupo do Greengrass (CLI)

Siga estas etapas para desassociar a função de grupo do seu grupo do Greengrass.

1. Obtenha o ID do grupo-alvo na lista de seus grupos.

```
aws greengrass list-groups
```

Esta é uma resposta de exemplo do `list-groups`. Cada grupo na resposta inclui uma propriedade `Id` que contém o ID do grupo.

```
{
    "Groups": [
        {
            "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/00dedaaa-ac16-484d-ad77-c3eedEXAMPLE/versions/4cbc3f07-fc5e-48c4-
a50e-7d356EXAMPLE",
            "Name": "MyFirstGroup",
            "LastUpdatedTimestamp": "2019-11-11T05:47:31.435Z",
            "LatestVersion": "4cbc3f07-fc5e-48c4-a50e-7d356EXAMPLE",
            "CreationTimestamp": "2019-11-11T05:47:31.435Z",
            "Id": "00dedaaa-ac16-484d-ad77-c3eedEXAMPLE",
            "Arn": "arn:aws:us-west-2:123456789012:/greengrass/groups/00dedaaa-
ac16-484d-ad77-c3eedEXAMPLE"
        },
        {
            "LatestVersionArn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE/versions/8fe9e8ec-64d1-4647-
b0b0-01dc8EXAMPLE",
            "Name": "GreenhouseSensors",
            "LastUpdatedTimestamp": "2020-01-07T19:58:36.774Z",
            "LatestVersion": "8fe9e8ec-64d1-4647-b0b0-01dc8EXAMPLE",
            "CreationTimestamp": "2020-01-07T19:58:36.774Z",
            "Id": "036ceaf9-9319-4716-ba2a-237f9EXAMPLE",
            "Arn": "arn:aws:us-west-2:123456789012:/greengrass/
groups/036ceaf9-9319-4716-ba2a-237f9EXAMPLE"
        },
        ...
    ]
}
```

Para obter mais informações, incluindo exemplos que usam a opção `query` para filtrar resultados, consulte the section called “Obter o ID do grupo”.

2. Copie da saída o `Id` do grupo de destino.
3. Desassocie a função do seu grupo. *`group-id`*Substitua pelo ID do grupo-alvo.

```
aws greengrass disassociate-role-from-group --group-id group-id
```

Se houver êxito, a resposta a seguir será retornada.

```
{
  "DisassociatedAt": "timestamp"
}
```

> **Note**
>
> Você pode excluir a função de grupo se não estiver usando. Primeiro, use delete-role-policy para desanexar cada política gerenciada 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

- Tópicos relacionados no Guia do usuário do IAM
  - Criação de uma função para delegar permissões a um serviço AWS
  - Modificar uma função
  - Adicionar e remover permissões de identidade IAM
  - Excluir funções ou perfis de instância
- AWS IoT Greengrass comandos na Referência de AWS CLI Comandos
  - list-groups
  - associate-role-to-group
  - disassociate-role-from-group
  - get-associated-role
- Comandos do IAM disponíveis na Referência de comandos do AWS CLI

- attach-role-policy
- create-role
- delete-role
- delete-role-policy

Cross-service prevenção delegada confusa

O problema "confused deputy" é um problema de segurança em que uma entidade que não tem permissão para executar uma ação pode coagir uma entidade mais privilegiada a executar a ação. Em AWS, a falsificação de identidade entre serviços pode resultar no problema confuso do deputado. Cross-service a representação 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 usar as chaves de contexto de condição `aws:SourceAccount`global `aws:SourceArn`e as chaves de contexto nas políticas de recursos para limitar as permissões que AWS IoT Greengrass concedem outro serviço ao 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 do Greengrass associado à solicitação `sts:AssumeRole`.

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 especificar 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:`*`region`*`:`*`account-id`*`:*`

Para obter exemplos de políticas que usam as chaves de contexto de condição global `aws:SourceArn` e `aws:SourceAccount`, consulte os seguintes tópicos:

- Criar a função de serviço do Greengrass
- Criar a função de grupo do Greengrass

- Criar e configurar uma função de execução do IAM para implantação em lote

Identity-based exemplos de políticas para AWS IoT Greengrass

Por padrão, os usuários e os perfis do IAM não têm permissão para criar ou modificar recursos do AWS IoT Greengrass . Eles também não podem realizar tarefas usando a AWS API Console de gerenciamento da AWS AWS CLI, ou. 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 política

Identity-based as políticas determinam se alguém pode criar, acessar ou excluir AWS IoT Greengrass recursos em sua conta. Essas ações podem incorrer em custos para sua Conta da AWS. Ao criar ou editar políticas baseadas em identidade, siga estas diretrizes e recomendações:

- Comece com as políticas AWS gerenciadas e avance para as permissões de privilégios mínimos — Para começar a conceder permissões aos seus usuários e cargas de trabalho, use as políticas AWS gerenciadas que concedem permissões para muitos casos de uso comuns. Eles estão disponíveis no seu Conta da AWS. Recomendamos que você reduza ainda mais as permissões definindo políticas gerenciadas pelo AWS cliente que sejam específicas para seus casos de uso. Para saber mais, 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 saber mais 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: é possível adicionar uma condição às políticas para limitar o acesso a ações e recursos. Por exemplo, é possível escrever uma condição de política para especificar que todas as solicitações devem ser enviadas usando SSL. Você também pode usar condições para conceder acesso às ações de serviço se elas forem usadas por meio de uma ação específica AWS service (Serviço da AWS), como CloudFormation. Para saber mais, consulte Elementos da política JSON do IAM: condição no Guia 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 cem verificações de política e recomendações práticas para ajudar a criar políticas seguras e funcionais. Para saber mais, consulte Validação de políticas do IAM Access Analyzer no Guia do Usuário do IAM.
- Exigir autenticação multifator (MFA) — Se você tiver um cenário que exija usuários do IAM ou um usuário root, ative Conta da AWS a MFA para obter segurança adicional. Para exigir MFA quando as operações de API forem chamadas, adicione condições de MFA às suas políticas. Para saber mais, consulte Configuração de acesso à API protegido por MFA no Guia do Usuário do IAM.

Para saber mais sobre as práticas recomendadas do IAM, consulte Práticas recomendadas de segurança no IAM no Guia do usuário do IAM.

AWS políticas gerenciadas para AWS IoT Greengrass

AWS IoT Greengrass mantém as seguintes políticas AWS gerenciadas que você pode usar para conceder permissões aos usuários e funções do IAM.

Política	Description
AWSGreengrassFullAccess	Permite todas AWS IoT Greengrass as ações para todos os seus AWS recursos. Essa política é recomendada para administradores AWS IoT Greengrass de serviços ou para fins de teste.
AWSGreengrassReadOnlyAccess	`List`Permissões e `Get` AWS IoT Greengrass ações para todos os seus AWS recursos.
AWSGreengrassResourceAccessRolePolicy	Permite acesso a recursos de AWS serviços, incluindo AWS Lambda AWS IoT Device Shadow. Esta é a política padrão usada para a função de serviço do Greengrass. Esta política foi concebida para proporcionar facilidade geral de acesso. Você pode definir uma política personalizada que seja mais restritiva.

Política	Description
GreengrassOTAUpdateArtifactAccess	Permite acesso somente de leitura a artefatos de atualização remota (OTA) para o software AWS IoT Greengrass Core em todos os sistemas. Região da AWS

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 utilizando 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 criar uma política que permita 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 programaticamente usando a API AWS CLI ou 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": "*"
        }
    ]
}
```

Solução de problemas de identidade e acesso para AWS IoT Greengrass

Use as informações a seguir para ajudá-lo a diagnosticar e corrigir problemas comuns que você pode encontrar ao trabalhar com AWS IoT Greengrass um IAM.

Problemas

- Não estou autorizado a realizar uma ação em AWS IoT Greengrass
- Erro: O Greengrass não está autorizado a assumir o perfil de serviço associado a essa conta ou o erro: Falha: perfil de serviço TES não está associado a essa conta.
- Erro: permissão negada ao tentar usar a função arn:aws:iam::<account-id>:role/<role-name> para acessar o URL s3. https://<region>-greengrass-updates.s3.<region>.amazonaws.com/core/<architecture>/greengrass-core-<distribution-version>.tar.gz
- O shadow do dispositivo não sincroniza com a nuvem.
- Não estou autorizado a realizar iam: PassRole
- Sou administrador e quero permitir que outras pessoas acessem AWS IoT Greengrass
- Quero permitir que pessoas fora da minha Conta da AWS para acessar meu AWS IoT Greengrass recursos

Para obter ajuda geral com a solução de problemas, consulte Solução de problemas.

Não estou autorizado a realizar uma ação em 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 erro de exemplo a seguir ocorre quando o usuário do IAM `mateojackson` do tenta visualizar detalhes sobre uma versão de definição de núcleo, mas não tem as permissões `greengrass:GetCoreDefinitionVersion`.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 greengrass:GetCoreDefinitionVersion on resource: resource: arn:aws:greengrass:us-
west-2:123456789012:/greengrass/definition/cores/78cd17f3-bc68-ee18-47bd-5bda5EXAMPLE/
versions/368e9ffa-4939-6c75-859c-0bd4cEXAMPLE
```

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:/greengrass/definition/cores/78cd17f3-bc68-ee18-47bd-5bda5EXAMPLE/versions/368e9ffa-4939-6c75-859c-0bd4cEXAMPLE` usando a ação `greengrass:GetCoreDefinitionVersion`.

Erro: O Greengrass não está autorizado a assumir o perfil de serviço associado a essa conta ou o erro: Falha: perfil de serviço TES não está associado a essa conta.

Solução: esse erro pode ser visto quando a implantação falhar. Verifique se uma função de serviço do Greengrass está associada à sua Conta da AWS na Região da AWS atual. Para acessar mais informações, consulte the section called “Gerenciar a função de serviço (CLI)” ou the section called “Gerenciar a função de serviço (console)”.

Erro: permissão negada ao tentar usar a função arn:aws:iam::<account-id>:role/<role-name> para acessar o URL s3. https://<region>-greengrass-updates.s3.<region>.amazonaws.com/core/<architecture>/greengrass-core-<distribution-version>.tar.gz

Solução: esse erro pode ser visto quando uma atualização OTA (over-the-air) falhar. Na política de função assinante, adicione a Região da AWS de destino como `Resource`. Essa função de signatário é usada para predefinir a URL do S3 para a atualização do AWS IoT Greengrass software. Para obter mais informações, consulte Função assinante do URL do S3.

O shadow do dispositivo não sincroniza com a nuvem.

Solução: Certifique-se de que AWS IoT Greengrass tenha permissões `iot:UpdateThingShadow` e `iot:GetThingShadow` ações na função de serviço do Greengrass. Se a função de serviço usa a política gerenciada `AWSGreengrassResourceAccessRolePolicy`, essas permissões são incluídas por padrão.

Consulte Solução de problemas de intervalo de sincronização de shadow.

Veja a seguir os problemas gerais do IAM que você pode encontrar ao trabalhar com AWS IoT Greengrass.

Não estou autorizado a realizar iam: PassRole

Se você receber uma mensagem de erro informando que não está autorizado a executar a ação `iam:PassRole`, as suas políticas devem ser atualizadas para permitir que você passe uma função para o AWS IoT Greengrass.

Alguns Serviços da AWS permitem que você passe uma função existente para esse serviço em vez de criar uma nova função de serviço ou uma função vinculada ao serviço. Para fazê-lo, você deve ter permissões para passar o perfil para o serviço.

O exemplo de erro a seguir ocorre quando uma usuária do IAM chamada `marymajor` tenta utilizar 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 o perfil 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 precisar de ajuda, entre em contato com seu AWS administrador. Seu administrador é a pessoa que forneceu suas credenciais de login.

Sou administrador e quero permitir que outras pessoas acessem AWS IoT Greengrass

Para permitir que outras pessoas acessem AWS IoT Greengrass, você deve conceder permissão às pessoas ou aplicativos que precisam de acesso. Se você estiver usando o Centro de Identidade do AWS IAM para gerenciar pessoas e aplicações, atribua conjuntos de permissões a

usuários ou grupos para definir o nível de acesso. Os conjuntos de permissões criam e atribuem automaticamente políticas do IAM aos perfis do IAM associados à pessoa ou aplicação. Para ter mais informações, consulte Conjuntos de permissões no Guia do usuário do Centro de Identidade do AWS IAM .

Se você não estiver usando o Centro de Identidade do IAM, deverá criar entidades do IAM (usuários ou perfis) para as pessoas ou aplicações que precisam de acesso. Você deve anexar uma política à entidade que concede a eles as permissões corretas no AWS IoT Greengrass. Depois que as permissões forem concedidas, forneça as credenciais ao usuário ou desenvolvedor da aplicação. Eles usarão essas credenciais para acessar AWS. Para saber mais sobre como criar grupos, políticas, permissões e usuários do IAM, consulte Identidades do IAM e Políticas e permissões no IAM no Guia do usuário do IAM.

Quero permitir que pessoas fora da minha Conta da AWS para acessar meu AWS IoT Greengrass recursos

Você pode criar uma função do IAM que usuários de outras contas ou pessoas fora da sua organização possam usar para acessar seus AWS recursos. Você pode especificar quem é confiável para assumir a função. Para obter mais informações, consulte Fornecer acesso a um usuário do IAM em outro Conta da AWS de sua propriedade e Fornecer acesso a contas da Amazon Web Services pertencentes a terceiros no Guia do usuário do IAM.

AWS IoT Greengrass não oferece suporte ao acesso entre contas com base em políticas baseadas em recursos ou listas de controle de acesso (ACLs).

Validação de conformidade AWS IoT Greengrass

Nossa nova experiência AWS de inscrição não foi projetada para cargas de trabalho regulamentadas. Se você estiver usando nossa nova experiência de AWS inscrição, mas quiser usá-la AWS para cargas de trabalho regulamentadas, você pode se inscrever AWS (avançado) ou ativar recursos avançados para seu AWS ambiente.

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 e escolha o programa de conformidade no qual você está interessado. Para obter informações gerais, consulte Programas de AWS conformidade Programas .

Você pode baixar relatórios de auditoria de terceiros usando AWS Artifact. Para obter mais informações, consulte Baixando relatórios em AWS Artifact .

Sua responsabilidade de conformidade durante o uso Serviços da AWS é determinada pela confidencialidade de seus dados, pelos objetivos de conformidade da sua empresa e pelas leis e regulamentações aplicáveis. Para obter mais informações sobre sua responsabilidade de conformidade ao usar Serviços da AWS, consulte a Documentação AWS de segurança.

Resiliência em AWS IoT Greengrass

A infraestrutura AWS global é construída em torno das regiões e zonas de disponibilidade da Amazon Web Services. Cada uma Região da AWS fornece várias zonas de disponibilidade fisicamente separadas e isoladas, conectadas a redes de baixa latência, alta taxa de transferência e alta redundância. 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 altamente disponíveis, tolerantes a falhas e escaláveis que uma ou várias infraestruturas de data center tradicionais.

Para obter mais informações sobre Regiões e Zonas de disponibilidade da Amazon Web Services, consulte Infraestrutura global da AWS.

Além da infraestrutura AWS global, AWS IoT Greengrass oferece vários recursos para ajudar a suportar suas necessidades de resiliência e backup de dados.

- Se o núcleo perder a conectividade com a Internet, os dispositivos cliente podem continuar a se comunicar pela rede local.
- Você pode configurar o núcleo para armazenar mensagens não processadas destinadas aos Nuvem AWS destinos em um cache de armazenamento local em vez de armazenamento na memória. O cache de armazenamento local pode persistir nas reinicializações principais (por exemplo, após uma implantação em grupo ou a reinicialização de um dispositivo), portanto, AWS IoT Greengrass pode continuar processando as mensagens destinadas a. AWS IoT Core Para obter mais informações, consulte the section called “Fila de mensagens MQTT”.
- Você pode configurar o núcleo para estabelecer uma sessão persistente com o agente de AWS IoT Core mensagens. Isso permite que o núcleo receba mensagens enviadas enquanto o núcleo está off-line. Para obter mais informações, consulte the section called “Sessões persistentes do MQTT com AWS IoT Core”.
- Você pode configurar um grupo do Greengrass para gravar registros no sistema de arquivos local e nos registros. CloudWatch Se o núcleo perder conectividade, o registro local pode continuar, mas os logs do CloudWatch são enviados com um número limitado de tentativas. Depois que

as novas tentativas são esgotadas, o evento é descartado. Você também deve estar ciente das limitações de registro.

- Você pode criar funções do Lambda que leiam fluxos do gerenciador de fluxo e enviam os dados para destinos de armazenamento local.

Segurança da infraestrutura em AWS IoT Greengrass

Como serviço gerenciado, AWS IoT Greengrass é protegido pela segurança de rede AWS global. Para obter informações sobre serviços AWS de segurança e como AWS proteger a infraestrutura, consulte AWS Cloud Security. Para projetar seu AWS ambiente usando as melhores práticas de segurança de infraestrutura, consulte Proteção de infraestrutura no Security Pillar AWS Well-Architected Framework.

Você usa chamadas de API AWS publicadas para acessar AWS IoT Greengrass pela rede. Os clientes devem oferecer compatibilidade com:

- Transport Layer Security (TLS). Exigimos TLS 1.2 e recomendamos TLS 1.3.
- Suítes de criptografia com sigilo direto perfeito (PFS), como DHE (Ephemeral) ou ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Diffie-Hellman A maioria dos sistemas modernos, como Java 7 e versões posteriores, comporta esses modos.

Em um AWS IoT Greengrass ambiente, os dispositivos usam X.509 certificados e chaves criptográficas para se conectar e autenticar no. Nuvem AWS Para obter mais informações, consulte the section called “Autorização e autenticação do dispositivo”.

Análise de configuração e vulnerabilidade em 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.

AWS IoT Device Defender aborda esses desafios fornecendo ferramentas para identificar problemas de segurança e desvios das melhores práticas. Você pode usar AWS IoT Device Defender para

analisar, auditar e monitorar dispositivos conectados para detectar comportamentos anormais e mitigar riscos de segurança. AWS IoT Device Defender pode auditar dispositivos para garantir que eles sigam as melhores práticas de segurança e detectem comportamentos anormais nos dispositivos. Isso possibilita a aplicação de políticas de segurança consistentes em todos os dispositivos e responder rapidamente quando eles estiverem comprometidos. Em conexões com AWS IoT Core, AWS IoT Greengrass gera IDs de cliente previsíveis que você pode usar com AWS IoT Device Defender recursos. Para saber mais, consulte AWS IoT Device Defender no Guia do desenvolvedor do AWS IoT Core .

Em AWS IoT Greengrass ambientes, 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 Greengrass não impõe isolamento de rede para funções Lambda definidas pelo usuário, sejam elas executadas ou não em um contêiner do Greengrass. Portanto, é possível que as funções do Lambda se comuniquem com qualquer outro processo em execução no sistema ou fora da rede.

Se você perder o controle de um dispositivo de núcleo do Greengrass e quiser impedir que dispositivos cliente transmitam dados para o núcleo, faça o seguinte:

1. Remova o núcleo do Greengrass do grupo do Greengrass.
2. Rode o certificado CA do grupo No AWS IoT console, você pode girar o certificado CA na página de configurações do grupo. Na AWS IoT Greengrass API, você pode usar a CreateGroupCertificateAuthorityação.

   Também recomendamos o uso de criptografia total do disco se o disco rígido do dispositivo principal estiver vulnerável a roubo.

AWS IoT Greengrass e endpoints VPC de interface (AWS PrivateLink)

Você pode estabelecer uma conexão privada entre sua VPC e o plano de AWS IoT Greengrass controle criando uma interface VPC endpoint. Você pode usar esse endpoint para gerenciar grupos, funções Lambda, implantações e outros recursos no serviço. AWS IoT Greengrass Os endpoints de interface são alimentados por AWS PrivateLinkuma tecnologia que permite acessar AWS

IoT Greengrass APIs de forma privada sem um gateway de internet, dispositivo NAT, conexão VPN ou conexão Direct AWS Connect. As instâncias na sua VPC não precisam de endereços IP públicos para se comunicar com AWS IoT Greengrass as APIs. O tráfego entre sua VPC e AWS IoT Greengrass o tráfego não sai da rede Amazon.

> **Note**
>
> Atualmente, você não pode configurar os dispositivos de núcleo do Greengrass para operar completamente em sua VPC.

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 para AWS IoT Greengrass Endpoints da VPC
- Crie uma interface VPC endpoint para AWS IoT Greengrass operações do plano de controle
- Criação de uma política de VPC endpoint para AWS IoT Greengrass

Considerações para AWS IoT Greengrass Endpoints da VPC

Antes de configurar uma interface para o VPC endpoint AWS IoT Greengrass, revise as propriedades e limitações do endpoint da interface no Guia do usuário do Amazon VPC. Além disso, esteja ciente das seguintes considerações:

- AWS IoT Greengrass suporta fazer chamadas para todas as ações da API do plano de controle a partir de sua VPC. O plano de controle inclui operações como CreateDeploymentStartBulkDeployment e. O plano de controle não inclui operações como GetDeployment e Discover, que são operações do plano de dados.
- Atualmente, não AWS IoT Greengrass há suporte para VPC endpoints nas regiões da China AWS .

Crie uma interface VPC endpoint para AWS IoT Greengrass operações do plano de controle

Você pode criar um VPC endpoint para o plano de AWS IoT Greengrass controle usando o console Amazon VPC ou o (). AWS Command Line Interface AWS CLI Para obter mais informações, consulte Criar um endpoint de interface no Guia do usuário da Amazon VPC.

Crie um VPC endpoint para AWS IoT Greengrass usar o seguinte nome de serviço:

- com.amazonaws. *region*.erva verde

Se você habilitar o DNS privado para o endpoint, poderá fazer solicitações de API 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 mais informações, consulte Acessar um serviço por um endpoint de interface no Guia do usuário da Amazon VPC.

Criação de uma política de VPC endpoint para 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 realizar 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 VPC endpoint para AWS IoT Greengrass actions

Veja a seguir um exemplo de uma política de endpoint para AWS IoT Greengrass. Quando anexada a um endpoint, essa política concede acesso às AWS IoT Greengrass ações listadas para todos os diretores em todos os recursos.

```
{
```

```
    "Statement": [
        {
            "Principal": "*",
            "Effect": "Allow",
            "Action": [
                "greengrass:CreateDeployment",
                "greengrass:StartBulkDeployment"
            ],
            "Resource": "*"
        }
    ]
}
```

Práticas recomendadas de segurança para AWS IoT Greengrass

Este tópico contém as melhores práticas de segurança para AWS IoT Greengrass.

Conceder o mínimo possível de permissões

Siga o princípio de privilégio mínimo usando o conjunto mínimo de permissões em funções do IAM. Limite o uso do caractere curinga `*` para as propriedades `Action` e `Resource` 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 política”.

A melhor prática de privilégios mínimos também se aplica às AWS IoT políticas que você anexa aos seus dispositivos principais e clientes do Greengrass.

Não codificar credenciais em funções do Lambda

Não codifique credenciais em suas funções do Lambda definidas pelo usuário. Como proteger melhor suas credenciais:

- Para interagir com AWS os serviços, defina permissões para ações e recursos específicos na função de grupo do Greengrass.
- Use segredos locais para armazenar suas credenciais. Ou, se a função usar o AWS SDK, use as 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 o acesso aos CloudWatch registros exija permissões do IAM.

- Não use informações confidenciais em caminhos de tópico MQTT.
- Não use informações confidenciais em nomes, tipos e atributos de dispositivos (coisas) no AWS IoT Core registro.
- Não registre informações confidenciais em suas funções do Lambda definidas pelo usuário.
- Não use informações confidenciais nos nomes e IDs dos recursos do Greengrass:
  - Conectores
  - Núcleos
  - Dispositivos
  - Funções
  - Groups (Grupos)
  - Loggers
  - Recursos (local, machine learning ou segredo)
  - Assinaturas

Criar assinaturas direcionadas

As assinaturas controlam o fluxo de informações em um grupo do Greengrass definindo como as mensagens são trocadas entre serviços, dispositivos e funções do Lambda. Para garantir que um aplicativo possa fazer apenas o que deve fazer, suas assinaturas devem permitir que os editores enviem mensagens apenas para tópicos específicos e limitar os assinantes para receber mensagens apenas de tópicos necessários para sua funcionalidade.

Manter o relógio do dispositivo sincronizado

É importante ter uma hora precisa em seu dispositivo. X.509 os certificados têm data e hora de validade. 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 .

Gerenciar a autenticação de dispositivos com o núcleo do Greengrass

Os dispositivos cliente do Greengrass podem executar FreeRTOS ou usar o SDK do dispositivo da AWS IoT ou a API de descoberta do AWS IoT Greengrass para obter informações de descoberta usadas para conexão e autenticação com o núcleo no mesmo grupo do Greengrass. As informações de detecção incluem:

- Informações de conectividade para o núcleo do Greengrass que esteja no mesmo grupo do Greengrass que o dispositivo cliente. Essas informações incluem o endereço do host e o número da porta de cada endpoint do dispositivo de núcleo.
- O certificado CA de grupo usado para assinar o certificado de servidor MQTT local. Os dispositivos cliente usam o certificado CA de grupo para validar o certificado de servidor MQTT apresentado pelo núcleo.

Veja a seguir as melhores práticas de dispositivos cliente para gerenciar a autenticação mútua com um núcleo do Greengrass. Essas práticas podem ajudar a reduzir seus riscos se seu dispositivo principal estiver comprometido.

Valide o certificado de servidor MQTT local para cada conexão.

> Os dispositivos cliente devem validar o certificado do servidor MQTT apresentado pelo núcleo sempre que estabelecerem uma conexão com o núcleo. Esta validação é o lado do dispositivo cliente da autenticação mútua entre um dispositivo de núcleo e dispositivos cliente. Os dispositivos cliente devem ser capazes de detectar uma falha e encerrar a conexão.

Não codifique informações de detecção.

> Os dispositivos cliente devem confiar em operações de detecção para obter informações de conectividade do núcleo e o certificado CA do grupo, mesmo que o núcleo use um endereço IP estático. Os dispositivos cliente não devem codificar essas informações de detecção.

Atualize periodicamente as informações de detecção.

> Os dispositivos cliente devem executar periodicamente a detecção para atualizar as informações de conectividade do núcleo e o certificado CA do grupo. Recomendamos que os dispositivos cliente atualizem essas informações antes de estabelecerem uma conexão com o núcleo. Como as durações mais curtas entre as operações de detecção podem minimizar o tempo de

exposição potencial, recomendamos que os dispositivos cliente se desconectem e se reconectem periodicamente para acionar a atualização.

Se você perder o controle de um dispositivo de núcleo do Greengrass e quiser impedir que dispositivos cliente transmitam dados para o núcleo, faça o seguinte:

1. Remova o núcleo do Greengrass do grupo do Greengrass.
2. Rode o certificado CA do grupo No AWS IoT console, você pode girar o certificado CA na página de configurações do grupo. Na AWS IoT Greengrass API, você pode usar a CreateGroupCertificateAuthorityação.

   Também recomendamos o uso de criptografia total do disco se o disco rígido do dispositivo principal estiver vulnerável a roubo.

Para obter mais informações, consulte the section called “Autorização e autenticação do dispositivo”.

Consulte também

- Melhores práticas de segurança para o AWS IoT Core no Guia do desenvolvedor do AWS IoT
- Dez regras de ouro de segurança para soluções de IoT industrial na Internet das Coisas no blog oficial AWS

Registro e monitoramento em AWS IoT Greengrass

O monitoramento é uma parte importante da manutenção da confiabilidade, disponibilidade e desempenho de AWS IoT Greengrass suas AWS soluções. Você deve coletar dados de monitoramento de todas as partes da sua AWS solução para poder depurar com mais facilidade uma falha multiponto, caso ocorra. Antes de iniciar o monitoramento AWS IoT Greengrass, você deve criar um plano de monitoramento que inclua respostas às 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?

Ferramentas de monitoramento

AWS fornece ferramentas que você pode usar para monitorar 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.

Você pode usar as seguintes ferramentas de monitoramento automatizado para monitorar AWS IoT Greengrass e relatar 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 CloudTrail Monitoramento 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.
- Amazon EventBridge — Use regras de EventBridge eventos para receber notificações sobre mudanças de estado em suas implantações de grupos do Greengrass ou chamadas de API

registradas com. CloudTrail Para obter mais informações, consulte o the section called “Obter notificações de implantação” artigo O que é a Amazon EventBridge? no Guia do EventBridge usuário da Amazon.

- Telemetria da integridade do sistema Greengrass: inscreva-se para receber dados de telemetria enviados a partir do núcleo do Greengrass. Para obter mais informações, consulte the section called “Coletar dados de telemetria da integridade do sistema”.
- Verificação de integridade local — use as APIs de saúde para obter uma visão geral do estado dos AWS IoT Greengrass processos locais no dispositivo principal. Para obter mais informações, consulte the section called “Chamada da API de verificação de integridade local”.

Consulte também

- the section called “Monitoramento com logs do AWS IoT Greengrass ”
- the section called “Registro em log AWS IoT Greengrass Chamadas de API com AWS CloudTrail”
- the section called “Obter notificações de implantação”

Monitoramento com AWS IoT Greengrass logs

AWS IoT Greengrass consiste no serviço de nuvem e no software AWS IoT Greengrass principal. O software AWS IoT Greengrass Core pode gravar registros na Amazon CloudWatch e no sistema de arquivos local do seu dispositivo principal. As funções e conectores Lambda executados no núcleo também podem gravar registros nos CloudWatch registros e no sistema de arquivos local. Você pode usar os logs para monitorar eventos e solucionar problemas. Todas as entradas de AWS IoT Greengrass registro incluem um registro de data e hora, nível de registro e informações sobre o evento. As alterações nas configurações de registro em log entram em vigor após a implantação do grupo.

O registro é configurado no nível do grupo. Para obter as etapas que mostram como configurar o registro em log para um grupo do Greengrass, consulte the section called “Configurar o registro para AWS IoT Greengrass”.

Acessando CloudWatch registros

Se você configurar o CloudWatch registro, poderá visualizar os registros na página Logs do CloudWatch console da Amazon. Os grupos de AWS IoT Greengrass registros para registros usam as seguintes convenções de nomenclatura:

```
/aws/greengrass/GreengrassSystem/greengrass-system-component-name
/aws/greengrass/Lambda/aws-region/account-id/lambda-function-name
```

Cada grupo de log contém fluxos de log que usam a seguinte convenção de nomenclatura:

```
date/account-id/greengrass-group-id/name-of-core-that-generated-log
```

As considerações a seguir se aplicam quando você usa o CloudWatch Logs:

- Os registros são enviados para o CloudWatch Logs com um número limitado de novas tentativas, caso não haja conexão com a Internet. Depois que as novas tentativas são esgotadas, o evento é descartado.
- Transação, memória e outras limitações se aplicam. Para obter mais informações, consulte the section called “Limitações de registro em log”.
- Sua função de grupo do Greengrass deve permitir AWS IoT Greengrass a gravação em registros. CloudWatch Para conceder permissões, incorpore a seguinte política em linha em sua função de grupo.

  JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:*:*:*"
            ]
        }
    ]
}
```

> **Note**
>
> Você pode conceder acesso mais granular a seus recursos de log. Para obter mais informações, consulte Uso de políticas baseadas em identidade (políticas do IAM) para CloudWatch registros no Guia CloudWatch do usuário da Amazon.

A função do grupo é um perfil do IAM que você cria e associa ao grupo do Greengrass. Você pode usar o console ou a AWS IoT Greengrass API para gerenciar a função do grupo.

Como usar o console

1. No painel de navegação do AWS IoT console, em Gerenciar, expanda dispositivos Greengrass e escolha Grupos (V1).
2. Selecione o grupo de destino.
3. Selecione Visualizar configurações . Em Função do grupo, você pode visualizar, associar ou desassociar a função do grupo.

   Para saber as etapas que mostram como associar a função do grupo, consulte função do grupo.

Uso da CLI

- Para localizar a função do grupo, use o comando get-associated-role.
- Para associar a função do grupo, use o comando associate-role-to-group.
- Para remover a função do grupo, use o comando disassociate-role-from-group.

Para saber como obter o ID do grupo para usar com esses comandos, consulte the section called “Obter o ID do grupo”.

Acessar os logs do sistema de arquivos

Se você configurar o registro em log de sistema de arquivos, os arquivos de log são armazenados em *greengrass-root*/ggc/var/log no dispositivo de núcleo. A estrutura de diretório a seguir é de alto nível:

```
greengrass-root/ggc/var/log
    - crash.log
    - system
        - log files for each Greengrass system component
    - user
        - region
            - account-id
                - log files generated by each user-defined Lambda function
            - aws
                - log files generated by each connector
```

> **Note**
>
> Por padrão *greengrass-root* é o diretório /greengrass. Caso um diretório de gravação seja configurado, os logs estão nesse diretório.

As seguintes considerações se aplicam ao usar logs de sistema de arquivos:

- A leitura de AWS IoT Greengrass registros no sistema de arquivos requer permissões de root.
- AWS IoT Greengrass suporta rotação baseada em tamanho e limpeza automática quando a quantidade de dados de registro está próxima do limite configurado.
- O arquivo crash.log está disponível apenas no sistema de arquivos de logs. Esse registro não é gravado em CloudWatch Registros.
- Limitações de uso de disco se aplicam. Para obter mais informações, consulte the section called “Limitações de registro em log”.

> **Note**
>
> Os registros AWS IoT Greengrass do software Core v1.0 são armazenados no *greengrass-root*/var/log diretório.

Configuração de registro em log padrão

Se as configurações de registro não estiverem definidas explicitamente, AWS IoT Greengrass usa a seguinte configuração de registro padrão após a primeira implantação em grupo.

AWS IoT Greengrass Componentes do sistema

- Digite - `FileSystem`
- Componente - `GreengrassSystem`
- Nível - `INFO`
- Espaço - `128 KB`

User-defined Funções Lambda

- Digite - `FileSystem`
- Componente - `Lambda`
- Nível - `INFO`
- Espaço - `128 KB`

> **Note**
>
> Antes da primeira implantação, apenas os componentes do sistema gravam logs ao sistema de arquivos porque não há funções do Lambda definidas pelo usuário que são implantadas.

Configurar o registro para AWS IoT Greengrass

Você pode usar o AWS IoT console ou as AWS IoT Greengrass APIs para configurar o AWS IoT Greengrass registro.

> **Note**
>
> Para permitir AWS IoT Greengrass a gravação de registros em CloudWatch registros, sua função de grupo deve permitir as ações de CloudWatch registros necessárias.

Configurar o registro em log (console)

Você pode configurar o registro de log na página Settings (Configurações) do grupo.

1. No painel de navegação do AWS IoT console, em Gerenciar, expanda dispositivos Greengrass e escolha Grupos (V1).
2. Selecione o grupo no qual você deseja configurar o registro em log.
3. Na página de configuração do grupo, selecione a guia Logs.
4. Selecione o registro local, da seguinte forma:

   - Para configurar o CloudWatch registro, para configuração de CloudWatch registros, escolha Editar.
   - Para configurar o registro do sistema de arquivo, para Local logs configuration (Configuração de registro local), selecione Edit (Editar).

   Você pode configurar o registro de um local ou em ambos os locais.
5. No modal de configuração de edição de logs, selecione o nível de logs do sistema Greengrass ou o nível de logs das funções do usuário do Lambda. Você pode escolher um componente ou ambos os componentes.
6. Selecione o nível mais baixo de eventos que você deseja registrar. Os eventos abaixo desse limite serão excluídos e não serão armazenados.
7. Selecione Salvar. As alterações entrarão em vigor após a implantação do grupo.

Configurar o registro em log (API)

Você pode usar as APIs do AWS IoT Greengrass registrador para configurar o registro programaticamente. Por exemplo, use a ação `CreateLoggerDefinition` para criar uma definição de registrador com base em uma carga útil `LoggerDefinitionVersion`, que usa a seguinte sintaxe:

```
{
  "Loggers": [
    {
      "Id": "string",
      "Type": "FileSystem|AWSCloudWatch",
      "Component": "GreengrassSystem|Lambda",
      "Level": "DEBUG|INFO|WARN|ERROR|FATAL",
      "Space": "integer"
    },
    {
      "Id": "string",
```

```
      ...
    }
  ]
}
```

`LoggerDefinitionVersion` é um conjunto de um ou mais objetos de `Logger` que tem as seguintes propriedades:

`Id`

Um identificador de registrador.

`Type`

O mecanismo de armazenamento para eventos de log. Quando `AWSCloudWatch` usado, os eventos de registro são enviados para o CloudWatch Logs. Quando `FileSystem` é usado, os eventos de log são armazenados no sistema de arquivos local.

Valores válidos: `AWSCloudWatch`, `FileSystem`

`Component`

A origem do evento de log. Quando `GreengrassSystem` é usado, os eventos de componentes do sistema Greengrass são registrados em log. Quando `Lambda` é usado, os eventos das funções do Lambda definidos pelo usuário são registrados em log.

Valores válidos: `GreengrassSystem`, `Lambda`

`Level`

O limiar no nível de log. Os eventos de log que estiverem abaixo desse limite serão excluídos e não serão armazenados.

Valores válidos: `DEBUG`, `INFO` (recomendado), `WARN`, `ERROR`, `FATAL`

`Space`

A quantidade máxima de armazenamento local, em KB, a ser usado para armazenamento de logs. Este campo é aplicável somente quando `Type` é definido como `FileSystem`.

Exemplo de configuração

O exemplo a seguir `LoggerDefinitionVersion` especifica uma configuração de registro de log que:

- Ativa o sistema de arquivos `ERROR` e versões posteriores do registro para componentes AWS IoT Greengrass do sistema.
- Ativa o arquivo do sistema `INFO` e superior fazendo o registro em log para funções do Lambda definidas pelo usuário.
- Ativa CloudWatch `INFO` (e superior) o registro em log para funções Lambda definidas pelo usuário.

```
{
  "Name": "LoggingExample",
  "InitialVersion": {
    "Loggers": [
      {
        "Id": "1",
        "Component": "GreengrassSystem",
        "Level": "ERROR",
        "Space": 10240,
        "Type": "FileSystem"
      },
      {
        "Id": "2",
        "Component": "Lambda",
        "Level": "INFO",
        "Space": 10240,
        "Type": "FileSystem"
      },
      {
        "Id": "3",
        "Component": "Lambda",
        "Level": "INFO",
        "Type": "AWSCloudWatch"
      }
    ]
  }
}
```

Depois de criar uma versão de definição de registrador, você pode usar o ARN da versão para criar uma versão de grupo antes de implantar o grupo.

Limitações de registro em log

AWS IoT Greengrass tem as seguintes limitações de registro.

Transações por segundo

Quando o login CloudWatch está ativado, o componente de registro agrupa os eventos de log localmente antes de enviá-los para CloudWatch, para que você possa fazer login a uma taxa superior a cinco solicitações por segundo por stream de log.

Memória

Se AWS IoT Greengrass estiver configurado para enviar registros para CloudWatch e uma função Lambda registrar mais de 5 MB/second por um período prolongado de tempo, o pipeline de processamento interno eventualmente será preenchido. O pior caso teórico é 6 MB por função do Lambda.

Distorção do relógio

Quando o login CloudWatch está ativado, o componente de registro assina solicitações CloudWatch usando o processo normal de assinatura do Signature Version 4. Se o horário do sistema no dispositivo do núcleo AWS IoT Greengrass ficar fora de sincronia por mais de 15 minutos, as solicitações serão rejeitadas.

Uso do disco

Use a fórmula a seguir para calcular a quantidade total máxima de uso do disco para registro.

```
greengrass-system-component-space * 8    // 7 if automatic IP detection is disabled
  + 128KB                                // the internal log for the local logging
 component
  + lambda-space * lambda-count          // different versions of a Lambda function are
 treated as one
```

Em que:

`greengrass-system-component-space`

A quantidade máxima de armazenamento local para os registros dos componentes AWS IoT Greengrass do sistema.

`lambda-space`

A quantidade máxima de armazenamento local para logs de função do Lambda.

`lambda-count`

O número de funções do Lambda implantadas.

Perda de log

Se seu dispositivo AWS IoT Greengrass principal estiver configurado para fazer login somente CloudWatch e não houver conexão com a Internet, você não terá como recuperar os registros atualmente na memória.

Quando as funções do Lambda são encerradas (por exemplo, durante a implantação), alguns segundos de registros não são gravados. CloudWatch

CloudTrail troncos

AWS IoT Greengrass é executado com AWS CloudTrail, um serviço que fornece um registro das ações realizadas por um usuário, função ou AWS serviço em AWS IoT Greengrass. Para obter mais informações, consulte the section called “Registro em log AWS IoT Greengrass Chamadas de API com AWS CloudTrail”.

Registro em log AWS IoT Greengrass Chamadas de API com AWS CloudTrail

AWS IoT Greengrass 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. 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 AWS IoT Greengrass da API. Se você criar uma trilha, poderá habilitar a entrega contínua de CloudTrail eventos para um bucket do Amazon 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 saber mais sobre isso CloudTrail, consulte o Guia AWS CloudTrail do usuário.

AWS IoT Greengrass 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 visualizar, pesquisar e baixar eventos recentes no seu 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 Amazon 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 Amazon 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 saber mais, 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 as ações são registradas CloudTrail e documentadas na referência da AWS IoT Greengrass API. Por exemplo, chamadas para as `CreateFunctionDefinition` ações `AssociateServiceRoleToAccount``GetGroupVersion`,`GetConnectivityInfo`, e 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 o seguinte:

- 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 .

Noções básicas AWS IoT Greengrass entradas do arquivo de log

Uma trilha é uma configuração que permite a entrega de eventos como arquivos de log para um bucket do Amazon S3 que você especificar. CloudTrail os arquivos de log contêm uma ou mais entradas de log. Um evento representa uma única solicitação de qualquer fonte e 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 `AssociateServiceRoleToAccount` ação.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/Mary_Major",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "Mary_Major"
    },
    "eventTime": "2018-10-17T17:04:02Z",
    "eventSource": "greengrass.amazonaws.com",
    "eventName": "AssociateServiceRoleToAccount",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "203.0.113.12",
    "userAgent": "apimanager.amazonaws.com",
    "errorCode": "BadRequestException",
    "requestParameters": null,
    "responseElements": {
        "Message": "That role ARN is invalid."
    },
    "requestID": "a5990ec6-d22e-11e8-8ae5-c7d2eEXAMPLE",
    "eventID": "b9070ce2-0238-451a-a9db-2dbf1EXAMPLE",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```

O exemplo a seguir mostra uma entrada de CloudTrail registro que demonstra a `GetGroupVersion` ação.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/Mary_Major",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "Mary_Major",
        "sessionContext": {
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2018-10-17T18:14:57Z"
            }
        },
        "invokedBy": "apimanager.amazonaws.com"
    },
    "eventTime": "2018-10-17T18:15:11Z",
    "eventSource": "greengrass.amazonaws.com",
    "eventName": "GetGroupVersion",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "203.0.113.12",
    "userAgent": "apimanager.amazonaws.com",
    "requestParameters": {
        "GroupVersionId": "6c477753-dbf2-4cb8-acc3-5ba4eEXAMPLE",
        "GroupId": "90fcf6df-413c-4515-93a8-00056EXAMPLE"
    },
    "responseElements": null,
    "requestID": "95dcffce-d238-11e8-9240-a3993EXAMPLE",
    "eventID": "8a608034-82ed-431b-b5e0-87fbdEXAMPLE",
    "readOnly": true,
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```


O exemplo a seguir mostra uma entrada de CloudTrail registro que demonstra a `GetConnectivityInfo` ação.


```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
```

```
        "arn": "arn:aws:iam::123456789012:user/Mary_Major",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "Mary_Major"
    },
    "eventTime": "2018-10-17T17:02:12Z",
    "eventSource": "greengrass.amazonaws.com",
    "eventName": "GetConnectivityInfo",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "203.0.113.12",
    "userAgent": "apimanager.amazonaws.com",
    "requestParameters": {
        "ThingName": "us-east-1_CIS_1539795000000_"
    },
    "responseElements": null,
    "requestID": "63e3ebe3-d22e-11e8-9ddd-5baf3EXAMPLE",
    "eventID": "db2260d1-a8cc-4a65-b92a-13f65EXAMPLE",
    "readOnly": true,
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```

O exemplo a seguir mostra uma entrada de CloudTrail registro que demonstra a `CreateFunctionDefinition` ação.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::123456789012:user/Mary_Major",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "Mary_Major"
    },
    "eventTime": "2018-10-17T18:01:11Z",
    "eventSource": "greengrass.amazonaws.com",
    "eventName": "CreateFunctionDefinition",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "203.0.113.12",
    "userAgent": "apimanager.amazonaws.com",
    "requestParameters": {
        "InitialVersion": "***"
```

```
    },
    "responseElements": {
        "CreationTimestamp": "2018-10-17T18:01:11.449Z",
        "LatestVersion": "dae06a61-c32c-41e9-b983-ee5cfEXAMPLE",
        "LatestVersionArn": "arn:aws:greengrass:us-east-1:123456789012:/greengrass/
definition/functions/7a94847d-d4d2-406c-9796-a3529EXAMPLE/versions/dae06a61-c32c-41e9-
b983-ee5cfEXAMPLE",
        "LastUpdatedTimestamp": "2018-10-17T18:01:11.449Z",
        "Id": "7a94847d-d4d2-406c-9796-a3529EXAMPLE",
        "Arn": "arn:aws:greengrass:us-east-1:123456789012:/greengrass/definition/
functions/7a94847d-d4d2-406c-9796-a3529EXAMPLE"
    },
    "requestID": "a17d4b96-d236-11e8-a74e-3db27EXAMPLE",
    "eventID": "bdbf6677-a47a-4c78-b227-c5f64EXAMPLE",
    "readOnly": false,
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```


Consulte também

- O que AWS CloudTrailé no Guia do AWS CloudTrail usuário
- Criação de uma EventBridge regra que é acionada em uma chamada de AWS API usando o CloudTrail Amazon EventBridge User Guide
- AWS IoT Greengrass Referência de API

Coletando dados de telemetria de integridade do sistema de AWS IoT Greengrass dispositivos principais

Os dados de telemetria da integridade do sistema são dados de diagnóstico que podem ajudá-lo a monitorar a performance de operações críticas em seus dispositivos de Greengrass nucleus. O agente de telemetria no núcleo 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.

> **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 dispositivos principais do Greengrass e notificações de implantação. Para obter mais informações, consulte O que é a Amazon EventBridge? no Guia do EventBridge usuário da Amazon.

É possível criar projetos e aplicações para recuperar, analisar, transformar e relatar dados de telemetria de seus dispositivos de borda. Especialistas em domínio, como engenheiros de processo, podem usar essas aplicações para obter informações sobre a integridade da frota.

Para garantir que os componentes do Greengrass Edge funcionem 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 Greengrass só retém dados de telemetria por até sete dias.

Esse recurso está disponível no software AWS IoT Greengrass Core v1.11.0 e é ativado por padrão para todos os núcleos do Greengrass, incluindo os núcleos existentes. Você começa a receber dados automaticamente assim que atualiza para o software AWS IoT Greengrass Core v1.11.0 ou posterior.

Para obter informações sobre como acessar ou gerenciar dados de telemetria publicados, consulte the section called “Fazendo a assinatura para receber dados de telemetria”.

O atendente de telemetria coleta e publica as seguintes métricas do sistema.

Métricas de telemetria

Name (Nome)	Description	Fonte
`SystemMemUsage`	A quantidade de memória atualmente em uso por todos os aplicativos no dispositivo de núcleo do Greengrass, incluindo o sistema operacional.	Sistema
`CpuUsage`	A quantidade de CPU atualmente em uso por todos	Sistema

Name (Nome)	Description	Fonte
	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.	Sistema
`LambdaOutOfMemory`	O número de execuções que resultam na falta de memória da função do Lambda.	Sistema
`DroppedMessageCount`	O número de mensagens descartadas que são destinadas a AWS IoT Core.	Componente do sistema do `GGCloudSpooler`
`LambdaTimeout`	O número de tempos limite para execução da função do Lambda definida pelo usuário.	User-defined Função e sistema Lambda Nuvem AWS
`LambdaUngracefullyKilled`	O número de execuções que a função do Lambda definida pelo usuário não consegue concluir.	User-defined Função e sistema Lambda Nuvem AWS
`LambdaError`	O número de execuções que resultam na gravação de registros de erros da função do Lambda definida pelo usuário.	User-defined Função e sistema Lambda Nuvem AWS

Name (Nome)	Description	Fonte
`BytesAppended`	O número de bytes de dados anexados ao gerenciador de fluxo.	Componente do sistema do `GGStreamManager`
`BytesUploadedToIoTAnalytics`	O número de bytes de dados que o gerenciador de fluxo exporta para os canais em AWS IoT Analytics.	Componente do sistema do `GGStreamManager`
`BytesUploadedToKinesis`	O número de bytes de dados que o gerenciador de fluxos exporta para streams no Amazon Kinesis Data Streams.	Componente do sistema do `GGStreamManager`
`BytesUploadedToIoTSiteWise`	O número de bytes de dados que o gerenciador de fluxo exporta para as propriedades do ativo AWS IoT SiteWise.	Componente do sistema do `GGStreamManager`
`BytesUploadedToS3ExportTaskExecutor`	O número de bytes de dados que o gerenciador de fluxos exporta para objetos no Amazon S3.	Componente do sistema do `GGStreamManager`
`BytesUploadedToHTTP`	O número de bytes de dados que o gerenciador de fluxos exporta para HTTP.	Componente do sistema do `GGStreamManager`

Definindo configurações de telemetria

A telemetria do Greengrass usa as seguintes configurações:

- O atendente de telemetria agrega dados de telemetria a cada hora.
- O atendente de telemetria publica uma mensagem de telemetria a cada 24 horas.

> **Note**
>
> As configurações são imutáveis.

Você pode habilitar ou desabilitar o atributo de telemetria para um dispositivo de núcleo do Greengrass. AWS IoT Greengrass usa sombras para gerenciar a configuração de telemetria. Suas alterações entram em vigor imediatamente quando o núcleo tem uma conexão com AWS IoT Core o.

O atendente de telemetria publica dados usando o protocolo MQTT com um nível de qualidade de serviço (QoS) de 0. Isso significa que ele não confirma a entrega nem repete as tentativas de publicação. 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 agente publica em um tópico AWS reservado. No entanto, dependendo do seu caso de uso, você pode incorrer em custos ao receber ou processar os dados.

Requisitos

Os requisitos a seguir se aplicam quando você define as configurações de telemetria:

- Você deve usar o software AWS IoT Greengrass Core v1.11.0 ou posterior.

  > **Note**
  >
  > Se você estiver executando uma versão anterior e não desejar usar telemetria, você não precisará executar ações nesse momento.

- Você deve fornecer permissões do IAM para atualizar a sombra do núcleo (coisa) e chamar as APIs de configuração antes de atualizar as configurações de telemetria.

  O exemplo de política do IAM a seguir permite gerenciar a configuração de sombra e de runtime de um núcleo específico:

  JSON

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

```
            "Sid": "AllowManageShadow",
            "Effect": "Allow",
            "Action": [
                "iot:GetThingShadow",
                "iot:UpdateThingShadow",
                "iot:DeleteThingShadow",
                "iot:DescribeThing"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:thing/core-name-*"
            ]
        },
        {
            "Sid": "AllowManageRuntimeConfig",
            "Effect": "Allow",
            "Action": [
                "greengrass:GetCoreRuntimeConfiguration",
                "greengrass:UpdateCoreRuntimeConfiguration"
            ],
            "Resource": [
                "arn:aws:iot:us-east-1:123456789012:thing/core-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.

Defina as configurações de telemetria (console)

A seguir, mostramos como atualizar as configurações de telemetria de um núcleo do Greengrass no console. AWS IoT Greengrass

1. No painel de navegação do AWS IoT console, em Gerenciar, expanda dispositivos Greengrass e escolha Grupos (V1).
2. Em Grupos do Greengrass, selecione seu grupo alvo.
3. Na página de configuração do núcleo, na seção Visão geral, selecione seu Greengrass core.
4. Na página de configuração do núcleo, selecione a guia Telemetria.

5. Na seção Telemetria da integridade do sistema, selecione Configure.
6. Em Configurar telemetria, selecione Telemetry para ativar ou desativar o status de telemetria.

> **Important**
>
> Por padrão, o recurso de telemetria está habilitado para o software AWS IoT Greengrass Core v1.11.0 ou posterior.

As alterações entram em vigor no runtime. Não é necessário implantar o grupo.

Defina as configurações de telemetria (CLI)

Na AWS IoT Greengrass API, o `TelemetryConfiguration` objeto representa as configurações de telemetria de um núcleo do Greengrass. Esse objeto faz parte do objeto `RuntimeConfiguration` associado ao núcleo. Você pode usar a AWS IoT Greengrass API ou o AWS SDK para gerenciar a telemetria do Greengrass. AWS CLI Os exemplos desta seção usam a AWS CLI.

Para verificar as configurações de telemetria

O comando a seguir obtém as configurações de telemetria de um núcleo do Greengrass.

- *`core-thing-name`*Substitua pelo nome do núcleo de destino.

  Para obter o nome da coisa, você usa o comando get-core-definition-version. O comando retorna o ARN da coisa que contém o nome da coisa.

```
aws greengrass get-thing-runtime-configuration --thing-name core-thing-name
```

O comando retorna um objeto `GetCoreRuntimeConfigurationResponse` na resposta JSON. Por exemplo:

```
{
    "RuntimeConfiguration": {
        "TelemetryConfiguration": {
            "ConfigurationSyncStatus": "OutOfSync",
            "Telemetry": "On"
        }
    }
}
```

Para definir as configurações de telemetria

O comando a seguir atualiza as configurações de telemetria para um núcleo do Greengrass.

- *core-thing-name*Substitua pelo nome do núcleo de destino.

  Para obter o nome da coisa, você usa o comando get-core-definition-version. O comando retorna o ARN da coisa que contém o nome da coisa.

JSON expanded

```
aws greengrass update-thing-runtime-configuration --thing-name core-thing-name --
telemetry-configuration  '{
    "RuntimeConfiguration": {
        "TelemetryConfiguration": {
            "ConfigurationSyncStatus": "InSync",
            "Telemetry": "Off"
        }
    }
}
```

JSON single-line

```
aws greengrass update-thing-runtime-configuration --thing-name core-thing-name --
telemetry-configuration "{\"TelemetryConfiguration\":{\"ConfigurationSyncStatus
\":\"InSync\",\"Telemetry\":\"Off\"}}"
```

As alterações nas configurações de telemetria serão aplicadas, se `ConfigurationSyncStatus` for `InSync`. As alterações entram em vigor no runtime. Não é necessário implantar o grupo.

TelemetryConfiguration objeto

O objeto `TelemetryConfiguration` tem as seguintes propriedades:

`ConfigurationSyncStatus`

Verifica se as configurações de telemetria estão sincronizadas. Talvez você não faça alterações nessa propriedade.

Tipo: string

Valores válidos: `InSync` ou `OutOfSync`

`Telemetry`

Ativa ou desativa a telemetria. O padrão é `On`.

Tipo: string

Valores válidos: `On` ou `Off`

Fazendo a assinatura para receber dados de telemetria

Você pode criar regras na Amazon EventBridge que definam como processar dados de telemetria publicados a partir do 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.

Evento de telemetria

O evento para uma alteração no estado da implantação, incluindo os dados de telemetria, usa o seguinte formato:

```
{
    "version": "0",
    "id": "f70f943b-9ae2-e7a5-fec4-4c22178a3e6a",
    "detail-type": "Greengrass Telemetry Data",
    "source": "aws.greengrass",
    "account": "123456789012",
    "time": "2020-07-28T20:45:53Z",
    "region": "us-west-1",
    "resources": [],
    "detail": {
        "ThingName": "CoolThing",
        "Schema": "2020-06-30",
        "ADP": [
            {
                "TS": 123231546,
                "NS": "StreamManager",
                "M": [
                    {
                        "N": "BytesAppended|BytesUploadedToKinesis",
                        "Sum": 11,
                        "U": "Bytes"
                    }
```

```
            ]
        },
        {
            "TS": 123231546,
            "NS": "StreamManager",
            "M": [
                {
                    "N": "BytesAppended|BytesUploadedToS3ExportTaskExecutor",
                    "Sum": 11,
                    "U": "Bytes"
                }
            ]
        },
        {
            "TS": 123231546,
            "NS": "StreamManager",
            "M": [
                {
                    "N": "BytesAppended|BytesUploadedToHTTP",
                    "Sum": 11,
                    "U": "Bytes"
                }
            ]
        },
        {
            "TS": 123231546,
            "NS": "StreamManager",
            "M": [
                {
                    "N": "BytesAppended|BytesUploadedToIoTAnalytics",
                    "Sum": 11,
                    "U": "Bytes"
                }
            ]
        },
        {
            "TS": 123231546,
            "NS": "StreamManager",
            "M": [
                {
                    "N": "BytesAppended|BytesUploadedToIoTSiteWise",
                    "Sum": 11,
                    "U": "Bytes"
                }
```

```
            ]
        },
        {
            "TS": 123231546,
            "NS": "arn:aws:lambda:us-west-1:123456789012:function:my-function",
            "M": [
                {
                    "N": "LambdaTimeout",
                    "Sum": 15,
                    "U": "Count"
                }
            ]
        },
        {
            "TS": 123231546,
            "NS": "CloudSpooler",
            "M": [
                {
                    "N": "DroppedMessageCount",
                    "Sum": 15,
                    "U": "Count"
                }
            ]
        },
        {
            "TS": 1593727692,
            "NS": "SystemMetrics",
            "M": [
                {
                    "N": "SystemMemUsage",
                    "Sum": 11.23,
                    "U": "Megabytes"
                },
                {
                    "N": "CpuUsage",
                    "Sum": 35.63,
                    "U": "Percent"
                },
                {
                    "N": "TotalNumberOfFDs",
                    "Sum": 416,
                    "U": "Count"
                }
            ]
```

```
            },
            {
                "TS": 1593727692,
                "NS": "arn:aws:lambda:us-west-1:123456789012:function:my-function",
                "M": [
                    {
                        "N": "LambdaOutOfMemory",
                        "Sum": 12,
                        "U": "Count"
                    },
                    {
                        "N": "LambdaUngracefullyKilled",
                        "Sum": 100,
                        "U": "Count"
                    },
                    {
                        "N": "LambdaError",
                        "Sum": 7,
                        "U": "Count"
                    }
                ]
            }
        ]
    }
}
```

A matriz ADP contém uma lista de pontos de dados agregados que têm as seguintes propriedades:

`TS`

Obrigatório. O timestamp de quando os dados foram agregados.

`NS`

Obrigatório. O namespace do sistema.

`M`

Obrigatório. Lista de métricas. Uma métrica contém as seguintes propriedades:

`N`

O nome da métrica.

`Sum`

O valor agregado da métrica. Oatendente de telemetria adiciona novos valores ao total anterior, então a soma é um valor cada vez maior. Você pode usar o timestamp para encontrar o valor de uma agregação específica. Por exemplo, para encontrar o valor agregado mais recente, subtraia o valor com timestamp anterior do valor com timestamp mais recente.

`U`

A unidade do valor da métrica.

`ThingName`

Obrigatório. O nome da coisa que você tem como alvo.

Pré-requisitos para criar regras EventBridge

Antes de criar uma EventBridge regra para AWS 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 streams do Amazon Kinesis, funções 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 Console de gerenciamento da AWS para criar uma EventBridge regra que receba dados de telemetria publicados pelo núcleo 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. Selecione Barramento de eventos e ative a regra no barramento de eventos selecionado.
4. Selecione o Tipo de regra e, em seguida selecione Rule with an event pattern (Regra com um padrão de eventos).
5. Escolha Próximo.
6. Em Origem do evento, escolha AWS eventos ou eventos de EventBridge parceiros.
7. Em Exemplo de evento, selecione AWS events e, em seguida selecione Dados de telemetria do Greengrass.
8. Para Padrão de eventos, faça as seguintes seleções:

   a. Em Event source (Origem do evento), selecione AWS services (Serviços da).
   b. Em Serviço da AWS , selecione Greengrass.
   c. Em Tipo de evento, selecione Dados de telemetria do Greengrass.
9. Selecione Next (Próximo).
10. Em Destino 1, selecione serviço da AWS .
11. Em Selecionar um destino, selecione Fila SQS.
12. Em Fila, selecione seu perfil.

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 pelo núcleo do Greengrass. Isso permite que servidores web, endereços de e-mail e outros assinantes de tópicos respondam ao evento.

1. Crie a regra.

   - *thing-name*Substitua pelo nome da coisa do núcleo.

     Para obter o nome da coisa, você usa o comando get-core-definition-version. O comando retorna o ARN da coisa que contém o nome da coisa.

```
aws events put-rule \
  --name TestRule \
  --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.

   - *queue-arn*Substitua pelo ARN da sua fila do Amazon SQS.

```
aws events put-targets \
  --rule TestRule \
  --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.

Solução de problemas AWS IoT Greengrass telemetria

Use as informações a seguir para ajudar a solucionar problemas com a configuração da telemetria. AWS IoT Greengrass

Erro: a resposta contém "ConfigurationStatus“:" OutOfSync "depois de executar o comando get-thing-runtime-configuration

Soluções:

- O serviço AWS IoT Device Shadow leva tempo para processar as atualizações de configuração em tempo de execução e entregar as atualizações ao dispositivo principal do Greengrass. Você pode esperar e verificar se as configurações de telemetria estão sincronizadas posteriormente.
- Verifique se dispositivo de núcleo está online.
- Ative o Amazon CloudWatch Logs in AWS IoT Core para monitorar a sombra.
- Use as métricas AWS IoT para monitorar sua coisa.

Chamada da API de verificação de integridade local

AWS IoT Greengrass contém uma API HTTP local que fornece um instantâneo do estado atual dos processos de trabalho locais que foram iniciados por AWS IoT Greengrass. Esse snapshot inclui funções do Lambda do sistema e definidas pelo usuário. As funções do System Lambda fazem parte do software AWS IoT Greengrass Core. Eles são executados como processos de operadores locais no dispositivo principal e gerenciam operações como roteamento de mensagens, sincronização de sombras locais e detecção automática do endereço IP.

A API de verificação de integridade é compatível com as seguintes solicitações:

- Enviar uma solicitação `GET` para obter informações de integridade sobre todos os operadores.
- Enviar uma solicitação `POST` para obter informações de integridade sobre operadores específicos.

As solicitações são enviadas localmente no dispositivo e não exigem conexão com a Internet.

Obter informações de integridade sobre todos os operadores

Enviar uma solicitação `GET` para obter informações de integridade sobre todos os operadores em execução.

- *`port`*Substitua pelo número da porta do IPC.

```
GET http://localhost:port/2016-11-01/health/workers
```

`port`

O número da porta do IPC.

O valor pode variar entre 1024 e 65535. O valor padrão é 8000.

Para alterar o número desta porta, você pode atualizar a propriedade `ggDaemonPort` no arquivo `config.json`. Para obter mais informações, consulte AWS IoT Greengrass arquivo de configuração principal.

Exemplo de solicitação

O exemplo de solicitação `curl` a seguir obtém informações de integridade de todos os operadores.

```
curl http://localhost:8000/2016-11-01/health/workers
```

Resposta do JSON

Essa solicitação retorna uma matriz de objetos de informações de integridade dos operadores.

Exemplo de resposta

O exemplo de resposta a seguir lista objetos de informações de saúde para todos os processos de trabalho que foram iniciados por AWS IoT Greengrass.

```
[
    {
      "FuncArn": "arn:aws:lambda:::function:GGShadowService:1",
      "WorkerId" : "65515053-2f70-43dc-7cc0-1712bEXAMPLE",
      "ProcessId": "1234",
      "WorkerState": "Waiting"
    },
    {
      "FuncArn": "arn:aws:lambda:::function:GGSecretManager:1",
      "WorkerId": "a9916cc2-1b4d-4f0e-4b12-b1872EXAMPLE",
      "ProcessId": "9798",
      "WorkerState": "Waiting"
    },
    {
      "FuncArn": "arn:aws:lambda:us-west-2:123456789012:function:my-lambda-function:3",
      "WorkerId": "2e6f785e-66a5-42c9-67df-42073EXAMPLE",
      "ProcessId": "11837",
      "WorkerState": "Waiting"
    },
    ...
]
```

Obtenha informações de integridade sobre determinados operadores

Enviar uma solicitação POST para obter informações de integridade sobre determinados operadores. *port* Substitua pelo número da porta do IPC. O padrão é 8000.

```
POST http://localhost:port/2016-11-01/health/workers
```

Exemplo de solicitação

O exemplo de solicitação `curl` a seguir obtém informações de integridade de determinados operadores.

```
curl --data "@body.json" http://localhost:8000/2016-11-01/health/workers
```

Veja um exemplo da estrutura de uma solicitação `body.json` abaixo:

```
{
    "FuncArns": [
        "arn:aws:lambda:::function:GGShadowService:1",
        "arn:aws:lambda:us-west-2:123456789012:function:my-lambda-function:3"
    ]
}
```

A estrutura da solicitação contém uma matriz `FuncArns`.

`FuncArns`

Uma lista de nomes de recursos da Amazon (ARNs) para funções do Lambda que representam os operadores de destino.

- Para funções do Lambda definidas pelo usuário, especifique o ARN da versão atualmente implantada. Se você adicionou funções do Lambda ao grupo usando um ARN do alias, pode usar a solicitação GET para obter todos os operadores e, em seguida, escolher os ARNs que deseja consultar.
- Para funções do Lambda do sistema, especifique o ARN da função do Lambda correspondente. Para obter mais informações, consulte the section called “Funções do Lambda do sistema”.

Tipo: matriz de strings

Tamanho mínimo: 1

Comprimento máximo: o número total de trabalhadores iniciados AWS IoT Greengrass no dispositivo principal.

Resposta do JSON

Essa solicitação retorna uma matriz de `Workers` e uma matriz de `InvalidArns`.

`Workers`

Uma lista de objetos de informações de integridade dos operadores especificados.

Tipo: matriz de objetos de informações de integridade

`InvalidArns`

Uma lista de ARNs de função que são inválidos, incluindo ARNs de função que não têm operadores associados.

Tipo: matriz de strings

Exemplo de resposta

O exemplo de resposta a seguir lista objetos de informações de integridade para determinados operadores.

```
{
    "Workers": [
        {
            "FuncArn": "arn:aws:lambda:::function:GGShadowService:1",
            "WorkerId" : "65515053-2f70-43dc-7cc0-1712bEXAMPLE",
            "ProcessId": "1234",
            "WorkerState": "Waiting"
        },
        {
            "FuncArn": "arn:aws:lambda:us-west-2:123456789012:function:my-lambda-
function:3",
            "WorkerId": "2e6f785e-66a5-42c9-67df-42073ESAMPLE",
            "ProcessId": "11837",
            "WorkerState": "Waiting"
        }
    ],
    "InvalidArns" : [
        "some-malformed-arn",
        "arn:aws:lambda:us-west-2:123456789012:function:some-unknown-function:1"
    ]
}
```

Esta consulta retorna os seguintes erros:

400 Solicitação inválida

O corpo da solicitação está malformado. Para resolver esse problema, utilize o formato a seguir e reenvie a solicitação:

```
{"FuncArns":["function-1-arn","function-2-arn"]}
```

400 A solicitação excede o número máximo de operadores

O número de ARNs especificado na matriz de `FuncArns` excede o número de operadores.

Informações de integridade dos operadores

Um objeto de informações de integridade contém as seguintes propriedades:

`FuncArn`

O ARN da função do Lambda do sistema que representa o operador.

Tipo: `string`

`WorkerId`

A ID do operador. Esta propriedade pode ser útil para depuração. O arquivo `runtime.log` e os logs da função do Lambda contêm a ID do operador, portanto, essa propriedade pode ser especialmente útil para depurar uma função do Lambda sob demanda que cria várias instâncias.

Tipo: `string`

`ProcessId`

A ID do processo (PID) do processo de operadores.

Tipo: `int`

`WorkerState`

O estado do operador.

Tipo: `string`

Os estados possíveis são os seguintes:

`Working`

Processando uma mensagem.

`Waiting`

Aguardando uma mensagem. Aplica-se a funções do Lambda de longa duração executadas como um daemon ou processo autônomo.

`Starting`

Criado, começando.

`FailedInitialization`

Falha ao inicializar.

`Terminated`

Parado pelo daemon do Greengrass

`NotStarted`

Falha ao iniciar, fazendo outra tentativa inicial.

`Initialized`

Inicializado com sucesso.

Funções do Lambda do sistema

Você pode solicitar informações de integridade às seguintes funções do Lambda do sistema:

`GGCloudSpooler`

Gerencia a fila para mensagens MQTT que têm AWS IoT Core como origem ou destino.

ARN: `arn:aws:lambda:::function:GGCloudSpooler:1`

`GGConnManager`

Encaminha mensagens MQTT entre o núcleo do Greengrass e os dispositivos do cliente.

ARN: `arn:aws:lambda:::function:GGConnManager`

`GGDeviceCertificateManager`

Escuta a AWS IoT sombra em busca de alterações nos endpoints IP do núcleo e gera o certificado do lado do servidor usado para autenticação mútua. GGConnManager

ARN: `arn:aws:lambda:::function:GGDeviceCertificateManager`

`GGIPDetector`

Detecção automática de endereço IP que permite aos dispositivos no grupo do Greengrass descobrirem o dispositivo principal do Greengrass. Esse serviço não é aplicável quando você fornece endereços IP manualmente.

ARN: `arn:aws:lambda:::function:GGIPDetector:1`

`GGSecretManager`

Gerencia o armazenamento seguro de segredos locais e o acesso por meio de conectores e Lambda definidos pelo usuário.

ARN: `arn:aws:lambda:::function:GGSecretManager:1`

`GGShadowService`

Gerencia sombras locais para dispositivos cliente.

ARN: `arn:aws:lambda:::function:GGShadowService`

`GGShadowSyncManager`

Sincroniza as sombras locais com Nuvem AWS as do dispositivo principal e dos dispositivos cliente, se a `syncShadow` propriedade do dispositivo estiver definida como. `true`

ARN: `arn:aws:lambda:::function:GGShadowSyncManager`

`GGStreamManager`

Processa fluxos de dados localmente e executa exportações automáticas para o. Nuvem AWS

ARN: `arn:aws:lambda:::function:GGStreamManager:1`

`GGTES`

O serviço local de troca de tokens que recupera as credenciais do IAM definidas na função do grupo Greengrass que o código local usa para acessar os serviços. AWS

ARN: `arn:aws:lambda:::function:GGTES`

Marcando seu AWS IoT Greengrass recursos

As tags podem ajudar você a organizar e gerenciar seus AWS IoT Greengrass grupos. Você pode usar tags para atribuir metadados a grupos, implantações em massa, núcleos, dispositivos e outros recursos que são adicionados aos grupos. As tags também podem ser usadas em políticas do IAM para definir o acesso condicional aos recursos do Greengrass.

> **Note**
>
> Atualmente, as tags de recursos do Greengrass não são suportadas para grupos de AWS IoT cobrança ou relatórios de alocação de custos.

Conceitos Básicos de Tags

As tags permitem que você categorize seus AWS IoT Greengrass recursos, por exemplo, por finalidade, proprietário e ambiente. Quando há muitos recursos do mesmo tipo, você pode identificar rapidamente um recurso com base nas tags que estão anexadas a ele. Uma tag consiste em uma chave e um valor opcional, ambos definidos por você. Recomendamos que você desenvolva um conjunto de chave de tags para cada tipo de recurso. Usar um conjunto consistente de chaves de tags facilita para você gerenciar seus recursos. Por exemplo, é possível definir um conjunto de tags para os grupos que ajudam você a monitorar a localização da fábrica de seus dispositivos principais. Para obter mais informações, consulte Estratégias de marcação da AWS.

Suporte de marcação no AWS IoT console

Você pode criar, visualizar e gerenciar tags para seus `Group` recursos do Greengrass no AWS IoT console. 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 Amazon Web Services.

Para atribuir tags ao criar um grupo

1. Você pode atribuir tags a um grupo ao criar o grupo. Selecione Adicionar nova tag na seção Tags para mostrar os campos de entrada de marcação.

Como visualizar e gerenciar tags na página de configuração do grupo

Você pode visualizar e gerenciar tags na página de configuração do grupo selecionando Visualizar configurações. Na seção Tags do grupo, selecione Gerenciar tags para adicionar, editar ou remover tags de grupo.

Suporte de marcação no AWS IoT Greengrass solicitações de

Você pode usar a AWS IoT Greengrass API para criar, listar e gerenciar tags para AWS IoT Greengrass recursos que oferecem suporte à marcação. 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 Amazon Web Services.

- Para adicionar tags durante a criação do recurso, defina-as na propriedade `tags` do recurso.
- Para adicionar tags após a criação de um recurso ou para atualizar valores de tag, use a ação `TagResource`.
- Para remover tags de um recurso, use a ação `UntagResource`.
- Para recuperar as tags que estão associadas a um recurso, use a ação `ListTagsForResource` ou obtenha o recurso e inspecione a propriedade `tags` dele.

A tabela a seguir lista os recursos que você pode marcar na AWS IoT Greengrass API e suas `Get` ações `Create` e ações correspondentes.

Recurso	Criar	Obtenção
`Group`	`CreateGroup`	`GetGroup`
`ConnectorDefinition`	`CreateConnectorDefinition`	`GetConnectorDefinition`
`CoreDefinition`	`CreateCoreDefinition`	`GetCoreDefinition`
`DeviceDefinition`	`CreateDeviceDefinition`	`GetDeviceDefinition`
`FunctionDefinition`	`CreateFunctionDefinition`	`GetFunctionDefinition`

Recurso	Criar	Obtenção
`LoggerDefinition`	`CreateLoggerDefinition`	`GetLoggerDefinition`
`ResourceDefinition`	`CreateResourceDefinition`	`GetResourceDefinition`
`SubscriptionDefinition`	`CreateSubscriptionDefinition`	`GetSubscriptionDefinition`
`BulkDeployment`	`StartBulkDeployment`	`GetBulkDeploymentStatus`

Use as ações a seguir para listar e gerenciar as tags de recursos que ofereçam suporte à atribuição de tags:

- `TagResource`. Adiciona tags a um recurso. Também é usada para alterar o valor do par de chave/valor da tag.
- `ListTagsForResource`. Lista as tags de um recurso.
- `UntagResource`. Remove tags de um recurso.

Você pode adicionar tags a um recurso ou removê-las 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 substituirá o valor antigo. Você pode definir um valor a uma string vazia, mas não pode definir o valor como nulo.

Quando você excluir um recurso, as tags associadas ao recurso também serão excluídas.

> **Note**
>
> Não confunda tags de recursos com os atributos que você pode atribuir às AWS IoT coisas. Embora os núcleos do Greengrass sejam AWS IoT coisas, as tags de recursos descritas neste tópico estão anexadas a uma`CoreDefinition`, não à coisa principal.

Utilização de tags com políticas do IAM

Em suas políticas do IAM, você pode usar tags de recurso para controlar o acesso de usuários e permissões. 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. Você pode marcar recursos durante a criação (chamado marcar ao criar), para que não seja necessário executar scripts de tags personalizadas posteriormente. Quando novos ambientes são iniciados com tags, as permissões correspondentes do IAM são aplicadas automaticamente.

As chaves de contexto de condição e os valores a seguir podem ser usados no elemento `Condition` (também chamado de bloco `Condition`) da política.

`greengrass:ResourceTag/`*`tag-key`*`: `*`tag-value`*

Permite ou nega ações do usuário em recursos com tags específicas.

`aws:RequestTag/`*`tag-key`*`: `*`tag-value`*

Exige que uma tag específica seja (ou não) usada ao fazer solicitações de API para criar ou modificar tags em um recurso marcável.

`aws:TagKeys: [`*`tag-key`*`, ...]`

Exige que um conjunto específico de chaves de tag seja (ou não) usado ao fazer uma solicitação de API para criar ou modificar recurso marcável.

As chaves e valores do contexto de condição só podem ser usados em AWS IoT Greengrass ações que atuam em um recurso marcável. Essas ações consideram o recurso como um parâmetro obrigatório. Por exemplo, você pode definir o acesso condicional no `GetGroupVersion`. Não é possível definir o acesso condicional em `AssociateServiceRoleToAccount` porque não foi feita nenhuma referência a um recurso marcável (por exemplo, grupo, definição de núcleo ou definição de dispositivo) na solicitação.

Para obter mais informações, consulte Controlando o acesso usando tags e Referência de política JSON do IAM no Guia do usuário do IAM. A referência de políticas JSON inclui a sintaxe detalhada, descrições e exemplos dos elementos, variáveis e lógica de avaliação de políticas JSON no IAM.

Políticas de exemplo do IAM

A política de exemplo a seguir aplica permissões baseadas em tags que restringem um usuário beta a ações somente em recursos beta.

- A primeira instrução permite que um usuário do IAM atue em recursos que tenham somente a tag env=beta.
- A segunda instrução impede que um usuário do IAM remova a tag env=beta de recursos. Isso impede que o usuário remova seu próprio acesso.

> **Note**
>
> Se usar tags para controlar o acesso a recursos, você também deverá gerenciar as permissões que permitem que os usuários adicionem ou removam tags desses mesmos recursos. Caso contrário, em alguns casos, talvez seja possível que os usuários contornem suas restrições e ganhem acesso a um recurso modificando as tags.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "greengrass:*",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/env": "beta"
                }
            }
        },
        {
            "Effect": "Deny",
            "Action": "greengrass:UntagResource",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/env": "beta"
```

```
                }
            }
        }
    ]
}
```

Para permitir que os usuários adicionem marcações ao criar, você deverá fornecer as permissões apropriadas. O exemplo de política a seguir inclui a condição `"aws:RequestTag/env": "beta"` nas ações `greengrass:TagResource` e `greengrass:CreateGroup`, o que permite que os usuários criem um grupo somente se marcarem o grupo com a tag env=beta. Isso efetivamente força os usuários a marcar novos grupos.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "greengrass:TagResource",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/env": "beta"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": "greengrass:CreateGroup",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:RequestTag/env": "beta"
                }
            }
        }
    ]
}
```

O trecho a seguir mostra como você pode especificar vários valores de tag para determinada chave de tag, colocando-os em uma lista:

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

Consulte também

- Marcando AWS recursos no Referência geral da Amazon Web Services

CloudFormation suporte para AWS IoT Greengrass

CloudFormation é um serviço que pode ajudá-lo a criar, gerenciar e replicar seus AWS recursos. Você pode usar CloudFormation modelos para definir AWS IoT Greengrass grupos e os dispositivos clientes, assinaturas e outros componentes que você deseja implantar. Para ver um exemplo, consulte the section called “Exemplo de modelo”.

Os recursos e a infraestrutura que você gera a partir de um modelo são chamados de pilha. É possível definir todos os recursos em um modelo ou fazer referência a recursos de outras pilhas. Para obter mais informações sobre CloudFormation modelos e recursos, consulte O que é CloudFormation? no Guia do AWS CloudFormation usuário.

Criar recursos do

CloudFormation os modelos são documentos JSON ou YAML que descrevem as propriedades e os relacionamentos dos AWS recursos. Os seguintes AWS IoT Greengrass recursos são compatíveis:

- Groups (Grupos)
- Núcleos
- Dispositivos cliente (dispositivos)
- funções do Lambda
- Conectores
- Recursos (local, machine learning e segredo)
- Assinaturas
- Registradores (configurações de registro)

Nos CloudFormation modelos, a estrutura e a sintaxe dos recursos do Greengrass são baseadas na AWS IoT Greengrass API. Por exemplo, o exemplo de modelo associa um `DeviceDefinition` de alto nível a um `DeviceDefinitionVersion` que contém um dispositivo cliente individual. Para obter mais informações, consulte the section called “Visão geral do modelo de objeto de grupo”.

A referência de tipos de AWS IoT Greengrass recursos no Guia AWS CloudFormation do Usuário descreve os recursos do Greengrass com os quais você pode gerenciar. CloudFormation Ao usar CloudFormation modelos para criar recursos do Greengrass, recomendamos que você os

gerencie somente a partir de. CloudFormation Por exemplo, você deve atualizar seu modelo se quiser adicionar, alterar ou remover um dispositivo (em vez de usar a AWS IoT Greengrass API ou o AWS IoT console). Isso permite que você use a reversão e outros recursos de gerenciamento de alterações do CloudFormation . Para obter mais informações sobre como usar CloudFormation para criar e gerenciar seus recursos e pilhas, consulte Como trabalhar com pilhas no Guia do AWS CloudFormation usuário.

Para ver um passo a passo que mostra como criar e implantar AWS IoT Greengrass recursos em um CloudFormation modelo, consulte Automatizando a AWS IoT Greengrass configuração com a Internet das Coisas CloudFormation no blog oficial. AWS

Implantar recursos

Depois de criar uma CloudFormation pilha que contém a versão do seu grupo, você pode usar o AWS IoT console AWS CLI ou para implantá-la.

> **Note**
>
> Para implantar um grupo, você deve ter uma função de serviço do Greengrass associada à sua. Conta da AWS A função de serviço AWS IoT Greengrass permite acessar seus recursos em AWS Lambda e outros AWS serviços. Essa função deve existir se você já implantou um grupo do Greengrass na Região da AWS atual. Para obter mais informações, consulte the section called “Função de serviço do Greengrass”.

Para implantar o grupo (AWS CLI)

- Execute o comando create-deployment.

  ```
  aws greengrass create-deployment --group-id GroupId --group-version-
  id GroupVersionId --deployment-type NewDeployment
  ```

  > **Note**
  >
  > A instrução `CommandToDeployGroup` no exemplo de modelo mostra como gerar o comando com os IDs do grupo e da versão do grupo ao criar uma pilha.

Para implantar o grupo (console)

1. No painel de navegação do AWS IoT console, em Gerenciar, expanda dispositivos Greengrass e escolha Grupos (V1).
2. Selecione o grupo.
3. Na página de configuração do grupo, selecione Implantar.

Exemplo de modelo

O exemplo de modelo a seguir cria um grupo do Greengrass que contém núcleo, dispositivo cliente, função, registrador, inscrição e dois recursos. Para fazer isso, o modelo segue o modelo de objeto da AWS IoT Greengrass API. Por exemplo, os dispositivos cliente que você deseja adicionar ao grupo estão contidos em um recurso `DeviceDefinitionVersion`, que é associado a um recurso `DeviceDefinition`. Para adicionar os dispositivos ao grupo, a versão do grupo faz referência ao ARN do `DeviceDefinitionVersion`.

O modelo inclui parâmetros que permitem especificar os ARNs do certificado para o núcleo e o dispositivo e o ARN da versão da função Lambda de origem (que é um recurso). AWS Lambda Ele usa as funções intrínsecas `Ref` e `GetAtt` para fazer referência a IDs, ARNs e outros atributos necessários para criar recursos do Greengrass.

O modelo também define dois AWS IoT dispositivos (coisas), que representam o dispositivo principal e o dispositivo cliente que são adicionados ao grupo Greengrass.

Depois de criar a pilha com seus recursos do Greengrass, você pode usar AWS CLI o console ou o console para implantar AWS IoT o grupo.

> **Note**
>
> A instrução `CommandToDeployGroup` no exemplo mostra como gerar um comando create-deployment completo da CLI que pode ser usado para implantar o grupo.

JSON

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Description": "AWS IoT Greengrass example template that creates a group version
 with a core, device, function, logger, subscription, and resources.",
```

```
    "Parameters": {
        "CoreCertificateArn": {
            "Type": "String"
        },
        "DeviceCertificateArn": {
            "Type": "String"
        },
        "LambdaVersionArn": {
            "Type": "String"
        }
    },
    "Resources": {
        "TestCore1": {
            "Type": "AWS::IoT::Thing",
            "Properties": {
                "ThingName": "TestCore1"
            }
        },
        "TestCoreDefinition": {
            "Type": "AWS::Greengrass::CoreDefinition",
            "Properties": {
                "Name": "DemoTestCoreDefinition"
            }
        },
        "TestCoreDefinitionVersion": {
            "Type": "AWS::Greengrass::CoreDefinitionVersion",
            "Properties": {
                "CoreDefinitionId": {
                    "Ref": "TestCoreDefinition"
                },
                "Cores": [
                    {
                        "Id": "TestCore1",
                        "CertificateArn": {
                            "Ref": "CoreCertificateArn"
                        },
                        "SyncShadow": "false",
                        "ThingArn": {
                            "Fn::Join": [
                                ":",
                                [
                                    "arn:aws:iot",
                                    {
                                        "Ref": "AWS::Region"
```

```
                                },
                                {
                                    "Ref": "AWS::AccountId"
                                },
                                "thing/TestCore1"
                            ]
                        ]
                    }
                }
            ]
        }
    },
    "TestClientDevice1": {
        "Type": "AWS::IoT::Thing",
        "Properties": {
            "ThingName": "TestClientDevice1"
        }
    },
    "TestDeviceDefinition": {
        "Type": "AWS::Greengrass::DeviceDefinition",
        "Properties": {
            "Name": "DemoTestDeviceDefinition"
        }
    },
    "TestDeviceDefinitionVersion": {
        "Type": "AWS::Greengrass::DeviceDefinitionVersion",
        "Properties": {
            "DeviceDefinitionId": {
                "Fn::GetAtt": [
                    "TestDeviceDefinition",
                    "Id"
                ]
            },
            "Devices": [
                {
                    "Id": "TestClientDevice1",
                    "CertificateArn": {
                        "Ref": "DeviceCertificateArn"
                    },
                    "SyncShadow": "true",
                    "ThingArn": {
                        "Fn::Join": [
                            ":",
                            [
```

```
                                "arn:aws:iot",
                                {
                                    "Ref": "AWS::Region"
                                },
                                {
                                    "Ref": "AWS::AccountId"
                                },
                                "thing/TestClientDevice1"
                            ]
                        ]
                    }
                }
            ]
        }
    },
    "TestFunctionDefinition": {
        "Type": "AWS::Greengrass::FunctionDefinition",
        "Properties": {
            "Name": "DemoTestFunctionDefinition"
        }
    },
    "TestFunctionDefinitionVersion": {
        "Type": "AWS::Greengrass::FunctionDefinitionVersion",
        "Properties": {
            "FunctionDefinitionId": {
                "Fn::GetAtt": [
                    "TestFunctionDefinition",
                    "Id"
                ]
            },
            "DefaultConfig": {
                "Execution": {
                    "IsolationMode": "GreengrassContainer"
                }
            },
            "Functions": [
                {
                    "Id": "TestLambda1",
                    "FunctionArn": {
                        "Ref": "LambdaVersionArn"
                    },
                    "FunctionConfiguration": {
                        "Pinned": "true",
                        "Executable": "run.exe",
```

```
                        "ExecArgs": "argument1",
                        "MemorySize": "512",
                        "Timeout": "2000",
                        "EncodingType": "binary",
                        "Environment": {
                            "Variables": {
                                "variable1": "value1"
                            },
                            "ResourceAccessPolicies": [
                                {
                                    "ResourceId": "ResourceId1",
                                    "Permission": "ro"
                                },
                                {
                                    "ResourceId": "ResourceId2",
                                    "Permission": "rw"
                                }
                            ],
                            "AccessSysfs": "false",
                            "Execution": {
                                "IsolationMode": "GreengrassContainer",
                                "RunAs": {
                                    "Uid": "1",
                                    "Gid": "10"
                                }
                            }
                        }
                    }
                }
            ]
        }
    },
    "TestLoggerDefinition": {
        "Type": "AWS::Greengrass::LoggerDefinition",
        "Properties": {
            "Name": "DemoTestLoggerDefinition"
        }
    },
    "TestLoggerDefinitionVersion": {
        "Type": "AWS::Greengrass::LoggerDefinitionVersion",
        "Properties": {
            "LoggerDefinitionId": {
                "Ref": "TestLoggerDefinition"
            },
```

```
                "Loggers": [
                    {
                        "Id": "TestLogger1",
                        "Type": "AWSCloudWatch",
                        "Component": "GreengrassSystem",
                        "Level": "INFO"
                    }
                ]
            }
        },
        "TestResourceDefinition": {
            "Type": "AWS::Greengrass::ResourceDefinition",
            "Properties": {
                "Name": "DemoTestResourceDefinition"
            }
        },
        "TestResourceDefinitionVersion": {
            "Type": "AWS::Greengrass::ResourceDefinitionVersion",
            "Properties": {
                "ResourceDefinitionId": {
                    "Ref": "TestResourceDefinition"
                },
                "Resources": [
                    {
                        "Id": "ResourceId1",
                        "Name": "LocalDeviceResource",
                        "ResourceDataContainer": {
                            "LocalDeviceResourceData": {
                                "SourcePath": "/dev/TestSourcePath1",
                                "GroupOwnerSetting": {
                                    "AutoAddGroupOwner": "false",
                                    "GroupOwner": "TestOwner"
                                }
                            }
                        }
                    },
                    {
                        "Id": "ResourceId2",
                        "Name": "LocalVolumeResourceData",
                        "ResourceDataContainer": {
                            "LocalVolumeResourceData": {
                                "SourcePath": "/dev/TestSourcePath2",
                                "DestinationPath": "/volumes/TestDestinationPath2",
                                "GroupOwnerSetting": {
```

```
                                "AutoAddGroupOwner": "false",
                                "GroupOwner": "TestOwner"
                            }
                        }
                    }
                }
            ]
        }
    },
    "TestSubscriptionDefinition": {
        "Type": "AWS::Greengrass::SubscriptionDefinition",
        "Properties": {
            "Name": "DemoTestSubscriptionDefinition"
        }
    },
    "TestSubscriptionDefinitionVersion": {
        "Type": "AWS::Greengrass::SubscriptionDefinitionVersion",
        "Properties": {
            "SubscriptionDefinitionId": {
                "Ref": "TestSubscriptionDefinition"
            },
            "Subscriptions": [
                {
                    "Id": "TestSubscription1",
                    "Source": {
                        "Fn::Join": [
                            ":",
                            [
                                "arn:aws:iot",
                                {
                                    "Ref": "AWS::Region"
                                },
                                {
                                    "Ref": "AWS::AccountId"
                                },
                                "thing/TestClientDevice1"
                            ]
                        ]
                    },
                    "Subject": "TestSubjectUpdated",
                    "Target": {
                        "Ref": "LambdaVersionArn"
                    }
                }
```

```
                ]
            }
        },
        "TestGroup": {
            "Type": "AWS::Greengrass::Group",
            "Properties": {
                "Name": "DemoTestGroupNewName",
                "RoleArn": {
                    "Fn::Join": [
                        ":",
                        [
                            "arn:aws:iam:",
                            {
                                "Ref": "AWS::AccountId"
                            },
                            "role/TestUser"
                        ]
                    ]
                },
                "InitialVersion": {
                    "CoreDefinitionVersionArn": {
                        "Ref": "TestCoreDefinitionVersion"
                    },
                    "DeviceDefinitionVersionArn": {
                        "Ref": "TestDeviceDefinitionVersion"
                    },
                    "FunctionDefinitionVersionArn": {
                        "Ref": "TestFunctionDefinitionVersion"
                    },
                    "SubscriptionDefinitionVersionArn": {
                        "Ref": "TestSubscriptionDefinitionVersion"
                    },
                    "LoggerDefinitionVersionArn": {
                        "Ref": "TestLoggerDefinitionVersion"
                    },
                    "ResourceDefinitionVersionArn": {
                        "Ref": "TestResourceDefinitionVersion"
                    }
                },
                "Tags": {
                    "KeyName0": "value",
                    "KeyName1": "value",
                    "KeyName2": "value"
                }
```

```
            }
        }
    },
    "Outputs": {
        "CommandToDeployGroup": {
            "Value": {
                "Fn::Join": [
                    " ",
                    [
                        "groupVersion=$(cut -d'/' -f6 <<<",
                        {
                            "Fn::GetAtt": [
                                "TestGroup",
                                "LatestVersionArn"
                            ]
                        },
                        ");",
                        "aws --region",
                        {
                            "Ref": "AWS::Region"
                        },
                        "greengrass create-deployment --group-id",
                        {
                            "Ref": "TestGroup"
                        },
                        "--deployment-type NewDeployment --group-version-id",
                        "$groupVersion"
                    ]
                ]
            }
        }
    }
}
```

YAML

```
AWSTemplateFormatVersion: 2010-09-09
Description: >-
  AWS IoT Greengrass example template that creates a group version with a core,
  device, function, logger, subscription, and resources.
Parameters:
  CoreCertificateArn:
    Type: String
```

```
  DeviceCertificateArn:
    Type: String
  LambdaVersionArn:
    Type: String
Resources:
  TestCore1:
    Type: 'AWS::IoT::Thing'
    Properties:
      ThingName: TestCore1
  TestCoreDefinition:
    Type: 'AWS::Greengrass::CoreDefinition'
    Properties:
      Name: DemoTestCoreDefinition
  TestCoreDefinitionVersion:
    Type: 'AWS::Greengrass::CoreDefinitionVersion'
    Properties:
      CoreDefinitionId: !Ref TestCoreDefinition
      Cores:
        - Id: TestCore1
          CertificateArn: !Ref CoreCertificateArn
          SyncShadow: 'false'
          ThingArn: !Join
            - ':'
            - - 'arn:aws:iot'
              - !Ref 'AWS::Region'
              - !Ref 'AWS::AccountId'
              - thing/TestCore1
  TestClientDevice1:
    Type: 'AWS::IoT::Thing'
    Properties:
      ThingName: TestClientDevice1
  TestDeviceDefinition:
    Type: 'AWS::Greengrass::DeviceDefinition'
    Properties:
      Name: DemoTestDeviceDefinition
  TestDeviceDefinitionVersion:
    Type: 'AWS::Greengrass::DeviceDefinitionVersion'
    Properties:
      DeviceDefinitionId: !GetAtt
        - TestDeviceDefinition
        - Id
      Devices:
        - Id: TestClientDevice1
          CertificateArn: !Ref DeviceCertificateArn
```

```
          SyncShadow: 'true'
          ThingArn: !Join
            - ':'
            - - 'arn:aws:iot'
              - !Ref 'AWS::Region'
              - !Ref 'AWS::AccountId'
              - thing/TestClientDevice1
  TestFunctionDefinition:
    Type: 'AWS::Greengrass::FunctionDefinition'
    Properties:
      Name: DemoTestFunctionDefinition
  TestFunctionDefinitionVersion:
    Type: 'AWS::Greengrass::FunctionDefinitionVersion'
    Properties:
      FunctionDefinitionId: !GetAtt
        - TestFunctionDefinition
        - Id
      DefaultConfig:
        Execution:
          IsolationMode: GreengrassContainer
      Functions:
        - Id: TestLambda1
          FunctionArn: !Ref LambdaVersionArn
          FunctionConfiguration:
            Pinned: 'true'
            Executable: run.exe
            ExecArgs: argument1
            MemorySize: '512'
            Timeout: '2000'
            EncodingType: binary
            Environment:
              Variables:
                variable1: value1
              ResourceAccessPolicies:
                - ResourceId: ResourceId1
                  Permission: ro
                - ResourceId: ResourceId2
                  Permission: rw
              AccessSysfs: 'false'
              Execution:
                IsolationMode: GreengrassContainer
                RunAs:
                  Uid: '1'
                  Gid: '10'
```

```
  TestLoggerDefinition:
    Type: 'AWS::Greengrass::LoggerDefinition'
    Properties:
      Name: DemoTestLoggerDefinition
  TestLoggerDefinitionVersion:
    Type: 'AWS::Greengrass::LoggerDefinitionVersion'
    Properties:
      LoggerDefinitionId: !Ref TestLoggerDefinition
      Loggers:
        - Id: TestLogger1
          Type: AWSCloudWatch
          Component: GreengrassSystem
          Level: INFO
  TestResourceDefinition:
    Type: 'AWS::Greengrass::ResourceDefinition'
    Properties:
      Name: DemoTestResourceDefinition
  TestResourceDefinitionVersion:
    Type: 'AWS::Greengrass::ResourceDefinitionVersion'
    Properties:
      ResourceDefinitionId: !Ref TestResourceDefinition
      Resources:
        - Id: ResourceId1
          Name: LocalDeviceResource
          ResourceDataContainer:
            LocalDeviceResourceData:
              SourcePath: /dev/TestSourcePath1
              GroupOwnerSetting:
                AutoAddGroupOwner: 'false'
                GroupOwner: TestOwner
        - Id: ResourceId2
          Name: LocalVolumeResourceData
          ResourceDataContainer:
            LocalVolumeResourceData:
              SourcePath: /dev/TestSourcePath2
              DestinationPath: /volumes/TestDestinationPath2
              GroupOwnerSetting:
                AutoAddGroupOwner: 'false'
                GroupOwner: TestOwner
  TestSubscriptionDefinition:
    Type: 'AWS::Greengrass::SubscriptionDefinition'
    Properties:
      Name: DemoTestSubscriptionDefinition
  TestSubscriptionDefinitionVersion:
```

```
    Type: 'AWS::Greengrass::SubscriptionDefinitionVersion'
    Properties:
      SubscriptionDefinitionId: !Ref TestSubscriptionDefinition
      Subscriptions:
        - Id: TestSubscription1
          Source: !Join
            - ':'
            - - 'arn:aws:iot'
              - !Ref 'AWS::Region'
              - !Ref 'AWS::AccountId'
              - thing/TestClientDevice1
          Subject: TestSubjectUpdated
          Target: !Ref LambdaVersionArn
  TestGroup:
    Type: 'AWS::Greengrass::Group'
    Properties:
      Name: DemoTestGroupNewName
      RoleArn: !Join
        - ':'
        - - 'arn:aws:iam:'
          - !Ref 'AWS::AccountId'
          - role/TestUser
      InitialVersion:
        CoreDefinitionVersionArn: !Ref TestCoreDefinitionVersion
        DeviceDefinitionVersionArn: !Ref TestDeviceDefinitionVersion
        FunctionDefinitionVersionArn: !Ref TestFunctionDefinitionVersion
        SubscriptionDefinitionVersionArn: !Ref TestSubscriptionDefinitionVersion
        LoggerDefinitionVersionArn: !Ref TestLoggerDefinitionVersion
        ResourceDefinitionVersionArn: !Ref TestResourceDefinitionVersion
      Tags:
        KeyName0: value
        KeyName1: value
        KeyName2: value
Outputs:
  CommandToDeployGroup:
    Value: !Join
      - ' '
      - - groupVersion=$(cut -d'/' -f6 <<<
        - !GetAtt
          - TestGroup
          - LatestVersionArn
        - );
        - aws --region
        - !Ref 'AWS::Region'
```

```
        - greengrass create-deployment --group-id
        - !Ref TestGroup
        - '--deployment-type NewDeployment --group-version-id'
        - $groupVersion
```

Compatível Região da AWS s

Atualmente, você pode criar e gerenciar AWS IoT Greengrass recursos somente nas seguintes opções Região da AWS:

- Leste dos EUA (Ohio)
- Leste dos EUA (N. da Virgínia)
- Oeste dos EUA (Oregon)
- Ásia-Pacífico (Mumbai)
- Ásia-Pacífico (Seul)
- Ásia-Pacífico (Singapura)
- Ásia-Pacífico (Sydney)
- Ásia-Pacífico (Tóquio)
- China (Pequim)
- Europa (Frankfurt)
- Europa (Irlanda)
- Europa (Londres)
- AWS GovCloud (US-West)

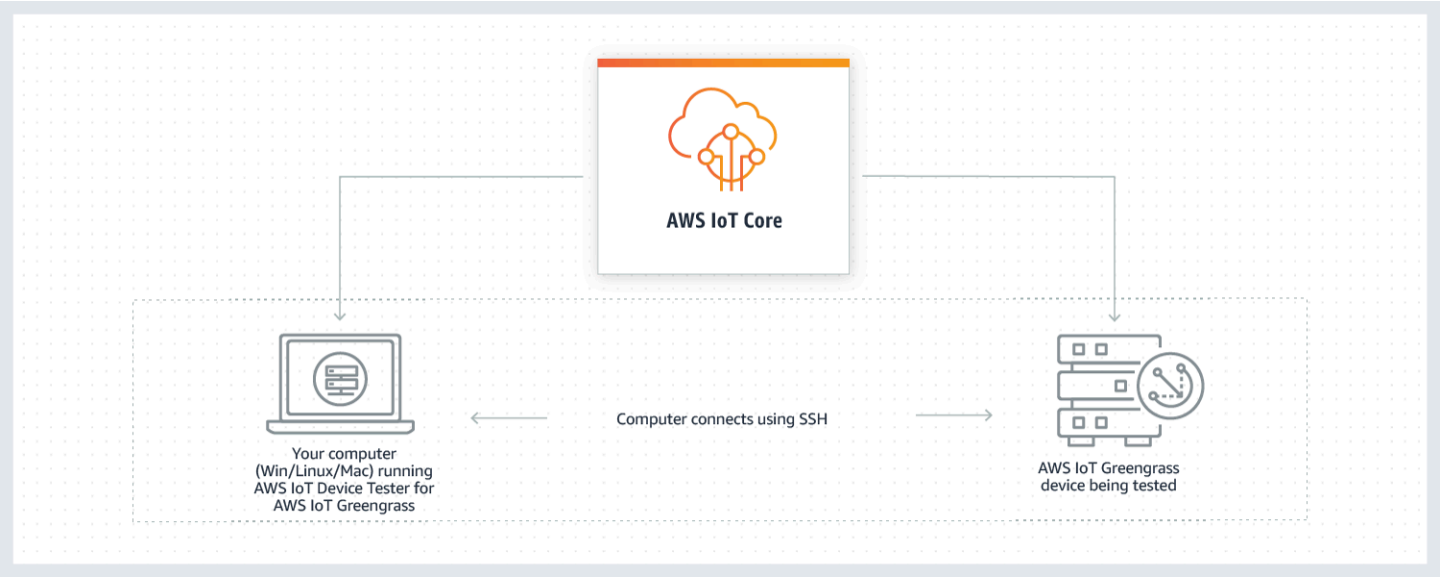
Utilizar AWS IoT Testador de dispositivos para AWS IoT Greengrass V1

AWS IoT O Device Tester (IDT) é uma estrutura de teste disponível para download que permite validar dispositivos de IoT. Como AWS IoT Greengrass Version 1 foi transferido para o modo de manutenção, o IDT for AWS IoT Greengrass V1 não gera mais relatórios de qualificação assinados. Você não poderá mais qualificar novos AWS IoT Greengrass V1 dispositivos para serem listados no Catálogo de AWS Partner Dispositivos por meio do Programa de Qualificação de AWS Dispositivos. No entanto, você pode continuar usando o IDT AWS IoT Greengrass V1 para testar seus dispositivos Greengrass V1. Recomendamos que você use o IDT para AWS IoT Greengrass V2 para qualificar e listar dispositivos Greengrass no AWS Partner Catálogo de 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 o IDT AWS IoT Greengrass 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 testes de ponta a ponta com. AWS IoT Core Por exemplo, ele verifica que o dispositivo pode enviar e receber mensagens MQTT e processá-las corretamente.



AWS IoT O Device Tester for AWS IoT Greengrass organiza testes usando os conceitos de suítes de teste 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 teste é o conjunto de testes individuais relacionados a um atributo, como implantações de grupo do Greengrass e mensagens MQTT.

Para obter mais informações, consulte Use o IDT para executar o AWS IoT Greengrass suíte de qualificação.

Conjuntos de teste personalizados

A partir do IDT v4.0.0, o IDT for AWS IoT Greengrass combina uma configuração padronizada e um formato de resultado com um ambiente de suíte de testes que permite desenvolver suítes de teste 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 obter mais informações, consulte Usar o IDT para desenvolver e executar os próprios pacotes de testes.

Versões compatíveis do AWS IoT Testador de dispositivos para AWS IoT Greengrass V1

Como AWS IoT Greengrass Version 1 foi movido para o modo de manutenção, o IDT AWS IoT Greengrass V1 não gera mais relatórios de qualificação assinados. Recomendamos usar o IDT para AWS IoT Greengrass V2.

Para obter informações sobre o IDT para AWS IoT Greengrass V2, consulte Usando o AWS IoT Device Tester for AWS IoT Greengrass V2 no Guia do AWS IoT Greengrass V2 desenvolvedor.

> **Note**
>
> Você recebe uma notificação ao iniciar um teste se o IDT for não AWS IoT Greengrass for compatível com a versão AWS IoT Greengrass que você está usando.

Ao fazer download do software, você concorda com o AWS IoT Acordo de licença do Device Tester.

Versões IDT não suportadas para for AWS IoT Greengrass

Este tópico lista as versões não suportadas do IDT para. AWS IoT Greengrass Versões não compatíveis não recebem correções de bugs ou atualizações. Para obter mais informações, consulte the section called “Política de suporte para AWS IoT Testador de dispositivos para AWS IoT Greengrass V1”.

IDT v4.4.1 para as AWS IoT Greengrass versões v1.11.6, v1.10.5

Notas de release:

- Permite validar e qualificar dispositivos que executam o software AWS IoT Greengrass principal v1.11.6 e v1.10.5.
- Contém pequenas correções de erros.

Versão do conjunto de testes:

`GGQ_1.3.1`

- Lançado em 20/12/2021

IDT v4.1.0 para as AWS IoT Greengrass versões v1.11.4, v1.10.4

Notas de release:

- Permite validar e qualificar dispositivos que executam o software AWS IoT Greengrass principal v1.11.4 e v1.10.4.
- Correção de um problema que fazia com que os registros exibidos durante uma execução de teste usassem tags redundantes.

Versão do conjunto de testes:

`GGQ_1.3.0`

- Lançado em 23/06/2021
- Adiciona novas tentativas para chamadas de API para Lambda, IAM AWS STS e para melhorar o tratamento de problemas de limitação ou de servidor.
- Adição de suporte para o Python 3.8 aos casos de teste do ML e Docker.

IDT v4.0.2 para as AWS IoT Greengrass versões v1.11.1, v1.11.0, v1.10.3

Notas de release:

- Correção de um problema que fazia com que o IDT mascarasse erros de integração de segurança de hardware (HSI).
- Permite que você desenvolva e execute suas suítes de teste personalizadas usando o AWS IoT Device Tester for AWS IoT Greengrass. Para obter 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, se uma mensagem de aviso de segurança for exibida, talvez seja necessário conceder uma exceção de segurança para o IDT. Para obter mais informações, consulte Exceção de segurança no macOS.

> **Note**
>
> AWS IoT Greengrass não fornece um Dockerfile ou uma imagem Docker para a versão 1.11.1 do software principal. AWS IoT Greengrass Para testar a qualificação do Docker em seu dispositivo, use uma versão anterior do software Core AWS IoT Greengrass .

IDT v3.2.0 para as AWS IoT Greengrass versões v1.11.0, v1.10.1, v1.10.0

Notas de release:

- Por padrão, o IDT só executa os testes necessários para qualificação. Para se qualificar para atributos adicionais, você pode modificar o arquivo `device.json`.
- Adição de um número de porta em `device.json` que você pode configurar para conexões SSH.
- O Docker só oferece suporte ao gerenciador de fluxo e ao machine learning (ML) sem conteinerização. Os contêineres, o Docker e a integração de segurança de hardware (HSI) não estão disponíveis para os dispositivos Docker.
- Nós fundimos `device-ml.json` e `device-hsm.json` no `device.json`.

IDT v3.1.3 para AWS IoT Greengrass versões: v1.10.x, v1.9.x, v1.8.x

Notas de release:

- Foi adicionado suporte para qualificação de recursos de ML para AWS IoT Greengrass v1.10.x e v1.9.x. Agora você pode usar o IDT para validar que seus dispositivos podem realizar inferência de ML localmente com modelos armazenados e treinados na nuvem.

- `--stop-on-first-failure` adicionado para o comando `run-suite`. Você pode usar essa opção para configurar o IDT a fim de interromper a execução na primeira falha. Recomendamos usar essa opção durante o estágio de depuração no nível de grupos de teste.
- Adição de uma verificação de desvio de relógio para testes MQTT a fim de garantir que o dispositivo em teste use a hora correta do sistema. O tempo usado deve estar dentro de um intervalo de tempo aceitável.
- `--update-idt` adicionado para o comando `run-suite`. Você pode usar essa opção para definir a resposta do prompt para atualização do IDT.
- `--update-managed-policy` adicionado para o comando `run-suite`. Você pode usar essa opção para definir a resposta do prompt para atualização da política gerenciada.
- Adição de uma correção de bug para atualizações automáticas das versões do pacote de testes do IDT. A correção garante que o IDT possa executar os conjuntos de testes mais recentes disponíveis para sua versão do AWS IoT Greengrass .

IDT v3.0.1 para AWS IoT Greengrass

Notas de release:

- Foi adicionado suporte para AWS IoT Greengrass v1.10.1.
- Atualizações automáticas das versões do conjunto de testes do IDT. O IDT pode baixar as suítes de teste mais recentes que estão disponíveis para sua AWS IoT Greengrass versão. Com este atributo:
  - Os conjuntos de testes são versionados usando um formato *`major`*`.`*`minor`*`.`*`patch`*. A versão inicial do conjunto de testes é `GGQ_1.0.0`.
  - Você pode baixar novos conjuntos de testes interativamente na interface de linha de comando ou definir o sinalizador `upgrade-test-suite` ao iniciar o IDT.

  Para obter mais informações, consulte the section called “Versões do pacote de testes”.
- Adição do `list-supported-products`. Você pode usar esse comando para listar o AWS IoT Greengrass e as versões do conjunto de testes compatíveis com a versão instalada do IDT.
- Adição do `list-test-cases`. Você pode usar esse comando para listar os casos de teste que estão disponíveis em um grupo de teste.
- `test-id` adicionado para o comando `run-suite`. Você pode usar essa opção para executar casos de teste individuais em um grupo de teste.

IDT v2.3.0 para AWS IoT Greengrass v1.10, v1.9.x e v1.8.x

Ao testar em um dispositivo físico, há suporte para AWS IoT Greengrass v1.10, v1.9.x e v1.8.x.

Ao testar em um contêiner Docker, as versões AWS IoT Greengrass v1.10 e v1.9.x são suportadas.

Notas de release:

- O suporte adicionado para the section called “Executar AWS IoT Greengrass em um contêiner Docker”. Agora você pode usar o IDT para qualificar e validar se seus dispositivos podem ser executados AWS IoT Greengrass em um contêiner Docker.
- Foi adicionada uma política AWS gerenciada (`AWSIoTDeviceTesterForGreengrassFullAccess`) que define as permissões necessárias para executar o AWS IoT Device Tester. Se novas versões exigirem permissões adicionais, AWS adicione-as a essa política gerenciada para que você não precise atualizar suas permissões do IAM.
- Verificações introduzidas para validar se o ambiente (por exemplo, conectividade do dispositivo e conectividade com a Internet) está configurado corretamente antes de executar os casos de teste.
- Melhoria do verificador de dependência do Greengrass no IDT para torná-lo mais flexível durante a verificação de libc em dispositivos.

IDT v2.2.0 para AWS IoT Greengrass v1.10, v1.9.x e v1.8.x

Notas de release:

- Foi adicionado suporte para AWS IoT Greengrass v1.10.
- Suporte adicionado para o conector de implantação do aplicativo Docker do Greengrass .
- Foi adicionado suporte para o gerenciador de AWS IoT Greengrass streams.
- Foi adicionado suporte para AWS IoT Greengrass na região da China (Pequim).

IDT v2.1.0 para AWS IoT Greengrass v1.9.x, v1.8.x e v1.7.x

Notas de release:

- Foi adicionado suporte para AWS IoT Greengrass v1.9.4.
- Foi adicionado suporte para Linux-ARMv6l dispositivos.

IDT v2.0.0 para AWS IoT Greengrass v1.9.3, v1.9.2, v.1.9.1, v1.9.0, v1.8.4, v1.8.3 e v1.8.2

Notas de release:

- Remoção da dependência do Python para o dispositivo em teste.
- O tempo de execução do pacote de testes foi reduzido em mais de 50%, agilizando o processo de qualificação.
- Tamanho do executável reduzido em mais de 50%, agilizando o download e a instalação.
- Aprimoramento do suporte para multiplicador de tempo limite para todos os casos de teste.
- Mensagens de pós-diagnóstico aprimoradas para agilizar a solução de problemas de erros.
- Atualização do modelo de política de permissões necessário para executar o IDT.
- Foi adicionado suporte para AWS IoT Greengrass v1.9.3.

IDT v1.3.3 para v1.9.2, v1.9.1, AWS IoT Greengrass v1.9.0, v1.8.3 e v1.8.2

Notas de release:

- Suporte para Greengrass v1.9.2 e v1.8.3 adicionado.
- Foi adicionado suporte para o Greengrass OpenWrt.
- Adição de login ao dispositivo com nome de usuário e senha SSH.
- Foi adicionada uma correção de bug de teste nativa para a OpenWrt-ARMv7l plataforma.

IDT v1.2 para v1.8.1 AWS IoT Greengrass

Notas de release:

- Adição de um multiplicador de tempo limite configurável para abordar e solucionar problemas de tempo limite (por exemplo, conexões de largura de banda baixa).

IDT v1.1 para v1.8.0 AWS IoT Greengrass

Notas de release:

- Foi adicionado suporte para AWS IoT Greengrass Hardware Security Integration (HSI).
- Foi adicionado suporte para AWS IoT Greengrass contêiner e sem contêiner.

- Foi adicionada a criação automatizada de funções de AWS IoT Greengrass serviço.
- Melhoria na limpeza de recursos de teste.
- Adição do relatório de resumo de execução de teste.

IDT v1.1 para v1.7.1 AWS IoT Greengrass

Notas de release:

- Foi adicionado suporte para AWS IoT Greengrass Hardware Security Integration (HSI).
- Foi adicionado suporte para AWS IoT Greengrass contêiner e sem contêiner.
- Foi adicionada a criação automatizada de funções de AWS IoT Greengrass serviço.
- Melhoria na limpeza de recursos de teste.
- Adição do relatório de resumo de execução de teste.

IDT v1.0 para v1.6.1 AWS IoT Greengrass

Notas de release:

- Foi adicionada uma correção de bug de teste OTA para compatibilidade de AWS IoT Greengrass versões futuras.

> **Note**
>
> Se você estiver usando o IDT v1.0 para AWS IoT Greengrass v1.6.1, deverá criar uma função de serviço do Greengrass. Em versões posteriores, o IDT cria o perfil de serviço para você.

Use o IDT para executar o AWS IoT Greengrass suíte de qualificação

Você pode usar o AWS IoT Device Tester (IDT) AWS IoT Greengrass para verificar se o software AWS IoT Greengrass Core é executado em seu hardware e pode se comunicar com o. Nuvem AWS Ele também executa testes de ponta a ponta com o AWS IoT Core. Por exemplo, ele verifica que o dispositivo pode enviar e receber mensagens MQTT e processá-las corretamente.

Como AWS IoT Greengrass Version 1 foi movido para o modo de manutenção, o IDT AWS IoT Greengrass V1 não gera mais relatórios de qualificação assinados. Se você quiser adicionar seu hardware ao Catálogo de AWS Partner dispositivos, execute o pacote de AWS IoT Greengrass V2 qualificação para gerar relatórios de teste para os quais você possa enviar AWS IoT. Para obter mais informações, consulte Programa de qualificação de dispositivos do AWS e Versões compatíveis do IDT para AWS IoT Greengrass V2.

Além de testar dispositivos, o IDT for AWS IoT Greengrass cria recursos (por exemplo, AWS IoT coisas, AWS IoT Greengrass grupos, funções Lambda e assim por diante) em Conta da AWS você para facilitar o processo de qualificação.

Para criar esses recursos, o IDT for AWS IoT Greengrass 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 AWS IoT Greengrass para executar o conjunto de AWS IoT Greengrass qualificação, o IDT executa as seguintes etapas:

1. Carrega e valida o dispositivo e as configuração de credencial.
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 o dispositivo passou nos testes necessários para a qualificação.

Versões do pacote de testes

O IDT for AWS IoT Greengrass organiza 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 teste é o conjunto de testes individuais relacionados a um atributo, como implantações de grupo do Greengrass e mensagens MQTT.

Começando com o IDT v3.0.0, os conjuntos de testes são versionados usando um formato *`major`*`.`*`minor`*`.`*`patch`*, por exemplo `GGQ_1.0.0`. Quando você faz download do IDT, o pacote inclui a versão mais recente do conjunto de testes.

> **Important**
>
> O IDT é compatível com as três versões mais recentes do conjunto de testes para qualificação do dispositivo. Para obter mais informações, consulte the section called “Política de suporte para AWS IoT Testador de dispositivos para AWS IoT Greengrass V1”.
> 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. 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.

Atualizações para definições de configuração do IDT

Novos testes podem apresentar novas definições de configuração do IDT.

- Se as configurações forem opcionais, o IDT continuará executando os testes.
- Se as configurações forem necessárias, o IDT notificará você e interromperá a execução. Depois de definir as configurações, reinicie a execução de teste.

  As definições de configuração estão localizadas na pasta *<device-tester-extract-location>*`/configs`. Para obter mais informações, consulte the section called “Defina as configurações do IDT”.

Se uma versão atualizada do conjunto de testes adicionar definições de configuração, o IDT criará uma cópia do arquivo de configuração original em *<device-tester-extract-location>*`/configs`.

Descrições dos grupos de testes

IDT v2.0.0 and later

Grupos de teste necessários para a qualificação de núcleo

Esses grupos de teste são necessários para qualificar seu AWS IoT Greengrass dispositivo para o Catálogo de AWS Partner dispositivos.

AWS IoT Greengrass Dependências principais

Valida se seu dispositivo atende a todos os requisitos de software e hardware do software AWS IoT Greengrass Core.

O caso de teste `Software Packages Dependencies` nesse grupo de teste não é aplicável ao testar em um contêiner do Docker.

Implantação

Valida que as funções do Lambda podem ser implantadas no dispositivo.

MQTT

Verifica a funcionalidade do roteador de AWS IoT Greengrass mensagens verificando a comunicação local entre os dispositivos principais e clientes do Greengrass, que são dispositivos IoT locais.

Over-the-Air (OTA)

Valida que seu dispositivo pode realizar com êxito uma atualização OTA do software AWS IoT Greengrass Core.

Esse grupo de teste não é aplicável ao testar em um contêiner do Docker.

Versão

Verifica se a versão AWS IoT Greengrass fornecida é compatível com a versão do AWS IoT Device Tester que você está usando.

Grupos de testes opcionais

Esses grupos de teste são opcionais. Se você optar por se qualificar para testes opcionais, seu dispositivo será listado com recursos adicionais no Catálogo de AWS Partner dispositivos.

Dependências de contêiner

Verifica se o dispositivo atende a todos os requisitos de software e hardware para executar as funções do Lambda no modo de contêiner em um núcleo do Greengrass.

Esse grupo de teste não é aplicável ao testar em um contêiner do Docker.

Contêiner de implantação

Valida que as funções do Lambda podem ser implantadas no dispositivo e executadas no modo de contêiner em um núcleo do Greengrass.

Esse grupo de teste não é aplicável ao testar em um contêiner do Docker.

Dependências do Docker (compatíveis com IDT v2.2.0 e versões posteriores)

Valida que o dispositivo atende a todas as dependências técnicas necessárias para usar o conector de implantação do aplicativo Docker do Greengrass a fim de executar contêineres.

Esse grupo de teste não é aplicável ao testar em um contêiner do Docker.

Integração de segurança de hardware (HSI)

Verifica se a biblioteca compartilhada HSI fornecida pode interagir com o módulo de segurança de hardware (HSM) e implementa as APIs PKCS#11 necessárias corretamente. A biblioteca do HSM e compartilhada deve assinar uma CSR, executar operações TLS e fornecer os tamanhos de chaves e o algoritmo de chave pública corretos.

Dependências do gerenciador de fluxo (compatíveis com IDT v2.2.0 e versões posteriores)

Valida se o dispositivo atende a todas as dependências técnicas necessárias para executar o gerenciador de AWS IoT Greengrass streams.

Dependências de Machine Learning (compatíveis com a versão 3.1.0 e posterior do IDT)

Valida que o dispositivo atende a todas as dependências técnicas necessárias para executar a inferência de ML localmente.

Testes de inferência de Machine Learning (compatíveis com a versão 3.1.0 e posterior do IDT)

Valida que a inferência de ML pode ser realizada no dispositivo em teste determinado. Para obter mais informações, consulte the section called “Opcional: configurar o dispositivo para qualificação de ML”.

Testes de contêiner de inferência de Machine Learning (compatíveis com a versão 3.1.0 e posterior do IDT)

Valida que a inferência de ML pode ser realizada no dispositivo em teste determinado e executada em modo de contêiner em um núcleo do Greengrass. Para obter mais informações, consulte the section called “Opcional: configurar o dispositivo para qualificação de ML”.

IDT v1.3.3 and earlier

Grupos de teste necessários para a qualificação de núcleo

Esses testes são necessários para qualificar seu AWS IoT Greengrass dispositivo para o Catálogo de AWS Partner dispositivos.

AWS IoT Greengrass Dependências principais

Valida se seu dispositivo atende a todos os requisitos de software e hardware do software AWS IoT Greengrass Core.

Combinação (interação de segurança de dispositivo)

Verifica a funcionalidade do gerenciador de certificados do dispositivo e a detecção de IP no dispositivo de núcleo do Greengrass alterando as informações de conectividade no grupo do Greengrass na nuvem. O grupo de teste alterna o certificado do AWS IoT Greengrass servidor e verifica se ele AWS IoT Greengrass permite conexões.

Implantação (necessária para IDT v1.2 e anterior)

Valida que as funções do Lambda podem ser implantadas no dispositivo.

Device Certificate Manager (DCM)

Verifica se o gerenciador de certificados do AWS IoT Greengrass dispositivo pode gerar um certificado de servidor na inicialização e alternar os certificados se eles estiverem prestes a expirar.

Detecção de IP (IPD)

Verifica se as informações de conectividade do núcleo estão atualizadas quando há alterações de endereço IP em um dispositivo de núcleo do Greengrass. Para obter mais informações, consulte Ativar detecção automática de IP.

Registro em log

Verifica se o serviço de AWS IoT Greengrass registro pode gravar em um arquivo de log usando uma função Lambda do usuário escrita em Python.

MQTT

Verifica a AWS IoT Greengrass funcionalidade do roteador de mensagens enviando mensagens sobre um tópico que é roteado para duas funções Lambda.

Nativo

Verifica se é AWS IoT Greengrass possível executar funções Lambda nativas (compiladas).

Over-the-Air (OTA)

Valida que seu dispositivo pode realizar com êxito uma atualização OTA do software AWS IoT Greengrass Core.

Penetração

Valida que o software AWS IoT Greengrass Core falhará ao iniciar se a proteção de link/soft link físico e o seccomp não estiverem habilitados. Ele também é usado para verificar outros atributos relacionados à segurança.

Shadow

Verifica a funcionalidade de sombra local e de sincronização de nuvem de sombra.

Spooler

Valida se as mensagens MQTT são colocadas em fila com a configuração padrão de spooler.

Token Exchange Service (TES)

Verifica se AWS IoT Greengrass pode trocar seu certificado principal por AWS credenciais válidas.

Versão

Verifica se a versão AWS IoT Greengrass fornecida é compatível com a versão do AWS IoT Device Tester que você está usando.

Grupos de testes opcionais

Estes testes são opcionais. Se você optar por se qualificar para testes opcionais, seu dispositivo será listado com recursos adicionais no Catálogo de AWS Partner dispositivos.

Dependências de contêiner

Verifica se o dispositivo atende a todas as dependências necessárias para executar funções do Lambda no modo de contêiner.

Integração de segurança de hardware (HSI)

Verifica se a biblioteca compartilhada HSI fornecida pode interagir com o módulo de segurança de hardware (HSM) e implementa as APIs PKCS#11 necessárias corretamente. A biblioteca do HSM e compartilhada deve assinar uma CSR, executar operações TLS e fornecer os tamanhos de chaves e o algoritmo de chave pública corretos.

Acesso aos recursos locais

Verifica o recurso de acesso a recursos locais (LRA) AWS IoT Greengrass fornecendo acesso a arquivos e diretórios locais de propriedade de vários usuários e grupos Linux para funções Lambda em contêineres por meio de APIs do LRA. AWS IoT Greengrass As funções do Lambdas devem ter acesso permitido ou negado a recursos locais com base na configuração do acesso aos recurso locais.

Rede

Verifica se as conexões de soquete podem ser estabelecidas a partir de uma função do Lambda. Essas conexões de soquete devem ser permitidas ou negadas com base na configuração do núcleo do Greengrass.

Pré-requisitos para executar o AWS IoT Greengrass suíte de qualificação

Esta seção descreve os pré-requisitos para usar o AWS IoT Device Tester (IDT) AWS IoT Greengrass para executar o pacote de qualificação. AWS IoT Greengrass

Baixe a versão mais recente do AWS IoT Testador de dispositivos para AWS IoT Greengrass

Baixe a versão mais recente do IDT e extraia o software em um local no seu sistema de arquivos onde você tenha permissões de leitura e 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.

Crie e configure um Conta da AWS

Antes de usar o IDT para AWS IoT Greengrass, você deve executar as seguintes etapas:

1. Crie 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 coisas, grupos do Greengrass e funções do Lambda, em seu nome.

Para criar esses recursos, o IDT for AWS IoT Greengrass 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 gratuito da Amazon Web Services, é necessário fornecer um cartão de crédito ao se cadastrar em uma conta da 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 2: Configurar permissões para o IDT

Nesta etapa, configure as permissões que o IDT for AWS IoT Greengrass usa para executar testes e coletar dados de uso do IDT. Você pode usar o Console de gerenciamento da AWS or 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 IDT, vá para the section called “Configure seu dispositivo para executar testes de IDT” ou the section called “(Opcional): configurar o contêiner do Docker”.

- 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.

      JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ManageRolePoliciesForIDTGreengrass",
            "Effect": "Allow",
            "Action": [
                "iam:DetachRolePolicy",
                "iam:AttachRolePolicy"
            ],
            "Resource": [
                "arn:aws:iam::*:role/idt-*",
                "arn:aws:iam::*:role/GreengrassServiceRole"
            ],
            "Condition": {
                "ArnEquals": {
                    "iam:PolicyARN": [
                        "arn:aws:iam::aws:policy/service-role/
AWSGreengrassResourceAccessRolePolicy",
                        "arn:aws:iam::aws:policy/service-role/
GreengrassOTAUpdateArtifactAccess",
                        "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole"
                    ]
                }
            }
        },
        {
```

```
            "Sid": "ManageRolesForIDTGreengrass",
            "Effect": "Allow",
            "Action": [
                "iam:CreateRole",
                "iam:DeleteRole",
                "iam:PassRole",
                "iam:GetRole"
            ],
            "Resource": [
              "arn:aws:iam::123456789012:role/idt-*",
              "arn:aws:iam::123456789012:role/GreengrassServiceRole"
            ]
        }
    ]
}
```

> **Important**
>
> A política a seguir concede permissão para criar e gerenciar funções exigidas pelo IDT para o AWS IoT Greengrass. Isso inclui permissões para anexar as seguintes políticas AWS gerenciadas:
>
> - AWSGreengrassResourceAccessRolePolicy
> - GreengrassOTAUpdateArtifactAccess
> - AWSLambdaBasicExecutionRole

c. Escolha Próximo: tags.

d. Selecione Next: Review (Próximo: revisar).

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 Definir permissões, selecione Anexar políticas existentes diretamente.

ii. Procure a IDTGreengrassIAMPermissionspolítica que você criou na etapa anterior. Marque a caixa de seleção.

iii. Pesquise a AWSIoTDeviceTesterForGreengrassFullAccesspolítica. Marque a caixa de seleção.

> **Note**
>
> AWSIoTDeviceTesterForGreengrassFullAccessÉ uma política AWS gerenciada que define as permissões que o IDT exige para criar e acessar AWS recursos usados para testes. Para obter mais informações, consulte the section called "AWS política gerenciada para IDT".

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, escolha Download.csve 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.

Para 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. Se você já configurou permissões no console, vá para the section called "Configure seu dispositivo para executar testes de IDT" ou the section called "(Opcional): configurar o contêiner do Docker".

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 .

   Linux, macOS, or Unix

```
aws iam create-policy --policy-name IDTGreengrassIAMPermissions --policy-
document '{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ManageRolePoliciesForIDTGreengrass",
            "Effect": "Allow",
            "Action": [
                "iam:DetachRolePolicy",
                "iam:AttachRolePolicy"
            ],
            "Resource": [
                "arn:aws:iam::*:role/idt-*",
                "arn:aws:iam::*:role/GreengrassServiceRole"
            ],
            "Condition": {
                "ArnEquals": {
                    "iam:PolicyARN": [
                        "arn:aws:iam::aws:policy/service-role/
AWSGreengrassResourceAccessRolePolicy",
                        "arn:aws:iam::aws:policy/service-role/
GreengrassOTAUpdateArtifactAccess",
                        "arn:aws:iam::aws:policy/service-role/
AWSLambdaBasicExecutionRole"
                    ]
                }
            }
        },
        {
            "Sid": "ManageRolesForIDTGreengrass",
            "Effect": "Allow",
            "Action": [
```

```
                "iam:CreateRole",
                "iam:DeleteRole",
                "iam:PassRole",
                "iam:GetRole"
            ],
            "Resource": [
              "arn:aws:iam::123456789012:role/idt-*",
              "arn:aws:iam::123456789012:role/GreengrassServiceRole"
            ]
        }
    ]
}'
```

Windows command prompt

```
aws iam create-policy --policy-name IDTGreengrassIAMPermissions --
policy-document '{\"Version\": \"2012-10-17\",        \"Statement\":
 [{\"Sid\": \"ManageRolePoliciesForIDTGreengrass\",\"Effect\": \"Allow
\",\"Action\": [\"iam:DetachRolePolicy\", \"iam:AttachRolePolicy\"],
 \"Resource\": [\"arn:aws:iam::*:role/idt-*\",\"arn:aws:iam::*:role/
GreengrassServiceRole\"],\"Condition\": {\"ArnEquals\": {\"iam:PolicyARN\":
 [\"arn:aws:iam::aws:policy/service-role/AWSGreengrassResourceAccessRolePolicy
\",\"arn:aws:iam::aws:policy/service-role/GreengrassOTAUpdateArtifactAccess
\",\"arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole\"]}}},
{\"Sid\": \"ManageRolesForIDTGreengrass\",\"Effect\": \"Allow\",\"Action\":
 [\"iam:CreateRole\",\"iam:DeleteRole\", \"iam:PassRole\", \"iam:GetRole
\"],\"Resource\": [\"arn:aws:iam::*:role/idt-*\",\"arn:aws:iam::*:role/
GreengrassServiceRole\"]}]}'
```

> **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.

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
```

c. Anexe a política `AWSIoTDeviceTesterForGreengrassFullAccess` ao seu usuário do IAM.

```
aws iam attach-user-policy --user-name IDTGreengrassUser --policy-arn
 arn:aws:iam::aws:policy/AWSIoTDeviceTesterForGreengrassFullAccess
```

> **Note**
>
> AWSIoTDeviceTesterForGreengrassFullAccessÉ uma política AWS gerenciada que define as permissões que o IDT exige para criar e acessar AWS recursos usados para testes. Para obter mais informações, consulte the section called “AWS política gerenciada para IDT”.

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 política gerenciada para AWS IoT Testador de dispositivos

A política AWSIoTDeviceTesterForGreengrassFullAccessgerenciada permite que a IDT execute operações e colete métricas de uso. Essa política concede as seguintes permissões de IDT:

- `iot-device-tester:CheckVersion`. Verifique se um conjunto de versões AWS IoT Greengrass, suíte de testes e IDT são compatíveis.
- `iot-device-tester:DownloadTestSuite`. Fazer download dos conjuntos de testes.
- `iot-device-tester:LatestIdt`. Obter informações sobre a versão mais recente do IDT disponível para download.

- `iot-device-tester:SendMetrics`. Publicar dados de uso que o IDT coleta sobre seus testes.
- `iot-device-tester:SupportedVersion`. Obtenha a lista AWS IoT Greengrass e as versões do pacote de testes compatíveis com o IDT. Essas informações são exibidas na janela da linha de comando.

Configure seu dispositivo para executar testes de IDT

Para configurar seu dispositivo, você deve instalar AWS IoT Greengrass dependências, configurar o software AWS IoT Greengrass Core, configurar seu computador host para acessar seu dispositivo e configurar as permissões de usuário em seu dispositivo.

Verificar AWS IoT Greengrass dependências no dispositivo em teste

Antes que o IDT for AWS IoT Greengrass possa testar seus dispositivos, verifique se você configurou seu dispositivo conforme descrito em Introdução ao AWS IoT Greengrass. Para obter mais informações sobre as plataformas compatíveis, consulte Plataformas compatíveis.

Configurar o AWS IoT Greengrass software

O IDT for AWS IoT Greengrass testa a compatibilidade de seu dispositivo com uma versão específica do AWS IoT Greengrass. O IDT oferece duas opções para testes AWS IoT Greengrass em seus dispositivos:

- Faça download e use uma versão do software de núcleo do AWS IoT Greengrass. O IDT instala o software para você.
- Use uma versão do software AWS IoT Greengrass Core já instalada em seu dispositivo.

> **Note**
>
> Cada versão do AWS IoT Greengrass tem uma versão IDT correspondente. Você deve baixar a versão do IDT que corresponde à versão que AWS IoT Greengrass você está usando.

As seções a seguir descrevem essas opções. Você só precisa fazer uma delas.

Opção 1: Faça o download do AWS IoT Greengrass Software básico e configuração AWS IoT Testador de dispositivos para usá-lo

Você pode baixar o software AWS IoT Greengrass principal na página de downloads do software AWS IoT Greengrass principal.

1. Localize a arquitetura correta e distribuição do Linux e, em seguida selecione Download (Fazer download).
2. Copie o arquivo tar.gz para *<device-tester-extract-location>*`/products/greengrass/ggc`.

> **Note**
>
> Não altere o nome do arquivo AWS IoT Greengrass tar.gz. Não coloque vários arquivos nesse diretório para o mesmo sistema operacional e arquitetura. Por exemplo, a presença dos arquivos `greengrass-linux-armv7l-1.7.1.tar.gz` e `greengrass-linux-armv7l-1.8.1.tar.gz` nesse diretório fará com que os testes falhem.

Opção 2: Usar uma instalação existente do AWS IoT Greengrass por AWS IoT Testador de dispositivos

Configure o IDT para testar o software AWS IoT Greengrass Core instalado em seu dispositivo adicionando o `greengrassLocation` atributo ao `device.json` arquivo na *<device-tester-extract-location>*`/configs` pasta. Por exemplo:

```
"greengrassLocation" : "<path-to-greengrass-on-device>"
```

Para obter mais informações sobre o arquivo `device.json`, consulte Configurar device.json.

Em dispositivos Linux, a localização padrão do software AWS IoT Greengrass Core é`/greengrass`.

> **Note**
>
> Seu dispositivo deve ter uma instalação do software AWS IoT Greengrass Core que não tenha sido iniciada.
> Certifique-se de ter adicionado o usuário `ggc_user` e `ggc_group` no seu dispositivo. Para obter mais informações, consulte Configuração de ambiente para o AWS IoT Greengrass.

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 obter mais informações, consulte Configurar device.json.

Você pode usar qualquer implementação SSL para criar uma chave SSH. As instruções a seguir mostram como usar SSH-KEYGENo 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 dispositivo em teste e *remote-device-ip* 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 como criar chaves SSH e copiar 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 obter 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 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 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 seu dispositivo para testar atributos opcionais

Os tópicos a seguir descrevem como configurar seus dispositivos de modo a executar testes IDT para atributos opcionais. Siga os passos de configuração a seguir apenas se você quiser testar estes recursos. Caso contrário, avance para the section called “Defina as configurações do IDT”.

Tópicos

- Opcional: configurar seu contêiner Docker para IDT para AWS IoT Greengrass
- Opcional: configurar o dispositivo para qualificação de ML

Opcional: configurar seu contêiner Docker para IDT para AWS IoT Greengrass

AWS IoT Greengrass fornece uma imagem do Docker e um Dockerfile que facilitam a execução do software AWS IoT Greengrass Core em um contêiner do Docker. Depois de configurar o AWS IoT Greengrass contêiner, você pode executar testes de IDT. No momento, somente arquiteturas x86_64 do Docker são compatíveis com a execução do IDT para o AWS IoT Greengrass.

Esse atributo exige o IDT v2.3.0 ou posterior.

O processo de configuração do contêiner Docker para executar testes de IDT depende se você usa a imagem do Docker ou o Dockerfile fornecido pelo. AWS IoT Greengrass

- Use a imagem do Docker. A imagem do Docker tem o software AWS IoT Greengrass principal e as dependências instalados.
- Use o arquivo do Docker. O Dockerfile contém o código-fonte que você pode usar para criar imagens de AWS IoT Greengrass contêiner personalizadas. A imagem pode ser modificada para funcionar com diferentes arquiteturas de plataforma ou para reduzir o tamanho da imagem.

> **Note**
>
> AWS IoT Greengrass não fornece Dockerfiles ou imagens Docker para a versão 1.11.1 AWS IoT Greengrass do software principal. Para executar testes de IDT em suas próprias imagens de contêiner personalizadas, sua imagem deve incluir as dependências definidas no Dockerfile fornecido pela. AWS IoT Greengrass

Os seguintes recursos não estão disponíveis quando você executa AWS IoT Greengrass em um contêiner Docker:

- Conectores executados no modo de contêiner do Greengrass. Para executar um conector em um contêiner do Docker, o conector deve ser executado no modo Sem contêiner. Para localizar conectores compatíveis com o modo Sem contêiner consulte the section called “Conectores do Greengrass fornecidos pela AWS”. Alguns desses conectores têm um parâmetro de modo de isolamento que você deve definir como Sem contêiner.
- Recursos de volume e dispositivo locais. Suas funções do Lambda definidas pelo usuário executadas no contêiner do Docker devem acessar dispositivos e volumes diretamente no núcleo.

Configure a imagem do Docker fornecida pelo AWS IoT Greengrass

Siga estas etapas para configurar a imagem do AWS IoT Greengrass Docker para executar testes de IDT.

Pré-requisitos

Antes de começar este tutorial, você deve fazer o seguinte.

- Você deve instalar os seguintes softwares e versões em seu computador host com base na versão AWS Command Line Interface (AWS CLI) que você escolher.

  AWS CLI version 2

  - Docker, versão 18.09 ou posterior. Versões anteriores também podem funcionar, mas recomendamos a versão 18.09 ou posterior.
  - AWS CLI versão 2.0.0 ou posterior.
    - Para instalar a AWS CLI versão 2, consulte Instalando a AWS CLI versão 2.
    - Para configurar o AWS CLI, consulte Configurando o. AWS CLI

> **Note**
>
> Para atualizar para uma AWS CLI versão posterior 2 em um computador Windows, você deve repetir o processo de instalação do MSI.

AWS CLI version 1

- Docker, versão 18.09 ou posterior. Versões anteriores também podem funcionar, mas recomendamos a versão 18.09 ou posterior.
- Python, versão 3.6 ou posterior.
- pip versão 18.1 ou posterior.
- AWS CLI versão 1.17.10 ou posterior
  - Para instalar a AWS CLI versão 1, consulte Instalando a AWS CLI versão 1.
  - Para configurar o AWS CLI, consulte Configurando o. AWS CLI
  - Para atualizar para a versão mais recente da AWS CLI versão 1, execute o comando a seguir.

```
pip install awscli --upgrade --user
```

> **Note**
>
> Se você usa a instalação MSI da AWS CLI versão 1 no Windows, esteja ciente do seguinte:
>
> - Se a instalação da AWS CLI versão 1 falhar ao instalar o botocore, tente usar a instalação do Python e do pip.
> - Para atualizar para uma AWS CLI versão posterior 1, você deve repetir o processo de instalação do MSI.

- Para acessar os recursos do Amazon Elastic Container Registry (Amazon ECR), você deve conceder a seguinte permissão.
  - O Amazon ECR exige que os usuários concedam a `ecr:GetAuthorizationToken` permissão por meio de uma política AWS Identity and Access Management (IAM) antes de poderem se autenticar em um registro e enviar ou extrair imagens de um repositório do Amazon ECR. Para obter mais informações, consulte Exemplos de políticas do repositório do Amazon ECR e

Acessando um repositório do Amazon ECR no Guia do usuário do Amazon Elastic Container Registry.

1. Faça download da imagem do Docker e configure o contêiner. Você pode fazer download da imagem pré-criada no Hub do Docker ou no Amazon Elastic Container Registry (Amazon ECR) e executá-la em plataformas do Windows, macOS e Linux (x86_64).

   Para fazer download da imagem do Docker do Amazon ECR, conclua todas as etapas em the section called “Obtenha a imagem do AWS IoT Greengrass contêiner do Amazon ECR”. Depois, retorne a este tópico para continuar a configuração.
2. Somente usuários do Linux: verifique se o usuário que executa o IDT tem permissão para executar comandos do Docker. Para obter mais informações, consulte Manage Docker as a non-root user na documentação do Docker.
3. Para executar o AWS IoT Greengrass contêiner, use o comando para seu sistema operacional:

   Linux

   ```
   docker run --rm --init -it -d --name aws-iot-greengrass \
   -p 8883:8883 \
   -v <host-path-to-kernel-config-file>:<container-path> \
   <image-repository>:<tag>
   ```

   - *<host-path-to-kernel-config-file>*Substitua pelo caminho para o arquivo de configuração do kernel no host e *<container-path>* pelo caminho em que o volume está montado no contêiner.

     O arquivo de configuração do kernel no host geralmente está localizado em `/proc/config.gz` ou em `/boot/config-`*<kernel-release-date>*. Você pode correr `uname -r` para encontrar o *<kernel-release-date>* valor.

     Exemplo: para montar o arquivo de configuração de `/boot/config-`*<kernel-release-date>*

     ```
     -v /boot/config-4.15.0-74-generic:/boot/config-4.15.0-74-generic \
     ```

     Exemplo: para montar o arquivo de configuração de `proc/config.gz`

```
-v /proc/config.gz:/proc/config.gz \
```

- Substitua*<image-repository>*: *<tag>* no comando pelo nome do repositório e pela tag da imagem de destino.

  Exemplo: Para apontar para a versão mais recente do software AWS IoT Greengrass Core

  ```
  216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
  ```

  Para obter a lista de imagens do AWS IoT Greengrass Docker, execute o comando a seguir.

  ```
  aws ecr list-images --region us-west-2 --registry-id 216483018798 --
  repository-name aws-iot-greengrass
  ```

macOS

```
docker run --rm --init -it -d --name aws-iot-greengrass \
-p 8883:8883 \
<image-repository>:<tag>
```

- Substitua*<image-repository>*: *<tag>* no comando pelo nome do repositório e pela tag da imagem de destino.

  Exemplo: Para apontar para a versão mais recente do software AWS IoT Greengrass Core

  ```
  216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
  ```

  Para obter a lista de imagens do AWS IoT Greengrass Docker, execute o seguinte comando:

  ```
  aws ecr list-images --region us-west-2 --registry-id 216483018798 --
  repository-name aws-iot-greengrass
  ```

Windows

```
docker run --rm --init -it -d --name aws-iot-greengrass \
-p 8883:8883 \
<image-repository>:<tag>
```

- Substitua*<image-repository>*: *<tag>* no comando pelo nome do repositório e pela tag da imagem de destino.

  Exemplo: Para apontar para a versão mais recente do software AWS IoT Greengrass Core

  ```
  216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
  ```

  Para obter a lista de imagens do AWS IoT Greengrass Docker, execute o seguinte comando:

  ```
  aws ecr list-images --region us-west-2 --registry-id 216483018798 --
  repository-name aws-iot-greengrass
  ```

> **Important**
>
> Ao testar com o IDT, não inclua o `--entrypoint /greengrass-entrypoint.sh \` argumento usado para executar a imagem para AWS IoT Greengrass uso geral.

4. Próxima etapa: configure suas AWS credenciais e seu `device.json` arquivo.

Configure o dockerfile fornecido pelo AWS IoT Greengrass

Siga estas etapas para configurar a imagem do Docker criada a partir do AWS IoT Greengrass Dockerfile para executar testes de IDT.

1. Em the section called “AWS IoT Greengrass Software Docker”, faça download do pacote do arquivo do Docker para o computador host e extraia-o.
2. Abra o `README.md`. As próximas três etapas se referem a seções desse arquivo.
3. Verifique se você atende aos requisitos na seção Pré-requisitos.

4. Somente usuários do Linux: conclua as etapas Habilitar as proteções symlink e hardlink e Habilitar o encaminhamento da rede IPv4.
5. Para criar a imagem do Docker, conclua todas as etapas na Etapa 1. Crie a imagem do AWS IoT Greengrass Docker. Depois, retorne a este tópico para continuar a configuração.
6. Para executar o AWS IoT Greengrass contêiner, use o comando para seu sistema operacional:

   Linux

   ```
   docker run --rm --init -it -d --name aws-iot-greengrass \
   -p 8883:8883 \
   -v <host-path-to-kernel-config-file>:<container-path> \
   <image-repository>:<tag>
   ```

   - *<host-path-to-kernel-config-file>*Substitua pelo caminho para o arquivo de configuração do kernel no host e *<container-path>* pelo caminho em que o volume está montado no contêiner.

     O arquivo de configuração do kernel no host geralmente está localizado em `/proc/config.gz` ou em `/boot/config-`*<kernel-release-date>*. Você pode correr `uname -r` para encontrar o *<kernel-release-date>* valor.

     Exemplo: para montar o arquivo de configuração de `/boot/config-`*<kernel-release-date>*

     ```
     -v /boot/config-4.15.0-74-generic:/boot/config-4.15.0-74-generic \
     ```

     Exemplo: para montar o arquivo de configuração de `proc/config.gz`

     ```
     -v /proc/config.gz:/proc/config.gz \
     ```

   - Substitua*<image-repository>*: *<tag>* no comando pelo nome do repositório e pela tag da imagem de destino.

     Exemplo: Para apontar para a versão mais recente do software AWS IoT Greengrass Core

     ```
     216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
     ```

Para obter a lista de imagens do AWS IoT Greengrass Docker, execute o comando a seguir.

```
aws ecr list-images --region us-west-2 --registry-id 216483018798 --
repository-name aws-iot-greengrass
```

macOS

```
docker run --rm --init -it -d --name aws-iot-greengrass \
-p 8883:8883 \
<image-repository>:<tag>
```

- Substitua*<image-repository>*: *<tag>* no comando pelo nome do repositório e pela tag da imagem de destino.

  Exemplo: Para apontar para a versão mais recente do software AWS IoT Greengrass Core

  ```
  216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
  ```

  Para obter a lista de imagens do AWS IoT Greengrass Docker, execute o seguinte comando:

  ```
  aws ecr list-images --region us-west-2 --registry-id 216483018798 --
  repository-name aws-iot-greengrass
  ```

Windows

```
docker run --rm --init -it -d --name aws-iot-greengrass \
-p 8883:8883 \
<image-repository>:<tag>
```

- Substitua*<image-repository>*: *<tag>* no comando pelo nome do repositório e pela tag da imagem de destino.

  Exemplo: Para apontar para a versão mais recente do software AWS IoT Greengrass Core

```
216483018798.dkr.ecr.us-west-2.amazonaws.com/aws-iot-greengrass:latest
```

Para obter a lista de imagens do AWS IoT Greengrass Docker, execute o seguinte comando:

```
aws ecr list-images --region us-west-2 --registry-id 216483018798 --
repository-name aws-iot-greengrass
```

> **Important**
>
> Ao testar com o IDT, não inclua o `--entrypoint /greengrass-entrypoint.sh \` argumento usado para executar a imagem para AWS IoT Greengrass uso geral.

7. Próxima etapa: configure suas AWS credenciais e seu `device.json` arquivo.

Solução de problemas na configuração do contêiner Docker para IDT para AWS IoT Greengrass

Use as informações a seguir para ajudar a solucionar problemas com a execução de um contêiner Docker para IDT para testes. AWS IoT Greengrass

AVISO: Erro ao carregar o arquivo de configuração:/home/user/. docker/config.json - stat /home/user /. < > docker/config.json: permissão negada

Se você receber esse erro ao executar comandos `docker` no Linux, execute o comando a seguir. *`<user>`*Substitua o comando a seguir pelo usuário que executa o IDT.

```
sudo chown <user>:<user> /home/<user>/.docker -R
sudo chmod g+rwx /home/<user>/.docker -R
```

Opcional: configurar o dispositivo para qualificação de ML

O IDT for AWS IoT Greengrass fornece testes de qualificação de aprendizado de máquina (ML) para validar se seus dispositivos podem realizar inferência de ML localmente usando modelos treinados em nuvem.

Para executar testes de qualificação de ML, primeiro é preciso configurar os dispositivos conforme descrito em the section called “Configure seu dispositivo para executar testes de IDT”. Depois,

siga as etapas deste tópico para instalar dependências para as estruturas de ML que você deseja executar.

É necessária a versão 3.1.0 ou posterior do IDT para executar testes de qualificação de ML.

Instalar dependências de estrutura do ML

Todas as dependências de estrutura do ML devem ser instaladas no diretório `/usr/local/lib/python3.x/site-packages`. Para certificar-se de que estão instaladas no diretório correto, é recomendado usar permissões raiz `sudo` ao instalar as dependências. Os ambientes virtuais não oferecem suporte a testes de qualificação.

> **Note**
>
> Se você estiver testando funções do Lambda executadas com conteinerização (no modo de Contêiner do Greengrass, a criação de symlinks para bibliotecas Python em `/usr/local/lib/python3.x` não é compatível. Para evitar erros, instale as dependências no diretório correto.

Siga as etapas para instalar as dependências na estrutura de destino:

- Instalar dependências do MXNet
- the section called “Instalar TensorFlow dependências”
- Instalar dependências do DLR

Instalar dependências do Apache MxNet

Os testes de qualificação do IDT para esta estrutura têm as seguintes dependências:

- Python 3.6 ou Python 3.7.

> **Note**
>
> Se estiver usando Python 3.6, você deve criar um symblink de Python 3.7 para binários Python 3.6. Isso configura seu dispositivo para atender ao requisito Python para AWS IoT Greengrass. Por exemplo:

```
sudo ln -s path-to-python-3.6/python3.6 path-to-python-3.7/python3.7
```

- Apache MXNet v1.2.1 ou posterior.
- NumPy. A versão deve ser compatível com sua versão do MXNet.

Instalar o MXNet

Siga as instruções na documentação do MXNet para instalar o MXNet.

> **Note**
>
> Se o Python 2.x e o Python 3.x estiverem instalados no seu dispositivo, use o Python 3.x nos comandos executados para instalar as dependências.

Validar a instalação do MXNet

Selecione uma das opções a seguir para validar a instalação do MXNet.

Opção 1: usar SSH para o seu dispositivo e executar scripts

1. SSH para o seu dispositivo.
2. Execute o script a seguir para verificar se as dependências estão instaladas corretamente.

   ```
   sudo python3.7 -c "import mxnet; print(mxnet.__version__)"
   ```

   ```
   sudo python3.7 -c "import numpy; print(numpy.__version__)"
   ```

   A saída imprime o número da versão e o script deve sair sem erro.

Opção 2: executar o teste de dependência de IDT

1. Certifique-se de que `device.json` esteja configurado para qualificação de ML. Para obter mais informações, consulte the section called “Configurar device.json para qualificação de ML”.
2. Execute o teste de dependências para a estrutura.

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

O resumo do teste exibe um resultado `PASSED` para `mldependencies`.

Instalar TensorFlow dependências

Os testes de qualificação do IDT para esta estrutura têm as seguintes dependências:

- Python 3.6 ou Python 3.7.

> **Note**
>
> Se estiver usando Python 3.6, você deve criar um symblink de Python 3.7 para binários Python 3.6. Isso configura seu dispositivo para atender ao requisito Python para AWS IoT Greengrass. Por exemplo:
>
> ```
> sudo ln -s path-to-python-3.6/python3.6 path-to-python-3.7/python3.7
> ```

- TensorFlow 1.x.

Instalando TensorFlow

Siga as instruções na TensorFlow documentação para instalar o TensorFlow 1.x com pip ou a partir do código-fonte.

> **Note**
>
> Se o Python 2.x e o Python 3.x estiverem instalados no seu dispositivo, use o Python 3.x nos comandos executados para instalar as dependências.

Validando a instalação TensorFlow

Escolha uma das opções a seguir para validar a TensorFlow instalação.

Opção 1: usar SSH para o seu dispositivo e executar um script

1. SSH para o seu dispositivo.
2. Execute o script a seguir para verificar se a dependência está instalada corretamente.

   ```
   sudo python3.7 -c "import tensorflow; print(tensorflow.__version__)"
   ```

   A saída imprime o número da versão e o script deve sair sem erro.

Opção 2: executar o teste de dependência de IDT

1. Certifique-se de que `device.json` esteja configurado para qualificação de ML. Para obter mais informações, consulte the section called “Configurar device.json para qualificação de ML”.
2. Execute o teste de dependências para a estrutura.

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

   O resumo do teste exibe um resultado `PASSED` para `mldependencies`.

Instale as dependências do Amazon SageMaker AI Neo Deep Learning Runtime (DLR)

Os testes de qualificação do IDT para esta estrutura têm as seguintes dependências:

- Python 3.6 ou Python 3.7.

  > **Note**
  >
  > Se estiver usando Python 3.6, você deve criar um symblink de Python 3.7 para binários Python 3.6. Isso configura seu dispositivo para atender ao requisito Python para AWS IoT Greengrass. Por exemplo:
  >
  > ```
  > sudo ln -s path-to-python-3.6/python3.6 path-to-python-3.7/python3.7
  > ```

- SageMaker AI Neo DLR.
- numpy.

Depois de instalar as dependências de teste do DLR, é preciso compilar o modelo.

Instalar o DLR

Siga as instruções na documentação do DLR para instalar o Neo DLR.

> **Note**
>
> Se o Python 2.x e o Python 3.x estiverem instalados no seu dispositivo, use o Python 3.x nos comandos executados para instalar as dependências.

Validar a instalação do DLR

Selecione uma das opções a seguir para validar a instalação do DLR.

Opção 1: usar SSH para o seu dispositivo e executar scripts

1. SSH para o seu dispositivo.
2. Execute o script a seguir para verificar se as dependências estão instaladas corretamente.

   ```
   sudo python3.7 -c "import dlr; print(dlr.__version__)"
   ```

   ```
   sudo python3.7 -c "import numpy; print(numpy.__version__)"
   ```

   A saída imprime o número da versão e o script deve sair sem erro.

Opção 2: executar o teste de dependência de IDT

1. Certifique-se de que `device.json` esteja configurado para qualificação de ML. Para obter mais informações, consulte the section called “Configurar device.json para qualificação de ML”.
2. Execute o teste de dependências para a estrutura.

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

   O resumo do teste exibe um resultado `PASSED` para `mldependencies`.

Compilar o modelo de DLR

Você deve compilar o modelo de DLR antes de usá-lo para testes de qualificação de ML. Selecione uma das seguintes opções para saber mais detalhes:

Opção 1: usar o Amazon SageMaker AI para compilar o modelo

Siga estas etapas para usar a SageMaker IA para compilar o modelo de ML fornecido pelo IDT. Este modelo é pré-treinado com Apache MXNet.

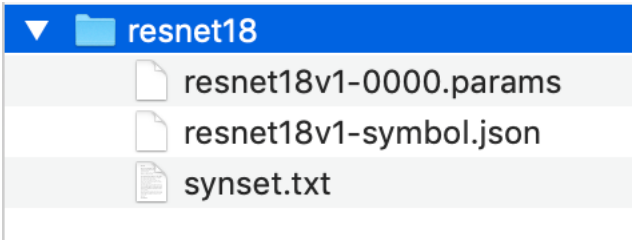
1. Verifique se o tipo de dispositivo é compatível com SageMaker IA. Para obter mais informações, consulte as opções do dispositivo de destino na Amazon SageMaker AI API Reference. Se o tipo de dispositivo atualmente não for suportado pela SageMaker IA, siga as etapas emthe section called “Opção 2: usar o TVM para compilar o modelo de DLR”.

   > **Note**
   >
   > A execução do teste DLR com um modelo compilado pela SageMaker IA pode levar de 4 a 5 minutos. Não interrompa o IDT durante esse período.

2. Faça download do arquivo tarball que contém o modelo MXNet pré-treinado e não compilado para DLR:

   - dlr-noncompiled-model-1.0.tar.gz

3. Descompacte o tarball. Esse comando gera a seguinte estrutura de diretório.

4. Mova o arquivo `synset.txt` do diretório `resnet18` para outro local. Anote o novo local. Posteriormente, copie este arquivo para o diretório do modelo compilado.
5. Compacte o conteúdo do diretório `resnet18`.

   ```
   tar cvfz model.tar.gz resnet18v1-symbol.json resnet18v1-0000.params
   ```

6. Faça o upload do arquivo compactado em um bucket do Amazon S3 em seu Conta da AWS e siga as etapas em Compilar um modelo (console) para criar um trabalho de compilação.

a. Em Configuração de entrada, use os seguintes valores:

   - Em Configuração de dados de entrada, digite `{"data": [1, 3, 224, 224]}`.
   - Em Estrutura de machine learning, selecione `MXNet`.

b. Em Configuração de saída, use os seguintes valores:

   - Em Local de saída do S3, insira o caminho para o bucket do Amazon S3 ou a pasta onde deseja armazenar o modelo compilado.
   - Em Dispositivo de destino, selecione o tipo de dispositivo.

7. Faça download do modelo compilado do local de saída especificado e descompacte o arquivo.
8. Copie `synset.txt` para o diretório do modelo compilado.
9. Altere o nome do diretório do modelo compilado para `resnet18`.

   O diretório do modelo compilado deve ter a seguinte estrutura de diretório.



Opção 2: usar o TVM para compilar o modelo de DLR

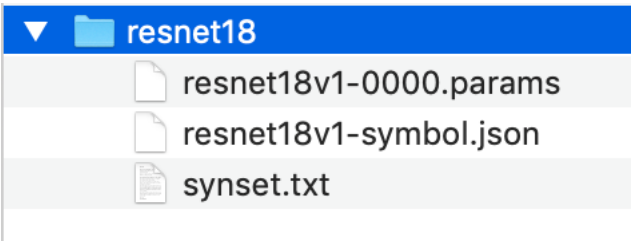
Siga estas etapas para usar o TVM para compilar o modelo de ML fornecido pelo IDT. Este modelo é pré-treinado com o Apache MXNet, portanto, é necessário instalar o MXNet no computador ou dispositivo onde o modelo será compilado. Para instalar o MxNet, siga as instruções na documentação do MxNet.

> **Note**
>
> Recomendamos que você compile o modelo no dispositivo de destino. Essa prática é opcional, mas pode ajudar a garantir a compatibilidade e mitigar possíveis problemas.

1. Faça download do arquivo tarball que contém o modelo MXNet pré-treinado e não compilado para DLR:

   - dlr-noncompiled-model-1.0.tar.gz

2. Descompacte o tarball. Esse comando gera a seguinte estrutura de diretório.



3. Siga as instruções na documentação do TVM para criar e instalar o TVM da origem para a sua plataforma.

4. Depois de criar o TVM, execute a compilação do TVM para o modelo resnet18. As etapas a seguir são baseadas no Quick Start Tutorial for Compiling Deep Learning Models na documentação do TVM.

   a. Abra o arquivo `relay_quick_start.py` a partir do repositório do TVM clonado.

   b. Atualize o código que define uma rede neural em retransmissão. Você pode usar uma das opções a seguir:

      - Opção 1: usar `mxnet.gluon.model_zoo.vision.get_model` para obter o módulo e os parâmetros de retransmissão:

```
from mxnet.gluon.model_zoo.vision import get_model
block = get_model('resnet18_v1', pretrained=True)
mod, params = relay.frontend.from_mxnet(block, {"data": data_shape})
```

      - Opção 2: copiar os seguintes arquivos do modelo não compilado que você baixou na etapa 1 para o mesmo diretório que o arquivo `relay_quick_start.py`. Esses arquivos contêm o módulo e os parâmetros de retransmissão.
        - `resnet18v1-symbol.json`
        - `resnet18v1-0000.params`

   c. Atualize o código que salva e carrega o módulo compilado para usar o código a seguir.

```
from tvm.contrib import util
path_lib = "deploy_lib.so"
#  Export the model library based on your device architecture
```

```
lib.export_library("deploy_lib.so", cc="aarch64-linux-gnu-g++")
with open("deploy_graph.json", "w") as fo:
    fo.write(graph)
with open("deploy_param.params", "wb") as fo:
    fo.write(relay.save_param_dict(params))
```

d. Crie o modelo:

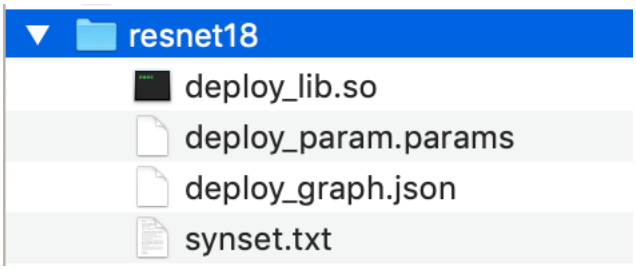
```
python3 tutorials/relay_quick_start.py --build-dir ./model
```

Este comando gera os seguintes arquivos.

- `deploy_graph.json`
- `deploy_lib.so`
- `deploy_param.params`

5. Copie os arquivos de modelo gerados em um diretório chamado `resnet18`. Este é o diretório do modelo compilado.
6. Copie o diretório do modelo compilado para o computador host. Depois, copie o arquivo `synset.txt` do modelo não compilado que você baixou na etapa 1 para o diretório do modelo compilado.

   O diretório do modelo compilado deve ter a seguinte estrutura de diretório.



Em seguida, configure suas AWS credenciais e seu `device.json` arquivo.

Defina as configurações de IDT para executar o AWS IoT Greengrass suíte de qualificação

Antes de executar os testes, você deve definir as configurações de AWS credenciais e dispositivos no computador host.

Configure seu AWS credenciais

Você deve configurar suas credenciais de usuário do IAM no arquivo *<device-tester-extract-location>* `/configs/config.json`. Use as credenciais do IDT para o AWS IoT Greengrass usuário criado em. the section called “Crie e configure um Conta da AWS” Você pode especificar suas credenciais de uma das seguintes formas:

- Arquivo de credenciais
- 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 AWS IoT Greengrass para usar AWS as credenciais do seu `credentials` arquivo, edite seu `config.json` arquivo da seguinte forma:

```
{
 "awsRegion": "us-west-2",
 "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 for AWS IoT Greengrass 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": "us-west-2",
 "auth": {
  "method": "environment"
 }
}
```

Configurar device.json

Além das AWS credenciais, o IDT for AWS IoT Greengrass precisa de informações sobre os dispositivos nos quais os testes são executados (por exemplo, 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`:

Physical device

```
[
  {
    "id": "<pool-id>",
    "sku": "<sku>",
    "features": [
      {
        "name": "os",
        "value": "linux | ubuntu | openwrt"
      },
      {
        "name": "arch",
        "value": "x86_64 | armv6l | armv7l | aarch64"
      },
      {
        "name": "container",
        "value": "yes | no"
      },
      {
        "name": "docker",
        "value": "yes | no"
      },
      {
        "name": "streamManagement",
        "value": "yes | no"
      },
      {
        "name": "hsi",
        "value": "yes | no"
      },
      {
        "name": "ml",
        "value": "mxnet | tensorflow | dlr | mxnet,dlr,tensorflow | no"
      },
      ********** Remove the section below if the device is not qualifying for ML
 **************,
      {
        "name": "mlLambdaContainerizationMode",
        "value": "container | process | both"
```

```
      },
      {
        "name": "processor",
        "value": "cpu | gpu"
      },

*******************************************************************************************
    ],
    ********** Remove the section below if the device is not qualifying for HSI
***************
    "hsm": {
      "p11Provider": "/path/to/pkcs11ProviderLibrary",
      "slotLabel": "<slot_label>",
      "slotUserPin": "<slot_pin>",
      "privateKeyLabel": "<key_label>",
      "openSSLEngine": "/path/to/openssl/engine"
    },

*******************************************************************************************
    ********** Remove the section below if the device is not qualifying for ML
***************
    "machineLearning": {
      "dlrModelPath": "/path/to/compiled/dlr/model",
      "environmentVariables": [
        {
          "key": "<environment-variable-name>",
          "value": "<Path:$PATH>"
        }
      ],
      "deviceResources": [
        {
          "name": "<resource-name>",
          "path": "<resource-path>",
          "type": "device | volume"
        }
      ]
    },

*******************************************************************************************
    "kernelConfigLocation": "",
    "greengrassLocation": "",
    "devices": [
      {
        "id": "<device-id>",
```

```
        "connectivity": {
          "protocol": "ssh",
          "ip": "<ip-address>",
          "port": 22,
          "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`.

Docker container

```
[
  {
    "id": "<pool-id>",
    "sku": "<sku>",
    "features": [
      {
        "name": "os",
        "value": "linux | ubuntu | openwrt"
      },
      {
        "name": "arch",
        "value": "x86_64"
      },
      {
        "name": "container",
        "value": "no"
```

```
      },
      {
        "name": "docker",
        "value": "no"
      },
      {
        "name": "streamManagement",
        "value": "yes | no"
      },
      {
        "name": "hsi",
        "value": "no"
      },
      {
        "name": "ml",
        "value": "mxnet | tensorflow | dlr | mxnet,dlr,tensorflow | no"
      },
      ********** Remove the section below if the device is not qualifying for ML
**************,
      {
        "name": "mlLambdaContainerizationMode",
        "value": "process"
      },
      {
        "name": "processor",
        "value": "cpu | gpu"
      },

*********************************************************************************************
    ],
    ********** Remove the section below if the device is not qualifying for ML
***************
    "machineLearning": {
      "dlrModelPath": "/path/to/compiled/dlr/model",
      "environmentVariables": [
        {
          "key": "<environment-variable-name>",
          "value": "<Path:$PATH>"
        }
      ],
      "deviceResources": [
        {
          "name": "<resource-name>",
          "path": "<resource-path>",
```

```
          "type": "device | volume"
        }
      ]
    },

 ********************************************************************************************
    "kernelConfigLocation": "",
    "greengrassLocation": "",
    "devices": [
      {
        "id": "<device-id>",
        "connectivity": {
          "protocol": "docker",
          "containerId": "<container-name | container-id>",
          "containerUser": "<user>"
        }
      }
    ]
  }
]
```

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 workload. 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 sua placa no Catálogo de AWS Partner dispositivos, o SKU especificado aqui deve 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.

`os` e `arch`

Combinações de sistema operacional (SO) e arquitetura compatíveis:

- `linux`, `x86_64`
- `linux`, `armv6l`
- `linux`, `armv7l`
- `linux`, `aarch64`
- `ubuntu`, `x86_64`
- `openwrt`, `armv7l`
- `openwrt`, `aarch64`

> **Note**
>
> Se você usa o IDT para testar a AWS IoT Greengrass execução em um contêiner Docker, somente a arquitetura x86_64 Docker é suportada.

`container`

Verifica se o dispositivo atende a todos os requisitos de software e hardware para executar as funções do Lambda no modo de contêiner em um núcleo do Greengrass.

O valor válido é `yes` ou `no`.

`docker`

Valida que o dispositivo atende a todas as dependências técnicas necessárias para usar o conector de implantação do aplicativo Docker do Greengrass a fim de executar contêineres.

O valor válido é `yes` ou `no`.

`streamManagement`

Valida se o dispositivo atende a todas as dependências técnicas necessárias para executar o gerenciador de AWS IoT Greengrass streams.

O valor válido é `yes` ou `no`.

`hsi`

Verifica se a biblioteca compartilhada HSI fornecida pode interagir com o módulo de segurança de hardware (HSM) e implementa as APIs PKCS#11 necessárias corretamente. A biblioteca do HSM e compartilhada deve assinar uma CSR, executar operações TLS e fornecer os tamanhos de chaves e o algoritmo de chave pública corretos.

O valor válido é `yes` ou `no`.

`ml`

Valida que o dispositivo atende a todas as dependências técnicas necessárias para executar a inferência de ML localmente.

O valor válido pode ser qualquer combinação de `mxnet`, `tensorflow`, `dlr` e `no` (por exemplo, `mxnet`, `mxnet,tensorflow`, `mxnet,tensorflow,dlr` ou `no`).

`mlLambdaContainerizationMode`

Verifica se o dispositivo atende a todas as dependências técnicas necessárias para executar a inferência de ML em modo contêiner em um dispositivo do Greengrass.

O valor válido é `container`, `process` ou `both`.

`processor`

Verifica se o dispositivo atende a todos os requisitos de hardware do tipo de processador especificado.

O valor válido é `cpu` ou `gpu`.

> **Note**
>
> Se você não quiser usar o atributo `container`, `docker`, `streamManager`, `hsi` ou `ml`, você pode definir o correspondente `value` a `no`.
> O Docker só oferece suporte à qualificação de atributos para `streamManagement` e `ml`.

`machineLearning`

Opcional. Informações de configuração para testes de qualificação de ML. Para obter mais informações, consulte the section called “Configurar device.json para qualificação de ML”.

`hsm`

Opcional. Informações de configuração para testes com um módulo AWS IoT Greengrass de segurança de hardware (HSM). Caso contrário, a propriedade `hsm` deve ser omitida. Para obter mais informações, consulte Integração de segurança de hardware.

Essa propriedade será aplicada somente se `connectivity.protocol` estiver definido como `ssh`.

`hsm.p11Provider`

O caminho absoluto para a biblioteca carregável libdl da implementação PKCS#11.

`hsm.slotLabel`

O rótulo de slot usado para identificar o módulo de hardware.

`hsm.slotUserPin`

O PIN do usuário usado para autenticar o AWS IoT Greengrass núcleo no módulo.

`hsm.privateKeyLabel`

O rótulo usado para identificar a chave no módulo de hardware.

`hsm.openSSLEngine`

O caminho absoluto para o arquivo `.so` do mecanismo OpenSSL para habilitar o suporte ao PKCS#11 no OpenSSL. Usado pelo agente de atualização AWS IoT Greengrass OTA.

`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. No momento, os únicos valores compatíveis são `ssh` 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.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 `docker`.

`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.auth.credentials.privKeyPath`

O caminho completo para a chave privada usada para fazer login no dispositivo que está sendo testado.

`connectivity.port`

Opcional. O número da porta usada nas conexões SSH.

O valor padrão é 22.

Essa propriedade só se aplica se `connectivity.protocol` estiver definido como `ssh`.

`greengrassLocation`

A localização do software AWS IoT Greengrass Core em seus dispositivos.

Para dispositivos físicos, esse valor é usado somente quando você usa uma instalação existente do AWS IoT Greengrass. Use esse atributo para instruir o IDT a usar a versão do software do núcleo do AWS IoT Greengrass instalada nos dispositivos.

Ao executar testes em um contêiner do Docker a partir da imagem do Docker ou do Dockerfile fornecido por AWS IoT Greengrass, defina esse valor como. `/greengrass`

`kernelConfigLocation`

Opcional. O caminho para o arquivo de configuração do kernel. AWS IoT O Device Tester usa esse arquivo para verificar se os dispositivos têm os recursos necessários do kernel habilitados. Se não for especificado, o IDT usa os seguintes caminhos para pesquisar o arquivo de configuração do kernel: e. `/proc/config.gz` `/boot/config-`*`<kernel-version>`* AWS IoT O Device Tester usará o primeiro caminho que encontrar.

Configurar device.json para qualificação de ML

Esta seção descreve as propriedades opcionais no arquivo de configuração do dispositivo que se aplicam à qualificação de ML. Se você planeja executar testes para qualificação de ML, é necessário definir as propriedades que se aplicam ao caso de uso.

Você pode usar o modelo `device-ml.json` para definir as configurações do dispositivo. Este modelo contém as propriedades de ML opcionais. Você também pode usar `device.json` e adicionar as propriedades de qualificação de ML. Esses arquivos estão localizados em *`<device-`*

tester-extract-location>/configs e incluem propriedades de qualificação de ML. Se usar device-ml.json, você deve renomear o arquivo como device.json antes de executar testes do IDT.

Para obter informações sobre propriedades de configuração de dispositivo que não se aplicam à qualificação de ML, consulte the section called “Configurar device.json”.

`ml` na matriz `features`

As estruturas de ML compatíveis com a sua placa. Esta propriedade requer a versão 3.1.0 ou posterior do IDT.

- Caso a sua placa seja compatível com uma estrutura somente, especifique a estrutura. Por exemplo:

```
{
    "name": "ml",
    "value": "mxnet"
}
```

- Caso a sua placa seja compatível com várias estruturas, especifique as estruturas como uma lista separada por vírgulas. Por exemplo:

```
{
    "name": "ml",
    "value": "mxnet,tensorflow"
}
```

`mlLambdaContainerizationMode` na matriz `features`

O modo de conteinerização com o qual você deseja testar. Esta propriedade requer a versão 3.1.0 ou posterior do IDT.

- Selecione `process` para executar o código de inferência de ML com uma função do Lambda sem contêineres. Essa opção requer a AWS IoT Greengrass versão v1.10.x ou posterior.
- Selecione `container` para executar o código de inferência de ML com uma função do Lambda em contêineres.
- Selecione `both` para executar o código de inferência de ML com ambos os modos. Essa opção requer a AWS IoT Greengrass versão v1.10.x ou posterior.

`processor` na matriz `features`

Indica o acelerador de hardware compatível com a placa. Esta propriedade requer a versão 3.1.0 ou posterior do IDT.

- Selecione `cpu` se sua placa usa uma CPU como processador.
- Selecione `gpu` se a sua placa usa uma GPU como processador.

`machineLearning`

Opcional. Informações de configuração para testes de qualificação de ML. Esta propriedade requer a versão 3.1.0 ou posterior do IDT.

`dlrModelPath`

É necessário usar a estrutura do `dlr`. O caminho absoluto para o diretório de modelo compilado do DLR, que deve ser chamado de `resnet18`. Para obter mais informações, consulte the section called “Compilar o modelo de DLR”.

> **Note**
>
> Veja a seguir um exemplo de caminho no macOS: `/Users/<user>/Downloads/resnet18`.

`environmentVariables`

Uma matriz de pares de chave/valor que pode passar dinamicamente configurações para testes de inferência de ML. Opcional para dispositivos de CPU. Você pode usar esta seção para adicionar variáveis de ambiente específicas da estrutura exigidas pelo tipo de dispositivo. Para obter informações sobre esses requisitos, consulte o site oficial da estrutura ou do dispositivo. Por exemplo, para executar testes de inferência do MXNet em alguns dispositivos, as seguintes variáveis de ambiente podem ser necessárias.

```
"environmentVariables": [
    ...
    {
        "key": "PYTHONPATH",
        "value": "$MXNET_HOME/python:$PYTHONPATH"
    },
    {
```

```
        "key": "MXNET_HOME",
        "value": "$HOME/mxnet/"
    },
    ...
]
```

> **Note**
>
> O campo `value` pode variar com base na instalação do MXNet.

Se você estiver testando funções do Lambda executadas com conteinerização em dispositivos de GPU, adicione variáveis de ambiente para a biblioteca de GPU. Isso possibilita a realização de cálculos pela GPU. Para usar bibliotecas de GPU diferentes, consulte a documentação oficial da biblioteca ou do dispositivo.

> **Note**
>
> Configure as chaves a seguir se o atributo `mlLambdaContainerizationMode` estiver definido como `container` ou `both`.

```
"environmentVariables": [
    {
        "key": "PATH",
        "value": "<path/to/software/bin>:$PATH"
    },
    {
        "key": "LD_LIBRARY_PATH",
        "value": "<path/to/ld/lib>"
    },
    ...
]
```

`deviceResources`

Obrigatório para dispositivos de GPU. Contém recursos locais que podem ser acessados por funções do Lambda. Use esta seção para adicionar recursos de dispositivo e de volume locais.

- Para recursos de dispositivo, especifique `"type": "device"`. Para dispositivos de GPU, os recursos do dispositivo devem estar abaixo `/dev` dos arquivos GPU-related do dispositivo.

  > **Note**
  >
  > O diretório `/dev/shm` é uma exceção. Ele pode ser configurado apenas como um recursos de volume.

- Para recursos de volume, especifique `"type": "volume"`.

Execute o AWS IoT Greengrass suíte de qualificação

Depois de definir a configuração necessária, você poderá iniciar os testes. O runtime 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.

Os comandos de exemplo `run-suite` a seguir mostram como executar os testes de qualificação para um grupo de dispositivos. Um grupo de dispositivos é um conjunto de dispositivos idênticos.

IDT v3.0.0 and later

Execute todos os grupos de teste em um conjunto de testes especificado.

```
devicetester_[linux | mac | win_x86-64] run-suite --suite-id GGQ_1.0.0 --pool-
id <pool-id>
```

Use o comando `list-suites` para listar os conjuntos de testes que estão na pasta `tests`.

Execute um grupo de testes específico em um conjunto de testes.

```
devicetester_[linux | mac | win_x86-64] run-suite --suite-id GGQ_1.0.0 --group-
id <group-id> --pool-id <pool-id>
```

Use o comando `list-groups` para listar os grupos de teste em um conjunto de testes.

Execute 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>
```

Execute 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>
```

Liste os casos de teste em um grupo de teste.

```
devicetester_[linux | mac | win_x86-64] list-test-cases --group-id <group-id>
```

As opções para o comando `run-suite` são opcionais. Por exemplo, você pode omitir `pool-id` se tiver apenas um único grupo de dispositivos definido no arquivo `device.json`. 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 obter mais informações, consulte the section called “Defina o comportamento de atualização padrão”.

Para obter mais informações sobre `run-suite` e outros comandos IDT, consulte the section called “Comandos do IDT”.

IDT v2.3.0 and earlier

Execute todos os grupos de teste em um conjunto especificado.

```
devicetester_[linux | mac | win_x86-64] run-suite --suite-id GGQ_1 --pool-
id <pool-id>
```

Execute um grupo de teste específico.

```
devicetester_[linux | mac | win_x86-64] run-suite --suite-id GGQ_1 --group-
id <group-id> --pool-id <pool-id>
```

`suite-id` e `pool-id` são opcionais se você estiver executando um único conjunto de testes em um único grupo de dispositivos. Isso significa que você tem apenas um grupo de dispositivos definido em seu arquivo `device.json`.

Verifique as dependências do Greengrass

Recomendamos executar o grupo de testes do verificador de dependência para garantir que todas as dependências do Greengrass estejam instaladas antes de executar grupos de testes relacionados. Por exemplo:

- Execute `ggcdependencies` antes de executar grupos de testes de qualificação de núcleo.
- Execute `containerdependencies` antes de executar grupos de teste específicos do contêiner.
- Execute `dockerdependencies` antes de executar grupos Docker-specific de teste.
- Execute `ggcstreammanagementdependencies` antes de executar grupos de testes específicos do gerenciador de fluxos.

Defina o comportamento de atualização padrão

Quando você inicia uma execução de teste, o IDT verifica online se há uma versão mais recente do conjunto de testes. Se houver, o IDT solicitará que você atualize para a versão mais recente disponível. Você pode definir o sinalizador `upgrade-test-suite` (ou u) para controlar o comportamento de atualização padrão. Os valores válidos são:

- y. O IDT faz download e usa a versão mais recente disponível.
- n (padrão). O IDT usa a versão especificada na opção `suite-id`. Se `suite-id` não for especificado, o IDT usará a versão mais recente na pasta `tests`.

Se você não incluir o sinalizador `upgrade-test-suite`, o IDT solicitará quando uma atualização estiver disponível e aguardará 30 segundos por sua entrada (y ou n). Se nenhuma entrada for inserida, ela assumirá n como padrão e continuará executando os testes.

Os exemplos a seguir mostram casos de uso comuns para esse atributo:

Usar automaticamente os testes mais recentes disponíveis para um grupo de testes.

```
devicetester_linux run-suite -u y --group-id mqtt --pool-id DevicePool1
```

Execute testes em uma versão específica do conjunto de testes.

```
devicetester_linux run-suite -u n --suite-id GGQ_1.0.0 --group-id mqtt --pool-id
 DevicePool1
```

Solicitar atualizações em tempo de execução.

```
devicetester_linux run-suite --pool-id DevicePool1
```

IDT para AWS IoT Greengrass Comandos da do

Os comandos do IDT estão localizados no diretório *<device-tester-extract-location>*/bin. Use-os para as seguintes operações:

IDT v3.0.0 and later

`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 suportados, neste caso AWS IoT Greengrass as versões, e as 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, como 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, como 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`.
- `upgrade-test-suite`. Controla como as atualizações da versão do conjunto de testes são tratadas. A partir do IDT v3.0.0, o IDT verifica online as versões atualizadas do conjunto de testes. Para obter mais informações, consulte the section called “Versões do pacote de testes”.
- `stop-on-first-failure`. Configura o IDT de modo a interromper a execução na primeira falha. Essa opção deve ser usada com para depurar os grupos de teste especificados `group-id`. Não use essa opção para gerar um relatório de qualificação ao executar um conjunto de testes completo.
- `update-idt`. Define a resposta para que o prompt atualize o IDT. O `Y` como entrada interrompe a execução do teste se o IDT detectar que há uma versão mais recente. O `N` como entrada continua a execução do teste.
- `update-managed-policy`. O `Y` como entrada interrompe a execução do teste se o IDT detectar que a política gerenciada do usuário não foi atualizada. O `N` como entrada continua a execução do teste.

Para obter mais informações sobre as opções `run-suite`, use a opção `help`:

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

IDT v2.3.0 and earlier

`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.

`run-suite`

Executa um conjunto de testes em um grupo de dispositivos.

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.

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 podem ser encontrados em *<device-tester-extract-location>*`/results/`*<execution-id>*`/`. Ambos os relatórios capturam os resultados da execução do pacote de teste de qualificação.

Esse `awsiotdevicetester_report.xml` é o relatório do teste de qualificação que você envia AWS para listar seu AWS Partner dispositivo no Catálogo de dispositivos. O relatório contém os seguintes elementos:

- A versão IDT.
- A AWS IoT Greengrass versão que foi testada.
- A SKU e o nome de grupo do dispositivo especificados no arquivo `device.json`.
- Os atributos do grupo do dispositivo especificados no arquivo `device.json`.
- O resumo agregado dos resultados de teste.
- Um detalhamento dos resultados de teste pelas bibliotecas que foram testadas, com base nos atributos do dispositivo (por exemplo, acesso de recurso local, shadow, MQTT e assim por diante).

O relatório `GGQ_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 AWS IoT Greengrass funcionalidade que foi testada.

Interpretando os relatórios do IDT

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, necessário para executar o conjunto de qualificação.

`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 recurso não é usado e pode ser ignorado.

O arquivo `awsiotdevicetester_report.xml` contém uma tag `<awsproduct>` com informações sobre o produto que está sendo testado e os atributos 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 atributos validados. Atributos marcados como `required` são necessários para enviar sua placa para qualificação. O trecho a seguir mostra como essas informações aparecem no arquivo `awsiotdevicetester_report.xml`.

```
<feature name="aws-iot-greengrass-no-container" value="supported" type="required"></
feature>
```

Os atributos marcados como `optional` não são necessários para qualificação. Os seguintes trechos mostram atributos opcionais.

```
<feature name="aws-iot-greengrass-container" value="supported" type="optional"></
feature>

<feature name="aws-iot-greengrass-hsi" value="not-supported" type="optional"></
feature>
```

Se não houver falhas de teste ou erros nos recursos necessários, seu dispositivo atende aos requisitos técnicos para ser executado AWS IoT Greengrass e pode interoperar com AWS IoT os serviços. Se quiser listar seu AWS 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 recurso `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 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

O IDT gera logs de teste em execução 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 do AWS IoT Device Tester (por exemplo, registros relacionados à configuração, sequenciamento de testes e geração de relatórios).

<test_case_id>`.log (for example, ota.log)`

Os logs do grupo de testes, incluindo logs do dispositivo em teste. Quando um teste falhar, um arquivo tar.gz com os logs do dispositivo em teste para o teste será criado (por exemplo, `ota_prod_test_1_ggc_logs.tar.gz`).

Para obter mais informações, consulte IDT para AWS IoT Greengrass solução de problemas.

Usar o IDT para desenvolver e executar os próprios pacotes de testes

A partir do IDT v4.0.0, o IDT for AWS IoT Greengrass 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.

Use o IDT para desenvolver e executar pacotes de testes personalizados da seguinte 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 pacotes de teste.

Para executar pacotes de testes personalizados

- Configure o dispositivo que deseja testar.
- Implemente as configurações conforme exigido pelos pacotes de testes que deseja usar.
- Use o IDT para executar seus pacotes de teste personalizados.
- Veja os resultados do teste e os logs de execução dos testes executados pelo IDT.

Baixe a versão mais recente do AWS IoT Testador de dispositivos para AWS IoT Greengrass

Baixe a versão mais recente do IDT e extraia o software em um local no seu sistema de arquivos onde você tenha permissões de leitura e 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 pacotes de teste

Os conjuntos de testes são compostos por três tipos de arquivos:

- Arquivos de configuração JSON que fornecem ao IDT informações sobre como executar o conjunto 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. Criação de arquivos de configuração JSON para seu conjunto de testes.
2. Crie executáveis de casos de teste que contenham a lógica de teste para seu conjunto de testes.
3. Verifique e documente as informações de configuração necessárias para que os executores de teste executem o pacote de teste.
4. Verifique se o IDT pode executar seu pacote de teste e produzir resultados de teste conforme o esperado.

Para criar rapidamente uma amostra de pacote 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 suíte de testes. Você pode concluir este tutorial para criar e executar a suíte de testes de amostra para entender como usar o AWS IoT Device Tester AWS IoT Greengrass para executar suítes de testes personalizadas.

Neste tutorial, você concluirá as seguintes etapas:

1. Compile o pacote de teste de amostra
2. Usar o IDT para executar o pacote de teste 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 seguinte comando:

    ```
    python3 --version
    ```

    No Windows, se o uso deste 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 seguinte comando:

    ```
    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.

    É recomendado o uso de um Raspberry Pi com o sistema operacional Raspberry Pi. Certifique-se de configurar o SSH no seu Raspberry Pi para se conectar remotamente a ele.

Configurar as informações do dispositivo para o IDT

Configure as informações do seu dispositivo para o IDT executar o teste. É preciso atualizar o modelo `device.json` localizado na pasta *<device-tester-extract-location>*`/configs` com as informações a seguir.

```
[
  {
    "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`.

Compile o pacote de teste de amostra

A pasta *<device-tester-extract-location>*`/samples/python` contém exemplos de arquivos de configuração, código-fonte e o SDK do cliente IDT que você pode combinar em um pacote de teste usando os scripts de compilação fornecidos. A seguinte árvore de diretórios 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 pacote de teste, 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 pasta `IDTSampleSuitePython_1.0.0` para entender como o conjunto de testes de amostra está estruturado e confira vários exemplos de executáveis de casos de teste e arquivos JSON de configuração de teste.

Próxima etapa: Usar o IDT para executar o conjunto de testes de amostra que você criou.

Usar o IDT para executar o pacote de teste de amostra

Para executar o pacote de teste 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 pacote de teste de amostra e transmite os resultados para o console. Quando a execução do teste é concluída, são vistas 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 todos os pré-requisitos deste tutorial são atendidos.

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 JSON que descrevem o conjunto de testes

Este tutorial mostra como usar o IDT AWS IoT Greengrass para desenvolver uma suíte de testes em Python que contém um único caso de teste. Neste tutorial, você concluirá as seguintes etapas:

1. Crie um diretório de pacotes de teste
2. Criação de arquivos de configuração JSON
3. Crie o executável do caso de teste
4. Execute o pacote de teste

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 seguinte comando:

    ```
    python3 --version
    ```

    No Windows, se o uso deste 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 seguinte comando:

    ```
    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.

    É recomendado o uso de um Raspberry Pi com o sistema operacional Raspberry Pi. Certifique-se de configurar o SSH no seu Raspberry Pi para se conectar remotamente a ele.

Crie um diretório de pacotes de teste

O IDT separa logicamente os casos de teste em grupos de teste dentro de cada pacote de teste. Cada caso de teste deve estar dentro de um grupo de teste. Para este tutorial, crie uma pasta chamada `MyTestSuite_1.0.0` e crie a seguinte árvore de diretórios nessa pasta:

```
MyTestSuite_1.0.0
### suite
    ### myTestGroup
        ### myTestCase
```

Criação de arquivos de configuração JSON

O conjunto de testes deve conter os seguintes arquivos de configuração JSON necessários:

Arquivos JSON necessários

`suite.json`

Contém informações sobre o conjunto de testes. Consulte Configuração do suite.json.

`group.json`

Contém informações sobre um grupo de teste. É preciso criar um arquivo `group.json` para cada grupo de teste em seu pacote de teste. Consulte Configurar group.json.

`test.json`

Contém informações sobre um caso de teste. É preciso criar um arquivo `test.json` para cada caso de teste em seu pacote de teste. Consulte Configurar 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
```

Obtenha o SDK do cliente do IDT

Use o SDK de cliente do IDT para permitir que o IDT interaja com o dispositivo testado e relate os resultados do teste. Neste tutorial, use a versão em 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 com êxito, execute o comando a seguir.

```
cd MyTestSuite_1.0.0/suite/myTestGroup/myTestCase
python3 -c 'import idt_client'
```

Crie o executável do caso de teste

Os executáveis do caso de teste contêm a lógica de teste que você deseja executar. Um pacote de teste pode conter vários executáveis de casos de teste. Para este tutorial, crie somente um executável de caso de teste.

1. Crie o arquivo do pacote de teste.

   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()
```

Configurar as informações do dispositivo para o IDT

Configure as informações do seu dispositivo para o IDT executar o teste. É preciso atualizar o modelo `device.json` localizado na pasta *<device-tester-extract-location>*`/configs` com as informações a seguir.

```
[
  {
    "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`.

Execute o pacote de teste

Depois de criar o pacote de teste, verifique se ele funciona conforme o esperado. Conclua as etapas a seguir para executar o pacote de teste em seu grupo de dispositivos existente.

1. Copie sua 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 pacote de teste e transmite os resultados para o console. Quando a execução do teste é concluída, são vistas 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 erro, verifique o seguinte:

- O SDK de cliente do IDT 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 que são incluídos ao escrever um pacote personalizado de testes.

Arquivos JSON necessários

`suite.json`

Contém informações sobre o conjunto de testes. Consulte Configuração do suite.json.

`group.json`

Contém informações sobre um grupo de teste. É preciso criar um arquivo `group.json` para cada grupo de teste em seu pacote de teste. Consulte Configurar group.json.

`test.json`

Contém informações sobre um caso de teste. É preciso criar um arquivo `test.json` para cada caso de teste em seu pacote de teste. Consulte Configurar test.json.

Arquivos JSON opcionais

`state_machine.json`

Define como os testes são executados quando o IDT executa o conjunto de testes. Consulte Configuração de state_machine.json.

`userdata_schema.json`

Define o esquema do arquivo `userdata.json` que os executores de teste podem incluir em sua definição de configuração. O arquivo `userdata.json` é usado em qualquer informação de configuração adicional necessária para executar o teste, mas que não esteja presente no arquivo `device.json`. Consulte Configuração de 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
    ### state_machine.json
    ### userdata_schema.json
    ### <test-group-folder>
        ### group.json
        ### <test-case-folder>
            ### test.json
```

Configuração do suite.json

O arquivo `suite.json` define as variáveis de ambiente e determina se os dados do usuário são necessários para executar o pacote de teste. 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 pacote de teste. O valor de `id` deve corresponder ao nome da pasta do pacote de teste na qual o arquivo `suite.json` está localizado. O nome e a versão do pacote 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 IDT-generated de teste.

`title`

Um nome definido pelo usuário para o produto ou recurso que está sendo testado por esse pacote de teste. O nome é exibido na CLI do IDT para os executores de teste.

`details`

Uma descrição breve da finalidade do pacote de teste.

`userDataRequired`

Define se os executores de teste precisam incluir informações personalizadas em um arquivo `userdata.json`. Se definir esse valor como `true`, também deverá incluir o arquivo `userdata_schema.json` na pasta do pacote de teste.

`environmentVariables`

Opcional. Uma matriz de variáveis de ambiente para configurar para esse pacote de teste.

`environmentVariables.key`

O nome da variável de ambiente.

`environmentVariables.value`

O valor da variável de ambiente.

Configurar 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`

Uma ID exclusiva, definida pelo usuário, para o grupo de testes. O valor de `id` deve corresponder ao nome da pasta do grupo de testes, na qual o arquivo `group.json` está localizado, e não deve conter sublinhados (_). O ID é usado nos relatórios IDT-generated de teste.

`title`

Um nome descritivo para o grupo de teste. O nome é exibido na CLI do IDT para os executores de teste.

`details`

Uma descrição breve da finalidade do grupo de teste.

`optional`

Opcional. Defina como `true` para exibir este grupo de teste como um grupo opcional depois que o IDT terminar de executar os testes necessários. O valor padrão é `false`.

Configurar 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 obter mais informações sobre como criar executáveis de casos de teste, consulte Criar 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`

Uma ID exclusiva definida pelo usuário para o caso de teste. O valor de `id` deve corresponder ao nome da pasta do caso de teste na qual o arquivo `test.json` está localizado e não deve conter sublinhados (_). O ID é usado nos relatórios IDT-generated de teste.

`title`

Um nome descritivo para o caso de teste. O nome é exibido na CLI do IDT para os executores de teste.

`details`

Uma descrição breve da finalidade do caso de teste.

`requireDUT`

Opcional. Defina como `true` se um dispositivo for necessário para executar este teste. Caso contrário, defina como `false`. O valor padrão é `true`. Os executores de teste configurarão os dispositivos que usarão para executar o teste nos arquivos `device.json`.

`requiredResources`

Opcional. Uma matriz que fornece informações sobre os dispositivos de recursos necessários para executar esse teste.

`requiredResources.name`

O nome exclusivo a ser dado ao dispositivo de recurso quando este teste está sendo executado.

`requiredResources.features`

Uma matriz de recursos de dispositivos de recursos definidos pelo usuário.

`requiredResources.features.name`

O nome do recurso. O recurso do dispositivo para o qual você deseja usar este dispositivo. Este nome é comparado ao nome do recurso fornecido pelo executor de teste no arquivo `resource.json`.

`requiredResources.features.version`

Opcional. A versão do recurso. Este valor é comparado à versão do recurso fornecida pelo executor de teste no arquivo `resource.json`. Se uma versão não for fornecida, o recurso não será verificado. Se um número de versão não for obrigatório para o recurso, deixe este campo em branco.

`requiredResources.features.jobSlots`

Opcional. O número de testes simultâneos que podem ser compatíveis com este recurso. O valor padrão é 1. Se você quiser que o IDT use dispositivos distintos para recursos individuais, recomendamos que você defina esse valor como 1.

`execution.timeout`

A quantidade de tempo (em milissegundos) que o IDT aguardará a conclusão da execução do teste. Para obter mais informações sobre este valor, consulte Criar 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 deseja executar para o sistema operacional especificado. Este 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 este caso de teste.

`environmentVariables.key`

O nome da variável de ambiente.

`environmentVariables.value`

O valor da variável de ambiente.

> **Note**
>
> Se você especificar a mesma variável de ambiente no arquivo `test.json` e no arquivo `suite.json`, o valor no arquivo `test.json` prevalecerá.

Configuração de state_machine.json

Uma máquina de estado é uma estrutura que controla o fluxo de execução do conjunto de testes. 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 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 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 pacote de teste em uma ordem aleatória.

- Gera relatórios e exibe 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 a máquina de estado do IDT funciona, consulte Configure a máquina de estados do IDT.

Configuração de 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 pacote de teste exigir informações que não estejam presentes no arquivo `device.json`. Por exemplo, seus testes podem precisar de credenciais de Wi-Fi rede, portas abertas específicas ou certificados que um usuário deve fornecer. Estas 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` incluído no pacote de teste.

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 válido do IETF Draft v4.

Quando o IDT executa seu conjunto de testes, ele lê automaticamente o esquema e o usa para validar o arquivo `userdata.json` fornecido pelo executor do teste. Se for válido, o conteúdo do arquivo `userdata.json` estará disponível no contexto do IDT e no contexto da máquina de estados.

Configure a máquina de estados do IDT

Uma máquina de estado 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 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 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 pacote de teste 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 de um pacote de teste da IDT deve atender aos seguintes critérios:

- Cada estado corresponde a uma ação a ser executada pelo IDT, como executar um grupo de teste ou produzir 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 da máquina de estado

É possível usar o modelo a seguir para configurar seu próprio arquivo *<custom-test-suite-folder>*`/suite/state_machine.json`:

```
{
  "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`

O nome do estado em que o IDT começa a executar o pacote de teste. O valor de `StartAt` deve ser definido como um dos estados listados no objeto `States`.

`States`

Um objeto que mapeia nomes de estados definidos pelo usuário para estados IDT válidos. Cada estado. *`state-name`*objeto contém a definição de um estado válido mapeado para o. *`state-name`*

O objeto `States` deve incluir os estados `Succeed` e `Fail`. Para obter informações sobre estados válidos, consulte Estados válidos e definições de estado.

Estados válidos e definições de estado

Esta seção descreve as definições de estado de todos os estados válidos que podem ser usados na máquina de estados IDT. Alguns dos estados a seguir são compatíveis com configurações no nível do caso de teste. No entanto, é recomendado configurar as 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 do estado

- RunTask
- Escolha
- Paralelo
- AddProductFeatures
- Relatório
- LogMessage
- SelectGroup
- Falha
- Êxito

RunTask

O estado `RunTask` executa casos de teste de um grupo de teste definido no pacote de teste.

```
{
    "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`

O nome do estado para o qual fazer a transição após a execução das ações no estado atual.

`TestGroup`

Opcional. O ID do grupo de teste a ser executado. Se este valor não for especificado, o IDT executará o grupo de teste selecionado pelo executor de testes.

`TestCases`

Opcional. Uma matriz de IDs de casos de teste do grupo especificado em `TestGroup`. Com base nos valores de `TestGroup` e `TestCases`, o IDT determina o comportamento da execução do teste da seguinte forma:

- Quando tanto o `TestGroup` como os `TestCases` são especificados, o IDT executa os casos de teste especificados do grupo de teste.
- Quando `TestCases` são especificados, mas `TestGroup` não são especificados, o IDT executa os casos de teste especificados.
- Quando `TestGroup` é especificado, mas `TestCases` não são especificados, o IDT executa os casos no grupo de teste especificado.
- Quando nem `TestGroup` nem `TestCases` são especificados, o IDT executa todos os casos de teste do grupo de teste que o executor de teste seleciona na CLI do IDT. Para habilitar a seleção de grupos para executores de teste, é preciso incluir ambos os estados `RunTask` e `Choice` em seu arquivo `statemachine.json`. Para ver um exemplo de como isso funciona, consulte Exemplo de máquina de estado: executar grupos de teste selecionados pelo usuário.

Para obter mais informações sobre como habilitar os comandos da CLI no IDT, consulte the section called “Habilitar comandos da CLI no IDT”.

`ResultVar`

O nome da variável de contexto a ser definida com os resultados da execução do teste. Não especifique este valor se você não especificou um valor para `TestGroup`. O IDT define o valor da variável que você define em `ResultVar` para `true` ou `false` com base no seguinte:

- Se o nome da variável estiver no formato *`text`*`_`*`text`*`_passed`, o valor será definido como se todos os testes do primeiro grupo de teste foram aprovados ou ignorados.
- Em todos os outros casos, o valor é definido como se todos os testes em todos os grupos de teste tivessem sido aprovados ou ignorados.

Normalmente, você usará o estado `RunTask` 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 paralelo, em uma ordem aleatória. No entanto, se todos os casos de teste exigirem a execução de um dispositivo e apenas um único dispositivo estiver disponível, os casos de teste serão executados em sequência.

Como tratar erros

Se algum dos grupos de teste ou IDs de casos de teste especificados não for válido, esse estado emitirá o erro de execução `RunTaskError`. Se o estado encontrar um erro de execução, ele também definirá a variável `hasExecutionError` no contexto da máquina de estado como `true`.

Escolha

O estado `Choice` permite definir dinamicamente o próximo estado para o qual fazer a transição com base em 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 fazer a transição se nenhuma das expressões definidas em `Choices` puder ser avaliada como `true`.

`FallthroughOnError`

Opcional. Especifica o comportamento quando o estado se depara com um erro na avaliação de expressões. Defina como `true` se deseja ignorar uma expressão, caso a avaliação resulte em um erro. Se não houver correspondência de expressão, a máquina de estado fará a transição para o estado `Default`. Se o valor `FallthroughOnError` não for especificado, o valor padrão será `false`.

`Choices`

Uma matriz de expressões e estados para determinar para qual estado fazer a transição depois de executar as ações no estado atual.

`Choices.Expression`

Uma string de expressão que deve ser avaliada para um valor booleano. Se a expressão for avaliada como `true`, a máquina de estado fará a transição para o estado definido em `Choices.Next`. As strings de expressão recuperam valores do contexto da máquina de estado e, em seguida, executam operações neles para chegar a um valor booleano. Para obter informações sobre como acessar o contexto da máquina de estado, consulte Contexto da máquina de estado.

`Choices.Next`

O nome do estado para o qual fazer a transição caso a expressão definida em `Choices.Expression` seja avaliada como `true`.

Como tratar erros

O estado `Choice` pode exigir o 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 bloco `Catch` para tratar erros nesse estado. Se quiser parar de executar a máquina de estado quando ela encontrar um erro, defina `FallthroughOnError` como `false`. No entanto, é recomendado configurar `FallthroughOnError` e `true`, e dependendo do seu caso de uso, executar uma das seguintes etapas:

- Se uma variável que você está acessando não existir em alguns casos, use o valor de `Default` e blocos `Choices` adicionais para especificar o próximo estado.
- Se uma variável que estiver acessando sempre existir, defina o estado `Default` como `Fail`.

Paralelo

O estado `Parallel` permite que você defina e execute novas máquinas de estado em paralelo.

```
{
    "Type": "Parallel",
    "Next": "<state-name>",
    "Branches": [
        <state-machine-definition>
    ]
}
```

Todos os campos que contêm valores são necessários, conforme descrito aqui:

`Next`

O nome do estado para o qual fazer a transição após a execução das ações no estado atual.

`Branches`

Uma matriz de definições de máquina de estado a serem executadas. Cada definição de máquina de estado deve conter os próprios estados `StartAt`, `Succeed` e `Fail`. As definições de máquina de estado nesta matriz não podem fazer referência a estados de fora da definição deles.

> **Note**
>
> Como cada máquina de estado de ramificação compartilha o mesmo contexto de máquina de estado, definir variáveis em uma ramificação e depois lê-las de outra ramificação pode resultar em um comportamento inesperado.

O estado `Parallel` passa para o próximo estado somente depois de executar todas as máquinas de estado da ramificação. Todo estado que necessita um dispositivo aguardará para ser executado até que o dispositivo esteja disponível. Se vários dispositivos estiverem disponíveis, este estado executará casos de teste a partir de vários grupos em paralelo. Se não houver dispositivos suficientes disponíveis, os casos de teste serão executados em sequência. Como os casos de teste são executados em uma ordem aleatória quando executados em paralelo, podem ser usados dispositivos diferentes para executar testes a partir do mesmo grupo de teste.

Como tratar erros

Certifique-se que tanto a máquina de estado da ramificação quanto a máquina de estado pai fazem a transição para o estado `Fail` para tratar erros de execução.

Como as máquinas de estado de ramificação não transmitem erros de execução para a máquina de estado principal, você não pode usar um bloco `Catch` para lidar com erros de execução em máquinas de estado de ramificação. Em vez disso, use o valor `hasExecutionErrors` no contexto da máquina de estado compartilhada. Para obter um exemplo de como isso funciona, consulte Exemplo de máquina de estado: execute dois grupos de testes em paralelo.

AddProductFeatures

O estado `AddProductFeatures` permite adicionar recursos do produto ao arquivo `awsiotdevicetester_report.xml` gerado pelo IDT.

Um recurso do produto é uma informação definida pelo usuário sobre critérios específicos que um dispositivo pode atender. Por exemplo, o recurso do produto `MQTT` pode indicar que o dispositivo publica mensagens MQTT corretamente. No relatório, os recursos do produto são definidas como `supported`, `not-supported` ou um valor personalizado, com base na aprovação dos testes especificados.

> **Note**
>
> O estado `AddProductFeatures` não gera relatórios por conta própria. Esse estado deve passar para o estado `Report` para 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`

O nome do estado para o qual fazer a transição após a execução das ações no estado atual.

`Features`

Uma matriz de recursos do produto para mostrar no arquivo `awsiotdevicetester_report.xml`.

`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`, poderá testar o mesmo recurso com condições diferentes e o IDT concatenará os valores do recurso para as condições compatíveis. Por exemplo, o trecho a seguir mostra o recurso `MyFeature` com dois valores de recurso separados:

```
...
{
    "Feature": "MyFeature",
    "FeatureValue": "first-feature-supported",
    "Groups": ["first-feature-group"]
},
{
    "Feature": "MyFeature",
    "FeatureValue": "second-feature-supported",
    "Groups": ["second-feature-group"]
},
...
```

Se os dois grupos de teste forem aprovados, o valor do recurso será definido como `first-feature-supported, second-feature-supported`.

`Groups`

Opcional. Uma matriz de IDs de grupos de teste. Todos os testes dentro de cada grupo de teste especificado devem ser aprovados para que o recurso seja compatível.

`OneOfGroups`

Opcional. Uma matriz de IDs de grupos de teste. Todos os testes dentro de pelo menos um dos grupos de teste especificados devem ser aprovados para que o recurso seja compatível.

`TestCases`

Opcional. Uma matriz de IDs de casos de teste. Se especificar este valor, o seguinte se aplica:

- Todos os casos de teste especificados devem ser aprovados para que o recurso seja compatível.

- `Groups` deve conter somente um ID de grupo de teste.
- `OneOfGroups` não deve ser especificado.

`IsRequired`

Opcional. Defina como `false` para marcar este recurso como um recurso opcional no relatório. O valor padrão é `true`.

`ExecutionMethods`

Opcional. Uma matriz de métodos de execução que correspondem ao valor `protocol` especificado no arquivo `device.json`. Se esse valor for especificado, os executores de teste deverão especificar um valor `protocol` que corresponda 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 o estado `AddProductFeatures`, você deve definir o valor de `ResultVar` no estado `RunTask` como um dos seguintes valores:

- Se especificou IDs de casos de teste individuais, defina `ResultVar` como *`group-id_test-id`*`_passed`.
- Se não especificou IDs de casos de teste individuais, defina `ResultVar` como *`group-id`*`_passed`.

O estado `AddProductFeatures` verifica os resultados dos testes da seguinte maneira:

- Se não especificou IDs de casos de teste, o resultado de cada grupo de teste será determinado a partir do valor da variável *`group-id`*`_passed` no contexto da máquina de estado.
- Se especificou IDs de casos de teste, o resultado de cada um dos testes será determinado a partir do valor da variável *`group-id_test-id`*`_passed` no contexto da máquina de estado.

Como tratar erros

Se um ID de grupo fornecido neste estado não for um ID de grupo válido, este estado resultará no erro de execução `AddProductFeaturesError`. Se o estado encontrar um erro de execução, ele também definirá a variável `hasExecutionErrors` no contexto da máquina de estado como `true`.

Relatório

O estado `Report` gera os arquivos *`suite-name`*`_Report.xml` e `awsiotdevicetester_report.xml`. Este 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`

O nome do estado para o qual fazer a transição após a execução das ações no estado atual.

É sempre necessário fazer a transição para o estado `Report` perto do final do fluxo de execução do teste para que os executores de teste possam visualizar os resultados do teste. Normalmente, o próximo estado após este estado é `Succeed`.

Como tratar erros

Se este estado se deparar com problemas ao gerar relatórios, ele emitirá o erro de execução `ReportError`.

LogMessage

O estado `LogMessage` gera o arquivo `test_manager.log` 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`

O nome do estado para o qual fazer a transição após a execução das ações no estado atual.

`Level`

O nível de erro no qual criar a mensagem de log. Se especificar um nível que não seja válido, este estado irá gerar uma mensagem de erro e a descartará.

`Message`

A mensagem a ser registrada.

SelectGroup

O estado `SelectGroup` atualiza o contexto da máquina de estado para indicar quais grupos estão selecionados. Os valores definidos por esse estado são usados por qualquer estado `Choice` seguinte.

```
{
    "Type": "SelectGroup",
    "Next": "<state-name>"
    "TestGroups": [
        <group-id>"
    ]
}
```

Todos os campos que contêm valores são necessários, conforme descrito aqui:

`Next`

O nome do estado para o qual fazer a transição 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, a variável *`group-id`*`_selected` é definida como `true` no contexto. Certifique-se de fornecer IDs de grupos de teste válidos porque o IDT não valida se os grupos especificados existem.

Falha

O estado `Fail` indica que a máquina de estado não foi executada corretamente. Este é um estado final para a máquina de estados e cada definição de máquina de estado deve incluir este estado.

```
{
```

```
    "Type": "Fail"
}
```

Êxito

O estado `Succeed` indica que a máquina de estado foi executada corretamente. Este é um estado final para a máquina de estados e cada definição de máquina de estado deve incluir este estado.

```
{
    "Type": "Succeed"
}
```

Contexto da máquina de estado

O contexto da máquina de estado é um documento JSON somente para leitura que contém dados que estão disponíveis para a máquina de estado durante a execução. O contexto da máquina de estado é acessível somente da máquina de estado e contém informações que determinam o fluxo de teste. Por exemplo, você pode usar as informações configuradas pelos executores de teste no arquivo `userdata.json` para determinar se é necessário executar um teste específico.

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 grupo de dispositivos selecionado para a execução do teste. Para um grupo de dispositivos selecionado, essas informações são recuperadas do elemento correspondente da matriz do grupo de dispositivos de nível superior definido no arquivo `device.json`.

`userData`

As informações no arquivo `userdata.json`.

`config`

As informações fixadas no arquivo `config.json`.

`suiteFailed`

O valor é definido como `false` quando a máquina de estado for iniciada. Se um grupo de teste falhar em um estado `RunTask`, este valor será definido como `true` para a duração restante da execução da máquina de estado.

`specificTestGroups`

Se o executor de teste selecionar grupos de teste específicos para execução em vez de todo o pacote de teste, esta chave será criada e conterá a lista de IDs de grupos de teste específicos.

`specificTestCases`

Se o executor de teste selecionar casos de teste específicos para execução em vez de todo o pacote de teste, esta chave será criada e conterá a lista de IDs de casos de teste específicos.

`hasExecutionErrors`

Não saia quando a máquina de estado é iniciada. Se algum estado encontrar um erro de execução, esta variável será criada e definida como `true` para a duração restante da execução da máquina de estado.

É possível consultar o contexto usando a notação JSONPath. A sintaxe para consultas JSONPath nas definições de estado é `{{$.query}}`. Você pode usar consultas JSONPath como strings de espaço reservado em alguns estados. O IDT substitui as strings de espaço reservado pelo valor da consulta JSONPath avaliada no contexto. É possível usar espaços reservados para os seguintes valores:

- O valor `TestCases` nos estados `RunTask`.
- O valor de `Expression` no estado `Choice`.

Ao 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 como uma string, um número ou um booleano.

Para obter mais informações sobre o uso da notação JSONPath para acessar dados do contexto, consulte Use o contexto do IDT.

Erros de execução

Os 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 arquivo `test_manager.log` e transmite a mensagem de log para o console.

É possível usar os seguintes métodos para lidar com erros de execução:

- Adicione um bloco `Catch` na definição do estado.
- Verifique o valor do valor `hasExecutionErrors` no contexto da máquina de estado.

Catch

Para usar `Catch`, adicione o seguinte à sua 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 fará a transição para o estado especificado em

`Catch.Next`. Consulte cada definição de estado para obter informações sobre o tipo de erro que ela produz.

`Catch.Next`

O próximo estado para o qual fazer a transição se o estado atual encontrar um erro de execução que corresponder a um dos valores especificados em `Catch.ErrorEquals`.

Os blocos de captura são manuseados de maneira sequencial até que um deles corresponda. Se os erros não corresponderem aos listados nos blocos Catch, as máquinas de estado continuarão a ser 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.

tem ExecutionError

Quando alguns estados encontram erros de execução, além de emitirem o erro, eles também definem o valor `hasExecutionError` como `true` no contexto da máquina de estado. É possível usar este valor para detectar quando ocorre um erro e, em seguida, usar um estado `Choice` para fazer a transição da máquina de estado para o estado `Fail`.

Este método tem as características a seguir.

- A máquina de estado não inicia com nenhum valor atribuído para `hasExecutionError` e este valor não está disponível até que um determinado estado a defina. Isto significa que é preciso definir explicitamente o `FallthroughOnError` como `false` para os estados `Choice` que acessam este valor para evitar que a máquina de estado pare se nenhum erro de execução ocorrer.
- Depois de definido como `true`, `hasExecutionError` nunca é definido como falso ou removido do contexto. Isto significa que esse valor é útil somente na primeira vez em que é definido como `true` e, para todos os estados subsequentes, não fornece um valor significativo.
- O valor `hasExecutionError` é compartilhado com todas as máquinas de estado da filial no estado`Parallel`, o que pode resultar em resultados inesperados, dependendo da ordem em que é acessado.

Por conta dessas características, não é recomendado usar este método, e sim, usar um bloco Catch.

Exemplo de máquinas de estado

Esta seção fornece alguns exemplos de configurações de máquina de estado.

Exemplos

- Exemplo de máquina de estado: execute um único grupo de teste
- Exemplo de máquina de estado: execute grupos de teste selecionados pelo usuário
- Exemplo de máquina de estado: execute um único grupo de teste com recursos do produto
- Exemplo de máquina de estado: execute dois grupos de testes em paralelo

Exemplo de máquina de estado: execute um único grupo de teste

Esta máquina de estado:

- Executa o grupo de teste com o ID `GroupA`, que deve estar presente no pacote em um arquivo `group.json`.
- Verifica se há erros de execução e transições de `Fail` para ver se algum foi encontrado.
- Gera um relatório e faz a transição para `Succeed` se não houver erros e, caso contrário, `Fail`.

```
{
    "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: execute grupos de teste selecionados pelo usuário

Esta máquina de estado:

- Verifica se o executor do teste selecionou grupos de teste específicos. A máquina de estado não verifica casos de teste específicos porque os executores de teste não podem selecionar casos de teste sem também selecionar um grupo de teste.
- Se os grupos de teste forem selecionados:
  - Executa os casos de teste nos grupos de teste selecionados. Para fazer isso, a máquina de estado não especifica explicitamente nenhum grupo de teste ou caso de teste no estado `RunTask`.
  - Gera um relatório após executar todos os testes e sai.
- Se os grupos de teste não forem selecionados:
  - Executa testes no grupo de teste `GroupA`.
  - Gera relatórios e sai.

```
{
    "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: execute um único grupo de teste com recursos do produto

Esta máquina de estado:

- Executa o grupo de teste `GroupA`.
- Verifica se há erros de execução e transições de `Fail` para ver se algum foi encontrado.
- Adiciona o recurso `FeatureThatDependsOnGroupA` ao arquivo `awsiotdevicetester_report.xml`:
  - Se `GroupA` for aprovado, o recurso será definido como `supported`.
  - O recurso não está marcado como opcional no relatório.
- Gera um relatório e faz a transição para `Succeed` se não houver erros e, caso contrário, `Fail`

```
{
    "Comment": "Runs GroupA and adds product features based on GroupA",
    "StartAt": "RunGroupA",
    "States": {
        "RunGroupA": {
            "Type": "RunTask",
            "Next": "AddProductFeatures",
            "TestGroup": "GroupA",
            "ResultVar": "GroupA_passed",
            "Catch": [
                {
                    "ErrorEquals": [
                        "RunTaskError"
                    ],
                    "Next": "Fail"
```

```
                }
            ]
        },
        "AddProductFeatures": {
            "Type": "AddProductFeatures",
            "Next": "Report",
            "Features": [
                {
                    "Feature": "FeatureThatDependsOnGroupA",
                    "Groups": [
                        "GroupA"
                    ],
                    "IsRequired": true
                }
            ]
        },
        "Report": {
            "Type": "Report",
            "Next": "Succeed",
            "Catch": [
                {
                    "ErrorEquals": [
                        "ReportError"
                    ],
                    "Next": "Fail"
                }
            ]
        },
        "Succeed": {
            "Type": "Succeed"
        },
        "Fail": {
            "Type": "Fail"
        }
    }
}
```

Exemplo de máquina de estado: execute dois grupos de testes em paralelo

Esta máquina de estado:

- Executa os grupos de teste `GroupA` e `GroupB` em paralelo. As variáveis `ResultVar` armazenadas no contexto pelos estados `RunTask` nas máquinas de estado da ramificação estão disponíveis para o estado `AddProductFeatures`.
- Verifica se há erros de execução e transições de `Fail` para ver se algum foi encontrado. Esta máquina de estado não usa um bloco `Catch` porque este método não detecta erros de execução em máquinas de estado de ramificação.
- Adiciona recursos ao arquivo `awsiotdevicetester_report.xml` com base nos grupos que passam
  - Se `GroupA` for aprovado, o recurso será definido como `supported`.
  - O recurso não está marcado como opcional no relatório.
- Gera um relatório e faz a transição para `Succeed` se não houver erros e, caso contrário, `Fail`

Se dois dispositivos estiverem configurados no grupo de dispositivos, tanto o `GroupA` como o `GroupB` poderão ser executados ao mesmo tempo. No entanto, se um `GroupA` ou `GroupB` tiver vários testes, os dois dispositivos poderão ser alocados para esses testes. Se somente um dispositivo estiver configurado, os grupos de testes 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"
```

```
        }
    }
}
```

Criar executáveis de casos de teste do IDT

Você pode criar e colocar executáveis de casos de teste em uma pasta de conjuntos de testes das seguintes formas:

- Para conjuntos de testes que usam argumentos ou variáveis de ambiente dos arquivos `test.json` para determinar quais testes executar, você pode criar um caso de teste executável único para todo o conjunto de testes ou um executável de teste para cada grupo de testes no conjunto de testes.
- Para um pacote de teste em que você deseja executar testes específicos com base em comandos especificados, você cria um caso de teste executável para cada caso de teste no pacote de teste.

Como redator de teste, é possível determinar qual abordagem é apropriada para seu caso de uso e estruturar o executável do caso de teste de acordo. Certifique-se de fornecer o caminho correto do executável do caso de teste em cada arquivo `test.json` 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 arquivo `test.json` e passa as variáveis de ambiente necessárias para o processo.

Use o SDK do cliente do IDT

Os SDKs do IDT Client permitem simplificar a forma como a lógica de teste é escrita em seu executável de teste com comandos de API que podem ser usados para interagir com o IDT e seus dispositivos em teste. No momento, o IDT fornece os seguintes SDKs:

- SDK do cliente IDT para Python
- SDK for Go SDK para Go

Esses SDKs estão localizados na pasta *<device-tester-extract-location>*/sdks. Ao criar um novo executável de caso de teste, é preciso 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 do dispositivo
- Interação do IDT
- Interação do host

Interação do dispositivo

Os comandos a seguir permitem que a comunicação com o dispositivo em teste sem precisar implementar nenhuma interação adicional com o dispositivo e as funções de gerenciamento de conectividade.

`ExecuteOnDevice`

Permite que pacotes de teste executem comandos shell em um dispositivo compatível com conexões SSH ou Docker shell.

`CopyToDevice`

Permite que os pacotes de teste copiem um arquivo local da máquina host que executa o IDT para um local especificado em um dispositivo que suporte conexões SSH ou Docker shell.

`ReadFromDevice`

Permite que os pacotes de teste leiam a partir da porta serial de dispositivos compatíveis com conexões UART.

> **Note**
>
> Como o IDT não gerencia conexões diretas com dispositivos que são feitas usando as informações de acesso a dispositivos do contexto, recomendamos usar esses comandos

da API de interação de dispositivos em seus executáveis de casos de teste. No entanto, se esses comandos não atenderem aos requisitos do 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 do pacote de teste.
Para fazer uma conexão direta, recupere as informações nos campos `device.connectivity` e `resource.devices.connectivity` do dispositivo em teste e dos dispositivos de recursos, respectivamente. Para obter mais informações sobre como usar contexto do IDT, consulte Use o contexto do IDT.

Interação do IDT

Os comandos a seguir permitem que os conjuntos de teste se comuniquem com o IDT.

`PollForNotifications`

Permite que os pacotes de teste verifiquem as notificações do IDT.

`GetContextValue` e `GetContextString`

Permite que os pacotes de teste recuperem valores do contexto do IDT. Para obter mais informações, consulte Use o contexto do IDT.

`SendResult`

Permite que os pacotes de teste relatem os resultados dos casos de teste ao IDT. Este comando deve ser chamado no final de cada caso de teste em um pacote de teste.

Interação do host

O comando a seguir permite que seus pacotes de teste se comuniquem com a máquina host.

`PollForNotifications`

Permite que os pacotes de teste verifiquem as notificações do IDT.

`GetContextValue` e `GetContextString`

Permite que os pacotes de teste recuperem valores do contexto do IDT. Para obter mais informações, consulte Use o contexto do IDT.

`ExecuteOnHost`

Permite que os pacotes de teste executem comandos na máquina local e permite que o IDT gerencie o ciclo de vida do executável do caso de teste.

Habilitar comandos da CLI no IDT

O comando `run-suite` da CLI do IDT fornece várias opções que permitem que o executor de teste personalize a execução do teste. Para permitir que os executores de teste usem estas opções para executar seu pacote de teste personalizado, você implementa o suporte para a CLI do IDT. Se 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 o comando `run-suite` 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 comando `run-suite`, o IDT o usa para calcular o valor da variável de ambiente IDT_TEST_TIMEOUT e define o campo `config.timeoutMultiplier` no contexto do IDT. Para apoiar este argumento, você deve fazer o seguinte:

- Em vez de usar diretamente o valor de tempo limite do arquivo `test.json`, leia a variável de ambiente IDT_TEST_TIMEOUT para obter o valor de tempo limite calculado corretamente.
- Recupere o valor `config.timeoutMultiplier` do contexto do IDT e aplique-o a tempos limite de execução prolongados.

Para obter mais informações sobre como sair mais cedo por conta de eventos de tempo limite, consulte Especifique 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 no comando `run-suite`, o IDT interrompe a execução dos testes assim que encontrar uma falha. No entanto, se os casos de teste estiverem sendo executados em paralelo, 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 sair antecipadamente em falhas, consulte Especifique 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 os comandos `run-suite` 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, a máquina de estados do seu pacote de teste deve incluir um conjunto específico de estados `RunTask` e `Choice` em sua máquina de estado. Se não estiver usando uma máquina de estado personalizada, a máquina de estado IDT padrão incluirá os estados necessários e você não precisará realizar nenhuma ação adicional. No entanto, se estiver usando uma máquina de estado personalizada, use Exemplo de máquina de estado: execute grupos de teste selecionados pelo usuário como amostra para adicionar os estados necessários à sua máquina de estado.

Para obter mais informações sobre os comandos da CLI no IDT, consulte Depure e execute conjuntos de teste personalizados.

Gravar logs de eventos

Enquanto o teste está sendo executado, você envia dados para `stdout` e `stderr` e grava logs de eventos e mensagens de erro no console. Para obter informações sobre o formato das mensagens do console, consulte Formato de mensagem do console.

Quando o IDT terminar de executar o pacote de teste, essas informações também estarão disponíveis no arquivo `test_manager.log` localizado na pasta *<devicetester-extract-location>*`/results/`*<execution-id>*`/logs`.

É possível configurar cada caso de teste para gravar os logs de sua execução de teste, incluindo logs do dispositivo em teste, no arquivo *<group-id>*`_`*<test-id>* localizado na pasta *<device-tester-extract-location>*`/results/`*execution-id*`/logs`. Para fazer isso, recupere o caminho para o arquivo de log do contexto do IDT com a consulta `testData.logFilePath`, crie

um arquivo nesse caminho e grave o conteúdo deseja 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.

É possível configurar seu executável de texto para criar arquivos de log adicionais, conforme necessário, na pasta *<device-tester-extract-location>*/logs. É recomendado especificar prefixos exclusivos para nomes de arquivos de log para que seus arquivos não sejam substituídos.

Relatar resultados ao IDT

O IDT grava os resultados do teste nos arquivos awsiotdevicetester_report.xml e *suite-name*_report.xml. Estes arquivos de relatório estão localizados em *<device-tester-extract-location>*/results/*<execution-id>*/. Ambos os relatórios capturam os resultados da execução do pacote de teste. Para obter mais informações sobre os esquemas que o IDT usa para esses relatórios, consulte Analise os resultados e logs dos testes do IDT

Para preencher o conteúdo do arquivo *suite-name*_report.xml, o comando SendResult deve ser usado para relatar os resultados do teste ao IDT antes da conclusão da execução do teste. Se o IDT não conseguir localizar os resultados de um teste, ele emitirá um erro para o caso de teste. O seguinte trecho em Python mostra os comandos para enviar um resultado de teste para o IDT:

```
request-variable = SendResultRequest(TestResult(result))
client.send_result(request-variable)
```

Se não reportar os resultados por meio da API, o IDT procurará os resultados do teste na pasta de artefatos de teste. O caminho para essa pasta é armazenado no campo testData.testArtifactsPath no contexto do IDT. Nesta pasta, o IDT usa o primeiro arquivo XML classificado em ordem alfabética localizado como resultado do teste.

Se sua lógica de teste produzir resultados XML JUnit, será possível gravar os resultados do teste em um arquivo XML na pasta de artefatos para fornecer os resultados diretamente ao IDT em vez de analisá-los e depois usar a API para enviá-los ao IDT.

Se usar este método, a lógica de teste deverá resumir com precisão os resultados do teste e formatar o arquivo de resultados no mesmo formato do arquivo *suite-name*_report.xml. O IDT não realiza nenhuma validação dos dados fornecidos, com as seguintes exceções:

- O IDT ignora todas as propriedades da tag testsuites. Em vez disso, ele calcula as propriedades da tag a partir dos resultados de outros grupos de teste relatados.
- Pelo menos uma tag testsuite deve existir nos testsuites.

Como o IDT usa a mesma pasta de artefatos para todos os casos de teste e não exclui os arquivos de resultados entre as execuções de teste, esse método também pode gerar relatórios incorretos, caso o IDT leia o arquivo incorreto. É recomendado o uso do 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 o IDT usar. Embora seja possível usar uma abordagem mista para gerar relatórios em pacotes de teste, ou seja, usar um arquivo de resultados XML para alguns casos de teste e enviar resultados por meio da API em outros casos, não recomendamos essa abordagem.

Especifique o comportamento de saída

Configure seu executável de texto para sempre sair com um código de saída 0, mesmo se um caso de teste relatar 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 caso o executável do caso de teste não tenha comunicado nenhum resultado ao IDT. Quando o IDT recebe um código de saída diferente de zero, ele marca que o caso de teste se deparou com um erro que o impediu de ser executado.

O IDT pode solicitar ou esperar a interrupção da execução de um caso de teste antes de concluir os eventos a seguir. Use estas 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 arquivo `test.json`. Se o executor do teste usou o argumento `timeout-multiplier` para especificar um multiplicador de tempo limite, o IDT calcula o valor do tempo limite com o multiplicador.

Para detectar este evento, use a variável de ambiente IDT_TEST_TIMEOUT. Quando um executor de teste inicializa 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. É possível ler o valor da variável para definir um cronômetro apropriado.

Interromper

Ocorre quando o executor do teste interrompe o IDT. Por exemplo, ao pressionar Ctrl+C.

Como os terminais propagam sinais para todos os processos secundários, você pode simplesmente configurar um manipulador de sinais em seus casos de teste para detectar sinais de interrupção.

Como alternativa, é possível sondar periodicamente a API para verificar o valor da `CancellationRequested` booleana na resposta da API `PollForNotifications`. Quando o IDT recebe um sinal de interrupção, ele define o valor do `CancellationRequested` booleano como `true`.

Interrompa na primeira falha

Ocorre quando um caso de teste executado paralelamente ao caso de teste atual falha e o executor de teste usa o argumento `stop-on-first-failure` para especificar que o IDT deve interromper ao se deparar com uma falha.

Para detectar esse evento, é possível pesquisar periodicamente a API para verificar o valor `CancellationRequested` do booleano na resposta da API `PollForNotifications`. Quando o IDT encontra uma falha e é configurado para interromper a primeira falha, ele define o valor do `CancellationRequested` booleano como `true`.

Quando um desses eventos ocorre, o IDT espera cinco minutos para que os casos de teste em execução no momento concluam a execução. Se todos os casos de teste em execução não saírem em cinco minutos, o IDT forçará a interrupção de cada um dos processos deles. Se o IDT não tiver recebido os resultados do teste antes da conclusão dos processos, ele marcará os casos de teste como expirados. Como prática recomendada, os seus casos de teste devem executar as seguintes ações quando encontrarem um dos eventos:

1. Interrompa a execução da lógica de teste normal.
2. Limpar todos os recursos temporários, como artefatos de teste no dispositivo em teste.
3. Relate um resultado de teste ao IDT, como uma falha ou erro no teste.
4. Sair.

Use o contexto do IDT

Quando o IDT executa um pacote de teste, o pacote de teste pode acessar um conjunto de dados que podem ser usados para determinar como cada teste é executado. Estes dados são chamados de contexto do IDT. Por exemplo, a configuração de dados do usuário fornecida pelos executores de teste em um arquivo `userdata.json` é disponibilizada para pacotes de teste no contexto do IDT.

O contexto do IDT pode ser considerado um documento JSON somente para leitura. Os pacotes de teste podem recuperar e gravar dados no contexto usando tipos de dados JSON padrão, como objetos, matrizes, números e assim por diante.

Esquema de contexto

O contexto do 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 do arquivo `config.json`. O campo `config` também contém o seguinte campo adicional:

`config.timeoutMultiplier`

O multiplicador de um valor de tempo limite usado pelo conjunto de testes. Este valor é especificado pelo executor de teste da CLI do IDT. O valor padrão é 1.

`device`

Informações sobre o dispositivo selecionado para a execução do teste. Estas informações são equivalentes à matriz `devices` definida no arquivo `device.json` do dispositivo selecionado.

`devicePool`

Informações sobre o grupo de dispositivos selecionado para a execução do teste. Estas informações são equivalentes ao elemento de matriz do grupo de dispositivos de nível superior definido no arquivo `device.json` do grupo de dispositivos selecionado.

`resource`

As informações sobre dispositivos de recursos do arquivo `resource.json`.

`resource.devices`

Estas informações são equivalentes à matriz `devices` definida no arquivo `resource.json`. Cada elemento `devices` inclui o seguinte campo adicional:

`resource.device.name`

O nome do dispositivo de recurso. Este valor é definido como o valor `requiredResource.name` no arquivo `test.json`.

`testData.awsCredentials`

As AWS credenciais usadas pelo teste para se conectar à AWS nuvem. Estas informações são obtidas do arquivo `config.json`.

`testData.logFilePath`

O caminho para o arquivo de log no qual o caso de teste grava mensagens de log. O pacote de teste criará este arquivo se ele não existir.

`userData`

As informações fornecidas pelo executor de teste no arquivo `userdata.json`.

Acessar dados no contexto

É possível consultar o contexto usando a notação JSONPath dos arquivos JSON e do executável de texto com as APIs `GetContextValue` e `GetContextString`. A sintaxe das cadeias de caracteres JSONPath para acessar o contexto do IDT varia da seguinte forma:

- Em `suite.json` e `test.json`, é usado `{{`*`query`*`}}`. Ou seja, não use o elemento raiz `$.` para iniciar sua expressão.
- Em `statemachine.json`, é usado `{{$.`*`query`*`}}`.
- Nos comandos da API, é usado *`query`* ou `{{$.`*`query`*`}}`, dependendo do comando. Para obter mais informações, consulte a documentação em linha dos SDKs.

A tabela a seguir descreve os operadores em uma expressão JSONPath típica:

Operador	Description
`$`	O elemento raiz. Como o valor de contexto de nível superior do IDT é um objeto, normalmente você usará `$.` para iniciar suas consultas.
`.childName`	Acessa o elemento filho com o nome `childName` de um objeto. Se aplicado a uma matriz, produz uma nova matriz com esse operador aplicado a cada elemento. O nome do elemento diferencia maiúsculas e minúsculas. Por exemplo, a consulta para acessar o valor `awsRegion` no objeto `config` é `$.config.awsRegion`.
`[start:end]`	Filtra elementos de uma matriz, recuperando itens que começam no índice `start` e vão até o índice `end`, ambos inclusive.
`[index1, index2, ... , indexN]`	Filtra elementos de uma matriz, recuperando itens somente dos índices especificados.

Operador	Description
`[?(expr)]`	Filtra elementos de uma matriz usando a expressão `expr`. Esta expressão que deve ser avaliada para um valor booleano.

Para criar expressões de filtro, use os seguintes elementos:

```
<jsonpath> | <value> operator <jsonpath> | <value>
```

Nesta sintaxe:

- `jsonpath` é um JSONPath que usa a sintaxe JSON padrão.
- `value` é qualquer valor personalizado que usa a sintaxe JSON padrão.
- `operator` é um dos seguintes operadores:
  - < (Menor que)
  - <= (Menor ou igual a)
  - == (Igual a)

    Se o JSONPath ou o valor em sua expressão for um valor de matriz, booleano ou objeto, este será o único operador binário compatível que será possível usar.
  - >= (Maior ou igual a)
  - > (Maior que)
  - =~ (Correspondência de expressão regular). Para usar este operador em uma expressão de filtro, o JSONPath ou valor no lado esquerdo da expressão deve ser avaliado como uma string e o lado direito deve ser um valor padrão que siga a sintaxe RE2.

Você pode usar consultas JSONPath no formato {{*query*}} como cadeias de caracteres de espaço reservado nos campos e nos arquivos `args` e `environmentVariables` nos campos nos `test.json` arquivos. `environmentVariables` `suite.json` O IDT realiza uma pesquisa de contexto e preenche os campos com o valor avaliado da consulta. Por exemplo, no arquivo `suite.json`, é possível 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, ao usar strings de caracteres de espaço reservado em arquivos `test.json` e `suite.json`, as seguintes considerações se aplicam às suas consultas:

- Cada ocorrência da chave `devicePool` em sua consulta deve estar em letras minúsculas. Ou seja, em vez disso, use `devicepool`.
- Para as matrizes, somente matrizes de strings podem ser usadas. Além disso, as matrizes usam um formato `item1, item2,...,itemN` não padrão. Se a matriz contiver somente um elemento, ela será serializada como `item`, tornando-a indistinguível de um campo de strings.
- Não é possível usar espaços reservados para recuperar objetos do contexto.

Por conta dessas considerações, é recomendado, sempre que possível, usar a API para acessar o contexto em sua lógica de teste em vez de strings de espaço reservado em arquivos `test.json` e `suite.json`. No entanto, em alguns casos, pode ser mais conveniente usar espaços reservados JSONPath para recuperar strings únicas para definir como variáveis de ambiente.

Definir configurações para executores de teste

Para executar conjuntos de testes personalizados, os executores de teste devem definir suas configurações com base no pacote de testes que desejam executar. As configurações são especificadas com base nos modelos de arquivo de configuração JSON localizados na pasta *<device-tester-extract-location>*`/configs/`. Se necessário, os executores de teste também devem configurar AWS as credenciais que a IDT usará para se conectar à nuvem. AWS

Como escritor de teste, será necessário configurar esses arquivos para depurar seu pacote de teste. É preciso fornecer instruções aos executores de teste para que eles possam definir as seguintes configurações conforme necessário para executar seus pacotes de testes.

Configurar device.json

O arquivo `device.json` contém informações sobre os dispositivos em que 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 workload. 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 quiser listar sua placa no Catálogo de AWS Partner dispositivos, o SKU especificado aqui deve corresponder ao SKU que você usa no processo de listagem.

`features`

Opcional. Uma matriz que contém recursos compatíveis com o dispositivo. Os recursos do dispositivo são valores definidos pelo usuário que você configura em seu pacote de teste. É preciso fornecer informações aos executores de teste sobre os nomes e valores dos recursos a serem incluídos no arquivo `device.json`. Por exemplo, se você quiser testar um dispositivo que funciona como um servidor MQTT para outros dispositivos, você pode configurar sua lógica de teste para validar níveis específicos suportados para um recurso chamado `MQTT_QOS`. Os executores de teste fornecem esse nome de recurso e definem o valor do recurso para os níveis de QOS compatíveis com o dispositivo deles. É possível recuperar as informações fornecidas do contexto do IDT com a consulta `devicePool.features` ou do contexto da máquina de estado com a consulta `pool.features`.

`features.name`

O nome do recurso.

`features.value`

Os valores dos recursos compatíveis.

`features.configs`

Definições de configuração, se necessárias, para o recurso.

`features.config.name`

O nome do ajuste de configurações.

`features.config.value`

Os valores de configuração compatíveis.

`devices`

Uma matriz de dispositivos no grupo a serem testados. 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 usada nas 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 `docker`.

`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 `docker`.

> **Note**
>
> Para verificar se os executores de teste configuram a conexão incorreta do dispositivo para um teste, você pode recuperar o `pool.Devices[0].Connectivity.Protocol` do contexto da máquina de estado e compará-la com o valor esperado em um estado `Choice`. Se um protocolo incorreto for usado, imprima uma mensagem usando o estado `LogMessage` e faça a transição para o estado `Fail`.
> Como alternativa, é possível usar o código de tratamento de erros para relatar uma falha no teste para tipos de dispositivos incorretos.

(Opcional) Configurar userdata.json

O arquivo `userdata.json` contém qualquer informação adicional exigida por um pacote de teste, mas não está especificada no arquivo `device.json`. O formato deste arquivo depende do [arquivo `userdata_scheme.json`] definido no pacote de teste. Se for um redator de testes, certifique-se de fornecer essas informações aos usuários que executarão os pacotes de testes escritos.

(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 dispositivos necessários para testar determinados recursos de um dispositivo em teste. Por exemplo, para testar a capacidade Bluetooth de um dispositivo, é possível usar um dispositivo de recurso para testar se seu dispositivo consegue se conectar com êxito. Os dispositivos de recursos são opcionais e pode exigir quantos dispositivos de recursos precisar. Como redator de teste, é possível usar 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 pool de dispositivos de recursos com os recursos necessários. Certifique-se de fornecer essas informações aos usuários que executarão os pacotes de testes escritos.

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-value>",
                "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 workload. Vários dispositivos são usados para executar testes diferentes.

`features`

Opcional. Uma matriz que contém recursos compatíveis com o dispositivo. As informações necessárias neste campo são definidas nos arquivos test.json no pacote de teste e determinam quais testes devem ser executados e como executá-los. Se o pacote de teste não exigir nenhum recurso, este campo não será obrigatório.

`features.name`

O nome do recurso.

`features.version`

A versão do recurso.

`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 serem testados. 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 usada nas 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 `docker`.

`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 `docker`.

(Opcional) Configurar 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 suas credenciais de AWS usuário para o IDT e, opcionalmente, para uma região. AWS Se AWS as credenciais com as permissões necessárias forem fornecidas, AWS IoT o Device Tester coleta e envia métricas de uso

para o. AWS Esse atributo é opcional e usado para aprimorar a funcionalidade do IDT. Para obter mais informações, consulte Métricas de uso do IDT.

Os executores de teste podem configurar suas AWS credenciais de uma das seguintes formas:

- 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 estão disponíveis após o encerramento da sessão. O IDT pode usar as variáveis de `AWS_SECRET_ACCESS_KEY` ambiente `AWS_ACCESS_KEY_ID` e para armazenar credenciais 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 AWS as credenciais para o IDT, os executores de teste editam a `auth` seção no `config.json` arquivo 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 no*<device-tester-extract-location>*.

`configFiles.root`

O caminho para a pasta que contém os arquivos de configuração.

`configFiles.device`

O caminho para a função `device.json`.

`testPath`

O caminho para a pasta que contém pacotes de teste.

`reportPath`

O caminho para a pasta que conterá os resultados do teste depois que o IDT executar um pacote de teste.

`awsRegion`

Opcional. A AWS região que as suítes de teste usarão. Se não for definida, os pacotes de teste usarão a região padrão especificada em cada pacote de teste.

`auth.method`

O método que o IDT usa para recuperar AWS as credenciais. Os valores aceitos são `file` recuperar credenciais de um arquivo de credenciais e recuperar credenciais `environment` usando variáveis de ambiente.

`auth.credentials.profile`

O perfil de credenciais a ser usado no 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 teste completo depende do hardware e da composição do pacote de teste. Para referência, são necessários aproximadamente 30 minutos para concluir o conjunto completo de testes de AWS IoT Greengrass qualificação em um Raspberry Pi 3B.

À medida que seu pacote de teste é escrito, é possível usar o IDT para executar o pacote de teste no modo de depuração para verificar seu código antes de executá-lo ou fornecê-lo aos executores de teste.

Executar o IDT no modo de depuração

Como os pacotes de teste dependem do IDT para interagir com dispositivos, fornecer o contexto e receber resultados, não é possível simplesmente depurar seus pacotes de teste em um IDE sem qualquer interação 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 comando a seguir 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 conjunto de testes nem executa a máquina de estado; em vez disso, ele interage com o IDE para responder às solicitações

feitas do conjunto de testes em execução no IDE e exibe 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 a máquina de estado e não gera arquivos 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. O IDT não lerá essas informações de `test.json` ou `suite.json`.
- Argumentos para selecionar os dispositivos de recursos. O IDT não lerá essas informações de `test.json`.

Para depurar seus pacotes de teste, conclua as seguintes etapas:

1. Crie os arquivos de ajuste de configuração necessários para executar o pacote de teste. Por exemplo, se seu pacote de teste exigir o `device.json`, `resource.json` e `user data.json` certifique-se de configurar 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 pacote de teste e responde a elas. O IDT também gera as variáveis de ambiente necessárias para o processo de caso do SDK do cliente de IDT.
3. No seu IDE, use a configuração `run` ou `debug` para fazer o seguinte:
   a. Defina os valores das variáveis de IDT-generated ambiente.
   b. Defina o valor de qualquer variável de ambiente ou argumento que você especificou em seu arquivo `test.json` e `suite.json`.
   c. Definir pontos de interrupção, conforme necessário.
4. Execute o pacote de teste em seu IDE.

   É possível depurar e executar novamente o pacote de teste quantas vezes 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

As seções a seguir descrevem 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 sua versão do IDT, neste caso AWS IoT Greengrass as versões, e as versões do pacote de testes de AWS IoT Greengrass qualificação 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. Algumas opções comumente usadas a seguir:

- `suite-id`. A versão do pacote de teste 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, como 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, como uma lista separada por vírgulas. Quando especificado, `group-id` deve especificar um único grupo.
- `pool-id`. O grupo de dispositivos a ser testado. É preciso especificar um grupo se houver 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 de modo a 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 pacote de teste. Isto é necessário somente se `userdataRequired` estiver definido como verdadeiro no arquivo `suite.json` do pacote de teste.

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 pacote de teste no modo de depuração. Para obter mais informações, consulte Executar o IDT no modo de depuração.

Analise os resultados e logs dos testes do IDT

Esta seção descreve o formato no qual o IDT gera logs de console e relatórios de teste.

Formato de mensagem do console

AWS IoT O Device Tester usa um formato padrão para imprimir mensagens no console quando inicia uma suíte 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 detalhadas do console consiste nos seguintes campos:

`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 `info`, `warn` ou `error`. O IDT emite uma mensagem `fatal` ou `panic` se encontrar um evento esperado que faça com que ele saia antecipadamente.

`msg`

A mensagem registrada.

`executionId`

Uma string de ID exclusiva para o processo atual do IDT. Este ID é usado para diferenciar entre execuções individuais de IDT.

As mensagens do console geradas de um pacote de teste fornecem informações adicionais sobre o dispositivo em teste e o pacote de teste, o grupo de teste e os casos de teste que o IDT executa. O trecho a seguir mostra um exemplo de uma mensagem de console gerada de um pacote de teste.

```
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 pacote de teste da mensagem do console contém os seguintes campos:

`suiteId`

O nome do pacote de teste em execução no momento.

`groupId`

O ID do grupo de teste em execução no momento.

`testCaseId`

O ID do caso de teste em execução no momento.

`deviceId`

Um ID do dispositivo em teste que está sendo usado pelo caso de teste atual no momento.

Para exibir um resumo do teste no console quando um IDT terminar de executar um teste, você deve incluir um [estado `Report`] na máquina de estado. O resumo do teste contém informações sobre o

conjunto de testes, os resultados de teste de cada grupo em execução e os locais dos arquivos de logs e relatórios gerados. O exemplo a seguir mostra uma mensagem de resume do 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 Esquema do relatório do Device Tester

`awsiotdevicetester_report.xml` é um relatório assinado que contém as seguintes informações:

- A versão IDT.
- A versão do pacote de teste.
- A assinatura do relatório e a chave usada para assinar o relatório.
- A SKU do dispositivo e o nome de grupo do dispositivo especificados no arquivo `device.json`.
- A versão do produto e os recursos do dispositivo que foram testados.
- O resumo agregado dos resultados de teste. Estas informações são as mesmas contidas no arquivo *`suite-name`*`_report.xml`.

```
<apnreport>
    <awsiotdevicetesterversion>idt-version</awsiotdevicetesterversion>
    <testsuiteversion>test-suite-version</testsuiteversion>
```

```
    <signature>signature</signature>
    <keyname>keyname</keyname>
    <session>
        <testsession>execution-id</testsession>
        <starttime>start-time</starttime>
        <endtime>end-time</endtime>
    </session>
    <awsproduct>
        <name>product-name</name>
        <version>product-version</version>
        <features>
            <feature name="<feature-name>" value="supported | not-supported | <feature-
value>" type="optional | required"/>
        </features>
    </awsproduct>
    <device>
        <sku>device-sku</sku>
        <name>device-name</name>
        <features>
            <feature name="<feature-name>" value="<feature-value>"/>
        </features>
        <executionMethod>ssh | uart | docker</executionMethod>
    </device>
    <devenvironment>
        <os name="<os-name>"/>
    </devenvironment>
    <report>
        <suite-name-report-contents>
    </report>
</apnreport>
```

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. Os recursos marcados como `required` são necessários para que o pacote de teste valide o dispositivo. O trecho a seguir mostra como essas informações aparecem no arquivo `awsiotdevicetester_report.xml`.

```
<feature name="ssh" value="supported" type="required"></feature>
```

Os recursos marcados como não `optional` 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 do relatório do pacote de teste

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 teste.

```
<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 tanto em `awsiotdevicetester_report.xml` como em *`suite-name`*`_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="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 pacote de teste.

`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 recurso 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 recurso `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, o AWS IoT Device Tester coleta e envia 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 comandos da CLI do IDT usados para executar testes

- O pacote de teste que é executado
- As suítes de teste na *<device-tester-extract-location>* pasta
- 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 arquivo `config.jso` n para acessá-las.

Configure seu AWS credenciais

Se você ainda não tiver um Conta da AWS, deverá 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 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 Console de gerenciamento da AWS 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.

      JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iot-device-tester:SendMetrics"
            ],
            "Resource": "*"
        }
    ]
}
```

   c. Selecione Next: Tags (Próximo: tags).

   d. Selecione Next: Review (Próximo: revisar).

   e. Em Name (Nome), insira **`IDTUsageMetricsIAMPermissions`**. Em Summary (Resumo), revise as permissões concedidas pela política.

   f. 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 Definir permissões, selecione Anexar políticas existentes diretamente.

      ii. Procure a IDTUsageMetricsIAMPermissionspolítica 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, escolha Download.csve salve o arquivo em um local seguro. Você usa essas informações posteriormente para configurar seu arquivo de AWS credenciais.

Para 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. Se você já configurou permissões no console, vá para the section called “Configure seu dispositivo para executar testes de IDT” ou the section called “(Opcional): configurar o contêiner do Docker”.

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, macOS, 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.

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. *user-name*Substitua pelo nome de usuário do IAM e, *<account-id>* no comando, pelo ID do seu 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 para o 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 as variáveis de ambiente, execute o seguinte comando:

   ```
   AWS_ACCESS_KEY_ID=access-key
   AWS_SECRET_ACCESS_KEY=secret-access-key
   ```

   b. Para o arquivo de credenciais, adicione as seguintes informações para `.aws/credentials file:`

   ```
   [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 obter mais informações, consulte (Opcional) Configurar config.json.

IDT para AWS IoT Greengrass solução de problemas

O IDT para AWS IoT Greengrass gravar esses erros em vários locais com base no tipo de erro. Os erros são gravados no console, arquivos de log e relatórios de teste.

Códigos de erro

A tabela a seguir lista os códigos de erro gerados pelo IDT para AWS IoT Greengrass.

Código de erro	Nome do código de erro	Possível causa raiz	Solução de problemas
101	InternalError	Ocorreu um erro interno.	Verifique os logs no diretório *<device-tester-extract-location>* /

Código de erro	Nome do código de erro	Possível causa raiz	Solução de problemas
			`results`. Se você não conseguir depurar o problema, entre em contato com o Suporte ao desenvolvedor da AWS.
102	TimeoutError	O teste não pode ser concluído em um período limitado. Isso poderá acontecer se: • Houver uma conexão de rede lenta entre computador de teste e dispositivo (por exemplo, se você estiver usando a rede da VPN). • Uma rede lenta atrasa a comunicação entre o dispositivo e a nuvem. • O campo `timeout` nos arquivos de configuração de teste (`test.json`) foi modificado incorretamente.	• Verifique a conexão de rede e a velocidade. • Verifique se você não modificou nenhum arquivo no diretório `/test`. • Tente executar manualmente o grupo de testes com falha usando o sinalizador `"--group-id"`. • Tente executar o conjunto de testes aumentando o tempo limite do teste. Para obter mais informações, consulte Erros de tempo limite.

Código de erro	Nome do código de erro	Possível causa raiz	Solução de problemas
103	PlatformNotSupportError	OS/architecture Combinação incorreta especificada em `device.json` .	Alterar sua configuração para uma das combinações compatíveis: • Linux, x86_64 • Linux, ARMv6l • Linux, ARMv7l • Linux, AArch64 • Ubuntu, x86_64 • OpenWRT, ARMv7l • OpenWRT, AArch64 Para obter mais informações, consulte Configurar device.json.

Código de erro	Nome do código de erro	Possível causa raiz	Solução de problemas
104	VersionNotSupportError	A versão do software AWS IoT Greengrass Core não é compatível com a versão do IDT que você está usando.	Use o device_tester_bin version comando para encontrar a versão compatível do software AWS IoT Greengrass Core. Por exemplo, se você estiver usando macOS, use: ./devicetester_mac_x86_64 version. Para encontrar a versão do software AWS IoT Greengrass Core que você está usando: • Se você estiver executando testes com o software AWS IoT Greengrass Core pré-instalado, use o SSH para se conectar ao seu dispositivo AWS IoT Greengrass principal e executar ***<path-to-preinstalled-green***

Código de erro	Nome do código de erro	Possível causa raiz	Solução de problemas
			grass-location> /greengrass/ggc/core/greengrassd --version • Se você estiver executando testes com uma versão diferente do software AWS IoT Greengrass Core, acesse o `devicetester_greengrass_` ***<os>***`/products/greengrass/gcc` diretório. A versão do software AWS IoT Greengrass Core faz parte do nome do arquivo.zip. Você pode testar uma versão diferente do software AWS IoT Greengrass Core. Para obter mais informações, consulte Conceitos básicos de AWS IoT Greengrass.

Código de erro	Nome do código de erro	Possível causa raiz	Solução de problemas
105	LanguageNotSupportError	O IDT oferece suporte a Python somente AWS IoT Greengrass para bibliotecas e SDKs.	Certifique-se de que: • O pacote do SDK em `devicetester_greengrass_ <os>/products/greengrass/ggsdk` é o Python SDK. • O conteúdo da pasta `bin` em `devicetester_greengrass_ <os>/tests/GGQ_1.0.0/suite/resources/run.runtimefarm/bin` não foi alterado.

Código de erro	Nome do código de erro	Possível causa raiz	Solução de problemas
106	ValidationError	Alguns campos em `device.json` ou `config.json` são inválidos.	Verifique a mensagem de erro no lado direito do código de erro no relatório. • Tipo de autenticação inválido para dispositivo: especifique o método correto para se conectar ao seu dispositivo. Para obter mais informações, consulte the section called “Configurar device.json”. • Caminho da chave privada inválido: especifique o caminho correto para a chave privada. Para obter mais informações, consulte Configurar device.json. • Inválido Região da AWS: especifique um válido Região da AWS em seu `config.json`

Código de erro	Nome do código de erro	Possível causa raiz	Solução de problemas
			arquivo. Para obter mais informações, consulte Endpoints de serviço da AWS. • AWS credencia is: defina AWS credenciais válidas em sua máquina de teste (usando variáveis de ambiente ou o `credentials` arquivo). Verifique se o campo `auth` está configura do corretame nte. Para obter mais informaçõ es, consulte the section called “Crie e configure um Conta da AWS”. • Entrada HSM inválida: verifique seus campos `p11Provid er` , `privateKe yLabel` , `slotLabel` , `slotUserPin` , e `openSSLEngine` no `device.json` .

Código de erro	Nome do código de erro	Possível causa raiz	Solução de problemas
107	SSHConnectionFailed	O computador de teste não pode se conectar ao dispositivo configurado.	Verifique se os campos a seguir do arquivo `device.json` estão corretos: • `ip` • `user` • `privKeyPath` • `password` Para obter mais informações, consulte Configurar device.json.

Código de erro	Nome do código de erro	Possível causa raiz	Solução de problemas
108	RunCommandError	O teste não conseguiu executar um comando no dispositivo em teste.	Verifique se o acesso raiz ao usuário configurado em `device.json` é permitido. Uma senha é exigida por alguns dispositivos ao executar comandos com acesso raiz. Verifique se o acesso raiz é permitido sem uma senha. Para obter mais informações, consulte a documentação do seu dispositivo. Tente executar o comando com falha manualmente no dispositivo e verifique se ocorre um erro.
109	PermissionDeniedError	Sem acesso raiz.	Defina o acesso raiz do usuário configurado no seu dispositivo.
110	CreateFileError	Não é possível criar um arquivo.	Verifique o espaço em disco do dispositivo e as permissões do diretório.

Código de erro	Nome do código de erro	Possível causa raiz	Solução de problemas
111	CreateDirError	Não é possível criar um diretório.	Verifique o espaço em disco do dispositivo e as permissões do diretório.
112	InvalidPathError	O caminho para o software AWS IoT Greengrass principal está incorreto.	Verifique se o caminho na mensagem de erro é válido. Não edite os arquivos do diretório `devicetester_greengrass_` *<os>*.
113	InvalidFileError	Um arquivo é inválido.	Verifique se o arquivo na mensagem de erro é válido.

Código de erro	Nome do código de erro	Possível causa raiz	Solução de problemas
114	ReadFileError	Não é possível ler o arquivo especificado.	Verifique o seguinte: • As permissões do arquivo estão corretas. • `limits.config` permite que arquivos suficientes sejam abertos. • O arquivo especificado na mensagem de erro existe e é válido. Se você estiver testando em um macOS, aumente o limite de arquivos abertos. O limite padrão é 256, que é o suficiente para testes.

Código de erro	Nome do código de erro	Possível causa raiz	Solução de problemas
115	FileNotFoundError	Um arquivo necessário não foi encontrado.	Verifique o seguinte: • Um arquivo do Greengrass compactado existe em `devicetester_greengrass_<os>/products/greengrass/ggc`. Você pode baixar o arquivo tar do AWS IoT Greengrass Core na página de downloads do AWS IoT Greengrass Core Software. • O pacote do SDK existe em `devicetester_greengrass_<os>/products/greengrass/ggsdk`. • Os arquivos em `devicetester_greengrass_<os>/`

Código de erro	Nome do código de erro	Possível causa raiz	Solução de problemas
			`tests` não foram modificados.
116	OpenFileFailed	Não foi possível abrir o arquivo especific ado.	Verifique o seguinte: • O arquivo especific ado na mensagem de erro existe e é válido. • `limits.co nfig` permite que arquivos suficientes sejam abertos. Se você estiver testando em um macOS, aumente o limite de arquivos abertos. O limite padrão é 256, que é o suficiente para testes.
117	WriteFileFailed	Falha ao gravar arquivo (pode ser o DUT ou computador de teste).	Verifique se o diretório especific ado na mensagem de erro existe e se você tem permissão de gravação.

Código de erro	Nome do código de erro	Possível causa raiz	Solução de problemas
118	FileCleanUpError	O teste não conseguir remover o arquivo especificado ou diretório nem desmontar o arquivo especificado no dispositivo remoto.	Se o arquivo binário ainda estiver em execução, ele poderá ser bloqueado. Termine o processo e exclua o arquivo especificado.
119	InvalidInputError	Configuração inválida.	Verifique se o arquivo `suite.json` é válido.
120	InvalidCredentialError	AWS Credenciais inválidas.	• Verifique suas AWS credencia is. Para obter mais informações, consulte the section called “Configure seu AWS credencia is”. • Verifique sua conexão de rede e execute o grupo de testes. Problemas de rede também podem causar esse erro.

Código de erro	Nome do código de erro	Possível causa raiz	Solução de problemas
121	AWSSessionError	Falha ao criar uma AWS sessão.	Esse erro pode ocorrer se AWS as credenciais forem inválidas ou se a conexão com a Internet estiver instável. Tente usar o AWS CLI para chamar uma operação de AWS API.
122	AWSApiCallError	Ocorreu um erro de AWS API.	Esse erro pode ser devido a um problema de rede. Verifique sua rede antes de repetir o grupo de testes.

Código de erro	Nome do código de erro	Possível causa raiz	Solução de problemas
123	IpNotExistError	O endereço IP não está incluído nas informações de conectividade.	Verifique a conexão com a Internet. Você pode usar o AWS IoT Greengrass console para verificar as informações de conectividade do item AWS IoT Greengras s principal que está sendo usado pelo teste. Se houver 10 endpoints incluídos nas informaçõ es de conectivi dade, você poderá remover alguns ou todos e executar o teste novamente . Para obter mais informações, consulte Informações de conectividade.
124	OTAJobNot CompleteError	Um OTA de trabalho não foi concluído.	Verifique sua conexão com a Internet e tente o grupo de testes OTA novamente.

Código de erro	Nome do código de erro	Possível causa raiz	Solução de problemas
125	CreateGreengrassServiceRoleError	Houve um dos problemas a seguir: • Ocorreu um erro ao criar uma função. • Ocorreu um erro ao anexar uma política à função AWS IoT Greengrass de serviço. • A política associada ao perfil de serviço é inválida. • Ocorreu um erro ao associar uma função a uma Conta da AWS.	Configure a função AWS IoT Greengrass de serviço. Para obter mais informações, consulte the section called “Função de serviço do Greengrass”.
126	DependenciesNotPresentError	Uma ou mais dependências necessárias para o teste específico não estão presentes no dispositivo.	Verifique o log de teste para ver quais dependências estão ausentes no dispositivo: *<device-tester-extract-location>* /results/ *<execution-id>*/logs/*<test-case-name.log>* .

Código de erro	Nome do código de erro	Possível causa raiz	Solução de problemas
127	InvalidHSMConfiguration	A HSM/PKCS configuração fornecida está incorreta.	No arquivo `device.json` , forneça a configuração correta necessária para interagir com o HSM usando PKCS#11.

Código de erro	Nome do código de erro	Possível causa raiz	Solução de problemas
128	OTAJobNot SuccededError	A tarefa OTA não foi bem-sucedida.	• Se você executou o grupo de testes `ota` individualmente, execute o grupo de testes `ggcdependencies` para verificar se todas as dependências (como `wget`) estão presentes. Depois, tente novamente o grupo de testes `ota`. • Analise os logs detalhados em *<device-tester-extract-location>* `/results/` *<execution-id>*`/logs/` para obter informações sobre solução de problemas e erros. Especificamente, verifique os seguintes logs: •

Código de erro	Nome do código de erro	Possível causa raiz	Solução de problemas
			Log do console (`test_manager.log`) • Log de caso de teste OTA (`ota_test.log`) • Log do daemon GGC (`ota_test_ggc_logs.tar.gz`) • Logs do atendente OTA (`ota_test_ota_logs.tar.gz`) • Verifique sua conexão com a Internet e tente o grupo de testes `ota` novamente. • Se o problema persistir, entre em contato com o Suporte ao desenvolvedor da AWS.

Código de erro	Nome do código de erro	Possível causa raiz	Solução de problemas
129	NoConnectivityError	O atendente de host não consegue se conectar à Internet.	Verifique suas configurações de conexão de rede e firewall. Tente novamente o grupo de testes depois que o problema de conectividade for resolvido.
130	NoPermissionError	O usuário do IAM para o qual você está usando para executar o IDT AWS IoT Greengrass não tem permissão para criar os AWS recursos necessários para executar o IDT.	Consulte Modelo de política de permissões para obter o modelo de política que concede as permissões necessárias para executar o IDT para AWS IoT Greengrass.

Código de erro	Nome do código de erro	Possível causa raiz	Solução de problemas
131	LeftoverAgentExistError	Seu dispositivo está executando AWS IoT Greengrass processos quando você tenta iniciar o IDT para AWS IoT Greengrass.	Verifique se não há nenhum daemon do Greengrass em execução no dispositivo. • Você pode usar este comando para interromper o daemon: sudo ./*<absolute-path-to-greengrass-daemon>* / greengrassd stop. • Você também pode encerrar o daemon do Greengrass por PID. **Note** Se você estiver usando uma instalação existente ou AWS IoT Greengrass configurada para iniciar automatic

Código de erro	Nome do código de erro	Possível causa raiz	Solução de problemas
			amente após a reinicial ização, deverá interromper o daemon após a reinicial ização e antes de executar a suíte de testes.
132	DeviceTimeOffsetEr ror	O dispositivo tem o horário incorreto.	Defina o horário incorreto no seu dispositivo.
133	InvalidMLConfigura tion	A configuração de fornecida está incorreta.	No arquivo `device.json` , forneça a configura ção correta necessári a para executar testes de inferência de ML. Para obter mais informações, consulte the section called “Opcional: configura r o dispositivo para qualificação de ML”.

Resolvendo o IDT para AWS IoT Greengrass erros

Ao usar o IDT, você deve instalar os arquivos de configuração corretos antes de executar o IDT for. AWS IoT Greengrass Se você estiver recebendo erros de análise e configuração, o primeiro passo 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

- Onde eu procuro erros?
- Erros de análise
- Erro de parâmetro necessário ausente
- Erro: Não foi possível iniciar teste
- Erro de não autorização para acessar um recurso
- Erros de permissão negada
- Erros de conexão SSH
- Erros de tempo limite
- Erros de comando não encontrados durante o teste
- Exceção de segurança no macOS

Onde eu procuro erros?

High-level os erros são exibidos no console durante a execução, e um resumo dos testes que falharam com o erro 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. Os arquivos de log para cada execução de teste são armazenados em um diretório nomeado com um UUID para a execução de teste que foi exibida no console durante a execução de teste.

O diretório dos logs de teste está localizado em *`<device-tester-extract-location>`*`/results/`*`<execution-id>`*`/logs/`. Esse diretório contém os seguintes arquivos que são úteis para depuração.

Arquivo	Description
`test_manager.log`	

Arquivo	Description
	Todos os logs que foram gravados no console durante a execução do teste. Um resumo dos resultados está localizado no final deste arquivo que 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-name>* .log	Logs detalhados para o teste específico.
<test-name> _ggc_logs.tar.gz	Uma coleção compactada de todos os registros que o daemon AWS IoT Greengrass principal gerou durante o teste. Para obter mais informações, consulte Solução de problemas AWS IoT Greengrass.
<test-name> _ota_logs.tar.gz	Uma coleção compactada de registros gerados pelo agente AWS IoT Greengrass OTA durante o teste. Somente para testes OTA.
<test-name> _basic_assertion_publisher_ggad_logs.tar.gz	Um conjunto de logs gerados pelo dispositivo editor do AWS IoT durante o teste.
<test-name> _basic_assertion_subscriber_ggad_logs.tar.gz	Um conjunto de logs gerados pelo dispositivo assinante do AWS IoT durante o teste.

Erros de análise

Ocasionalmente, um erro ortográfico em uma configuração JSON pode resultar em 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 executar a validação manualmente no IDE, em um editor de texto como o Atom ou o Sublime, ou por meio de uma ferramenta online, como a JSONLint.

Erro de parâmetro necessário ausente

Como novos atributos estão sendo adicionados ao IDT, os arquivos de configuração podem sofrer alterações. 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 garantir que a versão compatível mais recente tenha sido usada.

Erro: Não foi possível iniciar teste

Você pode encontrar erros que apontam para falhas ao iniciar o teste. Há várias causas possíveis e, portanto, faça o seguinte:

- Verifique se o nome do grupo incluído no comando de execução realmente existe. O nome do grupo é referenciado diretamente em seu arquivo `device.json`.
- Verifique se os dispositivos no grupo têm os parâmetros de configuração corretos.

Erro de não autorização para acessar um recurso

Você pode ver a mensagem de erro `<user or role> is not authorized to access this resource` na saída do terminal ou no arquivo `test_manager.log` em `/results/`*<execution-id>*`/logs`. Para resolver este problema, anexe a política gerenciada `AWSIoTDeviceTesterForGreengrassFullAccess` ao usuário de teste. Para obter mais informações, consulte the section called “Crie e configure um Conta da AWS”.

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`.

Erros de conexão SSH

Quando o IDT não puder se conectar a um dispositivo em teste, as falhas de conexão serão registradas em log em `/results/`*<execution-id>*`/logs/`*<test-case-id>*`.log`. As mensagens de falha de SSH aparecem na parte superior desse arquivo de log, pois a conexão com um dispositivo em teste é uma das primeiras operações executadas pelo IDT.

A maioria das configurações do Windows usa o aplicativo de terminal PuTTy para se conectar a hosts Linux. Esse aplicativo requer que os arquivos de chave privada PEM padrão sejam convertidos em um formato Windows proprietário chamado PPK. Quando IDT estiver configurado em seu arquivo `device.json`, use apenas os arquivos PEM. Se você usar um arquivo PPK, o IDT não poderá criar uma conexão SSH com o AWS IoT Greengrass dispositivo e não poderá executar testes.

Erros de tempo limite

Você pode aumentar o tempo limite para cada teste especificando um multiplicador de tempo limite, que será aplicado ao valor padrão de cada tempo limite do 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 GGQ_1.0.0 --pool-id DevicePool1 --timeout-
multiplier 2.5
```

Para obter mais informações, execute `run-suite --help`.

Erros de comando não encontrados durante o teste

Você precisa de uma versão mais antiga da biblioteca OpenSSL (libssl1.0.0) para executar testes em dispositivos. AWS IoT Greengrass A maioria das distribuições atuais do Linux usa o libssl versão 1.0.2 ou posterior (v1.1.0).

Por exemplo, em um Raspberry Pi, execute os seguintes comandos para instalar a versão necessária do libssl:

1.
```
wget http://ftp.us.debian.org/debian/pool/main/o/openssl/
libssl1.0.0_1.0.2l-1~bpo8+1_armhf.deb
```

2.
```
sudo dpkg -i libssl1.0.0_1.0.2l-1~bpo8+1_armhf.deb
```

Exceção de segurança no macOS

Quando você executa o IDT em uma máquina host que usa o macOS 10.15, o tíquete de reconhecimento de firma do IDT não é detectado corretamente e o IDT é impedido de ser executado. Para executar o IDT, você precisará conceder uma exceção de segurança ao executável `devicetester_mac_x86-64`.

Para conceder uma exceção de segurança ao executável do IDT

1. Inicie as Preferências do Sistema no menu Apple.
2. Selecione Segurança e privacidade e, em seguida, na guia Geral, clique no ícone de cadeado para fazer alterações nas configurações de segurança.
3. Procure a mensagem `"devicetester_mac_x86-64" was blocked from use because it is not from an identified developer.` e, em seguida selecione Permitir mesmo assim.
4. Aceite o aviso de segurança.

Se você tiver dúvidas sobre a política de suporte do IDT, entre em contato com o Suporte ao cliente da AWS.

Política de suporte para AWS IoT Testador de dispositivos para AWS IoT Greengrass V1

AWS IoT O Device Tester (IDT) for AWS IoT Greengrass é uma estrutura de teste disponível para download que permite validar e qualificar seus AWS IoT Greengrass dispositivos para inclusão no Catálogo de dispositivos.AWS Partner Recomendamos que você use a versão mais recente do AWS IoT Greengrass e do IDT para testar ou qualificar seus dispositivos. Para obter informações, consulte Versões compatíveis do IDT para AWS IoT Greengrass V2 no Guia do desenvolvedor do AWS IoT Greengrass Version 2 .

Você também pode usar qualquer uma das versões compatíveis do AWS IoT Greengrass e do IDT para testar ou qualificar seus dispositivos. Embora seja possível continuar a usar versões não compatíveis do IDT, essas versões não recebem correções de bugs ou atualizações.

> **Important**
>
> A partir de 4 de abril de 2022, AWS IoT o Device Tester (IDT) AWS IoT Greengrass V1 não gera mais relatórios de qualificação assinados. Você não pode mais qualificar novos AWS IoT Greengrass V1 dispositivos para serem listados no Catálogo de AWS Partner Dispositivos por meio do Programa de Qualificação de AWS Dispositivos. Embora você não possa qualificar dispositivos Greengrass V1, você pode continuar usando o IDT AWS IoT Greengrass V1 para testar seus dispositivos Greengrass V1. Recomendamos que você use o IDT para AWS IoT Greengrass V2 para qualificar e listar dispositivos Greengrass no AWS Partner Catálogo de dispositivos.

Se você tiver dúvidas sobre a política de suporte, entre em contato com o Suporte ao cliente da AWS.

Solução de problemas AWS IoT Greengrass

Esta seção fornece informações sobre solução de problemas e possíveis soluções para ajudar a resolver problemas com AWS IoT Greengrass.

Para obter informações sobre AWS IoT Greengrass cotas (limites), consulte Cotas de serviço no. Referência geral da Amazon Web Services

AWS IoT Greengrass Questões principais

Se o software AWS IoT Greengrass Core não iniciar, tente as seguintes etapas gerais de solução de problemas:

- Instale os binários que são apropriados para sua arquitetura. Para obter mais informações, consulte Software de núcleo do AWS IoT Greengrass.
- Verifique se o seu dispositivo de núcleo tem armazenamento local disponível. Para obter mais informações, consulte the section called “Solução de problemas de armazenamento”.
- Verifique se há mensagens de erro em `runtime.log` e `crash.log`. Para obter mais informações, consulte the section called “Solução de problemas com logs”.

Pesquise os seguintes sintomas e erros para encontrar informações que ajudem a solucionar problemas com um AWS IoT Greengrass núcleo.

Problemas

- Erro: O arquivo de configuração não contém o CaPath, CertPath ou KeyPath. O processo do daemon Greengrass com [pid = < pid>] morreu.
- Erro: Falha ao analisar/<greengrass-root/.json>. config/config
- Erro: Ocorreu um erro ao gerar a configuração TLS: ErrUnknownURIScheme
- Erro: Falha ao iniciar o tempo de execução: não foi possível iniciar os operadores: o teste de contêiner expirou.
- Erro: Falha ao invocar PutLogEvents no Cloudwatch local, LogGroup:/GreengrassSystem/connection_manager, erro:: falha na solicitação de envio causada por RequestError: Post http://<path>/cloudwatch/logs/: dial tcp < address: getsockopt: connection recused, response>: {}.

- Erro: Não foi possível criar o servidor devido a: falha ao carregar o grupo: chmod/<greengrass-root>/ggc/deployment/:aws:lambda: region: > < account-id:function: function-namelambda/arn: version/<file-name: arquivo ou diretório não existe. > < > < > < >
- A ferramenta AWS IoT Greengrass O software principal não inicia depois que você deixou de ser executado sem conteinerização para ser executado em um contêiner do Greengrass.
- Erro: Tamanho do spool deve ser pelo menos 262.144 bytes.
- Erro: [ERRO] – erro de mensagens na nuvem: Ocorreu um erro ao tentar publicar uma mensagem. {"errorString": "operation timed out"}
- Erro: container_linux.go:344: iniciar o processo do contêiner causou “process_linux.go:424: a inicialização do contêiner causou\” permissão negada\\\ "\"”. rootfs_linux.go:64: mounting \\\"/greengrass/ggc/socket/greengrass_ipc.sock\\\" to rootfs \\\"/greengrass/ggc/packages/<version>/rootfs/merged\\\" at \\\"/greengrass_ipc.sock\\\" caused \\\"stat /greengrass/ggc/socket/greengrass_ipc.sock:
- Erro: daemon do Greengrass em execução com PID: process-id. < > Não foi possível iniciar alguns componentes do sistema. Verifique "runtime.log" para erros.
- O shadow do dispositivo não sincroniza com a nuvem.
- ERRO: não foi possível aceitar a conexão TCP. accept tcp [::]:8000: accept4: muitos arquivos abertos.
- Erro: Erro de execução do tempo de execução: não foi possível iniciar o contêiner lambda. container_linux.go:259: o início do processo do contêiner causou "process_linux.go:345: init do contêiner causou \"rootfs_linux.go:50: rootfs de preparação causaram \\\"permissão negada\\\"\"".
- Aviso: [WARN] - [5] GK Remote: Erro ao recuperar dados da chave pública: ErrPrincipalNotConfigured: a chave privada para MqttCertificate não está definida.
- Erro: permissão negada ao tentar usar a função arn:aws:iam::<account-id>:role/<role-name> para acessar o URL s3. https://<region>-greengrass-updates.s3.<region>.amazonaws.com/core/<architecture>/greengrass-core-<distribution-version>.tar.gz
- A ferramenta AWS IoT Greengrass o núcleo está configurado para usar um proxy de rede e sua função Lambda não pode fazer conexões de saída.
- O núcleo está em um loop infinito de desconexão e conexão. O arquivo runtime.log contém uma série contínua de entradas de conexão e desconexão.
- Erro: não foi possível iniciar o contêiner do lambda. container_linux.go:259: o início do processo do contêiner causou "process_linux.go:345: o init do contêiner causou \"rootfs_linux.go:62: montagem de \\\"proc\\\" para rootfs \\\"

- [ERROR] -erro de execução em tempo de execução: não é possível iniciar o contêiner lambda. {"errorString": "falha ao inicializar montagens de contêiner: falha ao mascarar a raiz do greengrass no diretório superior de sobreposição: falha ao criar o dispositivo de máscara no diretório < ggc-path: o arquivo existe"} >
- [ERRO] - Falha na implantação. {"deploymentId": "<deployment-id > ", "errorString": ">falha no processo de teste do contêiner com pid < pid: estado do processo do contêiner: status de saída 1"}
- Erro: [ERROR] -erro de execução em tempo de execução: não é possível iniciar o contêiner lambda. {"errorString": "falha ao inicializar montagens de contêiner: falha ao criar sobreposição fs para contêiner: sobreposição de montagem em//packages/ ggc-version/falha: rootfs/merged falha na montagem com args source=\" no_source\" dest=\"greengrass/ggc/>/packages/ < ggc-version/ \" fstype=\ "overlay\" flags=\ "0\" data= "lowerdir=/ /packages/ ggc-version greengrass/ggc /dns:/, upperdir=/ /packages/ < ggc-version>/rootfs/merged, workdir=/ /packages/ ggc-version greengrass/ ggc < > greengrass/ggc < > rootfs/upper greengrass/ggc < >/rootfs/work\": muitos níveis de links simbólicos "}
- Erro: [DEPURAÇÃO] - Falha ao obter rotas. Descarte da mensagem.
- Erro: [Errno 24] Muitas < funções lambda abertas>, [Errno 24] Muitos arquivos abertos
- Erro: o servidor ds falhou ao iniciar a escuta do soquete: listen unix < ggc-path>/ggc/socket/ greengrass_ipc.sock: bind: argumento inválido
- [INFO] (Copiadora) aws.greengrass. StreamManager: robusto. Causado por: com.fasterxml.jackson.databind. JsonMappingException: Instantâneo excede o instante mínimo ou máximo
- Erro de GPG: https://dnw9lb6lzp2d8.cloudfront.net estável: as seguintes assinaturas eram inválidas InRelease: EXPKEYSIG 68D644ABD2327D47 Chave mestra do AWS Greengrass

Erro: O arquivo de configuração não contém o CaPath, CertPath ou KeyPath. O processo do daemon Greengrass com [pid = < pid>] morreu.

Solução: Você pode ver esse erro `crash.log` quando o software AWS IoT Greengrass Core não inicia. Isso pode ocorrer se você estiver executando v1.6 ou anterior. Execute um destes procedimentos:

- Atualize para a v1.7 ou posterior. Recomendamos que você sempre execute a versão mais recente do software AWS IoT Greengrass Core. Para fazer download de informações, consulte Software de núcleo do AWS IoT Greengrass.
- Use o `config.json` formato correto para sua versão AWS IoT Greengrass do software Core. Para obter mais informações, consulte the section called “AWS IoT Greengrass arquivo de configuração principal”.

> **Note**
>
> Para descobrir qual versão do software AWS IoT Greengrass Core está instalada no dispositivo principal, execute os seguintes comandos no terminal do seu dispositivo.
>
> ```
> cd /greengrass-root/ggc/core/
> sudo ./greengrassd --version
> ```

Erro: Falha ao analisar/<greengrass-root/.json>. config/config

Solução: esse erro pode ser visto quando o software de núcleo do AWS IoT Greengrass não iniciar. Certifique-se de que o arquivo de configuração do Greengrass está usando o formato JSON válido.

Abra `config.json` (localizado em `/greengrass-root/config`) e verifique o formato JSON. Por exemplo, certifique-se de que as vírgulas estão sendo usadas de forma correta.

Erro: Ocorreu um erro ao gerar a configuração TLS: ErrUnknownURIScheme

Solução: esse erro pode ser visto quando o software de núcleo do AWS IoT Greengrass não iniciar. Verifique se as propriedades da seção crypto do arquivo de configuração do Greengrass são válidas. A mensagem de erro deve fornecer mais informações.

Abra `config.json` (localizado em `/greengrass-root/config`) e verifique a seção `crypto`. Por exemplo, os caminhos de chave e certificado devem usar o formato URI correto e apontar para o local correto.

Erro: Falha ao iniciar o tempo de execução: não foi possível iniciar os operadores: o teste de contêiner expirou.

Solução: esse erro pode ser visto quando o software de núcleo do AWS IoT Greengrass não iniciar. Defina a propriedade `postStartHealthCheckTimeout` no arquivo de configuração do Greengrass. Essa propriedade opcional configura a quantidade de tempo (em milissegundos) que o daemon do Greengrass espera a verificação de saúde pós-início terminar. O valor padrão é de 30 segundos (30000 ms)

Abra `config.json` (localizado em `/greengrass-root/config`). No objeto `runtime`, adicione a propriedade `postStartHealthCheckTimeout` e defina o valor como um número maior que 30.000. Adicione uma vírgula onde for preciso para criar um arquivo JSON válido. Por exemplo:

```
...
"runtime" : {
  "cgroup" : {
    "useSystemd" : "yes"
  },
  "postStartHealthCheckTimeout" : 40000
},
...
```

Erro: Falha ao invocar PutLogEvents no Cloudwatch local, LogGroup:/GreengrassSystem/connection_manager, erro:: falha na solicitação de envio causada por RequestError: Post http://<path>/cloudwatch/logs/: dial tcp < address: getsockopt: connection recused, response>: {}.

Solução: esse erro pode ser visto quando o software de núcleo do AWS IoT Greengrass não iniciar. Isso pode ocorrer se você estiver executando AWS IoT Greengrass um Raspberry Pi e a configuração de memória necessária não tiver sido concluída. Para obter mais informações, consulte esta etapa.

Erro: Não foi possível criar o servidor devido a: falha ao carregar o grupo: chmod/<greengrass-root>/ggc/deployment/:aws:lambda: region: > < account-id:function: function-namelambda/arn: version/<file-name: arquivo ou diretório não existe. > < > < > < >

Solução: esse erro pode ser visto quando o software de núcleo do AWS IoT Greengrass não iniciar. Se você implantou um executável do Lambda no núcleo, verifique a `Handler` propriedade da função no `group.json` arquivo (localizado em/*greengrass-root*/ggc/deployment/group). Se o manipulador não tiver exatamente o mesmo nome do executável compilado, substitua o conteúdo do arquivo `group.json` por um objeto JSON vazio (`{}`) e execute os comandos a seguir para iniciar o AWS IoT Greengrass:

```
cd /greengrass/ggc/core/
sudo ./greengrassd start
```

Em seguida, use a API do AWS Lambda para atualizar o parâmetro `handler` da configuração da função, publique uma nova versão da função e atualize o alias. Para obter mais informações, consulte Versionamento e aliases da função do AWS Lambda.

Supondo que tenha adicionado a função ao grupo do Greengrass pelo alias (recomendado), você já pode reimplantar o grupo. (Caso contrário, você deve apontar para a nova versão da função ou do alias na definição e assinaturas do grupo antes de implantar o grupo.)

A ferramenta AWS IoT Greengrass O software principal não inicia depois que você deixou de ser executado sem conteinerização para ser executado em um contêiner do Greengrass.

Solução: verifique se não está faltando nenhuma dependência de contêiner.

Erro: Tamanho do spool deve ser pelo menos 262.144 bytes.

Solução: esse erro pode ser visto quando o software de núcleo do AWS IoT Greengrass não iniciar. Abra o arquivo `group.json` (localizado em */greengrass-root*/ggc/deployment/group),

substitua o conteúdo do arquivo por um objeto JSON vazio (`{}`) e execute os comandos a seguir para iniciar o AWS IoT Greengrass:

```
cd /greengrass/ggc/core/
sudo ./greengrassd start
```

Em seguida, execute as etapas do procedimento the section called “Como armazenar mensagens em cache no armazenamento local”. Não se esqueça de especificar um valor de `GG_CONFIG_MAX_SIZE_BYTES` para a função `GGCloudSpooler` que seja maior que ou igual a 262144.

Erro: [ERRO] – erro de mensagens na nuvem: Ocorreu um erro ao tentar publicar uma mensagem. {"errorString": "operation timed out"}

Solução: é possível que esse erro seja exibido no `GGCloudSpooler.log` quando o núcleo do Greengrass não conseguir enviar mensagens MQTT ao AWS IoT Core. Isso pode ocorrer se o ambiente do núcleo tiver largura de banda limitada e alta latência. Se você estiver executando a versão AWS IoT Greengrass v1.10.2 ou posterior, tente aumentar o `mqttOperationTimeout` valor no arquivo config.json. ??? Se a propriedade não estiver presente, adicione-a ao objeto `coreThing`. Por exemplo:

```
{
  "coreThing": {
    "mqttOperationTimeout": 10,
    "caPath": "root-ca.pem",
    "certPath": "hash.cert.pem",
    "keyPath": "hash.private.key",
    ...
  },
  ...
}
```

O valor padrão é 5 e o valor mínimo é 5.

Erro: container_linux.go:344: iniciar o processo do contêiner causou “process_linux.go:424: a inicialização do contêiner causou\” permissão negada\\\ "\"”. rootfs_linux.go:64: mounting \\\"/greengrass/ggc/socket/greengrass_ipc.sock\\\" to rootfs \\\"/greengrass/ggc/packages/<version>/rootfs/merged\\\" at \\\"/greengrass_ipc.sock\\\" caused \\\"stat /greengrass/ggc/socket/greengrass_ipc.sock:

Solução: Você pode ver esse erro `runtime.log` quando o software AWS IoT Greengrass Core não inicia. Isso ocorre se `umask` for maior que `0022`. Para resolver esse problema, você deve definir `umask` como `0022` ou menor. Um valor de `0022` concede a todos permissão de leitura dos novos arquivos por padrão.

Erro: daemon do Greengrass em execução com PID: process-id. < > Não foi possível iniciar alguns componentes do sistema. Verifique "runtime.log" para erros.

Solução: esse erro pode ser visto quando o software de núcleo do AWS IoT Greengrass não iniciar. Verifique `runtime.log` e `crash.log` para obter informações de erros específicos. Para obter mais informações, consulte the section called “Solução de problemas com logs”.

O shadow do dispositivo não sincroniza com a nuvem.

Solução: Verifique se ele AWS IoT Greengrass tem permissões `iot:UpdateThingShadow` e `iot:GetThingShadow` ações na função de serviço do Greengrass. Se a função de serviço usa a política gerenciada `AWSGreengrassResourceAccessRolePolicy`, essas permissões são incluídas por padrão.

Consulte Solução de problemas de intervalo de sincronização de shadow.

ERRO: não foi possível aceitar a conexão TCP. accept tcp [::]:8000: accept4: muitos arquivos abertos.

Solução: esse erro pode ser visto na saída do script `greengrassd`. Isso pode ocorrer se o limite do descritor de arquivo para o software AWS IoT Greengrass Core atingir o limite e precisar ser aumentado.

Use o comando a seguir e reinicie o software AWS IoT Greengrass Core.

```
ulimit -n 2048
```

> **Note**
>
> Neste exemplo, o limite é aumentado para 2048. Selecione um valor apropriado para seu caso de uso.

Erro: Erro de execução do tempo de execução: não foi possível iniciar o contêiner lambda. container_linux.go:259: o início do processo do contêiner causou "process_linux.go:345: init do contêiner causou \"rootfs_linux.go:50: rootfs de preparação causaram \\\"permissão negada\\\"\"".

Solução: instale AWS IoT Greengrass diretamente no diretório raiz ou certifique-se de que o diretório em que o software AWS IoT Greengrass Core está instalado e seus diretórios principais tenham `execute` permissões para todos.

Aviso: [WARN] - [5] GK Remote: Erro ao recuperar dados da chave pública: ErrPrincipalNotConfigured: a chave privada para MqttCertificate não está definida.

Solução: AWS IoT Greengrass usa um manipulador comum para validar as propriedades de todos os principais de segurança. Esse aviso em `runtime.log` é esperado, a menos que você

tenha especificado uma chave privada personalizada para o servidor MQTT local. Para obter mais informações, consulte the section called “Entidades principais de segurança”.

Erro: permissão negada ao tentar usar a função arn:aws:iam::<account-id>:role/<role-name> para acessar o URL s3. https://<region>-greengrass-updates.s3.<region>.amazonaws.com/core/<architecture>/greengrass-core-<distribution-version>.tar.gz

Solução: esse erro pode ser visto quando uma atualização OTA (over-the-air) falhar. Na política de função assinante, adicione a Região da AWS de destino como `Resource`. Essa função de signatário é usada para pré-assinar o URL do S3 para a atualização do AWS IoT Greengrass software. Para obter mais informações, consulte Função assinante do URL do S3.

A ferramenta AWS IoT Greengrass o núcleo está configurado para usar um proxy de rede e sua função Lambda não pode fazer conexões de saída.

Solução: dependendo do tempo de execução e dos executáveis usados pela função do Lambda para criar conexões, erros de tempo limite de conexão podem ser recebidos. Certifique-se de que suas funções do Lambda usem a configuração de proxy apropriada para se conectar por meio do proxy de rede. AWS IoT Greengrass passa a configuração do proxy para funções do Lambda definidas pelo usuário por meio das `http_proxy` variáveis de `no_proxy` ambiente`https_proxy`, e. Elas podem ser acessadas conforme mostrado no trecho Python a seguir.

```
import os
print(os.environ['http_proxy'])
```

Use a mesma letra que a variável definida em seu ambiente, por exemplo, todas minúsculas `http_proxy` ou todas maiúsculas `HTTP_PROXY`. Para essas variáveis, AWS IoT Greengrass oferece suporte a ambas.

> **Note**
>
> A maioria das bibliotecas comuns usadas para fazer conexões (como boto3 ou cURL e pacotes `requests` python) usam essas variáveis de ambiente por padrão.

O núcleo está em um loop infinito de desconexão e conexão. O arquivo runtime.log contém uma série contínua de entradas de conexão e desconexão.

Solução: isso pode acontecer quando outro dispositivo é codificado para usar o nome da coisa do núcleo como o ID de cliente para conexões MQTT com o AWS IoT. Conexões simultâneas são iguais Região da AWS e Conta da AWS devem usar IDs de cliente exclusivos. Por padrão, o núcleo usa o nome da coisa do núcleo como o ID do cliente para essas conexões.

Para resolver esse problema, você pode alterar o ID de cliente usado por outro dispositivo para a conexão (recomendado) ou substituir o valor padrão para o núcleo.

Para substituir o ID de cliente padrão para o dispositivo de núcleo

1. Execute o comando a seguir para interromper o daemon do Greengrass:

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

2. Abra *greengrass-root*`/config/config.json` para editar como o usuário su.
3. No objeto `coreThing`, adicione a propriedade `coreClientId` e defina o valor como o ID de cliente personalizado. O valor deve ter entre 1 e 128 caracteres. Ele deve ser exclusivo na Região da AWS atual para a Conta da AWS.

   ```
   "coreClientId": "MyCustomClientId"
   ```

4. Inicie o daemon.

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

Erro: não foi possível iniciar o contêiner do lambda. container_linux.go:259: o início do processo do contêiner causou "process_linux.go:345: o init do contêiner causou \"rootfs_linux.go:62: montagem de \\\"proc\\\" para rootfs \\\"

Solução: em algumas plataformas, você pode ver esse erro `runtime.log` ao AWS IoT Greengrass tentar montar o sistema de `/proc` arquivos para criar um contêiner Lambda. Ou você pode ver erros semelhantes, como `operation not permitted` ou `EPERM`. Esses erros podem ocorrer mesmo se os testes forem executados na plataforma pela passagem do script do verificador de dependências.

Tente uma das seguintes soluções possíveis:

- Habilite a opção `CONFIG_DEVPTS_MULTIPLE_INSTANCES` no kernel do Linux.
- Defina as opções de montagem `/proc` no host como `rw,relatim` apenas.
- Atualize o kernel do Linux para a versão 4.9 ou posterior.

> **Note**
>
> Esse problema não está relacionado à montagem de `/proc` para acesso a recursos locais.

[ERROR] -erro de execução em tempo de execução: não é possível iniciar o contêiner lambda. {"errorString”: “falha ao inicializar montagens de contêiner: falha ao mascarar a raiz do greengrass no diretório superior de sobreposição: falha ao criar o dispositivo de máscara no diretório < ggc-path: o arquivo existe"} >

Solução: esse erro pode ser visto em runtime.log quando a implantação falhar. Esse erro ocorre se uma função do Lambda no AWS IoT Greengrass grupo não puder acessar o `/usr` diretório no sistema de arquivos do núcleo.

Para resolver esse problema, adicione um recurso de volume local para o grupo e implante o grupo. Esse recurso deve:

- Especificar o `/usr` como o caminho de origem e caminho de destino.
- Adicionar automaticamente as permissões de grupo de SO do grupo Linux que possui o recurso.
- Ser afiliado à função do Lambda e permitir acesso somente leitura.

[ERRO] - Falha na implantação. {"deploymentId": "<deployment-id > “, “errorString”: “>falha no processo de teste do contêiner com pid < pid: estado do processo do contêiner: status de saída 1"}

Solução: esse erro pode ser visto em runtime.log quando a implantação falhar. Esse erro ocorre se uma função do Lambda no AWS IoT Greengrass grupo não puder acessar o `/usr` diretório no sistema de arquivos do núcleo.

Você pode confirmar se esse é o caso verificando em `GGCanary.log` se há erros adicionais. Se a função do Lambda não puder acessar o diretório `/usr`, `GGCanary.log` conterá o seguinte erro:

```
[ERROR]-standard_init_linux.go:207: exec user process caused "no such file or
 directory"
```

Para resolver esse problema, adicione um recurso de volume local para o grupo e implante o grupo. Esse recurso deve:

- Especificar o `/usr` como o caminho de origem e caminho de destino.
- Adicionar automaticamente as permissões de grupo de SO do grupo Linux que possui o recurso.
- Ser afiliado à função do Lambda e permitir acesso somente leitura.

Erro: [ERROR] -erro de execução em tempo de execução: não é possível iniciar o contêiner lambda. {"errorString": “falha ao inicializar montagens de contêiner: falha ao criar sobreposição fs para contêiner: sobreposição de montagem em//packages/ ggc-version/falha: rootfs/merged falha na montagem com args source=\" no_source\” dest=\”greengrass/ggc/>/packages/ < ggc-version/\” fstype=\ "overlay\” flags=\ "0\” data= “lowerdir=/ /packages/ ggc-version greengrass/ggc /dns:/, upperdir=/ /packages/ < ggc-version>/rootfs/merged, workdir=/ /packages/ ggc-version greengrass/ggc < > greengrass/ggc < > rootfs/upper greengrass/ggc < >/rootfs/work\”: muitos níveis de links simbólicos "}

Solução: você pode ver esse erro no `runtime.log` arquivo quando o software AWS IoT Greengrass Core não inicia. Esse problema pode ser mais comum em sistemas operacionais Debian.

Para resolver esse problema, faça o seguinte:

1. Atualize o software AWS IoT Greengrass Core para a versão v1.9.3 ou posterior. Isso deve resolver esse problema automaticamente.
2. Se você ainda receber esse erro depois de atualizar o software AWS IoT Greengrass Core, defina a `system.useOverlayWithTmpfs` propriedade como `true` no arquivo config.json.

   Example Exemplo

```
{
  "system": {
    "useOverlayWithTmpfs": true
  },
  "coreThing": {
    "caPath": "root-ca.pem",
    "certPath": "cloud.pem.crt",
    "keyPath": "cloud.pem.key",
    ...
  },
  ...
}
```

> **Note**
>
> Sua versão do software AWS IoT Greengrass Core é mostrada na mensagem de erro. Para encontrar sua versão de kernel do Linux, execute `uname -r`.

Erro: [DEPURAÇÃO] - Falha ao obter rotas. Descarte da mensagem.

Solução: verifique as assinaturas no grupo do e certifique-se de que a assinatura listada na mensagem `[DEBUG]` exista.

Erro: [Errno 24] Muitas < funções lambda abertas>, [Errno 24] Muitos arquivos abertos

Solução: Você pode ver esse erro no arquivo de log da função do Lambda se a função instancia `StreamManagerClient` no manipulador de funções. Recomendamos que você crie o cliente fora do manipulador. Para obter mais informações, consulte the section called “Use StreamManagerClient para trabalhar com streams”.

Erro: o servidor ds falhou ao iniciar a escuta do soquete: listen unix < ggc-path>/ggc/socket/greengrass_ipc.sock: bind: argumento inválido

Solução: esse erro pode ser visto quando o software de núcleo do AWS IoT Greengrass não iniciar. Esse erro ocorre quando o software AWS IoT Greengrass Core é instalado em uma pasta com um longo caminho de arquivo. Reinstale o software AWS IoT Greengrass Core em uma pasta com um caminho de arquivo que tenha menos de 79 bytes, se você não usar um diretório de gravação, ou 83 bytes, se usar um diretório de gravação.

[INFO] (Copiadora) aws.greengrass. StreamManager: robusto. Causado por: com.fasterxml.jackson.databind. JsonMappingException: Instantâneo excede o instante mínimo ou máximo

Ao atualizar o software AWS IoT Greengrass principal para a versão v1.11.3, você poderá ver o seguinte erro nos registros do gerenciador de fluxo se o gerenciador de fluxo 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 estiver usando uma versão do software AWS IoT Greengrass principal anterior à v1.11.3 e quiser atualizar para uma versão posterior, use uma atualização OTA para atualizar para a v1.11.4.

Erro de GPG: https://dnw9lb6lzp2d8.cloudfront.net estável: as seguintes assinaturas eram inválidas InRelease: EXPKEYSIG 68D644ABD2327D47 Chave mestra do AWS Greengrass

Quando você executa `apt update` em um dispositivo em que instalou o software AWS IoT Greengrass principal a partir de um repositório APT, você pode ver o erro a seguir.

```
Err:4 https://dnw9lb6lzp2d8.cloudfront.net stable InRelease
  The following signatures were invalid: EXPKEYSIG 68D644ABD2327D47 AWS Greengrass
 Master Key
Reading package lists... Done
W: GPG error: https://dnw9lb6lzp2d8.cloudfront.net stable InRelease: The following
 signatures were invalid: EXPKEYSIG 68D644ABD2327D47 AWS Greengrass Master Key
```

Esse erro ocorre porque AWS IoT Greengrass não oferece mais a opção de instalar ou atualizar o software AWS IoT Greengrass principal do repositório APT. Para executar com sucesso`apt update`, remova o AWS IoT Greengrass repositório da lista de fontes do dispositivo.

```
sudo rm /etc/apt/sources.list.d/greengrass.list
sudo apt update
```

Problemas de implantação

Use as seguintes informações para ajudar a solucionar os problemas de implantação.

Problemas

- Sua implantação atual não funciona e você deseja reverter para uma implantação anterior que funcione.
- Você verá o erro de implantação 403 Forbidden nos logs.
- Ocorre um ConcurrentDeployment erro quando você executa o comando create-deployment pela primeira vez.
- Erro: O Greengrass não está autorizado a assumir o perfil de serviço associado a essa conta ou o erro: Falha: perfil de serviço TES não está associado a essa conta.
- Erro: não é possível executar a etapa de download na implantação. erro ao fazer download: error ao fazer download do arquivo de definição de grupo: ... x509: o certificado expirou ou ainda não é válido
- A implantação não é concluída.
- Erro: Não foi possível encontrar executáveis java ou java8, ou o erro: Falha na identificação de < implantação > do tipo id de grupo NewDeployment para ID de < grupo: > falha na inicialização do trabalhador com ID de < trabalho, motivo pelo qual a versão instalada do Java deve ser maior > ou igual a 8
- A implantação não é concluída, e runtime.log contém várias entradas "esperar 1s para o contêiner ser interrompido".
- A implantação não termina e runtime.log contém “[ERROR] -Erro de implantação do Greengrass: falha ao relatar o status de implantação à nuvem {" deploymentId”: "<deployment-id > “, “errorString”: “Falha ao iniciar PUT, endpoint: https://deployment-status, error: Put https://<deployment-status>: proxyconnect tcp: x509: certificado assinado por autoridade < > desconhecida"}”
- Erro: > Falha no < id de implantação > do tipo id NewDeployment de < grupo Erro: Erro ao processar. A configuração do grupo é inválida: 112 ou [119 0] não têm permissão rw no caminho file:. < >

- Erro: < list-of-function-arns > está configurado para ser executado como root, mas o Greengrass não está configurado para executar funções do Lambda com permissões de root.
- Erro: ID de < implantação > do tipo NewDeployment de ID de < grupo > falhou erro: erro de implantação do Greengrass: não foi possível executar a etapa de download na implantação. erro durante o processamento: não foi possível carregar o arquivo de grupo baixado: não foi possível encontrar o UID com base no nome de usuário, nome de usuário: ggc_user: usuário desconhecido ggc_user.
- Erro: [ERROR] -erro de execução em tempo de execução: não é possível iniciar o contêiner lambda. {"errorString": “falha ao inicializar montagens de contêiner: falha ao mascarar a raiz do greengrass no diretório superior de sobreposição: falha ao criar o dispositivo de máscara no diretório < ggc-path: o arquivo existe"} >
- Erro: Falha na identificação de < implantação > do tipo NewDeployment ID de < grupo erro: falha no início do processo: > falha no início do processo: container_linux.go:259: iniciar o processo de contêiner causou “process_linux.go:250: a execução do processo exec setns para init causou\" wait: nenhum processo filho\ "”.
- Erro: [WARN] -MQTT [client] dial tcp: lookup host-prefix -ats.iot. < > <region > .amazonaws.com: esse host não existe... [ERRO] -Erro de implantação do Greengrass: falha ao relatar o status da implantação para a nuvem... net/http: solicitação cancelada enquanto aguarda a conexão (Client.Timeout excedida enquanto aguarda os cabeçalhos)

Sua implantação atual não funciona e você deseja reverter para uma implantação anterior que funcione.

Solução: use o AWS IoT console ou a AWS IoT Greengrass API para reimplantar uma implantação em funcionamento anterior. Isso implanta a versão do grupo correspondente no dispositivo de núcleo.

Para reimplantar uma implantação (console)

1. Na página de configuração do grupo, selecione a guia Implantações. Esta página exibe o histórico de implantação do grupo, incluindo a data e a hora, a versão do grupo e o status de cada tentativa de implantação.
2. Encontre a linha que contém a implantação que você deseja reimplantar. Selecione a implantação que você deseja reimplantar e selecione Reimplantar.



Para reimplantar uma implantação (CLI)

1. Use ListDeployments para encontrar o ID da implantação que você deseja reimplantar. Por exemplo:

```
aws greengrass list-deployments --group-id 74d0b623-c2f2-4cad-9acc-ef92f61fcaf7
```

   O comando retorna a lista de implantações para o grupo.

```
{
    "Deployments": [
        {
            "DeploymentId": "8d179428-f617-4a77-8a0c-3d61fb8446a6",
            "DeploymentType": "NewDeployment",
            "GroupArn": "arn:aws:greengrass:us-west-2:123456789012:/greengrass/
groups/74d0b623-c2f2-4cad-9acc-ef92f61fcaf7/versions/8dd1d899-4ac9-4f5d-
afe4-22de086efc62",
            "CreatedAt": "2019-07-01T20:56:49.641Z"
        },
        {
            "DeploymentId": "f8e4c455-8ac4-453a-8252-512dc3e9c596",
            "DeploymentType": "NewDeployment",
            "GroupArn": "arn:aws:greengrass:us-west-2::123456789012:/greengrass/
groups/74d0b623-c2f2-4cad-9acc-ef92f61fcaf7/versions/4ad66e5d-3808-446b-940a-
b1a788898382",
            "CreatedAt": "2019-07-01T20:41:47.048Z"
        },
        {
            "DeploymentId": "e4aca044-bbd8-41b4-b697-930ca7c40f3e",
            "DeploymentType": "NewDeployment",
```

```
            "GroupArn": "arn:aws:greengrass:us-west-2::123456789012:/greengrass/
groups/74d0b623-c2f2-4cad-9acc-ef92f61fcaf7/versions/1f3870b6-850e-4c97-8018-
c872e17b235b",
            "CreatedAt": "2019-06-18T15:16:02.965Z"
        }
    ]
}
```

> **Note**
>
> Esses AWS CLI comandos usam valores de exemplo para o grupo e o ID de implantação. Ao executar os comandos, certifique-se de substituir os valores de exemplo.

2. Use CreateDeployment para reimplantar a implantação de destino. Defina o tipo de implantação como `Redeployment`. Por exemplo:

```
aws greengrass create-deployment --deployment-type Redeployment \
  --group-id 74d0b623-c2f2-4cad-9acc-ef92f61fcaf7 \
  --deployment-id f8e4c455-8ac4-453a-8252-512dc3e9c596
```

   O comando retorna o ARN e o ID da nova implantação.

```
{
    "DeploymentId": "f9ed02b7-c28e-4df6-83b1-e9553ddd0fc2",
    "DeploymentArn": "arn:aws:greengrass:us-west-2::123456789012:/greengrass/
groups/74d0b623-c2f2-4cad-9acc-ef92f61fcaf7/deployments/f9ed02b7-c28e-4df6-83b1-
e9553ddd0fc2"
}
```

3. Use GetDeploymentStatus para obter o status da implantação.

Você verá o erro de implantação 403 Forbidden nos logs.

Solução: certifique-se de que a política do AWS IoT Greengrass núcleo na nuvem seja incluída `"greengrass:*"` como uma ação permitida.

Ocorre um ConcurrentDeployment erro quando você executa o comando create-deployment pela primeira vez.

Solução: uma implantação pode estar em andamento. Você pode executar get-deployment-status para verificar se uma implantação foi criada. Se ela não tiver sido criada, tente novamente.

Erro: O Greengrass não está autorizado a assumir o perfil de serviço associado a essa conta ou o erro: Falha: perfil de serviço TES não está associado a essa conta.

Solução: esse erro pode ser visto quando a implantação falhar. Verifique se uma função de serviço do Greengrass está associada à sua Conta da AWS na Região da AWS atual. Para acessar mais informações, consulte the section called “Gerenciar a função de serviço (CLI)” ou the section called “Gerenciar a função de serviço (console)”.

Erro: não é possível executar a etapa de download na implantação. erro ao fazer download: error ao fazer download do arquivo de definição de grupo: ... x509: o certificado expirou ou ainda não é válido

Solução: esse erro pode ser visto em `runtime.log` quando a implantação falhar. Se você receber um erro `Deployment failed` contendo a mensagem `x509: certificate has expired or is not yet valid`, verifique o relógio do dispositivo. O TLS e os X.509 certificados fornecem uma base segura para a criação de sistemas de IoT, mas exigem tempos precisos em servidores e clientes. Os dispositivos de IoT devem ter a hora correta (dentro de 15 minutos) antes de tentarem se conectar a AWS IoT Greengrass outros serviços TLS que usam certificados de servidor. Para obter mais informações, consulte Usando o Device Time para validar certificados de AWS IoT servidor na Internet das Coisas no blog AWS oficial.

A implantação não é concluída.

Solução: faça o seguinte:

- Certifique-se de que o AWS IoT Greengrass daemon esteja em execução no seu dispositivo principal. No terminal do dispositivo de núcleo, execute os comandos a seguir para verificar se o daemon está em execução e o inicie, caso necessário.

  1. Para verificar se o daemon está em execução:

     ```
     ps aux | grep -E 'greengrass.*daemon'
     ```

     Se a saída contém uma entrada `root` para `/greengrass/ggc/packages/1.11.6/bin/daemon`, o daemon está em execução.

     A versão no caminho depende da versão do software AWS IoT Greengrass Core instalada em seu dispositivo principal.

  2. Para iniciar o daemon:

     ```
     cd /greengrass/ggc/core/
     sudo ./greengrassd start
     ```

- Verifique se o dispositivo de núcleo está conectado e os endpoints de conexão do núcleo estão configurados corretamente.

Erro: Não foi possível encontrar executáveis java ou java8, ou o erro: Falha na identificação de < implantação > do tipo id de grupo NewDeployment para ID de < grupo: > falha na inicialização do trabalhador com ID de < trabalho, motivo pelo qual a versão instalada do Java deve ser maior > ou igual a 8

Solução: se o gerenciador de fluxo estiver habilitado para o AWS IoT Greengrass Core, você deverá instalar o Java 8 Runtime no dispositivo de núcleo antes de implantar o grupo. Para obter mais informações, consulte os requisitos para o gerenciador de fluxo. O gerenciador de fluxo é habilitado por padrão quando você usa a opção Criação de grupo padrão no console do AWS IoT .

Ou desabilite o gerenciador de fluxo e implante o grupo. Para obter mais informações, consulte the section called “Definir configurações (console)”.

A implantação não é concluída, e runtime.log contém várias entradas "esperar 1s para o contêiner ser interrompido".

Solução: execute os seguintes comandos no terminal do dispositivo principal para reiniciar o AWS IoT Greengrass daemon.

```
cd /greengrass/ggc/core/
sudo ./greengrassd stop
sudo ./greengrassd start
```

A implantação não termina e **`runtime.log`** contém “[ERROR] -Erro de implantação do Greengrass: falha ao relatar o status de implantação à nuvem {" deploymentId”: "<deployment-id > “, “errorString”: “Falha ao iniciar PUT, endpoint: https://deployment-status, error: Put https://<deployment-status>: proxyconnect tcp: x509: certificado assinado por autoridade < > desconhecida"}”

Solução: é possível visualizar esse erro em `runtime.log` quando o núcleo do Greengrass está configurado de modo a usar uma conexão proxy HTTPS e a cadeia de certificados do servidor de proxy não é confiável no sistema. Para tentar resolver esse problema, adicione a cadeia de certificados ao certificado raiz da CA. O núcleo do Greengrass adiciona os certificados deste arquivo ao pool de certificados usado para autenticação TLS em conexões HTTPS e MQTT com o AWS IoT Greengrass.

O exemplo a seguir mostra um certificado da CA do servidor de proxy adicionado ao arquivo de certificado raiz da CA:

```
# My proxy CA
-----BEGIN CERTIFICATE-----
MIIEFTCCAv2gAwIQWgIVAMHSAzWG/5YVRYtRQOxXUTEpHuEmApzGCSqGSIb3DQEK
\nCwUAhuL9MQswCQwJVUzEPMAVUzEYMBYGA1UECgwP1hem9uLmNvbSBJbmMuMRww
... content of proxy CA certificate ...
+vHIRlt0e5JAm5\noTIZGoFbK82A0/nO7f/t5PSIDAim9V3Gc3pSXxCCAQoFYnui
```

```
GaPUlGk1gCE84a0X\n7Rp/lND/PuMZ/s8YjlkY2NmYmNjMCAXDTE5MTEyN2cM216
gJMIADggEPADf2/m45hzEXAMPLE=
-----END CERTIFICATE-----

# Amazon Root CA 1
-----BEGIN CERTIFICATE-----
MIIDQTCCAimgF6AwIBAgITBmyfz/5mjAo54vB4ikPmljZKyjANJmApzyMZFo6qBg
ADA5MQswCQYDVQQGEwJVUzEPMA0tMVT8QtPHRh8jrdkGA1UEChMGDV3QQDExBBKW
... content of root CA certificate ...
o/ufQJQWUCyziar1hem9uMRkwFwYVPSHCb2XV4cdFyQzR1KldZwgJcIQ6XUDgHaa
5MsI+yMRQ+hDaXJiobldXgjUka642M4UwtBV8oK2xJNDd2ZhwLnoQdeXeGADKkpy
rqXRfKoQnoZsG4q5WTP46EXAMPLE
-----END CERTIFICATE-----
```

Por padrão, o arquivo de certificado raiz da CA está localizado em */greengrass-root*`/certs/root.ca.pem`. Para encontrar a localização no dispositivo de núcleo, verifique a propriedade `crypto.caPath` em config.json.

> **Note**
>
> *greengrass-root*representa o caminho em que o software AWS IoT Greengrass Core está instalado em seu dispositivo. Normalmente, esse é o diretório `/greengrass`.

Erro: > Falha no < id de implantação > do tipo id NewDeployment de < grupo Erro: Erro ao processar. A configuração do grupo é inválida: 112 ou [119 0] não têm permissão rw no caminho file:. < >

Solução: verifique se o grupo proprietário do *<path>* diretório tem permissões de leitura e gravação no diretório.

Erro: < list-of-function-arns > está configurado para ser executado como root, mas o Greengrass não está configurado para executar funções do Lambda com permissões de root.

Solução: esse erro pode ser visto em `runtime.log` quando a implantação falhar. Certifique-se de ter configurado AWS IoT Greengrass para permitir que as funções do Lambda sejam executadas com permissões de root. Altere o valor de `allowFunctionsToRunAsRoot` in `greengrass_root/config/config.json` para `yes` ou altere a função Lambda para ser executada como outra user/group. Para obter mais informações, consulte the section called “Executar uma função do Lambda como raiz”.

Erro: ID de < implantação > do tipo NewDeployment de ID de < grupo > falhou erro: erro de implantação do Greengrass: não foi possível executar a etapa de download na implantação. erro durante o processamento: não foi possível carregar o arquivo de grupo baixado: não foi possível encontrar o UID com base no nome de usuário, nome de usuário: ggc_user: usuário desconhecido ggc_user.

Solução: Se a identidade de acesso padrão do AWS IoT Greengrass grupo usar as contas padrão do sistema, o `ggc_user` usuário e o `ggc_group` grupo deverão estar presentes no dispositivo. Para obter instruções que mostram como adicionar o usuário e o grupo, consulte esta etapa. Certifique-se de inserir os nomes exatamente como é mostrado.

Erro: [ERROR] -erro de execução em tempo de execução: não é possível iniciar o contêiner lambda. {"errorString”: “falha ao inicializar montagens de contêiner: falha ao mascarar a raiz do greengrass no diretório superior de sobreposição: falha ao criar o dispositivo de máscara no diretório < ggc-path: o arquivo existe"} >

Solução: esse erro pode ser visto em `runtime.log` quando a implantação falhar. Esse erro ocorre se uma função do Lambda no grupo do Greengrass não conseguir acessar o diretório `/usr` no

sistema de arquivos do núcleo. Para resolver esse problema, adicione um recurso de volume local para o grupo e implante o grupo. O recurso deve:

- Especificar o `/usr` como o caminho de origem e caminho de destino.
- Adicionar automaticamente as permissões de grupo de SO do grupo Linux que possui o recurso.
- Ser afiliado à função do Lambda e permitir acesso somente leitura.

Erro: Falha na identificação de < implantação > do tipo NewDeployment ID de < grupo erro: falha no início do processo: > falha no início do processo: container_linux.go:259: iniciar o processo de contêiner causou “process_linux.go:250: a execução do processo exec setns para init causou \" wait: nenhum processo filho\ "”.

Solução: esse erro pode ser visto quando a implantação falhar. Repita a implantação.

Erro: [WARN] -MQTT [client] dial tcp: lookup host-prefix -ats.iot. < > <region > .amazonaws.com: esse host não existe... [ERRO] -Erro de implantação do Greengrass: falha ao relatar o status da implantação para a nuvem... net/http: solicitação cancelada enquanto aguarda a conexão (Client.Timeout excedida enquanto aguarda os cabeçalhos)

Solução: você poderá ver esse erro se estiver usando o `systemd-resolved`, que habilita a configuração `DNSSEC` por padrão. Como resultado, muitos domínios públicos não são reconhecidos. As tentativas de acessar o AWS IoT Greengrass endpoint não conseguem encontrar o host, então suas implantações permanecem no `In Progress` estado.

Você pode usar os comandos e a saída a seguir para testar esse problema. Substitua o *`region`* espaço reservado nos endpoints pelo seu. Região da AWS

```
$ ping greengrass-ats.iot.region.amazonaws.com
ping: greengrass-ats.iot.region.amazonaws.com: Name or service not known
```

```
$ systemd-resolve greengrass-ats.iot.region.amazonaws.com
greengrass-ats.iot.region.amazonaws.com: resolve call failed: DNSSEC validation failed:
 failed-auxiliary
```

Uma solução possível é desabilitar o DNSSEC. Quando o DNSSEC é `false`, as pesquisas de DNS não são validadas pelo DNSSEC. Para obter mais informações, consulte este problema conhecido para o `systemd`.

1. Adicione `DNSSEC=false` a `/etc/systemd/resolved.conf`.
2. Reinicie o `systemd-resolved`.

Para obter informações sobre `resolved.conf` e DNSSEC, execute `man resolved.conf` no seu terminal.

Problemas para criar grupo/criar função

Use as informações a seguir para ajudar a solucionar problemas com a criação de um AWS IoT Greengrass grupo ou função do Greengrass Lambda.

Problemas

- Erro: Sua configuração 'IsolationMode' para o grupo é inválida.
- Erro: Sua configuração 'IsolationMode' para função com arn < function-arn é > inválida.
- Erro: a MemorySize configuração da função com arn < function-arn não > é permitida em =. IsolationMode NoContainer
- Erro: acessar a configuração do Sysfs para a função com arn < function-arn > não é permitido em =. IsolationMode NoContainer
- Erro: a MemorySize configuração da função com arn < function-arn > é necessária em =. IsolationMode GreengrassContainer
- Erro: A função < function-arn > se refere ao recurso do tipo < resource-type > que não é permitido em =. IsolationMode NoContainer
- Erro: A configuração de execução da função com arn < function-arn > não é permitida.

Erro: Sua configuração 'IsolationMode' para o grupo é inválida.

Solução: esse erro ocorre quando o valor de `IsolationMode` no `DefaultConfig` de `function-definition-version` não é compatível. Os valores compatíveis são `GreengrassContainer` e `NoContainer`.

Erro: Sua configuração 'IsolationMode' para função com arn < function-arn é > inválida.

Solução: esse erro ocorre quando o valor de `IsolationMode` no <function-arn> da `function-definition-version` não é compatível. Os valores compatíveis são `GreengrassContainer` e `NoContainer`.

Erro: a MemorySize configuração da função com arn < function-arn não > é permitida em =. IsolationMode NoContainer

Solução: esse erro ocorre quando você especifica um valor `MemorySize` e opta por executar sem conteinerização. As funções do Lambda que são executadas sem conteinerização não podem ter limites de memória. Você pode remover o limite ou pode alterar a função do Lambda para que ela seja executada em um contêiner AWS IoT Greengrass .

Erro: acessar a configuração do Sysfs para a função com arn < function-arn > não é permitido em =. IsolationMode NoContainer

Solução: esse erro ocorre quando você especifica `true` para `AccessSysfs` e opta por executar sem conteinerização. As funções do Lambda executadas sem conteinerização devem ter seu código atualizado para acessar o sistema de arquivos diretamente e não podem usar `AccessSysfs`. Você pode especificar um valor de `false` for `AccessSysfs` ou alterar a função do Lambda para ser executada em um AWS IoT Greengrass contêiner.

Erro: a MemorySize configuração da função com arn < function-arn > é necessária em =. IsolationMode GreengrassContainer

Solução: Esse erro ocorre porque você não especificou um `MemorySize` limite para uma função do Lambda que está sendo executada em um AWS IoT Greengrass contêiner. Para resolver o erro, especifique um valor de `MemorySize`.

Erro: A função < function-arn > se refere ao recurso do tipo < resource-type > que não é permitido em =. IsolationMode NoContainer

Solução: não é possível acessar os tipos de recursos `Local.Device`, `Local.Volume`, `ML_Model.SageMaker.Job`, `ML_Model.S3_Object` ou `S3_Object.Generic_Archive` ao executar uma função do Lambda sem conteinerização. Se você precisar desses tipos de recursos, deverá executá-los em um AWS IoT Greengrass contêiner. Você também pode acessar dispositivos locais diretamente ao executar sem conteinerização alterando o código em sua função do Lambda.

Erro: A configuração de execução da função com arn < function-arn > não é permitida.

Solução: esse erro ocorre ao criar uma função do Lambda do sistema com `GGIPDetector` ou `GGCloudSpooler` e especificar uma configuração `IsolationMode` ou `RunAs`. Você deve omitir os parâmetros `Execution` para essa função do Lambda do sistema.

Problemas de descoberta

Use as informações a seguir para ajudar a solucionar problemas com o serviço AWS IoT Greengrass Discovery.

Problemas

- Erro: o dispositivo é membro de muitos grupos; os dispositivos não podem estar em mais de 10 grupos

Erro: o dispositivo é membro de muitos grupos; os dispositivos não podem estar em mais de 10 grupos

Solução: esta é uma limitação conhecida. Um dispositivo cliente pode ser membro de até 10 grupos.

Problemas com recursos de machine learning

Use as informações a seguir para ajudar a solucionar problemas com recursos de machine learning.

Problemas

- InvalidMLModelOwner - GroupOwnerSetting é fornecido no recurso do modelo ML, mas GroupOwner ou não GroupPermission está presente
- NoContainer A função não pode configurar a permissão ao anexar recursos de aprendizado de máquina. <function-arn > se refere ao recurso de aprendizado de máquina < resource-id com permissão na política de acesso > a recursos. < ro/rw >
- A função < function-arn > se refere ao recurso de aprendizado de máquina < resource-id sem permissão em ambos > os recursos. ResourceAccessPolicy OwnerSetting
- A função < function-arn > se refere ao recurso de aprendizado de máquina < resource-id > com permissão\ "rw\", enquanto a configuração do proprietário do recurso permite GroupPermission apenas\ "ro\".
- NoContainer A função < function-arn > se refere aos recursos do caminho de destino aninhado.
- O Lambda < function-arn > obtém acesso ao recurso < resource-id compartilhando o mesmo id > do proprietário do grupo

InvalidMLModelOwner - GroupOwnerSetting é fornecido no recurso do modelo ML, mas GroupOwner ou não GroupPermission está presente

Solução: Você recebe esse erro se um recurso de aprendizado de máquina contiver o ResourceDownloadOwnerSetting objeto, mas a `GroupPermission` propriedade necessária `GroupOwner` não estiver definida. Para resolver esse problema, defina a propriedade ausente.

NoContainer A função não pode configurar a permissão ao anexar recursos de aprendizado de máquina. <function-arn > se refere ao recurso de aprendizado de máquina < resource-id com permissão na política de acesso > a recursos. < ro/rw >

Solução: você recebe esse erro se uma função do Lambda não conteinerizada especificar permissões em nível de função para um recurso de aprendizado de máquina. Non-containerized as funções devem herdar as permissões do proprietário do recurso, as permissões definidas no recurso de aprendizado de máquina. Para resolver esse problema, selecione herdar permissões de proprietário do recurso (console) ou rremover as permissões da política de acesso aos recursos da função do Lambda (API) .

A função < function-arn > se refere ao recurso de aprendizado de máquina < resource-id sem permissão em ambos > os recursos. ResourceAccessPolicy OwnerSetting

Solução: você receberá esse erro se as permissões para o recurso de machine learning não estiverem configuradas para a função anexada do Lambda ou para o recurso. Para resolver esse problema, configure as permissões na `ResourceAccessPolicy` propriedade da função Lambda ou na `OwnerSetting` propriedade do recurso.

A função < function-arn > se refere ao recurso de aprendizado de máquina < resource-id > com permissão\ "rw\”, enquanto a configuração do proprietário do recurso permite GroupPermission apenas\ "ro\”.

Solução: você receberá esse erro se as permissões de acesso definidas para a função anexada do Lambda excederem as permissões de proprietário do recurso definidas para o recurso de machine learning. Para resolver esse problema, defina permissões mais restritivas para a função do Lambda ou permissões menos restritivas para o proprietário do recurso.

NoContainer A função < function-arn > se refere aos recursos do caminho de destino aninhado.

Solução: você receberá esse erro se vários recursos de machine learning anexados a uma função não conteinerizada do Lambda utilizarem o mesmo caminho de destino ou um caminho de destino aninhado. Para resolver esse problema, especifique caminhos de destino separados para os recursos.

O Lambda < function-arn > obtém acesso ao recurso < resource-id compartilhando o mesmo id > do proprietário do grupo

Solução: você receberá esse erro em `runtime.log` se o mesmo grupo de sistemas operacionais for especificado como a identidade da função do Lambda Executar como e o proprietário do recurso de um recurso de machine learning, mas o recurso não estiver vinculado à função do Lambda. Essa configuração dá à função Lambda permissões implícitas que ela pode usar para acessar o recurso sem AWS IoT Greengrass autorização.

Para resolver esse problema, use um grupo de SO diferente para uma das propriedades ou anexe o recurso de machine learning à função do Lambda.

AWS IoT Greengrass núcleo dos problemas do Docker

Use as informações a seguir para ajudar a solucionar problemas com a execução de um AWS IoT Greengrass núcleo em um contêiner Docker.

Problemas

- Erro: Opções desconhecidas: -no-include-email
- Aviso: IPv4 está desabilitado. As redes não funcionarão.
- 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: não arn:aws:iam::<account-id>:user/<user-name> está autorizado a executar: ecr:GetAuthorizationToken on resource: *
- Erro: Não é possível criar um contêiner para o serviço do greengrass: Conflito. O nome do contêiner "/aws-iot-greengrass" já está em uso.

- Erro: [FATAL]-Falha ao redefinir o namespace de montagem do thread devido a um erro inesperado: "operação não permitida". Para manter a consistência, o GGC travará e precisará ser reiniciado manualmente.

Erro: Opções desconhecidas: -no-include-email

Solução: este erro pode ocorrer ao executar o comando `aws ecr get-login`. Verifique se você tem a AWS CLI versão mais recente instalada (por exemplo, run:`pip install awscli --upgrade --user`). Se você estiver usando o Windows e tiver instalado a CLI usando o instalador MSI, repita o processo de instalação. Para obter informações, consulte Instalar a AWS Command Line Interface no Microsoft Windows no Guia do usuário do AWS Command Line Interface .

Aviso: IPv4 está desabilitado. As redes não funcionarão.

Solução: Você pode receber esse aviso ou uma mensagem semelhante ao executar AWS IoT Greengrass em um computador Linux. Ative o encaminhamento de rede IPv4 conforme descrito nesta etapa. ??? AWS IoT Greengrass a implantação na nuvem e as comunicações MQTT não funcionam quando o encaminhamento IPv4 não está ativado. Para obter mais informações, consulte Configurar parâmetros de kernel com namespace (sysctls) em runtime na documentação do Docker.

Erro: Um firewall está bloqueando o compartilhamento de arquivos entre janelas e os contêineres.

Solução: esse erro ou uma mensagem `Firewall Detected` pode ser recebida 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: não arn:aws:iam::<account-id>:user/<user-name> está autorizado a executar: ecr: GetAuthorizationToken on resource: *

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: Não é possível criar um contêiner para o serviço do greengrass: Conflito. O nome do contêiner "/aws-iot-greengrass" já está em uso.

Solução: isso pode ocorrer quando o nome do contêiner é usado em um contêiner mais antigo. Para resolver esse problema, remova o antigo contêiner do Docker executando o seguinte comando:

```
docker rm -f $(docker ps -a -q -f "name=aws-iot-greengrass")
```

Erro: [FATAL]-Falha ao redefinir o namespace de montagem do thread devido a um erro inesperado: "operação não permitida". Para manter a consistência, o GGC travará e precisará ser reiniciado manualmente.

Solução: Esse erro `runtime.log` pode ocorrer quando você tenta implantar uma função do `GreengrassContainer` Lambda em um AWS IoT Greengrass núcleo em execução em um contêiner Docker. Atualmente, apenas funções do Lambda do `NoContainer` podem ser implantadas em um contêiner do Docker do Greengrass.

Para resolver esse problema, certifique-se de que todas as funções do Lambda estão no modo `NoContainer` e inicie uma nova implantação. Então, ao iniciar o contêiner, não vincule o `deployment` diretório existente ao contêiner AWS IoT Greengrass principal do Docker. Em vez disso, crie um diretório `deployment` em seu lugar e salve-o no contêiner do Docker. Isso permite que o novo contêiner do Docker receba a implantação mais recente com funções do Lambda sendo executadas no modo `NoContainer`.

Para obter mais informações, consulte the section called “Executar AWS IoT Greengrass em um contêiner Docker”.

Solução de problemas com logs

Você pode definir as configurações de registro para um grupo do Greengrass, como enviar CloudWatch registros para o Logs, armazenar registros no sistema de arquivos local ou ambos. Para obter informações detalhadas ao solucionar problemas, altere temporariamente o nível de log para `DEBUG`. As alterações nas configurações de registro em log entram em vigor quando você implanta o grupo. Para obter mais informações, consulte the section called “Configurar o registro para AWS IoT Greengrass”.

No sistema de arquivos local, AWS IoT Greengrass armazena os registros nos seguintes locais. É necessário ter permissões raiz para ler os logs no sistema de arquivos.

greengrass-root`/ggc/var/log/crash.log`

Mostra as mensagens geradas quando um AWS IoT Greengrass núcleo falha.

greengrass-root`/ggc/var/log/system/runtime.log`

Mostra mensagens sobre qual componente apresentou falha.

greengrass-root`/ggc/var/log/system/`

Contém todos os registros dos componentes do AWS IoT Greengrass sistema, como o gerenciador de certificados e o gerenciador de conexões. Ao usar as mensagens em `ggc/var/log/system/` `eggc/var/log/system/runtime.log`, você deve ser capaz de descobrir qual erro ocorreu nos componentes AWS IoT Greengrass do sistema.

greengrass-root`/ggc/var/log/system/localwatch/`

Contém os registros do AWS IoT Greengrass componente que processa o upload dos registros do Greengrass para o Logs. CloudWatch Se você não conseguir visualizar os logins do Greengrass CloudWatch, poderá usar esses registros para solucionar problemas.

greengrass-root`/ggc/var/log/user/`

Contém todos os logs das funções do Lambda definidas pelo usuário. Marque essa pasta para encontrar mensagens de erro locais de suas funções do Lambda.

> **Note**
>
> Por padrão *greengrass-root* é o diretório `/greengrass`. Caso um diretório de gravação seja configurado, os logs estão nesse diretório.

Se os registros estiverem configurados para serem armazenados na nuvem, use CloudWatch os Registros para visualizar as mensagens de registro. `crash.log`é encontrado somente nos registros do sistema de arquivos no dispositivo AWS IoT Greengrass principal.

Se AWS IoT estiver configurado para gravar registros CloudWatch, verifique esses registros se ocorrerem erros de conexão quando os componentes do sistema tentarem se conectar AWS IoT.

Para obter mais informações sobre o AWS IoT Greengrass registro, consultethe section called "Monitoramento com logs do AWS IoT Greengrass ".

> **Note**
>
> Os registros AWS IoT Greengrass do software Core v1.0 são armazenados no *greengrass-root*`/var/log` diretório.

Solução de problemas de armazenamento

Quando o armazenamento de arquivo local estiver cheio, pode ser que alguns componentes comecem a apresentar falha:

- As atualizações de shadow locais não ocorrem.
- Os novos certificados AWS IoT Greengrass principais do servidor MQTT não podem ser baixados localmente.
- As implantações apresentam falha.

Você sempre deve estar ciente da quantidade de espaço livre disponível localmente. Você pode calcular o espaço livre com base no tamanho das funções do Lambda implantadas, na configuração de registro em log (consulte the section called "Solução de problemas com logs") e no número de shadows armazenados localmente.

Solução de problemas com mensagens

Todas as mensagens enviadas localmente AWS IoT Greengrass são enviadas com QoS 0. Por padrão, AWS IoT Greengrass armazena mensagens em uma fila na memória. Portanto, as mensagens não processadas são perdidas quando o núcleo do Greengrass é reiniciado, por exemplo, após a implantação de um grupo ou a reinicialização de um dispositivo. No entanto, você pode configurar AWS IoT Greengrass (v1.6 ou posterior) para armazenar mensagens em cache no sistema de arquivos para que elas persistam nas reinicializações principais. Você também pode configurar o tamanho da fila. Se você configurar um tamanho de fila, verifique se ela é maior que ou igual a 262.144 bytes (256 KB). Caso contrário, AWS IoT Greengrass talvez não inicie corretamente. Para obter mais informações, consulte the section called “Fila de mensagens MQTT”.

> **Note**
>
> Ao usar o padrão da fila na memória, recomendamos que você implante grupos ou reinicie o dispositivo quando a interrupção do serviço é a mais baixa.

Você também pode configurar o núcleo para estabelecer sessões persistentes com a AWS IoT. Isso permite que o núcleo receba mensagens enviadas Nuvem AWS enquanto o núcleo está offline. Para obter mais informações, consulte the section called “Sessões persistentes do MQTT com AWS IoT Core”.

Solução de problemas de intervalo de sincronização de shadow

Os atrasos significativos na comunicação entre um dispositivo de núcleo do Greengrass e a nuvem podem causar falha na sincronização shadow devido a um tempo limite. Nesse caso, você deve ver entradas de log semelhantes às seguintes:

```
[2017-07-20T10:01:58.006Z][ERROR]-cloud_shadow_client.go:57,Cloud shadow
 client error: unable to get cloud shadow what_the_thing_is_named for
 synchronization. Get https://1234567890abcd.iot.us-west-2.amazonaws.com:8443/things/
what_the_thing_is_named/shadow: net/http: request canceled (Client.Timeout exceeded
 while awaiting headers)
[2017-07-20T10:01:58.006Z][WARN]-sync_manager.go:263,Failed to get cloud
 copy: Get https://1234567890abcd.iot.us-west-2.amazonaws.com:8443/things/
what_the_thing_is_named/shadow: net/http: request canceled (Client.Timeout exceeded
 while awaiting headers)
```

```
[2017-07-20T10:01:58.006Z][ERROR]-sync_manager.go:375,Failed to execute sync operation
 {what_the_thing_is_named VersionDiscontinued []}"
```

Uma possível correção é configurar o intervalo de tempo durante o qual seu dispositivo de núcleo aguardará uma resposta do host. Abra o arquivo config.json em *greengrass-root*/config e adicione um campo `system.shadowSyncTimeout` com um valor de tempo limite em segundos. Por exemplo:

```
{
  "system": {
    "shadowSyncTimeout": 10
  },
  "coreThing": {
    "caPath": "root-ca.pem",
    "certPath": "cloud.pem.crt",
    "keyPath": "cloud.pem.key",
    ...
  },
  ...
}
```

Se nenhum valor `shadowSyncTimeout` for especificado em `config.json`, o padrão será de 5 segundos.

> **Note**
>
> Para o software AWS IoT Greengrass Core v1.6 e versões anteriores, o padrão `shadowSyncTimeout` é 1 segundo.

Verificar AWS Re: post

Se você não conseguir resolver seu problema usando as informações de solução de problemas neste tópico, você pode pesquisar Solução de problemas ou verificar a AWS IoT Greengrass tag em AWS re:POST para ver se há problemas relacionados ou postar uma nova pergunta. Os membros da AWS IoT Greengrass equipe monitoram ativamente o AWS Re:post.

Histórico do documento para AWS IoT Greengrass

A tabela a seguir descreve mudanças importantes no Guia do AWS IoT Greengrass desenvolvedor após junho de 2018. Para receber notificações sobre atualizações dessa documentação, é possível inscrever-se em um feed RSS.

Alteração	Descrição	Data
Aviso de fim do suporte para AWS IoT Greengrass Version 1	Aviso de fim do suporte: em 7 de outubro de 2026, AWS o suporte para o. AWS IoT Greengrass Version 1 Depois de 7 de outubro de 2026, você não poderá mais acessar os AWS IoT Greengrass V1 recursos. Para obter mais informações, visite Migrar de AWS IoT Greengrass Version 1.	7 de outubro de 2025
Atualização do fim do suporte da versão v1.11.x para o Snap	As informações de fim de suporte do AWS IoT Greengrass core v 1.11.x Snap foram atualizadas no snapcraft.io.	22 de setembro de 2023
Fim do suporte para o snap v1.11.x	Foram adicionadas informações sobre o fim do suporte para o AWS IoT Greengrass core v 1.11.x Snap no snapcraft.io.	19 de setembro de 2023
Imagens do Docker para v1.11.6 AWS IoT Greengrass	As imagens do Docker para o software AWS IoT Greengrass Core v1.11.6 estão disponíveis no Amazon Elastic Container Registry (Amazon ECR) e no Docker Hub. Recomendamos	12 de abril de 2022

	que você sempre execute a versão mais recente.	
AWS IoT Device Tester (IDT) para depreciação AWS IoT Greengrass V1	O IDT para AWS IoT Greengrass V1 não gerará mais relatórios de qualificação assinados.	4 de abril de 2022
Support update para AWS IoT Device Tester for AWS IoT Greengrass	O IDT para a AWS IoT Greengrass versão 4.4.1 agora suporta o uso do software AWS IoT Greengras s principal versão v1.11.6 para qualificação de dispositivos.	24 de março de 2022
AWS IoT Greengrass versão 1.11.6 lançada	A versão 1.11.6 do software AWS IoT Greengrass Core está disponível. Essa versão contém melhorias no desempenho e correções de erros. Recomendamos que você sempre execute a versão mais recente.	24 de março de 2022
Lançado o SiteWise conector IoT versão 12	A versão 12 do SiteWise conector IoT está disponível. Esta versão contém correções de erros.	23 de dezembro de 2021
Imagens do Docker para AWS IoT Greengrass v1.11.5 e v1.10.5	As imagens do Docker para o software AWS IoT Greengras s Core v1.11.5 e v1.10.5 estão disponíveis no Amazon Elastic Container Registry (Amazon ECR) e no Docker Hub. Recomendamos que você sempre execute a versão mais recente.	22 de dezembro de 2021

AWS IoT Greengrass V1 política de manutenção	A política de AWS IoT Greengrass V1 manutenção define os diferentes níveis de manutenção e atualizações do AWS IoT Greengrass V1 serviço e do software AWS IoT Greengrass principal v1.x.	20 de dezembro de 2021
AWS IoT Lançada a versão 4.4.1 do Device Tester	O IDT para a AWS IoT Greengrass versão 4.4.1 já está disponível. Essa versão inclui o pacote de AWS IoT Greengrass qualifi cação (GGQ) v1.3.1 e oferece suporte ao uso das versões AWS IoT Greengrass principai s do software v1.11.5 e v1.10.5 para qualificação de dispositivos.	20 de dezembro de 2021
AWS IoT Greengrass versões 1.11.5 e 1.10.5 lançadas	As versões 1.11.5 e 1.10.5 do software AWS IoT Greengrass Core estão disponíveis. Essas versões contêm melhorias no desempenho e correções de erros. Recomendamos que você sempre execute a versão mais recente.	12 de dezembro de 2021

Imagens do Docker AWS IoT Greengrass v1.11.4 e v1.10.4 republicadas	As imagens do Docker para as versões 1.11.4 e 1.10.4 do software AWS IoT Greengrass Core foram republicadas no Amazon Elastic Container Registry (Amazon ECR) e no Docker Hub para corrigir erros com. BusyBox Para usar as imagens mais recentes do Docker, use as tags `1.11.4-1` ou `1.10.4-1`. Para obter mais informações sobre as tags disponíveis, consulte amazon/aws-iot-greengrass no Docker Hub.	8 de dezembro de 2021
CloudWatch O conector de métricas oferece suporte a registros de data e hora duplicados nos dados de entrada	Agora você pode enviar dados de entrada com timestamps duplicados para esse conector.	19 de novembro de 2021
Cross-service atualização confusa de prevenção de deputados	AWS IoT Greengrass 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

Imagens do Docker para v1.11.4 AWS IoT Greengrass	As imagens do Docker para o software AWS IoT Greengrass Core v1.11.4 estão disponíveis no Amazon Elastic Container Registry (Amazon ECR) e no Docker Hub. Recomendamos que você sempre execute a versão mais recente.	24 de agosto de 2021
Publicado o AWS IoT Greengrass snap v1.11.4	A versão 1.11.4 do AWS IoT Greengrass snap está disponível. Recomendamos que você sempre execute a versão mais recente.	20 de agosto de 2021
Support update para AWS IoT Device Tester for AWS IoT Greengrass	O IDT para a AWS IoT Greengrass versão 4.1.0 agora oferece suporte ao uso do software AWS IoT Greengrass principal versão v1.11.4 para qualificação de dispositivos.	18 de agosto de 2021
AWS IoT Greengrass versão 1.11.4 lançada	A versão 1.11.4 do software AWS IoT Greengrass Core está disponível. Esta versão corrige um problema com o gerenciador de streams que impedia atualizações para a versão 1.11.3 de uma versão anterior do software Core. AWS IoT Greengras s Recomendamos que você sempre execute a versão mais recente.	17 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. É possível estabelecer uma conexão privada entre sua VPC e o ambiente de gerenciamento AWS IoT Greengrass .	16 de agosto de 2021
AWS IoT Lançada a versão 4.1.0 do Device Tester	A versão 4.1.0 do AWS IoT Device Tester for AWS IoT Greengrass está disponíve l. Esta versão suporta o uso das versões 1.11.3 e 1.10.4 do software AWS IoT Greengras s principal para qualificação de dispositivos.	23 de junho de 2021
Publicado o AWS IoT Greengrass snap v1.11.3	A versão 1.11.3 do AWS IoT Greengrass snap contém melhorias de desempenho e correções de erros. Como melhor prática, recomenda mos que você sempre execute a versão mais recente.	15 de junho de 2021

Imagens do Docker para AWS IoT Greengrass v1.11.3 e v1.10.4 lançadas	As imagens do Docker para o software AWS IoT Greengrass Core v1.11.3 e v1.10.4 estão disponíveis no Amazon Elastic Container Registry (Amazon ECR) e no Docker Hub. Essas versões do AWS IoT Greengrass Core contêm melhorias de desempenho e correções de erros. Recomendamos que você sempre execute a versão mais recente.	15 de junho de 2021
AWS IoT Greengrass versão 1.11.3 lançada	A versão 1.11.3 do software AWS IoT Greengrass Core está disponível. Essa versão contém melhorias no desempenho e correções de erros. Recomendamos que você sempre execute a versão mais recente.	14 de junho de 2021
AWS IoT Greengrass versão 1.10.4 lançada	A versão 1.10.4 do software AWS IoT Greengrass Core está disponível. Essa versão contém melhorias no desempenho e correções de erros. Recomendamos que você sempre execute a versão mais recente.	14 de junho de 2021

Modbus-TCP Lançado o adaptador de protocolo versão 2	A versão 2 do conector do adaptador de Modbus-TCP protocolo está disponível. Esta versão incluiu suporte para cadeias de caracteres de origem codificadas em ASCII, UTF8 e ISO8859.	24 de maio de 2021
Lançamento do conector de implantação do aplicativo do Docker versão 7	A versão 7 do conector de implantação do aplicativo do Docker do Greengrass está disponível.	5 de abril de 2021
AWS IoT Greengrass versão 1.11.1 lançada	A versão 1.11.1 do software AWS IoT Greengrass Core está disponível. Essa versão contém melhorias no desempenho e correções de erros. Recomendamos que você sempre execute a versão mais recente.	29 de março de 2021
AWS IoT Lançada a versão 4.0.2 do Device Tester	A versão 4.0.2 do AWS IoT Device Tester for AWS IoT Greengrass está disponível. Essa versão substitui o IDT v4.0.0 e adiciona suporte para a versão 1.11.1 do software Core. AWS IoT Greengrass Isso também corrige um problema que fazia com que o IDT mascarasse erros de integração de segurança de hardware (HSI).	29 de março de 2021

Lançado o SiteWise conector IoT versão 11	A versão 11 do SiteWise conector IoT está disponível. Isso inicia o suporte para strings que contêm caracteres ocultos ou não imprimíveis. Essa versão também inclui melhorias gerais de desempenho e correções de erros.	24 de março de 2021
Snap AWS IoT Greengrass v1.11.0 republicado	AWS IoT Greengrass A versão 1.11.0 do snap foi republicada no Snapcraft para solucionar correções de bugs e uma possível falha no aplicativo ao usar o interpretador Python. AWS IoT Greengrass não fornece snaps para as versões 1.10 e 1.9 do software.	19 de março de 2021
Support update para AWS IoT Device Tester for AWS IoT Greengrass	O IDT para a AWS IoT Greengrass versão 4.0.0 agora suporta o uso do software AWS IoT Greengrass principal versão v1.10.3 para qualificação de dispositivos.	18 de março de 2021
Snap AWS IoT Greengrass v1.8.4 republicado	AWS IoT Greengrass A versão 1.8.4 do snap foi republicada no Snapcraft para solucionar correções de bugs e uma possível falha no aplicativo ao usar o interpretador Python.	15 de março de 2021

Imagem do Docker AWS IoT Greengrass v1.11.0 republicada para ARMv7l	A imagem do Docker para o software AWS IoT Greengrass Core versão 1.11.0 para a plataforma ARMv7l foi republicada no Amazon Elastic Container Registry (Amazon ECR) e no Docker Hub para corrigir erros e uma possível falha no aplicativo ao usar o interpretador Python.	8 de março de 2021
AWS IoT Greengrass Imagens do Docker v1.10.3 lançadas	As imagens do Docker para o software AWS IoT Greengrass Core versão 1.10.3 agora estão disponíveis no Amazon Elastic Container Registry (Amazon ECR) e no Docker Hub.	8 de março de 2021
Imagens do Docker AWS IoT Greengrass v1.11.0 e v1.9.4 republicadas	As imagens do Docker para as versões 1.11.0 e 1.9.4 do software AWS IoT Greengrass Core foram republicadas no Amazon Elastic Container Registry (Amazon ECR) e no Docker Hub para corrigir erros e uma possível falha no aplicativo ao usar o interpretador Python. As imagens do Docker para ARMv7l não foram republicadas no momento.	26 de fevereiro de 2021

AWS IoT Greengrass versão 1.10.3 lançada	A versão 1.10.3 do software AWS IoT Greengrass Core está disponível. Esta versão adiciona a propriedade de configuração do núcleo `systemComponentAuthTimeout` e contém melhorias de desempenho e correções de bugs. Recomendamos que você sempre execute a versão mais recente.	24 de fevereiro de 2021
Lançado o SiteWise conector IoT versão 10	A versão 10 do SiteWise conector IoT está disponível. Esta versão resolve problemas de estabilidade com o StreamManager cliente quando a conexão é perdida e melhora o tratamento de OPC-UA valores quando a `SourceTimestamp` está ausente. Use o SiteWise conector de IoT para enviar dados locais de dispositivos e equipamentos para propriedades de ativos na IoT. SiteWise	22 de janeiro de 2021

Lançado o SiteWise conector IoT versão 9	A versão 9 do SiteWise conector IoT está disponíve l. Isso inicia o suporte para destinos de `StreamMan ager` stream personalizados do Greengrass, OPC-UA faixa morta, modo de escaneame nto personalizado e taxa de escaneamento personali zada. Isso também inclui desempenho aprimorado durante as atualizações de configuração feitas a partir do gateway de IoT SiteWise . Use o SiteWise conector de IoT para enviar dados locais de dispositivos e equipamentos para propriedades de ativos na IoT. SiteWise	15 de dezembro de 2020
AWS IoT Lançada a versão 4.0.0 do Device Tester	A versão 4.0.0 do AWS IoT Device Tester for AWS IoT Greengrass 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.	15 de dezembro de 2020

AWS IoT Greengrass snap v1.11	A versão 1.11.0 do AWS IoT Greengrass snap oferece suporte a funções Lambda não conteinerizadas. Como melhor prática, recomenda mos que você sempre execute a versão mais recente.	6 de dezembro de 2020
Lançado o SiteWise conector IoT versão 8	A versão 8 do SiteWise conector IoT está disponíve l. Esta versão melhora a estabilidade quando o conector experimenta conectividade de rede intermitente. Use o SiteWise conector de IoT para enviar dados locais de dispositivos e equipamentos para proprieda des de ativos na IoT. SiteWise	19 de novembro de 2020
O conector do Kinesis Firehose é compatível com o modo Sem contêiner	Você pode usar o parâmetro `IsolationMode` para configurar o modo de conteinerização para o conector.	19 de outubro de 2020
Lançamento do conector de implantação de aplicativo do Docker versão 6	A versão 6 do conector de implantação do aplicativo do Docker do Greengrass está disponível.	18 de setembro de 2020

AWS IoT Greengrass versão 1.11.0 lançada	A versão 1.11.0 do software AWS IoT Greengrass Core está disponível. Essa versão adiciona o atributo de telemetria da integridade do sistema e uma API de verificação de integridade local. O Stream Manager agora pode exportar dados para o Amazon Simple Storage Service (Amazon S3) e IoT. SiteWise Essa versão também contém melhorias no desempenho e correções de erros. Recomendamos que você sempre execute a versão mais recente.	16 de setembro de 2020
Lançado o SiteWise conector IoT versão 7	A versão 7 do SiteWise conector IoT está disponível. Essa versão corrige um problema com as métricas de gateway. Use o SiteWise conector de IoT para enviar dados locais de dispositivos e equipamentos para propriedades de ativos na IoT. SiteWise	14 de agosto de 2020
ServiceNow MetricBase Os conectores Integration, Splunk Integration e Twilio Notifications suportam o modo Sem contêiner	Você pode usar o parâmetro `IsolationMode` para configurar o modo de conteinerização para o conector.	30 de julho de 2020

O conector do SNS é compatível com o modo Sem contêiner	Você pode usar o parâmetro `IsolationMode` para configurar o modo de conteinerização para o conector.	6 de julho de 2020
CloudWatch O conector Metrics suporta o modo Sem contêiner	Você pode usar o parâmetro `IsolationMode` para configurar o modo de conteinerização para o conector.	17 de junho de 2020
AWS IoT Greengrass versão 1.10.2 lançada	A versão 1.10.2 do software AWS IoT Greengrass Core está disponível. Esta versão adiciona a propriedade de configuração do núcleo `mqttOperationTimeout` e contém melhorias de desempenho e correções de bugs. Recomendamos que você sempre execute a versão mais recente.	8 de junho de 2020
Instaladores de machine learning Tensorflow defasados	AWS IoT Greengrass Os instaladores de aprendizado de máquina pré-embalados do Tensorflow foram descontinuados. Os exemplos de machine learning foram atualizados para o Python 3.7.	29 de maio de 2020

Suporte à estrutura de trabalho do Chainer e instaladores de machine learning do Greengrass defasados	AWS IoT Greengrass Os instaladores e downloads pré-empacotados de aprendiza do de máquina para MXNet e DLR foram descontinuados. O suporte à estrutura de trabalho do Chainer e os downloads associados foram descontin uados.	4 de maio de 2020
Lançado o SiteWise conector IoT versão 6	A versão 6 do SiteWise conector IoT está disponível. Esta versão adiciona suporte para CloudWatch métricas e descoberta automática de novas OPC-UA tags. Isso significa que você não precisa reiniciar seu gateway quando as tags mudam para suas OPC-UA fontes. Essa versão do conector requer o gerenciador de streams e o software AWS IoT Greengrass Core v1.10.0 ou superior. Use o SiteWise conector de IoT para enviar dados locais de dispositivos e equipamentos para propriedades de ativos na IoT. SiteWise	29 de abril de 2020

Conectores atualizados para Python 3.7	Conectores que suportam o runtime Python foram atualizados para Python 3.7. Recomendamos que você atualize as versões do conector do Python 2.7 para o Python 3.7.	29 de abril de 2020
A configuração do dispositivo do Greengrass pode ser executada no modo silencioso	Você pode executar a configuração do dispositivo do Greengrass no modo silencioso para que o script não solicite nenhum valor.	27 de abril de 2020
Novas imagens base do Docker	Você pode baixar imagens do AWS IoT Greengrass Docker criadas em imagens base do Alpine Linux (x86_64, ARMv7l ou AArch64).	23 de abril de 2020
AWS IoT Greengrass versão 1.10.1 lançada	A versão 1.10.1 do software AWS IoT Greengrass Core está disponível. Essa versão contém melhorias no desempenho e correções de erros. Recomendamos que você sempre execute a versão mais recente.	16 de abril de 2020
Novo capítulo de segurança	AWS IoT Greengrass o conteúdo de segurança foi reorganizado, com novas informações adicionadas.	30 de março de 2020

Use o gerenciador de pacotes APT para instalar AWS IoT Greengrass	Em distribuições Debian-based Linux compatíveis, você pode usar `apt` para instalar o software AWS IoT Greengrass Core em seus dispositivos.	26 de fevereiro de 2020
Lançado o SiteWise conector IoT versão 5	A versão 5 do SiteWise conector IoT está disponível. Esta versão corrige um problema de compatibilidade com o software AWS IoT Greengrass Core v1.9.4. Use o SiteWise conector de IoT para enviar dados locais de dispositivos e equipamentos para propriedades de ativos na IoT. SiteWise	12 de fevereiro de 2020
Novo script para configurar rapidamente um dispositivo de núcleo	É possível usar a configuração do dispositivo do Greengrass para configurar seu dispositivo de núcleo em minutos. Além disso, AWS IoT Greengrass agora oferece suporte Node.js às funções Lambda 12.x.	20 de dezembro de 2019

AWS IoT Greengrass versão 1.10.0 lançada	A versão 1.10.0 do software AWS IoT Greengrass Core está disponível. Os novos atributos dessa versão incluem um gerenciador de fluxo, suporte a contêineres com o conector de implantação do aplicativo Docker, funções não conteinerizadas do Lambda que podem acessar recursos de machine learning, suporte a sessões persistentes do MQTT com AWS IoT e tráfego MQTT local por uma porta especificada.	25 de novembro de 2019
Suporte de console para notificações de implantação	Use o EventBridge console da Amazon para criar regras de eventos que são acionadas quando as implantações do seu grupo Greengrass mudam de estado.	14 de novembro de 2019
AWS IoT Greengrass versão 1.9.4 lançada	A versão 1.9.4 do software AWS IoT Greengrass Core está disponível. Essa versão contém melhorias no desempenho e correções de erros. Como melhor prática, recomendamos que você sempre execute a versão mais recente.	17 de outubro de 2019

Suporte por meio do console para gerenciar a função de serviço do Greengrass	Use recursos novos e aprimorados no AWS IoT console para gerenciar sua função de serviço no Greengrass.	4 de outubro de 2019
Suporte por meio do console para gerenciamento de tags no nível de grupo	Você pode criar, visualizar e gerenciar tags para os grupos do Greengrass no console do AWS IoT .	23 de setembro de 2019
Novos conectores de Machine Learning	Use o conector ML Feedback para publicar entradas e previsões do modelo e o conector ML Object Detection para executar um serviço de inferência de detecção de objetos local.	19 de setembro de 2019
AWS IoT Greengrass versão 1.9.3 lançada	A versão 1.9.3 do software AWS IoT Greengrass Core está disponível. Esta versão permite que você instale o software AWS IoT Greengras s Core em distribuições Raspbian em arquiteturas ARMv6l, suporta atualizaç ões OTA na porta 443 com ALPN e contém uma correção de bug para cargas binárias enviadas de funções Lambda do Python 2.7 para outras funções Lambda.	12 de setembro de 2019

AWS IoT Greengrass versão 1.8.4 lançada	A versão 1.8.4 do software AWS IoT Greengrass Core está disponível. Essa versão contém melhorias no desempenho e correções de erros. Se você estiver executando a v1.8.x, recomendamos atualizar para a v1.8.4 ou a v1.9.3. Para versões anteriores, recomendamos atualizar para a v1.9.3.	30 de agosto de 2019
AWS IoT Greengrass versão 1.9.2 lançada com suporte para OpenWrt	A versão 1.9.2 do software AWS IoT Greengrass Core está disponível. Esta versão permite que você instale o software AWS IoT Greengrass Core em OpenWrt distribuições com arquiteturas Armv8 (AArch64) e ARMv7l.	20 de junho de 2019
AWS IoT Greengrass versão 1.8.3 lançada	A versão 1.8.3 do software AWS IoT Greengrass Core está disponível. Essa versão contém melhorias gerais no desempenho e correções de erros. Se você estiver executando a v1.8.x, recomendamos atualizar para a v1.8.3 ou a v1.9.2. Para versões anteriores, recomendamos atualizar para a v1.9.2.	20 de junho de 2019

AWS IoT Greengrass versão 1.9.1 lançada	A versão 1.9.1 do software AWS IoT Greengrass Core está disponível. Esta versão contém uma correção de bug para mensagens AWS IoT que contêm um caractere curinga no tópico.	10 de maio de 2019
AWS IoT Greengrass versão 1.8.2 lançada	A versão 1.8.2 do software AWS IoT Greengrass Core está disponível. Essa versão contém melhorias gerais no desempenho e correções de erros. Se você estiver executando a v1.8.x, recomendamos atualizar para a v1.8.2 ou a v1.9.0. Para versões anteriores, recomendamos atualizar para a v1.9.0.	2 de maio de 2019
AWS IoT Greengrass versão 1.9.0 lançada	Novos recursos: Suporte para tempos de execução Node.js Lambda do Python 3.7 e 8.10, conexões MQTT otimizadas e suporte à chave Elliptic Curve (EC) para o servidor MQTT local.	1º de maio de 2019

AWS IoT Greengrass versão 1.8.1 lançada	A versão 1.8.1 do software AWS IoT Greengrass Core está disponível. Essa versão contém melhorias gerais no desempenho e correções de erros. Como melhor prática, recomendamos que você sempre execute a versão mais recente.	18 de abril de 2019
AWS IoT Greengrass snap disponível no snapcraft	Use o aplicativo AWS IoT Greengrass Snap Store para projetar, testar e implantar software rapidamente em dispositivos Linux com AWS IoT Greengrass.	1 de abril de 2019
Suporte a mais controle de acesso usando permissões baseadas em tags	Você pode usar tags nas políticas AWS Identity and Access Management (IAM) para controlar o acesso aos seus AWS IoT Greengrass recursos.	29 de março de 2019
Lançamento do conector de análises do IoT	Use o conector de análises do IoT para enviar dados de dispositivos locais para canais do AWS IoT Analytics .	15 de março de 2019
Suporte em lote no conector do Kinesis Firehose	O conector do Kinesis Firehose permite o envio de registros de dados em lote ao Amazon Data Firehose em um intervalo especificado.	15 de março de 2019

CloudFormation suporte para AWS IoT Greengrass recursos	Use CloudFormation modelos para criar e gerenciar AWS IoT Greengrass recursos.	15 de março de 2019
AWS IoT Greengrass versão 1.8.0 lançada	Novos recursos: identidade de acesso padrão configurá vel para funções Lambda, suporte para tráfego HTTPS pela porta 443 e IDs de cliente nomeados previsivelmente para conexões MQTT com. AWS IoT	7 de março de 2019
AWS IoT Greengrass versões 1.7.1 e 1.6.1 lançadas	As versões 1.7.1 e 1.6.1 do software AWS IoT Greengras s Core estão disponíveis. Essas versões exigem o kernel do Linux versão 3.17 ou posterior. Recomendamos que os clientes que executam qualquer versão do software de núcleo do Greengrass atualizem para a versão 1.7.1 imediatamente.	11 de fevereiro de 2019
SageMaker Tempo de execução de aprendizado profundo AI Neo	O tempo de execução de aprendizado profundo SageMaker AI Neo oferece suporte a modelos de aprendizado de máquina que foram otimizados pelo compilador de aprendizado profundo SageMaker AI Neo.	28 de novembro de 2018

Executar AWS IoT Greengrass em um contêiner Docker	Você pode executar AWS IoT Greengrass em um contêiner Docker configurando seu grupo Greengrass para ser executado sem conteiner ização.	26 de novembro de 2018
AWS IoT Greengrass versão 1.7.0 lançada	Novos atributos: conectore s do Greengrass, Secrets Manager local, configurações de isolamento e permissão para funções do Lambda, raiz de hardware de segurança confiável, conexão usando ALPN ou proxy de rede e suporte a Raspbian Stretch.	26 de novembro de 2018
AWS IoT Greengrass downloads de software	Os pacotes de software AWS IoT Greengrass Core, AWS IoT Greengrass Core SDK e AWS IoT Greengrass Machine Learning SDK estão disponíve is para download na Amazon. CloudFront	26 de novembro de 2018
AWS IoT Testador de dispositi vos para AWS IoT Greengrass	Use o AWS IoT Device Tester AWS IoT Greengrass para verificar se a arquitetura da CPU, a configuração do kernel e os drivers funcionam com. AWS IoT Greengrass	26 de novembro de 2018

AWS CloudTrail registro para chamadas AWS IoT Greengrass de API	AWS IoT Greengrass 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.	29 de outubro de 2018
Support para TensorFlow v1.10.1 no NVIDIA Jetson TX2	A biblioteca TensorFlow pré-compilada do NVIDIA Jetson TX2 que AWS IoT Greengrass fornece agora usa a versão v1.10.1. TensorFlow Assim, há suporte ao Jetpack 3.3 e CUDA Toolkit 9.0.	18 de outubro de 2018
Suporte a recursos de Machine Learning do MXNet v1.2.1	AWS IoT Greengrass suporta modelos de aprendizado de máquina treinados usando o MXNet v1.2.1.	29 de agosto de 2018
AWS IoT Greengrass versão 1.6.0 lançada	Novos atributos: executáveis do Lambda, fila de mensagens configurável, intervalo de repetição de reconexão configurável, recursos de volume em /proc e diretório de gravação configurável.	26 de julho de 2018

Atualizações anteriores

A tabela a seguir descreve mudanças importantes no Guia do AWS IoT Greengrass desenvolvedor antes de julho de 2018.

Alteração	Descrição	Data
AWS IoT Greengrass Lançada a versão 1.5.0	Novos atributos: • Interferência de aprendizagem de máquina local usando modelos treinados de nuvem. Para obter mais informações, consulte Executar a inferência de machine learning. • As funções do Lambda para Greengrass oferecem suporte a dados de entrada binários, além do JSON. Para obter mais informações, consulte Versões do núcleo do AWS IoT Greengrass.	29 de março de 2018
AWS IoT Greengrass Lançada a versão 1.3.0	Novos atributos: • Over-the-air Agente de atualização (OTA) capaz de lidar com trabalhos de atualização do Greengrass implantados na nuvem. Para obter mais informações, consulte Atualizações OTA de AWS IoT Greengrass Software principal. • Acessar periféricos e recursos locais das funções do Lambda do Greengrass. Para obter mais informações, consulte Acesso aos recursos locais com funções e conectores do Lambda.	27 de novembro de 2017
AWS IoT Greengrass Lançada a versão 1.1.0	Novos atributos: • Redefina os AWS IoT Greengrass grupos implantados. Para obter mais informações, consulte Redefinir implantações. • Support para tempos de execução Lambda Node.js 6.10 e Java 8, além do Python 2.7.	20 de setembro de 2017
AWS IoT Greengrass Lançada a versão 1.0.0	AWS IoT Greengrass está geralmente disponível.	7 de junho de 2017